Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Espaços seguros de tuplos partilhados para redes de sensores sem fios

Amável Martins Santo (26546)

Orientador: Doutor Henrique João Lopes Domingos
Arguente: Doutor Alysson Neves Bessani
Vogal: Doutora Margarida Paula Neves Mamede

Lisboa
(Julho de 2010)

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Espaços seguros de tuplos partilhados para redes de sensores sem fios

Amável Martins Santo (26546)

Lisboa
(Julho de 2010)

Aos meus pais, por toda a educação, estabilidade e apoio.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao meu orientador, Henrique João Domingos, pela oportunidade de realização desta dissertação, pelas suas ideias, conselhos e pela confiança demonstrada no meu trabalho.

Deixo também uma palavra de apreço a todo o Departamento de Informática da FCT-UNL, a minha segunda casa ao longo dos últimos anos, por me ter dado os conhecimentos fundamentais à realização desta dissertação.

Não posso também deixar de lembrar todos os colegas com que trabalhei e partilharam comigo as experiências de faculdade, muito especialmente o meu colega e amigo Miguel Almeida, parceiro de muitas batalhas travadas ao longo do curso.

Por último, gostaria de agradecer à minha família e amigos, por todo o apoio nos bons e maus momentos.

Resumo

As redes de sensores sem fios (RSSF) constituem hoje uma tecnologia emergente, tendo suscitado grande interesse de investigação, experimentação e utilização em diferentes tipos de contextos. De um modo geral, estas redes possibilitam a monitorização de propriedades do ambiente de forma autónoma, permitindo diminuir ou mesmo evitar a necessidade de intervenção humana.

No entanto a estruturação de suportes de comunicação e desenvolvimento de aplicações, cada vez mais complexas e críticas do ponto de vista de segurança, em bases reutilizáveis de serviços, ainda apresenta muitos aspectos em aberto na investigação actual. Apesar de já existirem protocolos para estas redes, que garantem comunicação ou encaminhamento seguro, a sua combinação com serviços de endereçamento a grupos lógicos de nós sensores que possuam um certo estado ainda não está muito desenvolvida.

Este trabalho apresenta uma plataforma middleware na forma de pilha de serviços, que disponibiliza um ambiente de programação baseado na noção de espaços seguros de tuplos partilhados, tendo subjacente garantias de comunicação, encaminhamento e agregação segura de dados. Estas garantias de segurança visam combater diferentes tipologias de ataques e falhas que podem ser originadas quer por ataques externos à rede, quer por ataques internos arbitrários, por intrusão ao nível dos nós.

A implementação e validação da plataforma foram efectuadas sobre um ambiente de simulação com suporte à norma de comunicação *IEEE 802.15.4*, parametrizado com as características de sensores do tipo *MicaMote*, com escala de milhares de nós. Verificaram-se excelentes resultados na cobertura da rede por canais seguros, baixo impacto energético dos serviços de segurança adicionados e alta fiabilidade nas operações da rede e resiliência face a falhas de nós durante o encaminhamento.

Palavras-chave: redes de sensores sem fios (RSSF), espaços de tuplos, vizinhanças lógicas, encaminhamento seguro, agregação segura, middleware.

Abstract

Wireless sensor networks (WSN) are today an emerging technology, attracting strong interest in research, testing and use in different contexts. In general, these networks enable the monitoring of properties of the environment autonomously, allowing to reduce or even avoid the need for human intervention.

However, communication support and application development structuring, increasingly complex and critical from the standpoint of security in terms of reusable services, still presents many open issues in current research. Although protocols for these networks exist, ensuring both secure communication and routing, its combination with addressing services of logical groups of sensor nodes which have a certain state, is still not very developed.

This work presents a middleware service platform in the form of a services stack, providing a programming environment based on the concept of secure shared tuple spaces, having underlying guarantees of communication, secure routing and aggregation of data. These security guarantees aim to combat different typologies of attacks and failures which can be caused either by external attacks to the network, either by internal arbitrary attacks through intrusion at the level of nodes.

The implementation and validation of the platform were made on a simulation environment, with support for *IEEE 802.15.4* communication standard, parameterized with the characteristics of sensors of type *MicaMote*, scaled for thousands of nodes. Excellent results were achieved in terms of network secure channels coverage, low energy impact of the security services added and high reliability in network operations and resilience to failures of nodes during routing.

Keywords: wireless sensor networks (WSN), tuple spaces, logical neighborhoods, secure routing, secure aggregation, middleware

---

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

1

Introdução

1.1 Contexto

Nos útimos anos os avanços significativos na micro-electrónica, na comunicação sem fios e na tecnologia de miniaturização permitiram o desenvolvimento de redes de sensores sem fios (RSSF), as quais são formadas por centenas ou milhares de pequenos nós que podem chegar às dezenas de milímetros cúbicos. Comunicam entre si por rádio através da norma *IEEE 802.15.4* e possuem sensores embutidos capazes de recolher amostras de propriedades do ambiente (e.g. temperatura, pressão), e coordenando-se entre si de forma a alcançar um dado objectivo [ASSC02, SMZ07]. Estes nós possuem no entanto enormes restrições em termos de recursos computacionais: energia disponível limitada e finita, poucas dezenas de metros de alcance rádio e sujeito a interferências e reflexões, processador de baixíssima frequência e limitações de memória.

Devido às restrições mencionadas, existe a necessidade das aplicações e protocolos desenvolvidos para esses aparelhos tentarem optimizar ao máximo o trabalho útil, optimizando o processamento e principalmente a comunicação, pois é esta última a operação que mais energia gasta. Normalmente as RSSF são constituídas por uma estação central na periferia (usualmente chamada *sink-node* [1]) e nós homogéneos no seu interior. O *sink-node* muitas vezes encontra-se ligado a uma rede exterior tal como a *Internet* e está isento das restrições hardware acima mencionadas, recebendo dados e

[1] Ao longo do presente trabalho podem ser utilizados para o mesmo significado qualquer um dos termos *sink-node* ou estação central.

enviando comandos para a rede. Já os nós homogéneos, dependendo das aplicações a correr na rede, podem-se auto-organizar segundo uma dada topologia, possuindo cada nó iguais capacidades de processamento e memória, apesar dos sensores embutidos no dispositivo poderem medir diferentes propriedades do ambiente. No entanto, existem também casos de redes constituídas por nós heterogéneos que não obedecem estritamente à ideia anterior e onde podem existir nós actuadores [SCV+06], que têm como única função desempenhar uma acção física (e.g. soar um alarme) resultado dum evento ou comando, ou então nós especiais agregadores com mais capacidades computacionais.

Durante o ciclo de funcionamento, os sensores monitorizam e podem pré-processar eventos, de acordo com condições específicas programáveis. Devido ao curto alcance de comunicação, os dados são encaminhados de nó em nó podendo mesmo ser sujeitos a agregações ou processamento intermédio até chegarem à estação central, de forma a serem finalmente processados.

São assim inúmeras as aplicabilidades das RSSF indo desde monitorização de habitats [MCP+02], passando por monitorização da saúde de estruturas ou equipamento [CFP+06], ou até mesmo vigilância num campo de batalha [Vig].

1.2 Motivação

Ao contrário das redes tradicionais de computadores, onde através da pilha OSI os serviços estão bem definidos e estruturados, nas RSSF tal não acontece pois a natureza da rede é significativamente diferente. A comunidade que gere os protocolos para as RSSF ainda não chegou a um consenso para normalização dos serviços necessária à comunicação segura e coordenação entre os nós, estruturando-os de forma semelhante às redes anteriores. Contudo a comunidade científica tem feito propostas de estruturação de serviços aos diversos níveis desta pilha mas tendo como motivação unicamente famílias de aplicações específicas. Ainda estamos portanto longe daquela visão de estruturação de middleware genéricos de ampla base reutilizável, que podem ser usados por qualquer aplicação. Nas RSSF, de momento apenas uma parte desta pilha está normalizada, as camadas *física*, que consiste essencialmente no hardware do nó, na sua bateria e nos seus sensores, e *enlace*, responsável entre outras coisas pela divisão justa do meio de comunicação sem fios entre os vários nós, de onde faz parte a sub-camada MAC. No entanto tudo é muito dependente da aplicação e inclusive a gestão do ciclo de vida do processador é ainda de muito baixo nível. O desejável seria agora adicionar uma camada de rede, onde constem serviços como encaminhamento seguro, e uma camada de middleware, que possuísse por exemplo lógicas de organização da rede e

agregação segura de dados.

Tal como já foi dito anteriormente, na visão mais usual das RSSF, estas baseiam-se numa arquitectura centralizada, tendo por base uma estação central que pode transmitir instruções e recolher dados da rede, tendo os nós iguais capacidades de amostragem, pré-processamento e recolha de dados. No entanto a evolução das RSSF deu origem a novas arquitecturas descentralizadas, onde são usadas várias estações centrais e nós heterogéneos. Estes tanto são capazes de recolher diferentes tipos de dados do ambiente, como realizar um variado número de acções físicas. Devido a esta evolução, como forma de facilitar o endereçamento por parte de um nó a um conjunto de outros que partilhem simultaneamente certos atributos ou propriedades e assim facilitar operações, tais como encaminhamento ou agregação de dados sobre um conjunto de nós, torna-se pois fulcral a introdução na rede da noção de grupos lógicos ou vizinhança lógica (VL) [MP06].

Uma VL consiste num agrupamento de nós que partilham características ou propriedades em comum (e.g. acções que podem desencadear, localização, temperaturas de igual valor registadas) e que se constituem com base em critérios funcionais ou de natureza semântica ao nível das aplicações. Alguns sistemas baseados em VLs estão associadas à noção de estabelecimento de espaços de tuplos partilhados [MPR06, Gel85], que são vistos pelas aplicações como repositórios de dados que disponibilizam, aos membros de cada VL, primitivas básicas para escrita, leitura e acesso a dados representados sob a forma de tuplos. Através desta noção é possível atingir um maior grau de transparência e diminuir o esforço de construção das aplicações em matéria de endereçamento e de quaisquer interacções em grupo, quer seja entre nós no interior da rede (de forma autónoma), quer seja a partir de pedidos do exterior.

As VLs podem também elas sofrer ataques do exterior ou mesmo do interior da rede, através de nós maliciosos (que possam ter sido capturados a partir de ataques por intrusão), assim como devido a processamento incorrecto provocado por falhas arbitrárias. Há por isso que arranjar mecanismos e serviços de segurança que as protejam, e que permitam estender a noção de VL para um conceito de vizinhança lógica segura, onde as propriedades cobrem também requisitos de segurança implícitos ao suporte de agrupamento e disseminação de dados, ao nível dos serviços de segurança suportados na arquitectura *middleware* subjacente às aplicações.

A investigação anterior das RSSF tem proposto alguns sistemas que são implementações de VLs, como é o caso do TeenyLime [CMMP06], e que podem ser instalados nos sensores. Estas propostas visam estabelecer a noção de VL como um paradigma baseado na noção de estabelecimento de espaços de tuplos partilhados [MPR06] por nós de uma dada VL. No entanto, nenhum destes sistemas contempla a possibilidade

da rede poder ser alvo de ataques ou falhas e por isso não possui quaisquer serviços de segurança para fazer face a estes problemas. Esta é assim uma área de inovação, com enormes potencialidades de desenvolvimento, através de novos estudos que levem à integração do paradigma de espaços de tuplos para suporte de vizinhanças lógicas seguras através de novos protocolos seguros de encaminhamento, difusão e agregação de dados, combinados a diferentes níveis da pilha de serviços das RSSF. A ideia central nesta abordagem é disponibilizar paradigmas e ambientes de programação em que o programador possa definir vizinhanças lógicas seguras suportadas em espaços de tuplos seguros, partilhados com base em propriedades de segurança utilizadas de forma transparente às aplicações e garantidas pelo empilhamento de arquitecturas de *middleware* para redes de sensores seguras.

1.3 Solução implementada

Verificam-se três lacunas principais nas RSSF hoje em dia. Em primeiro lugar, os serviços não estão estruturados numa pilha de maneira a serem reaproveitados pelas mais diversas aplicações, levando assim a que a construção de cada uma delas exija bastante tempo e linhas de código. Em segundo lugar, nota-se que existem ainda poucos protocolos que garantam segurança e os que existem nem sempre dão grandes garantias, fazendo com que muitas das vezes o bom funcionamento de protocolos de encaminhamento, difusão ou agregação seja quebrado. Por último, ainda não existe grande suporte para realização de operações numa RSSF sobre um conjunto de nós em simultâneo, segundo a noção de grupo lógico, o que facilitaria bastante o seu endereçamento.

Ora, tentando colmatar os problemas anteriores, esta dissertação apresenta uma pilha estruturada segundo as mais recentes propostas da investigação, de forma a que cada camada tenha um enorme grau de transparência e independência relativamente a qualquer outra, sendo assim fácil a sua substituição caso haja necessidade. Esta pilha é constituída segundo uma visão ascendente pelas seguintes camadas:

Comunicação segura garante os requisitos básicos de segurança, confidencialidade, autenticidade, integridade e não-replicação de dados;

Encaminhamento seguro garante, segundo uma aproximação centrada nos dados, que as pesquisas e retorno dos mesmos são efectuados de forma segura, contornando eventuais rotas que tenham falhado, auto-reorganizando-se os nós nessa situação;

Camada de infraestrutura onde existem serviços de comunicação e agregação de dados entre as aplicações, sendo que os nós destino das operações se organizam em grupos segundo a noção de vizinhança lógica e a comunicação é efectuada segundo a noção de espaços de tuplos. Esta camada exporta uma API que as aplicações finais usarão, invocando para isso as suas primitivas.

Para construção desta plataforma foi usado o ambiente Java, sendo que os protocolos existentes que serviram de base para cada uma das camadas foram implementados de raiz nesta linguagem. Até aqui apenas estavam implementados em linguagens de baixo nível para sensores, nomeadamente em *nesC*, e para as quais não havia simuladores capazes de testar a escalabilidade ou a correcção da implementação.

No final testou-se num simulador o comportamento da rede segundo critérios de cobertura segura de nós da rede, gastos energéticos de operações e contribuições de cada camada para os mesmos, e de fiabilidade de entrega de mensagens e resiliência contra falhas de encaminhamento.

Com esta implementação procurou-se provar, acima de tudo, que é possível garantir segurança de forma viável e com baixos custos energéticos a vários níveis da pilha, combatendo assim a grande maioria dos ataques, exportando ao mesmo tempo uma API com serviços às aplicações, diminuindo dessa forma o seu esforço de construção.

1.4 Principais contribuições

O trabalho realizado na presente dissertação consistiu na concepção, implementação e teste de uma plataforma modular, baseada num conjunto de serviços seguros, genéricos e adaptativos, disponibilizando-os através de uma API que pode ser invocada pelas aplicações, sendo a organização dos nós da rede baseada no conceito de VL. Através destes serviços e deste conceito, pensa-se ter diminuído o esforço de construção e gestão das aplicações, tornando-as mais seguras e facilitando assim a vida ao programador e utilizador final.

Mais concretamente, os serviços principais implementados foram os seguintes:

i) **Comunicação segura** - garantindo essencialmente que as mensagens a circular na rede permanecem confidenciais, íntegras e não são replicadas;

ii) **Difusão e encaminhamento seguros de dados** - garantindo que os dados que são difundidos para a rede, ou as rotas e resultados associados a um certo fluxo de dados não são sujeitos a adulteração ou corrupção;

iii) **Agregação segura de dados** - garantindo que é possível efectuar agregação segura e processamento interno na rede junto às fontes de dados, de forma a diminuir o tráfego da rede;

iv) **Combate contra intrusões** - permitindo que, mesmo que um atacante consiga aceder a um nó fisicamente e aceda aos seus segredos, com isso não consegue ganhar controlo de uma parte da rede;

v) **Organização da rede através de VLS** - onde é possível agrupar os nós da rede e operar sobre eles de forma segura, conforme uma certa especificação baseada no conceito de vizinhança lógica segura.

1.5 Organização do documento

Após este capítulo introdutório, o documento encontra-se organizado da seguinte forma: no capítulo 2 apresenta-se e analisa-se algum do trabalho relacionado efectuado no âmbito deste trabalho, no capítulo 3 consta a arquitectura da plataforma, no capítulo 4 apresentam-se em detalhe os aspectos da sua implementação, no capítulo 5 são apresentados os testes e discutidos os seus resultados no âmbito da sua validação e, finalmente, o capítulo 6 apresenta as conclusões deste trabalho e ideias para trabalho futuro.

2

Trabalho relacionado

Um dos grandes objectivos para o futuro das RSSF é a criação e estabilização de módulos de serviços estruturados numa pilha de tal forma que possam ser reutilizados pelas aplicações. Estes serviços têm de assegurar segurança nas comunicações e encaminhamento dentro das redes, de forma a fazer face a ataques por parte de alguém mal intencionado ou devido a falhas arbirárias dos nós. Isso levará a que o esforço inerente à construção, teste e instalação das aplicações diminua. Nas próximas secções do presente capítulo, apresentam-se e discutem-se muitas das soluções apresentadas hoje em dia para RSSF, no que toca à comunicação, encaminhamento, plataformas existentes, processamento interno, agregação e a segurança associada a cada um destes aspectos.

2.1 Comunicação em RSSF

Ao contrário das redes tradicionais de computadores, as RSSF possuem enormes restrições de energia, espaço em memória ou comunicação. Devido a esta diferente natureza, os serviços não podem ser implementados da mesma forma, nem darem as mesmas garantias, assim como a suposta estruturação dos mesmos numa pilha não pode ter as mesmas características. Assim apresenta-se nesta secção uma pilha de referência para as RSSF, as diferenças face à pilha das redes tradicionais e os ataques que a mesma pode sofrer ao nível das várias camadas.

2.1.1 Pilha de referência em RSSF

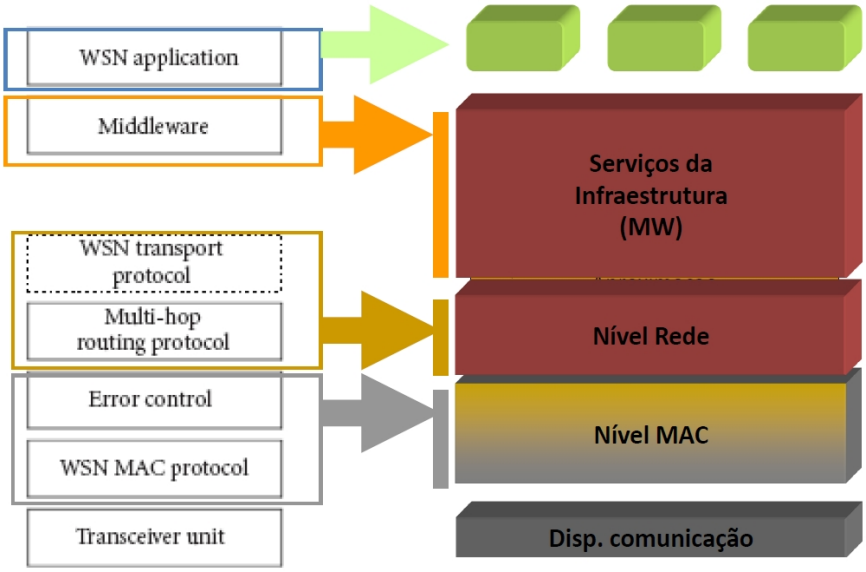


Figura 2.1: Pilha de uma RSSF.

Até aos dias de hoje os serviços para RSSF têm sido implementados juntamente com as próprias aplicações, ou reaproveitados apenas para certas famílias de aplicações com a mesma natureza. No futuro deseja-se que estes possam ser reaproveitados pelas mais diversas aplicações sem haver a necessidade de qualquer alteração às suas implementações. Tem havido no entanto nos últimos anos várias propostas da comunidade científica para a normalização destes serviços. Estas propostas estão a convergir para o modelo que se apresenta na Fig. 2.1, do qual se passa a descrever em pormenor as várias camadas e os principais serviços que implementam.

2.1.1.1 Camada Física

Esta camada é responsável pela selecção da frequência rádio, detecção de sinal e modulação e codificação de dados. Ao nível desta camada surgem os problemas relacionados com a propagação de sinal no meio e incorrecto desenho hardware levando a gastos energéticos suplementares.

2.1.1.2 Camada MAC

Esta camada gere a multiplexagem de sequências de dados, acesso ao meio de comunicação e controlo de erros. No entanto o controlo de erros ainda não está completamente desenvolvido, acarretando um significativo *overhead* na comunicação. Além

disso, comparativamente às redes convencionais, necessita ter em conta critérios adicionais na gestão do canal em termos de poupança energética, o qual é ainda muito rudimentar.

2.1.1.3 Camada de Rede

Esta camada trata do encaminhamento dos dados, reconfiguração topológica da rede na presença de falhas de comunicação e agregação de dados. Está no entanto em aberto na área de investigação o melhoramento dos protocolos de encaminhamento de forma a levar em conta critérios energéticos para escolha óptima de rotas de forma a prolongar a vida útil da rede.

2.1.1.4 Camada de Infraestrutura

Esta camada fornece uma transparência às aplicações para os níveis inferiores, exportando um conjunto de primitivas que as mesmas podem usar de forma a configurar a rede de acordo com os seus requisitos. Através dessa API as aplicações podem, entre outras coisas, escolher qual a topologia da rede e o algoritmo de encaminhamento de dados a utilizar. Podem ainda possuir serviços internos tais como sincronização dos relógios ou localização espacial.

2.1.2 Modelo de adversário

Devido à sua natureza própria, as RSSF estão sujeitas aos mais diversos tipos de ataques por parte de alguém mal intencionado. Estes ataques podem ser classificados em diversas categorias tal como se pode ver em [KW03]. Para o que se deseja explicar, consideremos apenas a categoria que engloba as classes de *ataques externos* e *ataques internos*:

Ataques externos são ataques lançados a partir de nós ou estações externas à rede, as quais não possuem chaves nem outros segredos necessários para participar de forma legítima nas operações da rede. Podem consistir somente na monitorização do tráfego da rede, obtendo o seu conteúdo e possivelmente os segredos através de técnicas de força bruta ou criptanálise. No entanto, caso sejam usados segredos seguros, com chaves de tamanho razoável, pode nunca ser possível em tempo útil a um atacante obter os mesmos através dessas técnicas. Esta classe de ataques está intimamente ligada ao modelo adversarial *Dolev-Yao* [DY83];

Ataques internos são ataques que ocorrem quando um nó participante na rede foi capturado, assim como os seus segredos. Uma vez na posse destes segredos o

atacante pode lançar ataques quer sejam a partir dos nós corrompidos que sejam posteriormente adicionados à rede, quer sejam a partir de aparelhos com mais alta capacidade. Os ataques internos são muito mais difíceis de defender relativamente aos externos devido ao facto do atacante agora ter na sua posse segredos da rede e poder controlar uma parte da mesma.

Intimamente ligados com a categoria anterior estão os modelos de adversário a que uma RSSF pode ter que fazer face. Existem três grandes modelos que têm que ser tidos em consideração quando se desenham os protocolos para estas redes, de modo a não nos deparamos com a rede completamente corrompida. Estes vão desde o modelo mais simples, *Dolev Yao* [DY83], passando pelos modelos *Bizantinos* [LSP82], até aos mais difíceis de combater, os modelos *Manets* [CW09]:

Modelo Dolev-Yao neste modelo o adversário é capaz de ler mensagens anónimas a circular na rede, adulterá-las e voltar a injectá-las provocando assim quebras no normal funcionamento dos protocolos, ou corrompendo dados. Mais concretamente o adversário pode apenas efectuar ataques ao nível do canal de comunicação. A maioria destes ataques podem ser facilmente protegidos através de criptografia simétrica;

Modelos Bizantinos na génese deste modelo está que um comportamento incorrecto de um nó não permite distinguir se foi um ataque ou apenas uma falha. De modo a combater estes ataques tem que se prever todas as possíveis falhas que um nó pode ter e incorporar isso nos protocolos, o que não é de todo possível. Assim o atacante pode capturar nós legítimos, acedendo aos segredos, injectando-os novamente com comportamentos incorrectos arbitrários sem que se descubra;

Modelos Manets este modelo engloba os dois anteriores e onde o atacante pode ainda efectuar outros tipos de ataques mais poderosos, próprios das RSSF[KW03], replicando os ataques por vários nós adulterados.

2.1.3 Ataques em RSSF

As RSSF podem ser alvo de ataques a vários níveis da pilha apresentada anteriormente na subsecção 2.1.1. Passam-se assim a descrever brevemente os ataques mais relevantes a cada camada.

2.1.3.1 Ataques na camada Física

A este nível da pilha os ataques consistem essencialmente em ataques do tipo *denial of service*, onde é emitido um sinal na mesma frequência da rede, fazendo com que os

sensores não consigam comunicar, ataques por *exaustão de bateria*, onde são emitidos pacotes inúteis para os sensores de forma a que estes gastem a bateria a tentarem lê-los, ou então *roubo de sensores*, onde o atacante acede à sua memória, onde constam entre outras coisas os segredos usados na rede.

2.1.3.2 Ataques na camada MAC

A este nível existem ataques específicos ao próprio funcionamento do protocolo MAC descrito em pormenor em [BJJ06]. Além disso existem ataques à comunicação, os quais se passam a descrever:

Descarte de mensagens onde um nó corrompido descarta silenciosamente mensagens úteis ao funcionamento da rede sem que outros nós se apercebam;

Replicação de mensagens onde um atacante na posse de uma mensagem anterior que circulou na rede, volta a colocá-la em trânsito provocando replicação de acções possivelmente desastrosas;

Impersonificação onde um nó se faz passar por outro e comunica em seu nome com os demais da rede;

Adulteração de mensagens onde o atacante captura uma mensagem, altera-lhe o seu conteúdo e reencaminha-a para os destinatários.

2.1.3.3 Ataques na camada de Rede

Nesta camada podem existir diversos tipos de ataques com características diferentes tal como apresentado em pormenor em [YY06]. Estes ataques podem afectar as fases de descoberta, formação ou estabelecimento de rotas, bem como a fase de manutenção posterior do mecanismo de encaminhamento, assim como a correcção dos dados que são transmitidos ao longo da rede. Passam-se a descrever os ataques mais relevantes:

Spoofing de informações de encaminhamento tendo como alvo os pacotes de controlo responsáveis pelas informações de encaminhamento, através de repetições ou modificação desses pacotes, este ataque pode criar ciclos na rede, atrair ou repelir tráfego, gerar mensagens de erro de rotas falsas, dividir a rede, entre outros danos;

Encaminhamento selectivo acontece quando um nó malicioso se recusa a reencaminhar todos ou determinados pacotes vindos dos seus vizinhos, descartando-os,

não colaborando assim com a rede. Este tipo de ataque pode ser realizado somente sobre algumas rotas seleccionadas e pode ser mais incisivo se o nó pertencer a uma rota principal de transmissão de dados;

Ataque Sinkhole este ataque acontece quando o tráfego é desviado para um determinado nó malicioso. Os nós vizinhos ou o próprio nó podem manipular os dados modificando-os. Esse tipo de ataque acontece porque os adversários podem capturar e alterar as mensagens de encaminhamento. Um nó pode assim tornar-se atraente aos nós vizinhos fazendo parte das suas rotas. Se a métrica utilizada para escolha do próximo nó for o número de saltos, um computador de maior poder e com maior potência de transmissão pode, por exemplo, indicar apenas um salto ao destino e assim atrair todo o tráfego para si;

Ataque Sybil para resistir a determinadas ameaças, alguns sistemas aplicam redundâncias ao nível de rotas, caso alguma seja comprometida. Um nó, conhecendo essas características, pode apresentar múltiplas identidades e assim fazer-se passar por outros controlando grande parte da rede. Quanto maior é o poder de alcance do nó, mais efectivo pode ser o seu ataque, pois a sua área de influência aumenta;

Ataque Wormhole este ataque consiste num túnel criado pelo atacante. As mensagens entram nesse túnel numa parte da rede e são propagadas até uma outra parte normalmente com uma latência menor. Um *wormhole* para ser instanciado necessita de dois nós, as extremidades do túnel, sendo que cada um deles tenta convencer os vizinhos que o túnel é o melhor caminho através da transmissão de pacotes de roteamento com métricas forjadas que façam o túnel mais atraente diante das outras possibilidades;

Ataque por inundação de HELLO muitos protocolos de encaminhamento emitem pacotes especiais de *HELLO* não autenticados entre vizinhos de forma a verificar a conectividade. Um nó malicioso com grande alcance pode enviar pacotes de *HELLO* para qualquer nó da rede. Assim os sensores ao receberem estes pacotes julgam esse nó como vizinho e começam a aceitar as rotas que são anunciadas por ele, dando-se portanto um corrompimento das mesmas e um desgaste energético dos nós.

2.1.3.4 Ataques na camada Infraestrutura

Os ataques nesta camada são essencialmente ataques ao bom funcionamento das aplicações e dos seus serviços, tais como agregação incorrecta de dados, ataques à difusão

de mensagens ou ataques de sincronização de relógios.

2.1.4 Modelo de segurança

Face aos ataques aos vários níveis da pilha apresentados na secção 2.1.3, passam-se agora a descrever brevemente contramedidas para mitigar ou mesmo anular os mesmos para as camadas Física, MAC e de Rede.

2.1.4.1 Contramedidas na camada Física

Nesta camada não há muito a fazer para evitar os ataques descritos. O ideal seria um mecanismo anti-roubo, que uma vez tentado abrir o sensor para lhe retirar os segredos por parte de um atacante, resultaria num reset à memória do sensor em resposta, o que não é de todo viável, pois o atacante pode retirar as baterias retirando energia ao sensor para efectuar tal acção. O ideal será mesmo tentar proteger o terreno onde os sensores estão depositados.

2.1.4.2 Contramedidas na camada MAC

Nesta camada é onde se têm que garantir os requisitos básicos de segurança confidencialidade, autenticidade, integridade e frescura dos dados, mitigando assim os principais ataques. Ora para garantir tais requisitos já foram desenvolvidos módulos com serviços de segurança tais como o TinySec [KSW04], ou ainda melhor para RSSF o Minisec [LMPG07], os quais são adicionados aos sistemas operativos dos sensores.

2.1.4.3 Contramedidas na camada de Rede

Nesta camada é onde se têm que garantir requisitos tais como encaminhamento seguro ou *broadcast* seguro, assim como outros necessários estabelecimentos de chaves para grupos, dependendo da organização da rede. Assim passam-se a descrever alguns mecanismos em mais detalhe e quais os ataques que previnem ou combatem:

Chaves partilhadas com cifra de mensagens previne todos os ataques vindos do exterior, quando o atacante não tem acesso aos segredos. Protege assim dos ataques *Sybil*, *Encaminhamento selectivo* e *Wormhole*. Devem ser usados em conjunto mecanismos de refrescamento de chaves, pois as chaves usadas neste tipo de redes têm comprimento curto;

Verificação de bidireccionalidade caso haja suporte para este mecanismo e se o ataque for efectuado do exterior da rede, é possível eliminar o ataque *Hello Flooding* que normalmente não é combatido com cifra de mensagens;

Bom desenho dos protocolos de encaminhamento permite evitar ataques tais como o *Wormhole* ou *Sinkhole*. No desenho destes protocolos tem de haver suporte para rotas dinâmicas, evitando assim a atracção definitiva de nós para vizinhos maliciosos;

Encaminhamento por caminhos alternativos permite combater o ataque por *Encaminhamento Selectivo*. Caso os nós possuam mecanismos para verificar a confiança dos nós que lhes enviaram dados, então as rotas de nós maliciosos podem ser definitivamente postas de parte;

***Broadcast* autenticado** com este mecanismo pode-se testar a autenticidade dos pacotes enviados pela estação central, ou então de nós do interior da rede. O protocolo μ*Tesla* [PST+02] descrito mais adiante é suficiente para levar a cabo esta verificação.

2.1.5 Análise comparativa

Começámos por ver nesta secção um modelo de estruturação de serviços proposto para RSSF, o qual apresenta bastantes diferenças face aos modelos das redes convencionais. De seguida apresentámos os vários modelos de adversário considerados hoje em dia para estes tipos de redes, onde o modelo *Manets* é o mais difícil de fazer frente com os seus ataques próprios para RSSF, indo muito mais além dos modelos mais simples tal como o modelo *Dolev-Yao*. Vimos depois os vários tipos de ataques aos mais diversos níveis da pilha, sendo as camadas *MAC* e de *Rede* aquelas onde os ataques são mais difíceis de se desmascararem e são mais destrutivos, pois podem resultar em devoluções de resultados errados para a estação base em consequência de uma pesquisa, assim como podem alterar toda a organização da rede, dificultando ou impedindo operações sobre a mesma. Para esses ataques vimos por último várias técnicas que se podem aplicar para mitigar ou mesmo impedir esses ataques recorrendo acima de tudo a técnicas criptográficas ou então recorrendo a rotas alternativas para devolução de dados.

Resta no entanto observar que os protocolos de segurança são bastante difíceis de desenhar para estas redes pois os sensores só suportam técnicas criptográficas simétricas com chaves pequenas, que são mais leves comparativamente às técnicas assimétricas. Assim existe a necessidade de técnicas de refrescamento de chaves que levem a que se consiga manter a rede segura por tempo infinito.

2.2 Difusão e encaminhamento em RSSF

Nesta secção apresentam-se os protocolos relevantes desenvolvidos até hoje para efectuar pesquisas e retornar resultados duma RSSF. Os protocolos de pesquisa inserem-se em duas grandes classes: pesquisa *on demand* [PR99, HPJ05], onde as pesquisas são injectadas na rede através da estação central no momento exacto em que se desejam obter as amostras dos sensores, ou então com base no registo de um *interesse*[1] [IGE00] em todos os nós. Em ambos os casos anteriores as pesquisas e os retornos de amostras é conseguido através de protocolos instalados nos sensores que permitem a sua difusão e encaminhamento e que obedecem às restrições da rede, tendo em conta possíveis falhas e ataques, ou escassez de recursos energéticos.

De seguida apresentam-se alguns protocolos usados hoje em dia nas RSSF para este fim. Serão apresentados, com base no seu método de efectuar pesquisas, encaminhamento e agregação dos dados, tolerância a falhas ou ataques e modo como tratam os aspectos de optimização de consumos de energia.

2.2.1 TinyDB

O *TinyDB* [MFHH03] não é um protocolo de difusão e encaminhamento, mas sim um sistema de processamento de pesquisas *on demand* para extrair informação de uma rede de sensores, tendo na sua génese um protocolo deste género. As pesquisas são enviadas para toda a rede através duma linguagem que é uma variante de SQL e executadas de forma concorrente e distribuída. Os dados consistem numa tabela distribuída de tuplos que vai sendo preenchida de tempos a tempos e onde as colunas representam os vários atributos e valores medidos pelos sensores, sendo que localmente a cada nó coexiste uma sub-parte da tabela. A rede organiza-se segundo uma topologia em árvore, sendo as pesquisas injectadas no nó raiz e reencaminhadas até às folhas.

O **encaminhamento**, devido à topologia da rede, é baseado no reencaminhamento de mensagens de pesquisa para os descendentes de um nó corrente, retornando os resultados pelo caminho inverso até ao nó raiz. Possui um mecanismo, recorrendo a tabelas auxiliares, que impede que as pesquisas sejam reencaminhadas para descendentes que não possuam os dados requeridos.

A **agregação de dados** conta com poderosas primitivas que consistem em pesquisas com uma sintaxe bem definida. A aproximação é a seguinte: à medida que os dados de

[1] Um interesse é formado por um conjunto de atributos a que os sensores têm de obedecer tais como "todas as temperaturas acima dos 60ºC no sector 1", onde neste caso apenas retornarão amostras os sensores de temperatura situados no sector 1 que registem temperaturas acima dos 60ºC.

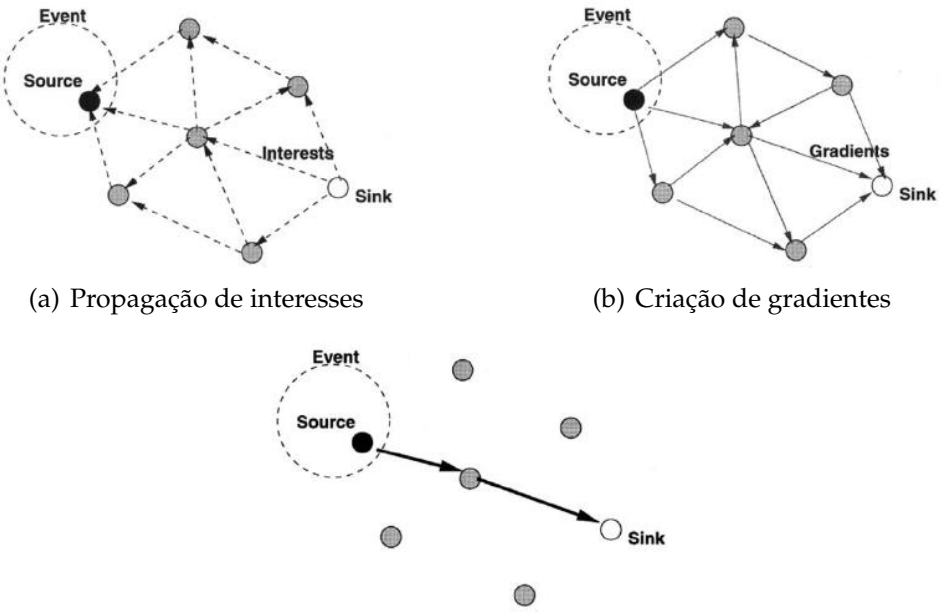
uma pesquisa de agregação são encaminhados ascendentemente na árvore, são agregados em cada nível de acordo com uma função (e.g. soma, média) e lógica de partição da tabela especificadas na pesquisa.

Quanto a **tolerância a falhas ou ataques**, o sistema apenas possui um mecanismo que permite o envio redundante de mensagens por mais que um caminho em simultâneo, de modo a evitar a falha de nós. Além disso a rede reorganiza-se na zona da falha caso seja necessário.

Para **poupança de energia** possui um modelo com mecanismos que adaptam a frequência e reajustam outras propriedades das pesquisas conforme a energia disponível, de modo à extensão do tempo de vida da rede. Além disso os nós estão sincronizados e apenas ficam acordados durante os momentos de recolha de dados e comunicação.

2.2.2 Directed Diffusion

O *Directed Diffusion* (DD) [IGE00] é um protocolo de pesquisas para RSSF baseado no conceito de pesquisas por interesses. É escalável e robusto e permite a coordenação dos sensores de forma a permitir a disseminação de interesses e posterior retorno das amostras de forma distribuída. A troca de mensagens faz-se directamente entre nós vizinhos, mas pode alcançar qualquer zona da rede através de uma estratégia multi-saltos. O protocolo funciona em três fases principais:



(a) Propagação de interesses (b) Criação de gradientes

(c) Retorno de dados

Figura 2.2: Esquema simplificado do *Directed Diffusion*

1) **Propagação de interesses** (Fig. 2.2(a)) - as mensagens de interesse são enviadas a partir de um certo nó (e.g. normalmente a estação central) para toda a rede através de inundação. Cada nó da rede faz cache do interesse, quem o enviou e qual a duração e frequência das amostras a retornar (gradientes) para o mesmo. Periodicamente os interesses são refrescados;

2) **Propagação de dados** (Fig. 2.2(b)) - os resultados de um interesse são retornados pelo caminho inverso a partir dos nós que recolhem as amostras até ao nó que originou o interesse. Os nós intermediários reencaminham os dados com a frequência definida no gradiente mas apenas no caso do interesse constar na sua cache;

3) **Reforço de caminhos** (Fig. 2.2(c)) - onde um qualquer nó, incluindo a estação central, que não deseje/deseje receber as mensagens com dados de um determinado vizinho, reenvia por *unicast* o interesse para o mesmo, mas desta vez com um intervalo de amostragem maior/menor. Inerente a este passo está que os dados são retornados sempre pelo caminho com um valor maior de reforço, correspondendo ao caminho empiricamente melhor segundo uma dada heurística (e.g. de menor delay).

Além disso são guardados em cache os dados dos interesses conhecidos de forma a combater as mensagens de dados replicados. O interesse dos dados recebidos é identificado baseado num identificador que acompanha as mensagens de dados. Com essa informação é possivel efectuar uma simples **agregação de dados** no caso de se receberem amostras com a mesma natureza de vários nós. Quanto a **tolerância a falhas**, o protocolo conta apenas com o facto de receber os dados por caminhos alternativos, apesar de com frequências diferentes e poder escolher qual o caminho de onde recebe certos dados através de reforço, contrariando assim a situação onde por exemplo nós intermediários de um certo caminho falhem. Em termos de **poupança de energia**, não possui qualquer mecanismo directo no protocolo a não ser a possibilidade de agregar dados, diminuindo a quantidade de informação a circular na rede e consequentemente a energia gasta na comunicação.

2.2.3 μTesla

O μ*Tesla* [PST+02] não é um protocolo de encaminhamento, mas sim um protocolo que providencia broadcast autenticado para ambientes com grandes restrições em termos de recursos energéticos, como é o caso de uma RSSF. É usada apenas criptografia simétrica e as mensagens são bastante reduzidas em número e tamanho.

O protocolo assume que o nó emissor (aquele de quem se quer autenticar as mensagens) e os receptores estão fortemente sincronizados temporalmente, mas mesmo

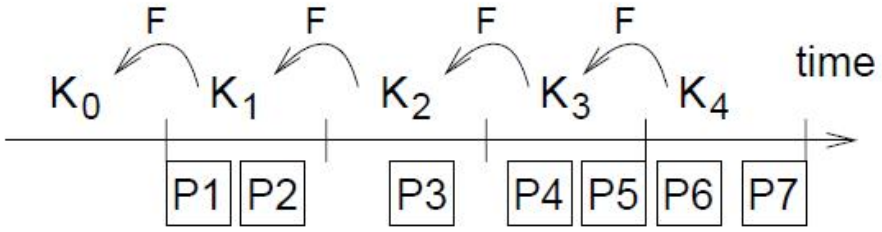


Figura 2.3: Esquema de funcionamento do μTesla.

assim conhecem um valor máximo de erro de sincronização. Assim consegue-se dividir o espaço temporal em épocas, tal como exemplificado na Fig. 2.3. No início de cada uma das épocas o emissor envia uma mensagem com a revogação da chave que permite autenticar todas as mensagens anteriormente enviadas[2], sobre as quais foi na altura gerado um MAC que seguiu na própria mensagem de forma a permitir autenticar posteriormente o emissor. No entanto, caso um nó receptor perca uma mensagem com uma revogação de chave, pode sempre obtê-la a partir de uma chave recebida posteriormente, bastando para isso submeter a mesma a uma função de *hash* de sentido único que gera passo a passo todas as anteriores, autenticando assim as mensagens da época correspondente. Consegue-se pois provar que foi o emissor o nó legítimo que enviou a mensagem, pois mais ninguém além dele conseguia gerar a próxima chave da cadeia.

Este protocolo tem o problema de todas as mensagens de uma dada época terem de ser guardadas em cache até receberem a chave que as autentica, não se podendo assim trabalhar sobre as mesmas durante uma ou várias épocas. Quanto maior a necessidade das mensagens serem rapidamente autenticadas, mais curta tem de ser a duração de uma época, aumentando a frequência das comunicações e gastando-se assim mais energia. Além disso tem a desvantagem do emissor ter de gerar prontamente um número elevado de chaves caso queiramos que todo o processo se prolongue por várias épocas, o que faz com que um nó do interior da rede para ser o emissor no processo terá de gastar energia a gerar bastantes chaves e armazená-las no seu pequeno espaço em memória. No entanto, apesar das suas desvantagens, até ao presente para RSSF este continua a ser o melhor mecanismo de *broadcast* seguro e faz parte da génese de muitos outros protocolos, dos quais fazem parte o *Secure Directed Diffusion* que a seguir se apresenta.

[2]Na imagem anterior um nó receptor ao receber K2 no inicio da época 3, consegue autenticar as mensagens da época correspondente à chave, ou seja, P1 e P2.

2.2.4 Secure Directed Diffusion

O *Secure Directed Diffusion* [HYWLZ06] é uma extensão ao *Directed Diffusion* que vem garantir conectividade da rede caso não hajam partições e contenção de tráfico malicioso na presença de nós comprometidos, fazendo face aos seguintes problemas de segurança:

- Um sensor pode injectar falsos interesses ou amostras na rede;
- Um sensor pode descartar mensagens de interesses ou amostras.

Na sua génese está que todo o terreno onde os nós estão espalhados está dividido em zonas segundo uma grelha e ao invés de se associarem chaves aos sensores, associam-se sim a cada zona, tendo uma zona um conjunto de chaves que servirão para a estação central autenticar no futuro os dados retornados pelos sensores dum dado sector. Existe também uma fase inicial de configuração da rede, onde são criadas chaves par-a-par entre todos os vizinhos de modo à verificação futura de autenticidade e integridade de mensagens gradientes, assim como dos resultados, não garantindo no entanto confidencialidade na sua implementação. Durante essa configuração é usada uma chave mestra previamente instalada nos nós necessária à geração das chaves, a qual é apagada de memória no final desse processo, impossibilitando assim que nós maliciosos estabeleçam ligações com outros nós. No entanto é possível instalar nós numa fase posterior na rede, pois existe um mecanismo assíncrono para tal, sendo que nesse caso os novos nós são instalados com a chave mestra na memória. Além disso através do protocolo μ*Tesla* garante-se a autenticação dos interesses enviados a partir da estação.

O protocolo permite **agregação segura de dados** sendo gerado um *MAC* sobre os mesmos em cada nó agregador no caminho até à estação central. No entanto esta não é talvez a técnica mais apropriada para RSSF, dado que a transmissão desta grande quantidade de dados origina grandes gastos de energia.

Quanto a **tolerância a falhas**, as chaves permitem que a estação central autentique os dados e assim verifique quais os caminhos onde podem ter sido corrompidos nós ou quais os nós que os corromperam, reforçando outros caminhos em resposta. No entanto o desejável seria que nó a nó se fizesse essa verificação para descartar os valores corrompidos, mas tal é impossível nesta solução pois cada nó apenas partilha uma chave com os vizinhos no seu alcance.

Quanto a **poupança de energia**, não há qualquer melhoria relativamente ao protocolo base. Aliás a adição dos serviços de segurança introduz ainda mais bytes transmitidos na rede como seria de esperar e consequentemente acarreta mais gastos energéticos. Além disso os principais problemas deste protocolo deparam-se com a necessidade de mecanismos que levem cada nó a descobrir a sua localização, de modo a poder calcular a chave, e o facto de não possuir qualquer mecanismo de renovação de chaves, não garantindo assim segurança futura.

2.2.5 Análise comparativa

Depois de apresentados os sistemas anteriores faz-se agora uma comparação quanto ao grau de segurança de cada um, apresentando-se quais os ataques externos e internos mais comuns que os mesmos toleram. Essas comparações resumem-se nas Tabelas 2.1 e 2.2 respectivamente.

Tabela 2.1: Tolerância dos protocolos de encaminhamento face a ataques externos.

Protocolos	Confidencialidade	Integridade	Frescura	Disponibilidade	Autenticidade
TinyDB				X	
DD				X	
μTesla (broadcast)		X	X		X
SDD		X	X	X	X

Tabela 2.2: Tolerância dos protocolos de encaminhamento face a ataques internos.

Protocolos	Spoofing	Encaminhamento selectivo	Sinkhole	Sybil	Wormhole	HELLO
TinyDB		X				
DD	X	X				
SDD	X	X	X	X	X	

Vemos que ao nível de ataques externos (Tab. 2.1) o protocolo *SDD* é o que oferece mais garantias, embora não garanta confidencialidade de dados. Quanto aos ataques à disponibilidade, praticamente todos os protocolos dispõem de mecanismos de redundância no encaminhamento em termos de rotas alternativas de transmissão. De notar também que, apesar do protocolo *μTesla* dar boas garantias, tem que se ter em consideração que o mesmo é apenas vocacionado para o caso especial de broadcast autenticado. No geral, estes resultados eram esperados pois nem o *TinyDB* nem o *DD* foram implementados tendo como base critérios de segurança.

Relativamente aos ataques internos (Tab. 2.2) e tendo por base apenas os protocolos de encaminhamento, nota-se que todos eles possuem mecanismos de tolerância a falhas de nós nas rotas e a mensagens perdidas, logo todos eles toleram o ataque por

encaminhamento selectivo, assumindo que apenas uma pequena parte dos nós são corrompidos. Isso é conseguido no *TinyDB* com rotas alternativas disjuntas e no DD e SDD com heurísticas de escolha de vizinhos que tenham entregue mais dados.

Já os ataques *Spoofing*, *Sinkhole*, *Sybil* e *Wormhole*, imaginando o caso onde são adicionados nós maliciosos externos ao interior da rede, são defendidos pelo *SDD*, pois neste protocolo é usado um segredo na comunicação entre vizinhos. No entanto, se o atacante roubar os segredos, através de acesso físico a um nó, será sempre possivel efectuar o ataque.

Quanto ao ataque *HELLO Flooding*, originado através do envio sucessivo de mensagens de *HELLO* para estabelecimento de ligações locais não seguras, fazendo com que os vizinhos gastem energia a processá-las, não é defendido por nenhum dos protocolos, pois sobre estas mensagens não é normalmente garantida segurança.

2.3 Plataformas de espaços de tuplos partilhados para redes AdHoc

A natureza e exigências características das RSSF, tais como energia limitada, pouca capacidade de processamento, curto alcance de comunicação, topologia dinâmica, escalabilidade, robustez, qualidade de serviço ou heterogeneidade dos nós, faz com que a tarefa de construção de aplicações para as mesmas não seja de todo fácil e rápida. Assim, e de modo a diminuir o esforço no seu desenvolvimento, torna-se clara a necessidade de adoptar abstracções de middleware sobre as quais as aplicações possam assentar.

Esta secção apresenta portanto vários sistemas middleware que têm sido propostos ao longo dos últimos anos, sendo que a grande maioria dos mesmos obedece à noção de comunicação através de *espaços de tuplos* [MPR06, Gel85] e à noção de *regiões lógicas* [MP06]. A primeira é interessante em RSSF pouco fiáveis dado que existe a possibilidade de ser totalmente distribuída e a segunda serve como um mecanismo para poupança de energia e abstracção de endereçamento, pois numa mesma região é possível efectuar esquemas tais como agregação de dados eliminando a redundância e minimizando o número de transmissões.

2.3.1 Lime

O *Lime* [MPR06] é um modelo *middleware* de suporte ao desenvolvimento de aplicações que exibem mobilidade física de "hosts", mobilidade lógica de agentes, ou ambos. No Lime, o espaço de tuplos está dividido em inúmeros espaços de tuplos transientes,

cada um sendo transportado por unidades móveis individuais. Além disso no modelo estão presentes também as noções de localização de espaços de tuplos e reacção dos programas a estados específicos. Desta forma é possível migrar tuplos entre componentes, especificando o destino, e registar um excerto de código a executar quando um tuplo correspondendo a um certo padrão aparece no espaço. Este modelo, apesar de não ser vocacionado para as redes de sensores, devido a não ter nenhum mecanismo para aceder aos dados capturados pelos sensores, foi a base para o desenvolvimento de grande parte dos modelos recentes para esta área.

2.3.2 Hood

O modelo *Hood* [WSBC04] é uma abstracção de programação para vizinhanças em redes de sensores, onde os nós conseguem identificar subconjuntos da sua vizinhança física de acordo com certos critérios, e partilharem estado com a mesma. Mais propriamente cada nó demonstra um interesse em alguns atributos exportados pelos seus nós vizinhos e localmente faz cache dos mesmos, filtrando os atributos que não lhe interessam. O *Hood* foi o primeiro modelo a definir claramente a relação entre vários conceitos fundamentais para a noção de vizinhança: *grupo de membros*, *partilha de dados*, *cache de dados* e *mensagens de grupo*. No entanto possui algumas deficiências pois recursos são gastos desnecessariamente devido às mensagens serem enviadas periodicamente para os vizinhos mesmo sem haver ninguém interessado nelas. Além disso o modelo não tem suporte para afectar remotamente o estado de outro nó e não oferece abstracções para reagir a mudanças no estado partilhado. Não possui também quaisquer mecanismos para agregação de dados.

2.3.3 Abstract Regions

O modelo *Abstract Regions* [WM04] providencia uma interface de comunicação colectiva baseada em regiões, onde os nós estão relacionados topologicamente ou geograficamente. Neste modelo pares <*chave, valor*> são partilhados entre os nós de uma dada região e manipulados através de operações *read/write*. É baseado no modelo *Hood*, providenciando tal como este mecanismos para procura e partilha de dados entre nós sensores. Além disso, implementa também mecanismos de *tradeoff* entre gastos no uso dos recursos e precisão dos resultados, assim como inúmeros operadores que permitem pesquisar o estado dos nós vizinhos e implementar operações eficientes de agregação, compressão, e sumários de dados locais a uma dada região. Semelhantemente ao *Hood* não há qualquer mecanismo de notificação quando um dado tuplo aparece no sistema, logo não se podem executar acções em resposta. Para além disso, cada

região é programada de forma independente e definida antes da fase de compilação, não podendo ser readaptada em tempo de execução, tendo-se para isso que parar todo o funcionamento da rede.

2.3.4 Context Shadow

O modelo *Context Shadow* [Jan02] explora múltiplos espaços de tuplos, cada um responsável apenas por informação local advinda dos sensores do nó, a qual representa um dado contexto. Espalhados pela rede existem nós que efectuam consultas a outros nós, cuja localização é predefinida, à procura de serviços que disponibilizem informação desejada. Como a informação está dividida por contextos, é possível ao sistema adaptar-se e tomar decisões dependendo destes. A grande desvantagem deste sistema consiste na necessidade de se saber à partida a localização dos nós que possuem esses serviços.

2.3.5 Aggila

O sistema *Agilla* [FRL05] consiste numa plataforma *middleware* que providencia um estilo de programação baseado em agentes móveis. Estes agentes podem proactivamente migrar o seu código e estado por toda a rede, posicionando-se onde realmente são necessários. Para redes onde no seu interior haja a necessidade de processamento exaustivo dos dados e caso o número periódico de migrações e o código dos agentes não seja exagerado, levar os agentes e assim a capacidade de processamento até aos nós que contêm os dados pode diminuir drasticamente o peso da comunicação na rede, assim como a latência das operações, evitando-se a situação onde os dados têm de ser encaminhados por toda a rede até a uma estação base que efectue o processamento e assim diminuindo drasticamente a energia gasta. Além do mais existe a possibilidade destes agentes transportarem consigo a lógica de agregação até certos nós. Podem existir tantas aplicações a correr sobre a rede quantas se queiram, sendo somente necessário adicionar e remover agentes.

2.3.6 TeenyLime

O *TeenyLime* [CMMP06] é uma plataforma *middleware* bastante modular que assenta sobre o sistema operativo *TinyOS*, o qual foi desenhado para cenários onde todos os componentes de uma RSSF possuem o poder de computação e decisão sem necessitarem de um nó central para se coordenarem. O caso mais claro é aquele onde temos sensores e actuadores e os últimos efectuam determinadas acções baseadas nas amostras

recolhidas pelos primeiros. As operações de coordenação na rede consistem essencialmente nas operações do LIME. Ao contrário deste último, os espaços de tuplos são partilhados apenas com os seus vizinhos da vizinhança lógica, sendo que o espaço de tuplos partilhado é diferente para cada nó. Como algumas propriedades deste *middleware* temos que: as operações são reactivas, começando a executar quando um dado tuplo aparece no sistema; os dados são filtrados na origem antes de serem enviados para os vizinhos; as amostras são recolhidas apenas quando realmente são precisas; e a comunicação é fiável. No entanto a camada de encaminhamento que vem com a implementação de origem não apresenta quaisquer mecanismos de segurança, nem de agregação de dados, podendo esta ser estendida ou substituída sem afectar a lógica do sistema.

2.3.7 Análise comparativa

Relativamente aos anteriores modelos apresentados, nenhum deles apresenta quaisquer mecanismos de segurança de modo a garantir desde as propriedades básicas de segurança até à agregação segura de dados, logo não merecem qualquer análise comparativa nesse sentido. Segue-se uma síntese das vantagens e desvantagens de cada um.

O *Lime* é vocacionado para redes com nós móveis, onde estes comunicam através de um espaço de tuplos. No entanto não é vocacionado para RSSF pois não apresenta quaisquer mecanismos para extracção de informação dos sensores de um nó.

O *Hood* é um modelo que implementa, entre outras, a noção de partilha de informação e vizinhança, fazendo com que nós possam partilhar informação. No entanto não possui suporte para reagir a eventos ou mudança de estado.

O modelo *Abstract Regions* tem suporte para dividir a rede em zonas lógicas e partilhar informação entre sensores, à semelhança do *Hood*, mas tem a grande desvantagem de zonas lógicas não poderem ser criadas em tempo de execução, mas apenas em tempo de compilação, tornando assim este sistema muito estático.

O *Context Shadow* é um sistema baseado em serviços e clientes, onde os primeiros estão espalhados pela rede em nós com uma localização predeterminada e são requisitados pelos segundos. A grande desvantagem deste sistema consiste mesmo no facto dos serviços terem uma localização estática.

O modelo *Agilla* é um sistema um pouco diferente dos outros, sendo baseado em agentes móveis. A grande desvantagem é que nem todos os problemas ganham com esta estratégia, pois nem sempre se quer levar o processamento até aos nós. Torna-se assim bastante limitado nos casos onde pode ser aplicado.

Por último, temos o modelo *TeenyLime*, o qual é baseado na comunicação por espaços de tuplos, tal como o *Lime*, mas onde estes são agora partilhados apenas com a sua vizinhança lógica. Possui também, entre outras propriedades, suporte para reacção a eventos e mudança de estado. Devido à sua arquitectura bastante modular, é bastante fácil de alterar ou estender componentes sem afectar a lógica do sistema.

2.4 Agregação segura de dados em RSSF

Como se sabe, numa RSSF a comunicação é a operação que mais energia gasta, sendo directamente proporcional ao número de bytes enviados. É por isso essencial, de forma a reduzir a energia consumida, usar na rede técnicas de *pré-processamento de dados* de forma a diminuir a quantidade necessária de dados a transmitir, e de *agregação de dados* que permitam a eliminação de dados redundantes que nela circulem. Esta secção começa primeiro por apresentar em detalhe os dois conceitos anteriores, seguidos de exemplos de ataques aos mecanismos de agregação e de alguns modelos existentes que combatem esses ataques. A secção terminará com uma análise final.

2.4.1 Processamento interno e agregação de dados

Muitas das aplicações das RSSF baseiam-se na obtenção e disseminação de medições associadas a fenómenos físicos do meio ambiente, observados pelos sensores a pedido de uma estação central. Dependendo da natureza desses dados e do processamento que é necessário para a sua análise, pode ser possível efectuar parte desse processamento no interior da rede e retornar apenas resultados pré-processados e agregados, os quais em certos casos possuem um tamanho bastante inferior ao conjunto dos dados. Esta é a ideia subjacente aos princípios de processamento interno e agregação de dados na rede e permite minimizar a transmissão de bytes, poupando assim energia. Este conceito é de reconhecida importância e foi uma das razões da criação de sistemas tais como o *Aggilla* [FRL05], no qual a computação é levada até ao nó através de agentes. Mais especificamente associado à noção de agregação de dados, a ideia é que os dados recebidos de diferentes nós sejam sintetizados num relatório de alto nível e de fácil reconhecimento, eliminando a redundância e minimizando o número de transmissões para a estação central, diminuindo assim os gastos de energia.

Diversos protocolos de agregação têm sido propostos na investigação das RSSF, mas ainda poucos existem que se adaptem bem às condições genéricas das mesmas. Em muitos casos apenas é efectuada agregação fora da rede, quando toda a informação está disponível na estação central. Outras aproximações recorrem a nós agregadores dedicados, os quais não possuem necessariamente sensores, mas sim mais capacidade

de bateria e processamento do que os nós normais, para prolongar a vida útil global da rede. Contudo, no caso de redes homogéneas onde os nós têm iguais capacidades, de mais baixo custo e maior flexibilidade para alterações dinâmicas de estruturação no interior da rede, o problema da agregação de dados é ainda um aspecto em aberto.

2.4.2 Ataques à agregação de dados numa RSSF

Diversos tipos de ataques existem que podem ser lançados sobre uma RSSF tal como visto na subsecção 2.1.3. No entanto questões particulares existem que podem afectar a rede no contexto da agregação de dados, sendo os seguintes ataques exemplos disso:

Ataque Sybil como o atacante pode apresentar mais do que uma identidade dentro da rede, pode afectar o esquema de agregação nas mais variadas formas. Primeiro pode criar diversas identidades para votar na eleição de um dado nó agregador. Em segundo lugar pode afectar o resultado da agregação por gerar diferentes entradas com diferentes valores. Em terceiro lugar pode tirar partido do facto de possuir várias identidades para corromper esquemas que se baseiam em consenso para validar os resultados;

Encaminhamento selectivo devido a poder escolher se reencaminha ou não as mensagens recebidas com os dados, um nó malicioso pode assim escolher um subconjunto delas e descartar as outras, afectando desse modo o resultado final de agregação. Como exemplo, o nó pode decidir descartar as mensagens que reportem nos seus dados temperaturas acima dos 60ºC, impossibilitando assim a eventual activação de um sistema de incêndio desencadeada a partir de outra zona da rede.

Existe ainda um tipo de ataque próprio dos esquemas de agregação:

Ataque furtivo (*Stealthy*) neste ataque o adversário injecta dados falsos na rede sem revelar a sua existência, sendo na sua génese um ataque com as mesmas características do ataque *Sybil*, tal como tipificado ao nível de encaminhamento seguro em RSSF. Na abordagem de segurança de algoritmos e protocolos de agregação de dados, ao impedir que o atacante seja detectado, nenhuma acção pode ser tomada contra ele, podendo continuar indefinidamente a injectar dados falsos na rede.

2.4.3 Modelos que garantem agregação segura de dados

Nos últimos anos diversos modelos foram criados de forma a combater ataques à agregação de dados. No entanto nenhum deles é perfeito, tendo as suas vantagens

e desvantagens e não garantindo simultaneamente todas os requisitos básicos de segurança. Um trabalho recente com uma boa análise sobre vários dos modelos pode ser encontrado em [AFN08]. Entretanto, seguem-se dois exemplos de protocolos propostos que garantem agregação segura de dados e que podem ser aplicados facilmente hoje em dia numa RSSF:

SIA este protocolo é descrito em mais detalhe em [PSP03], mas é essencialmente uma *Framework* que foi criada para combater os ataques onde a informação de agregação pode ser manipulada maliciosamente num nó sem que tal seja detectado. É composto pelas fases de capturar informação dos sensores, calcular localmente o resultado de agregação, calcular um MAC local com a sua chave, e reportar os resultados e o MAC ao nó agregador seguinte. Este protocolo providencia serviços que garantem a integridade dos dados, autenticação, frescura e confidencialidade. No entanto o protocolo não assume que o nó agregador seguinte possa estar corrompido, podendo este assim criar resultados falsos de agregação;

WDA descrito em mais detalhe em [DDHV03], é um protocolo que consegue garantir a validação dos dados transmitidos de um nó agregador até à estação central. É baseado no testemunho de vários nós que efectuam também eles a tarefa de agregação, mas apenas para gerar um testemunho (um MAC) sobre os resultados de forma a enviá-lo para o nó agregador, que por sua vez reencaminha o resultado da agregação juntamente com os testemunhos dos outros nós. O WDA oferece apenas integridade e autenticidade dos dados, sendo que outras propriedades podem ser no entanto garantidas noutro nível inferior da pilha de comunicação. No geral é um protocolo que coloca muita redundância nas operações de agregação, aumentando bastante o tamanho das mensagens enviadas devido à informação de controlo.

2.4.4 Análise comparativa

De modo a garantir uma agregação segura, é necessário que todos os nós correctos do grupo cooperem de tal forma que a informação que a estação central obtenha seja a correcta. Note-se o exemplo em que a estação central efectua uma pesquisa para a rede, sendo da responsabilidade de um conjunto de nós efectuarem o processo de agregação, retornando a síntese de todos os resultados. Existem diversas estratégias que se podem usar, das quais duas foram apresentadas anteriormente. Qualquer dessas estratégias tem as suas falhas, pois no caso do SIA não é assumido que o nó agregador seguinte possa estar corrompido e não há maneira da estação central conseguir apurar qual o nó que actuou maliciosamente, pois toda a informação que lhe chega sobre um

dado intervalo é aceite como válida. Já o protocolo WDA possibilita à estação central obter essa informação, pois todos os MAC's lhe chegam permitindo assim identificar o infractor. No entanto o facto de todos os MAC's circularem na rede faz com que o tamanho das mensagens aumente, consumindo-se mais energia.

Tabela 2.3: Tolerância dos protocolos de agregação face a ataques externos.

Protocolos	**Confidencialidade**	**Integridade**	**Frescura**	**Disponibilidade**	**Autenticidade**
SIA	X	X	X		X
WDA		X			X

Tabela 2.4: Tolerância dos protocolos de agregação face a ataques internos.

Protocolos	**Encaminhamento Selectivo**	**Sybil**	**Stealthy**
SIA		X	X
WDA			

Uma comparação dos dois protocolos vistos anteriormente, em termos de tolerância a ataques externos e internos à agregação, pode ser vista nas tabelas 2.3 e 2.4 respectivamente. Desta comparação diga-se que o protocolo *WDA* é aquele que dá menos garantias em termos de ataques internos, pois o facto de usar mecanismos de testemunhas faz com que um nó malicioso se possa passar por vários outros, não evitando assim o ataque *Sybil* e o *Stealthy*. Além disso dá muito poucas garantias quanto a ataques externos, mas tal pode ser colmatado se assentar sobre uma camada que já forneça serviços que combatam esses ataques, como é o caso do *Minisec* [LMPG07].

3

Arquitectura

Neste capítulo apresenta-se uma visão geral da arquitectura do sistema implementado, sem entrar em grandes pormenores no que toca à constituição interna dos componentes principais. Em primeiro lugar apresentar-se-á uma visão global de todo o sistema, seguindo-se a apresentação do ambiente de simulação e a descrição de cada um dos componentes. Terminar-se-á com algumas considerações finais à arquitectura apresentada.

3.1 Visão global do sistema

A plataforma implementada possui a forma de uma pilha de serviços e é constituída por três componentes principais e independentes entre si que os implementam. Segundo uma visão ascendente da pilha (fig. 3.1) temos o componente *MinisecB*, seguido pelo componente *SecureDiffusion* e finalmente o componente *Infrastructure* com o qual as aplicações comunicam. Toda a pilha foi implementada sobre o simulador para RSSF JProwler [jPr].

O componente *MinisecB* assenta directamente sobre a sub-camada MAC do simulador e tem por objectivo garantir autenticidade, confidencialidade, integridade e não replicação das mensagens a circular em toda a rede. Foi implementado tendo por base as especificações do protocolo Minisec [LMPG07].

O componente *SecureDiffusion* tem por objectivo garantir fiabilidade e resiliência a falhas ao nível das rotas de comunicação. Foi implementado segundo as especificações

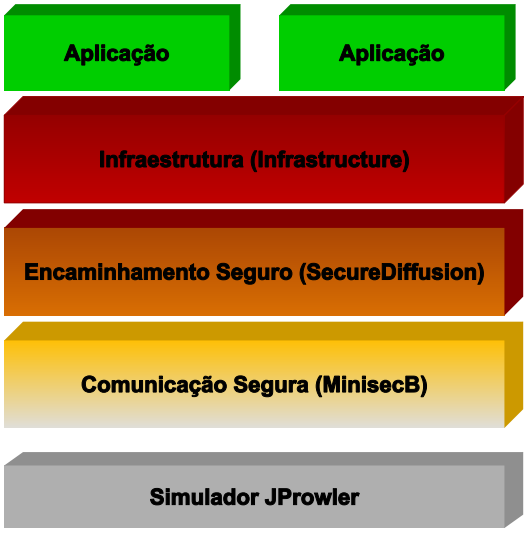


Figura 3.1: Visão da pilha implementada.

do protocolo Secure Directed Diffusion [HYWLZ06].

O componente *Infrastructure* é constituído essencialmente por um espaço de tuplos e por um conjunto de serviços que operam sobre esse espaço e os quais são exportados às aplicações de forma a estas comunicarem através da noção de *Vizinhanças Lógicas*. Para a construção desta camada foi readaptada para Java a implementação do sistema TeenyLime [CMMP06] na linguagem *nesC*.

Finalmente temos os módulos *Aplicação* que invocam as primitivas da API exportada pela camada Infraestrutura.

O sistema implementado possui as seguintes acções principais:

Criação de uma VL as aplicações da estação central ou os próprios nós do interior da rede, a pedido do utilizador, criam grupos lógicos na rede denominados de *vizinhanças lógicas*. Para este fim é enviada uma mensagem para a rede, que ao nível do componente *SecureDiffusion* é vista como um interesse à semelhança do protocolo SDD, e ao nível do componente *Infrastructure* é vista como um conjunto de tuplos que contêm definições de um grupo lógico;

Envio de comandos para as vizinhanças lógicas após as VLs estarem criadas, o nó criador envia tuplos para os nós pertencentes às mesmas, contendo comandos ou dados. Esses tuplos são adicionados ao espaço de tuplos de cada nó e entregues às aplicações desejadas;

Execução de aplicações as aplicações, após receberem um dado tuplo com comandos, podem decidir executar um excerto de código pré-instalado. Nesse código pode estar por exemplo captura de amostras dos sensores, ou ainda criação de outras

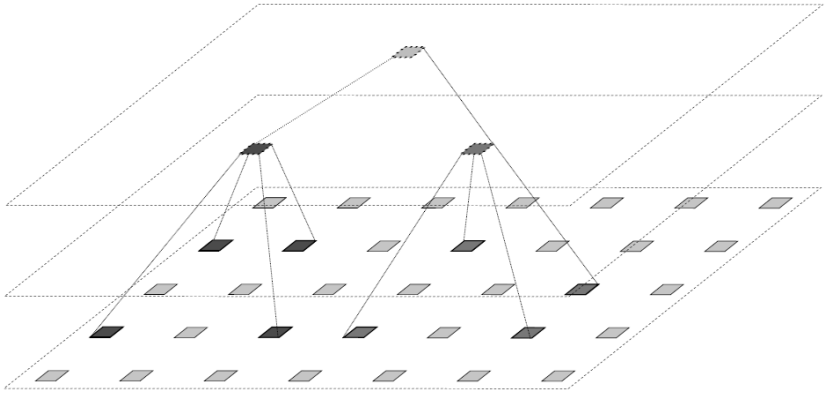


Figura 3.2: Rede *overlay* de nós, onde ao nível mais baixo estão os dispositivos físicos e nos níveis superiores se encontram criados grupos lógicos.

VLs na rede através do mesmo processo. A recursividade de criação destas por parte dos nós do interior da rede não tem qualquer limite. Quaisquer dados gerados são escritos para o espaço de tuplos local, podendo assim ser partilhados com outras aplicações;

Retorno de resultados das VLs caso o nó criador de uma VL deseje capturar os dados escritos pelas aplicações, tem de previamente enviar um tuplo com uma reacção para ser instalada em todos os nós da VL de forma a capturar os mesmos. Assim quando uma aplicação escreve um tuplo que seja desejado, ao invés de ficar a residir no espaço de tuplos local, é retornado para o nó criador.

Na imagem A.3 em anexo, é possível ver uma sequência de acções segundo os conceitos anteriores.

Podem assim ser criadas topologias *overlay* sem limite de profundidade, como se pode ver na figura 3.2.

3.2 Ambiente de simulação

O ambiente de simulação para RSSF sobre o qual o sistema foi implementado e testado foi o simulador *JProwler* [jPr]. Vejamos as caracteristicas mais importantes deste simulador:

- Implementado na linguagem *JAVA* de forma modular, permitindo a sua extensão;
- Executa segundo o modelo de eventos discretos;
- Fácil de prototipar e testar protocolos de comunicação;

- Boa escalabilidade e desempenho de rede, suportando milhares de nós em condições parecidas com simulações de tempo real;
- Possui suporte de modelação para um sensor do tipo *MicaMote*;
- Modelo de rádio probabilístico que simula a operação da camada MAC segundo a norma *802.15.4*. Apesar de simplista é bastante preciso.

Possui ainda uma interface gráfica de visualização da disposição dos nós e troca de mensagens, que pode ser vista na figura A.1 em anexo.

3.3 Camada Comunicação Segura

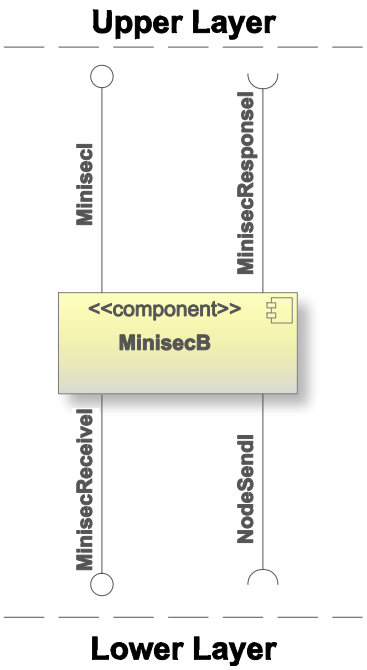


Figura 3.3: Componente MinisecB.

O componente *MinisecB* é a implementação da camada *Comunicação Segura* e assenta directamente sobre o simulador, garantindo autenticação, confidencialidade, integridade e não replicação de mensagens na rede, combatendo assim grande parte dos ataques ao nível MAC descritos na subsecção 2.1.3. O seu nome advém do protocolo Minisec e particularmente da sua variante de comunicação por *broadcast*, o qual foi escolhido para implementar esta camada. Pode ser visto um esquema da sua interacção com as outras camadas na Fig. 3.3. Podem ainda ser vistas nas listagens 3.1, 3.2 e 3.3 a constituição das interfaces usadas e exportadas. Como se pode ver, as interfaces são

bastante apelativas e não merecem grandes explicações em termos arquitecturais. A camada superior quando envia uma mensagem invoca o método `MinisecI.sendMsg`. Seguidamente esta mensagem é transmitida para o destinatário desejado ou por *broadcast* de forma segura. Quando é recebida uma mensagem segura, é entregue à camada superior através do método `MinisecReceiveI.receiveMsg`. De notar que esta camada tem ainda de comunicar com as primitivas do rádio, neste caso do simulador, sendo que para enviar uma mensagem invoca um método de envio na interface `NodeSendI`.

Listagem 3.1: MinisecI - API invocada pelo nível superior.

```
1 public interface MinisecI {
2   public void sendMsg(byte[] payload, short destiny, byte msgType);
3 }
```

Listagem 3.2: MinisecReceiveI - API invocada pelo nível inferior.

```
1 public interface MinisecReceiveI {
2   public void receiveMsg(byte[] payload);
3 }
```

Listagem 3.3: MinisecResponseI - API de resposta implementada pelo nível superior.

```
1 public interface MinisecResponseI {
2   public void receiveMsg(byte[] payload, short previousHop, byte msgType);
3 }
```

3.4 Camada Encaminhamento Seguro

O componente *SecureDiffusion* é a implementação da camada *Encaminhamento Seguro* e é o componente responsável por todo o encaminhamento que se efectua na rede, e mais particularmente por garantir segurança a esse nível. Essa segurança visa combater os ataques ao nível rede descritos anteriormente na subsecção 2.1.3. O nome deste componente advém do protocolo *Secure Directed Diffusion* escolhido para implementar esta camada, o qual foi apresentado na subsecção 2.2.4.

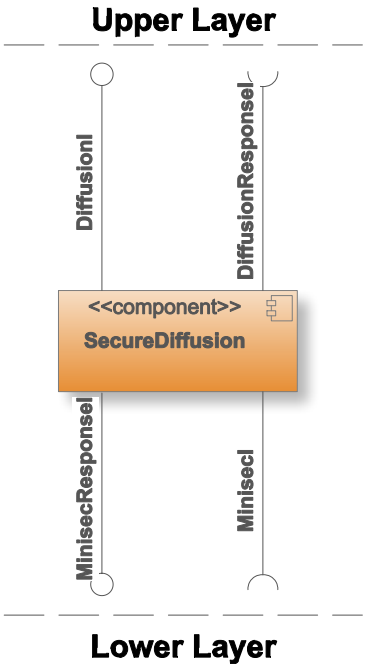


Figura 3.4: Componente SecureDiffusion.

Pode ser visto um esquema da sua interacção com as outras camadas na figura 3.4. Podem ainda ser vistas nas listagens 3.4 e 3.5 a constituição das interfaces usadas e exportadas.

Devido à complexidade desta camada, efectua-se em seguida uma breve explicação do seu funcionamento nas partes que se consideram mais relevantes. Sendo assim este componente trata as seguintes acções:

Registo e remoção de interesses por forma a registar um interesse na rede há que invocar o método `DiffusionI.regist`. Quando o criador verificar que já não há necessidade do interesse continuar na rede, invoca `DiffusionI.unregist` de forma ao mesmo ser removido da rede e libertar recursos. Quando um interesse registado na rede é recebido, a camada imediatamente acima desta é informada através do método `DiffusionResponseI.interestRegistered` e se essa mesma camada quiser subscrever o interesse invoca imediatamente o método `DiffusionI.subscribe` informando a presente camada para reservar os recursos necessários. Por outro lado quando um interesse é removido da rede é invocado o método `DiffusionResponseI.interestUnregistered` na camada superior para libertar todos os recursos;

Subscrição de interesses como foi dito anteriormente, após a camada superior do nó

Listagem 3.4: DiffusionI - API invocada pelo nível superior.

```
1  public interface DiffusionI {
2    /** Regists an interest in the network with the given attributes. */
3    public String regist(byte attrs[]);
4
5    /** Unregists a previously registered interest from the network. */
6    public Error_t unregist(String interestID);
7
8    /** Returns data to the creator of the given interest. */
9    public Error_t returnData(String interestID, byte[] data);
10
11   /** Sends data to the subscribers of a previously registered interest.
12    *  Informs about the new expected delivering data rate towards this node
13    *  or 0 if no data is expected to be returned.
14    *  NOTE: This method is called by the node who registered the interest
15    *  only. */
16   public Error_t diffuseData(String interestID, byte[] data, int dataRate);
17
18   /** Called in order to subscribe to an interest after testing the match of
19    *  local state against interest attributes. */
20   public Error_t subscribe(String interestID, byte[] attrs, byte[] dataKey);
21 }
```

Listagem 3.5: DiffusionResponseI - API de resposta implementada pelo nível superior.

```
1  public interface DiffusionResponseI {
2    /** Called when a new interest has been registered in the network. The
3     *  local state is tested against the interest attributes. If there is
4     *  a match the interest is subscribed and the Diffusion component must
5     *  be informed immediately through DiffusionI.subscribe(...). */
6    public void interestRegistered(String interestID, byte[] attrs,
7                byte[] dataKey);
8
9    /** Called after a new message of an existing interest has arrived. */
10   public void interestUpdated(String interestID, byte[] attrs);
11
12    /** Called after new results of a previously registered interest have
13     *  arrived. */
14   public void resultsArrived(short src, String interestID, byte[] attrs);
15
16   /** Called after receiving an acknowledge which says that a node has
17    *  subscribed a previously created interest. */
18   public void nodeSubscribedInterest(short src, String interestID,
19               byte[] attrs);
20
21   /** Called when an interest this node had previously subscribed is
22    *  unregistered from the network. */
23   public void interestUnregistered(String interestID);
24 }
```

subscritor do interesse ter informado a presente camada dessa vontade é devolvido para o criador do interesse uma mensagem de *acknowledge* informando sobre o novo subscritor. Essa mensagem irá eventualmente chegar ao nó criador e o método `DiffusionResponseI.nodeSubscribedInterest` será invocado;

Difusão para os subscritores de um interesse quando a camada superior no nó criador de um interesse deseja enviar dados para os seus subscritores, necessita invocar o método `DiffusionI.diffuseData` e nos nós subscritores é consequentemente invocado o método `DiffusionResponseI.interestUpdated`;

Retorno de dados para o criador de um interesse quando a camada superior tem a necessidade de retornar dados para o nó criador do mesmo, é invocado o método `DiffusionI.returnData` e de forma a informar o nó criador é invocado o método `DiffusionResponseI.resultsArrived` na camada acima desta.

3.5 Camada Infraestrutura

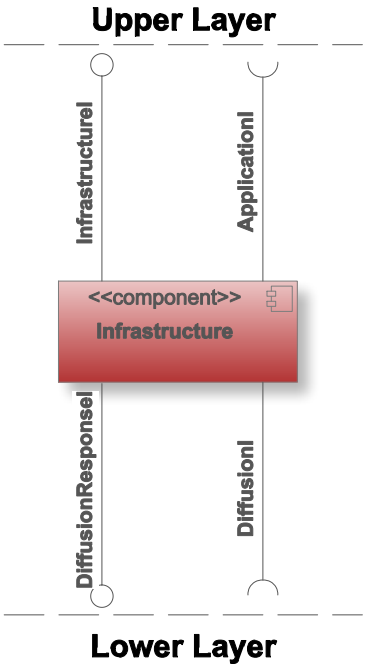


Figura 3.5: Componente Infrastruture.

O componente *Infrastructure* é a implementação da camada *Infraestrutura* e é o componente responsável essencialmente pela gestão do espaço de tuplos local a cada nó, exportando uma visão da rede como um conjunto de vizinhanças lógicas. A sua implementação é uma adaptação do sistema *TeenyLime* apresentado na subsecção 2.3.6.

Nesta camada não existem quaisquer serviços de segurança pois quer a segurança das mensagens de dados, assim como a segurança das mensagens para as vizinhanças lógicas, são tratadas pela camada *Difusão Segura*. Além disso não possui quaisquer mecanismos para efectuar agregação, pois essa noção é vista apenas pelas aplicações a correrem sobre os nós, as quais têm acesso directo aos códigos de agregação instalados localmente. Devido a isto, esta camada não segue estritamente a proposta de pilha apresentada na subsecção 2.1.1 do capítulo 2.

Pode ser visto um esquema da sua interacção com as outras camadas na Fig. 3.5. Pode ainda ser vista na listagem 3.6 a API exportada às aplicações que executam nos sensores. Os métodos desta API pertencem essencialmente a três classes distintas:

Gestão de vizinhanças lógicas com um método para construção de VLs e outro para a sua remoção;

Gestão do espaço de tuplos com dois métodos que dizem respeito a duas operações muito conhecidas sobre espaços de tuplos, nomeadamente, um método que escreve um tuplo e outro que lê um tuplo seguido de remoção no espaço;

Gestão de reacções que podem ser instaladas no espaço de tuplos de uma VL e que assim que um tuplo aparece no espaço com as definições fornecidas, é retornado para o nó que invocou a operação.

3.6 Módulo Aplicação

Os módulos *Aplicação* executam directamente sobre toda a pilha implementada, invocando directamente a API apresentada na listagem 3.6. Em resposta, a camada *Infraestrutura* comunica com as várias aplicações através da API que consta na listagem 3.7.

Sobre uma mesma aplicação podem ser invocadas diferentes tarefas (e.g. diferentes esquemas de agregação). Por essa razão foi adicionado o conceito de *Task* ou tarefa, independente na execução e que corresponde a diferentes excertos de código que podem ser instanciados por qualquer aplicação a correr sobre o sensor, reaproveitando dessa forma código. Todas as Tasks têm de estender o objecto *Task*, apresentado na listagem 3.8. Por sua vez, uma *Task* comunica com a *Aplicação* que a instanciou através da API vista na listagem 3.9.

Listagem 3.6: InfrastructureI - API invocada pelas aplicações.

```
1  public interface InfrastructureI {
2
3  // LOGICAL NEIGHBOURHOODS MANAGEMENT
4    /** Creates a logical neighbourhood in the network. */
5    Pair<TLOpId_t, String> createLN(short component, TemplateSet tmplSet);
6
7    /** Removes a previously created LN from the network. */
8    TLOpId_t removeLN(short component, String nghId);
9
10 // TUPLE SPACE MANAGEMENT
11   /** Adds a tuple to the tuple space of the given VL. */
12   TLOpId_t out(short component, String nghId, Tuple tuple, int dataRate);
13
14   /** Removes and returns one random tuple of each node's tuple space
15    *  belonging to the given LN, which matches the template given. */
16   TLOpId_t in(short component, String nghId, Tuple tmpl, int dataRate);
17
18 // REACTIONS MANAGEMENT
19   /** Adds a reaction to the tuple space of an existing VL. If a tuple
20    *  space inserted in that same TS matches the template given, it is
21    *  returned as a copy towards this node. */
22   TLOpId_t addReaction(short component, String nghId, Tuple tmpl,
23            int dataRate);
24
25   /** Removes a reaction previously added to a TS of the given LN. */
26   TLOpId_t removeReaction(short component, String nghId, TLOpId_t reaction,
27            int dataRate);
28 }
```

Listagem 3.7: ApplicationI - API de comunicação com uma aplicação.

```
1  public interface GenericAppI {
2    /** Starts an application. */
3    void start();
4
5    /** Stops an application. */
6    void stop();
7
8    /** Returns the identifier of the application. */
9    short getId();
10
11   /** Called whenever a new tuple appears in a tuple space and it matches a
12    *  reaction previously installed by this application, or when this
13    *  application has submitted a read request over it. */
14   void tupleReady(TLOpId_t opId, Tuple tuple);
15 }
```

Listagem 3.8: Task - procedimento genérico que pode ser invocado.

```
1 public abstract class Task {
2   public abstract void start();
3   public abstract void tupleReady(TLOpId_t opId, Tuple tuple);
4 }
```

Listagem 3.9: MasterAppI - comunicação de uma Task com uma Aplicação.

```
1 public interface MasterAppI {
2   public InfrastructureI getManager();
3   public NodeStateI nodeState();
4   public void terminateMe(Task t);
5 }
```

3.7 Modelo de segurança

A segurança na pilha está apenas ao nível das camadas *Encaminhamento Seguro* e *Comunicação Segura*. Temos assim para cada uma delas uma breve explicação das garantias de segurança da mesma, sem focar pormenores de implementação:

Camada de Comunicação Segura esta camada garante que todas as mensagens transmitidas na rede obedecem às quatro propriedades básicas de segurança: integridade, autenticidade, confidencialidade e não replicação. Todas essas garantias são dadas através de uma chave partilhada entre todos os nós, que pode ser ou não renovada, dependendo da implementação. De salientar que esse processo de renovação não faz parte do âmbito deste trabalho;

Camada de Encaminhamento Seguro esta camada garante disponibilidade de rotas na ausência de partições, através de encaminhamento replicado por várias rotas, mesmo no caso de ataque a um sensor numa das rotas de comunicação principais. Garante também integridade, confidencialidade e autenticidade a este nível, entre nós vizinhos. Além disso, como se verá mais adiante no capítulo da implementação, as mensagens em alguns casos especiais serão autenticadas da estação central para todos os outros nós da rede, e os resultados retornados para a estação central serão cifrados.

Neste sistema o modelo de adversário assumido é o modelo *Manets* apresentado na subsecção 2.1.2, onde o atacante pode efectuar tanto ataques aos canais de comunicação como aos nós fisicamente, roubando os seus segredos. No entanto, assume-se, neste último caso, que o atacante demora ainda alguns segundos até roubar as chaves de um sensor e a poder efectuar ataques com elas. Além disso durante a instalação de novos nós na rede e na fase de *bootstrap* consequente, o atacante não tem acesso físico aos nós.

Por último, assume-se que a estação central é uma entidade confiável, não podendo ser alvo de ataques.

3.8 Considerações finais

Depois de uma visão geral do sistema implementado, explicam-se agora algumas divergências relativamente à pilha de serviços proposta pela investigação, vista no capítulo 2, particularmente no que toca aos serviços que cada camada teria de implementar.

Quanto à camada *Infraestrutura*, foi falado que através da API exportada as aplicações poderiam ajustar critérios de rede. Ora, nesta implementação tal não foi considerado, sendo que todas as definições são comuns a todas as aplicações, quer sejam de protocolos ou outras. Além disso, nesta camada não foram implementados outros serviços além daqueles associados à visão de *espaço de tuplos* e à noção de *vizinhança lógica*, pois tal não seria necessário para o que se queria testar.

Também na camada *Difusão Segura* não constam quaisquer mecanismos de agregação. Esta noção foi transportada para o nível *Aplicação*, onde podem ser criadas VLs dedicadas que actuam como agregadoras, facilitando todo o processo.

De referir também que na API *InfrastructureI* (listagem 3.6) são apenas exportadas as operações *in* e *out* sobre um espaço de tuplos. Não foram implentadas outras operações de modo a concentrar esforços apenas naquelas necessárias ao teste da plataforma.

4

Implementação

Este capítulo descreve detalhadamente a implementação e o modo de funcionamento dos componentes apresentados no capítulo 3. Inicia-se no entanto com a descrição das melhorias efectuadas ao simulador. No final são feitas algumas considerações sobre o que foi apresentado.

4.1 Ambiente de simulação

Foi incorporado no simulador um módulo de simulação de energia readaptado dum trabalho anterior sobre RSSF [Ama09]. Este modelo foi utilizado para os testes de gastos energéticos apresentados mais à frente. É um modelo bastante simplista, mas que consegue simular a energia gasta pela comunicação e pelos mecanismos que asseguram a segurança dos dados e protocolos, assim como a energia gasta pelos diferentes estados que um sensor pode apresentar. Os gastos energéticos das várias operações consideradas para os testes são baseados nos do trabalho citado e podem ser vistos na tabela 4.1.

Foi também implementada uma nova interface de interacção com o utilizador, a qual pode ser vista na figura A.2 em anexo. Essa interface contempla as seguintes acções:

- Ajuste da velocidade de simulação;

Tabela 4.1: Consumos energéticos das várias operações de um nó sensor.

Operação	Consumo
Emissão	59,2 μJ/B
Recepção	28,6 μJ/B
Cifra/Decifra	1,79 μJ/B
Geração de MAC	2,47 μJ/B
Activo	0,096 μJ/s

- Ao nível rádio, visualização das ligações unidireccionais e bidireccionais de um nó com os seus vizinhos;
- Ao nível de encaminhamento, visualização para um interesse de quais os nós onde o mesmo foi registado, quais os que o subscreveram, quais as rotas de retorno de dados e ligações seguras com os vizinhos;
- Para cada nó, a sua informação, tal como localização lógica, carga da bateria, medição actual dos seus sensores, e possibilidade de desligar e ligar o rádio.
- Pesquisa de um nó na rede e visualização de todos os nós com cores diferentes consoante o seu tipo (e.g. temperatura, agregador).

Por último foi alterada a forma das comunicações serem efectuadas. Anteriormente, qualquer que fosse o tamanho da mensagem, a sua comunicação demorava sempre o mesmo tempo. Após a alteração, o tempo de envio é agora proporcional ao tamanho da mensagem segundo taxas de transmissão reais, adoptadas dos sensores *MicaZ* [Mic].

4.2 Componente MinisecB

Como dito anteriormente, este componente foi implementado segundo a especificação da variante *broadcast* do protocolo *Minisec*. Descrevem-se agora alguns dos pormenores da implementação:

Algoritmo e modo de cifra tal como especificado, foi usado o algoritmo *Skipjack* [Ski] para cifra dos dados. No entanto, ao contrário do modo *OCB* recomendado, o qual num único passo garante confidencialidade e autenticidade, foi usado o modo *CFB* que usa dois passos. Isto deveu-se ao facto de não haver disponível qualquer implementação para Java da versão 2.0 desse modo. O método substituto gasta portanto quase o dobro da energia que o recomendado gastaria, mas

que se pode desprezar pois numa RSSF o grande impacto energético como se sabe está na comunicação de dados;

Filtros de Bloom estes filtros [Blo70] combinados com um sistema de épocas estão dependentes da forte sincronização de relógios dos sensores, o que no simulador é garantido. Permitem garantir a *frescura* das mensagens e foram implementados seguindo à risca a especificação. As definições de inicialização deste filtro estão, acima de tudo, dependentes da frequência de comunicação, sendo que foram escolhidos e testados valores que dessem garantias em termos da percentagem de *falsos negativos*. Após se afinarem os filtros, conseguiu-se, com uma duração de época de 5 segundos, uma expectativa de um máximo de 10 mensagens recebidas por segundo, e com 256 bits de tamanho dos filtros, que o máximo verificado de falsos negativos em toda a rede fosse de 8% e a média abaixo de 1%;

Fila de mensagens para cifra foi implementada uma fila de mensagens, de modo a contornar o facto de que uma mensagem pode ser cifrada por este componente e depois ficar durante muito tempo nas filas da sub-camada MAC até ser realmente enviada. Ora, quando chegasse ao destinatário, o seu contador poderia não estar sincronizado com o dos filtros de Bloom. Assim, as mensagens são guardadas na fila e só são cifradas no preciso momento antes de serem enviadas;

Intervalo de comunicação efectuou-se um melhoramento a este nível para o modo como as mensagens são transmitidas. Como nas RSSF se registam muitas colisões, foi adoptado o mecanismo de transmissão baseado na divisão do espaço temporal em intervalos. Estes intervalos são calculados no momento de instalação dos sensores, sendo baseados em critérios aleatórios. O espaço temporal de referência para divisão em intervalos foi de 1 segundo. Assim, a cada 1 segundo mais um delay aleatório são despachadas tantas mensagens da fila quanto se possa, ficando o resto do tempo até ao próximo segundo impossibilitados de enviar dados. Os nós podem assim comunicar em períodos temporais não coicidentes com alta probabilidade;

Controlo da taxa de transmissão de dados foi implementado um mecanismo que, em concordância com o esquema de intervalo de comunicação, limita o número de bytes a serem enviados até um valor parametrizado. O valor de referência máximo é sempre o permitido pelo rádio, que foi ajustado no simulador para transmitir a um máximo de 10.500B/s;

Refrescamento de chaves o protocolo Minisec usa, como se sabe, uma chave comum a todos os nós da rede para efectuar as operações de cifra das mensagens. No

entanto, apesar de já existirem alguns protocolos de refrescamento de chaves, tal não foi contemplado na presente dissertação, pois isso exigia um alargamento de esforços na implementação fora do âmbito do que se queria testar.

Constituição de mensagens a constituição de mensagens foi baseada no descrito no protocolo Minisec do SO TinyOS. Essa constituição varia de SO para SO, logo resolveu-se seguir esse modelo e deixou-se essa adaptação para quando se fizer a instalação do sistema em sensores reais. No entanto, o tamanho mínimo das mensagens nesta implementação é de 16 bytes (12 bytes dos campos iniciais + 4 bytes do MIC), variando depois o campo dos dados.

4.3 Componente SecureDiffusion

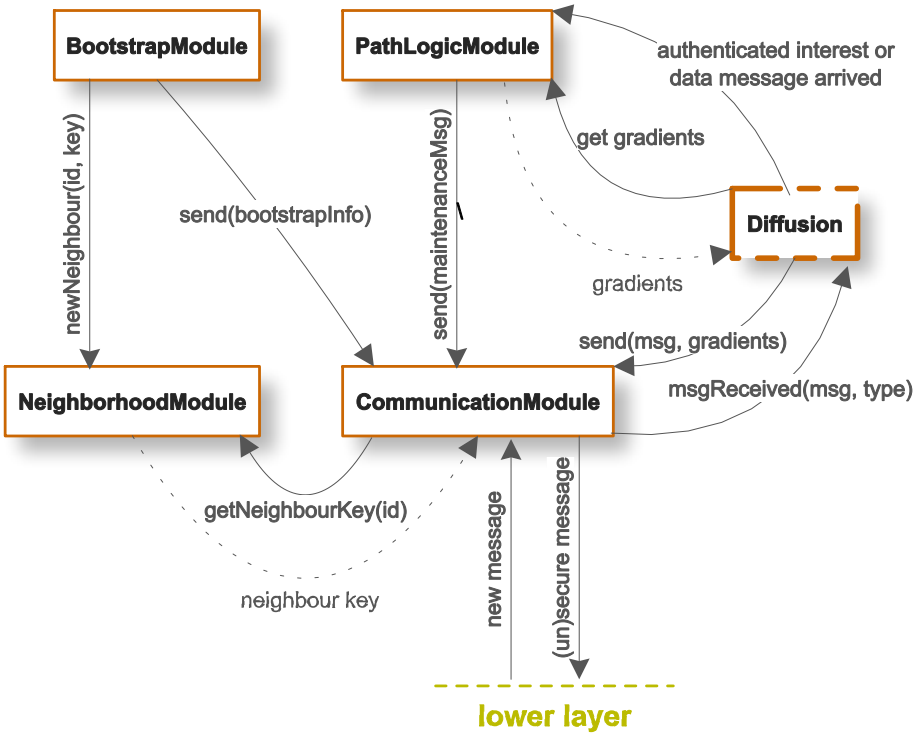


Figura 4.1: SecureDiffusion - módulos e suas dependências.

Este componente foi implementado tendo por base a especificação do protocolo Secure Directed Diffusion. Pode ser visto na Fig.4.1 um esquema simplificado dos módulos que o constituem e as suas interacções segundo a presente implementação. Em mais pormenor descrevem-se agora as suas funções, as suas interacções e o seu modo de funcionamento:

Bootstrap Module gere o estabelecimento de novas conexões com os vizinhos, executando o algoritmo de *bootstrap* segundo o mecanismo do SDD. Informa destas novas conexões o módulo *NeighborhoodModule*. Além disso invoca directamente as primitivas do módulo *CommunicationModule* de forma a comunicar com os vizinhos e estabelecer conexões seguras. Aquando da sua instalação, o sensor é programado para iniciar o *bootstrap* ao fim de um dado tempo;

Neighborhood Module gere todo o estado dos canais de comunicação com os vizinhos, incluindo os segredos dos canais seguros e outras informações tais como o comportamento do nó (e.g. nº de mensagens recebidas com sucesso, nº de mensagens corrompidas);

Communication Module gere todas as comunicações de entrada e saída com a camada inferior, serializando e cifrando os dados se necessário, pedindo nesse caso os segredos dos canais ao módulo *NeighborModule*. Nas comunicações de entrada, após serem desserializadas, as mensagens são entregues ao gestor, o módulo Diffusion;

PathLogic Module responsável pela manutenção dos caminhos, construindo-os com todas as mensagens de interesses e dados recebidas. Além disso, possui também um mecanismo de recuperação de caminhos, substituindo rotas degradadas por outras que entreguem dados em melhores condições. Esse mecanismo é baseado na definição de gradientes, retorno de dados e reforço de caminhos descrita na especificação do SDD. Na presente implementação, esse mecanismo verifica regularmente, consoante o data rate dos dados, se um dado nó reforçado lhe falhou na entrega dos dados, dando prioridade aos vizinhos que menos vezes falharam. Outros esquemas são possíveis, bastando para isso substituir este módulo por outra implementação.

Existe também o armazenamento da informação dos interesses, que é guardada em três repositórios distintos dependendo se o interesse foi registado localmente, se foi subscrito localmente, ou então nenhum dos casos anteriores. Os repositórios são cada um implementados segundo uma lista de objectos, tendo como indice o identificador do interesse. Passam-se a descrever:

InterestsList guarda a informação geral para cada interesse na rede (e.g. id, nº sequência, data rate). Além disso guarda também uma instância de autenticação μTesla do lado receptor de forma a autenticar o interesse recebido. Assim, mesmo que o interesse não seja subscrito terá de ser autenticado em cada nó para as suas definições poderem ser actualizadas, tal como o seu *data rate*;

RegisteredInterestsList guarda para cada interesse registado localmente a instância do lado emissor μTesla, assim como a chave para decifrar os dados retornados;

SubscribedInterestsList guarda para cada interesse subscrito a chave para poder cifrar os dados antes de retorná-los para o nó que o registou, assim como o número de sequência dos mesmos.

Uma vez recebida uma mensagem de interesse, é remetida pelo gestor *Diffusion* para o módulo de autenticação. Quando este módulo autenticar a mensagem retorna-a para o gestor *Diffusion* para ser tratada. No entanto a mensagem inicial, onde seguem as definições do interesse, é apenas autenticada com a chave de rede local desta camada, pois o protocolo μTesla ainda não está em funcionamento. Nessa primeira mensagem segue também cifrada a primeira chave da cadeia de autenticação μTesla e a chave para cifra de dados.

Apresentam-se agora os vários tipos de mensagens trocadas na rede segundo a visão deste componente:

Interesse mensagem enviada quando se pretende disseminar o registo de um novo interesse na rede, ou então uma actualização do mesmo. Note-se que o envio de dados para os subscritores de um interesse é efectuado através destas mensagens. A disseminação das mesmas é efectuada por inundação. Pode ser vista na listagem 4.1;

***Acknowledge de criação de interesse* de subscrição** mensagem retornada por um nó no momento da subscrição de um interesse para o nó que o registou. O retorno é efectuado por um subconjunto dos melhores caminhos reversos e serve para construir as rotas de encaminhamento numa primeira fase, antes dos dados reais serem retornados. Pode ser vista na listagem 4.2;

Revogação de chave de interesse mensagem de revogação de chave da cadeia de autenticação de interesse de acordo com o protocolo μTesla. É enviada por inundação. Pode ser vista na listagem 4.3;

Dados mensagem de dados retornada pelos subscritores de um interesse para o nó que o registou. É enviada pelos melhores 3 caminhos reversos, podendo esse número ser configurado. O algoritmo não impede que os caminhos possam convergir. Pode ser vista na listagem 4.4;

Manutenção de caminhos reversos mensagem enviada periodicamente, consoante o *dataRate* de retorno de dados de forma a manter activos os melhores caminhos. Pode ser vista na listagem 4.5.

Listagem 4.1: Mensagem de interesse.

```
1 public class SecureInterestMsg {
2   String id;
3   int seqNum;
4   int dataRate;
5   byte[] attrs;
6   byte[] securityInfo;
7   byte[] mac;
8 }
```

Listagem 4.2: Mensagem de *acknowledge*.

```
1 public class SecureAckMsg {
2   String id;
3   byte[] attrs;
4   short src;
5   byte[] mac;
6 }
```

Listagem 4.3: Mensagem de revogação de chave de interesse.

```
1 public class SecureKeyDisclosureMsg {
2   String id;
3   short epoch;
4   byte[] key;
5   byte[] mac;
6 }
```

Listagem 4.4: Mensagem de dados.

```
1 public class SecureDataMsg {
2   String id;
3   short src;
4   int seqNum;
5   byte[] data;
6   byte[] mac;
7 }
```

Listagem 4.5: Mensagem de dados manutenção de caminho reverso.

```
1 public class SecureRecoveryAckMsg {
2   String id;
3   short src;
4   int seqNum;
5   byte[] mac;
6 }
```

Existe ainda uma mensagem de remoção de interesse, enviada quando um nó decide remover um interesse previamente registado na rede. É enviada por inundação.

Existe também uma mensagem que encapsula outras, a qual limita a distância em termos de saltos a que são propagadas. Por exemplo, para criação de vizinhanças lógicas locais a um certo número máximo saltos.

4.4 Componente Infrastructure

Este componente na presente implementação conta apenas com um serviço que permite criar VLs na rede e gerir um espaço de tuplos, e com outro que testa o estado do nó local face a definições de uma vizinhança lógica. As acções que na presente implementação podem ser realizadas por esta camada são as seguintes:

Criação de uma VL é enviado um pedido à camadas inferior para disseminar o registo de uma VL para a rede. Todos os nós que recebam esse pedido e que respeitem as suas definições subscrevem o interesse e retornam uma confirmação mais tarde, apesar de não haver garantias que a mensagem seja recebida. Na invocação do método existe também a possibilidade de limitar a disseminação do pedido a um certo número de saltos. Isto é bastante útil caso se deseje criar VLs locais, por exemplo a pedido de um nó agregador numa zona, evitando-se enviar tráfego para toda a rede;

Remoção de uma VL é enviado um pedido para a camada inferior a pedir para remover todo o estado sobre a VL;

Escrita de tuplo um tuplo é adicionado através da operação *out*. Pode resultar no despoletar de uma reacção previamente instalada;

Leitura destrutiva de tuplo um tuplo é lido e removido através da operação *in*. Consequentemente o tuplo é retornado para a aplicação local ou remota que remeteu o pedido;

Instalação de reacção as reacções após registadas permitem retornar um tuplo assim que este apareça no espaço de tuplos no qual foi instalada. Caso as reacções sejam instaladas remotamente, os tuplos são retornados para a aplicação no nó que a instalou;

Remoção de reacção as reacções podem ser removidas de toda uma VL a pedido, ou então, caso uma VL seja removida da rede, todas as reacções instaladas sobre a mesma são automaticamente removidas.

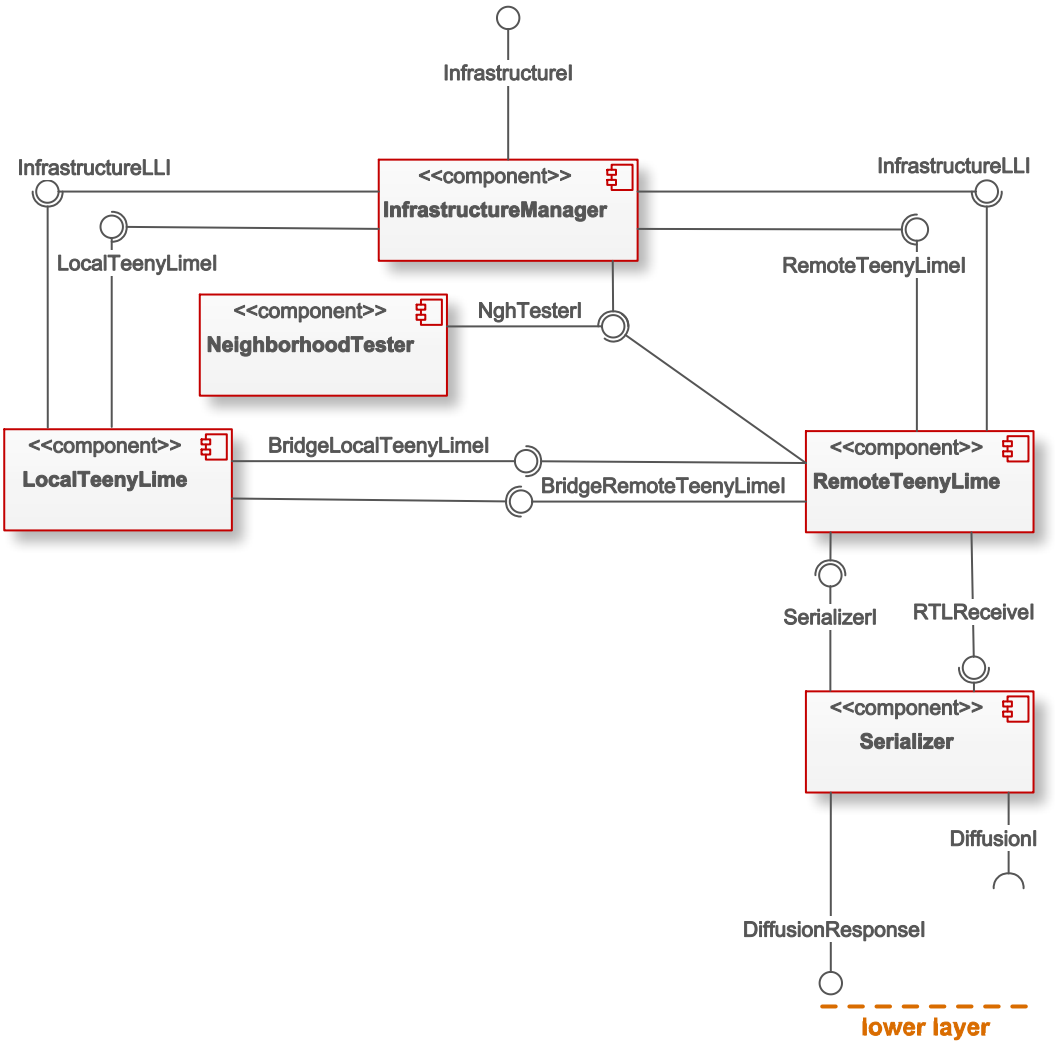


Figura 4.2: Infrastructure - módulos e suas dependências.

A implementação é semelhante à do sistema *TeenyLime*. Pode ser visto um esquema simplificado dos módulos que o constituem e as suas interacções, na Fig. 4.2.

Passam-se agora a descrever em pormenor cada um dos módulos e suas interacções:

InfrastructureManager é o módulo gestor do componente, o qual reencaminha os pedidos das aplicações para os serviços correctos. Reecaminha operações sobre o espaço de tuplos local para o módulo *LocalTeenyLime* e para a rede através do módulo *RemoteTeenyLime*. Além disso usa o módulo *NeighborhoodTester* de modo a testar se o nó local pertence a uma VL;

LocalTeenyLime gere o espaço local de tuplos e as reacções instaladas sobre o mesmo.

Listagem 4.6: Template de acesso ao estado de um nó.

```
1  public interface NodeTemplateI {
2    // sink or sensor
3    String device();
4    // temperature, aggregator, etc
5    public String type();
6    // logical location (e.g. 'Room 1A')
7    public String location();
8    // the unique id of the node in the network
9    public short address();
10   // returns the sample of the reading if the sensor is a sample prober or
11   // null otherwise.
12   public Object reading();
13   // returns the remaining power value of the battery in percentage
14   public double batteryPower();
15 }
```

Retorna tuplos quer para o *gestor* dirigido às aplicações, quer para o módulo *RemoteTeenyLime* dirigido a aplicações remotas, dependendo se a origem dos pedidos é local ou remota, respectivamente;

RemoteTeenyLime Gere todas as operações recebidas remotamente sobre as VLs a que este nó pertença, ou então as operações enviadas com destino às VLs criadas por este nó. Possui para isso uma ligação ao módulo *LocalTeenyLime* de modo a aceder ao espaço de tuplos e outra ao módulo *Serializer* que liga à camada de baixo e remete pedidos para a rede. Consulta também o módulo *NeighborhoodTester* para testar se o nó local pertence a uma certa VL, de forma a subscrevê-la. Por último são reencaminhados para as aplicações, através da ligação ao gestor *InfrastructureManager*, todos os resultados retornados devido a reacções que despoletaram nos nós remotos onde foram instaladas;

NeighborhoodTester possui a lógica para testar as definições de uma VL face ao estado local do nó. Uma VL pode ser construída com base em combinações lógicas de informação sobre o nó, tal como tipo de nó, localização lógica, tipo de amostras recolhidas, intervalo de valor da amostra, endereço ou carga da bateria. Essas definições são comparadas com um *template*, que possui as informações do nó, o qual pode ser visto na listagem 4.6;

Serializer serve de ponte entre esta camada e a camada inferior, pois ambas possuem lógicas completamente diferentes que necessitam de uma conversão em termos de invocação de métodos.

4.5 Módulo Aplicação

Os módulos de aplicação são instanciados nos sensores no momento da instalação. Cada módulo, como já se disse anteriormente, comunica com a API fornecida pelo componente *Infrastructure* e pode fazer uso de excertos de código auxiliares, denominados de *Tasks*, que ajudam em tarefas variadas como é o caso de agregações. Passam-se agora a descrever em mais pormenor certas propriedades destes módulos, dando especial ênfase ao modo como são programados:

Constituição dos tuplos os tuplos possuem campos, onde cada campo pode possuir um valor ou então um *wildcard*. Caso um tuplo possua pelo menos um campo *wildcard* é denominado de *template*. Mais informações sobre o funcionamento de tuplos podem ser encontradas em [MPR06]. Na presente implementação são suportados os seguintes tipos de campos: *String*, *Short*, *Integer*, *Float*, *Char*, *Byte* e *Blob* (conjunto de bytes). São ainda suportados *wildcards* sobre cada um deles;

Início de execução de aplicações é invocado um método `start()` sobre todas as aplicações de maneira a começarem a sua execução. Caso seja uma aplicação na estação central, é normal que as primeiras instruções sejam para criar uma VL. Caso seja uma aplicação nos nós do interior da rede é normal que comecem com a instalação de uma reacção no ET local de forma a poder receber as instruções na forma de tuplos, que são adicionados a esse mesmo espaço por aplicações remotas. Podem ser vistos nas listagens 4.7 e 4.8 exemplos do primeiro e segundo casos respectivamente. Apesar de essa ser uma regra geral, nada impede que os comportamentos se invertam, pois como se falou anteriormente, os nós do interior da rede podem também eles ter um papel de estação central. Observando ambas as figuras, pode-se ver que no primeiro caso foi criada uma VL sobre todos os sensores *agregadores* do *sector1* e no segundo caso vemos o *template* da reacção a ser instanciado com vários campos, que os tuplos de instrução dirigidos a todas as aplicações têm de respeitar;

Recepção de tuplos um tuplo é recebido pelas aplicações assincronamente através do método `tupleReady(TLOpId_t opId, Tuple tuple)`. O primeiro campo diz respeito ao identificador da operação local que fez o pedido e o segundo ao tuplo retornado. Podemos ver na listagem 4.9 um exemplo onde a aplicação recebe um tuplo com amostras dos sensores, consequência da instalação de uma reacção numa VL. Nesse exemplo, o identificador do nó origem dos dados vem no índice 3 e os dados no índice 4, sendo os anteriores índices de controlo.

Tarefas auxiliares denominadas também de *Tasks*, são excertos de código que podem ser usados por todas as aplicações. Pode ser visto na listagem 4.10 o código de uma Task que escreve para o ET um tuplo com os pares <*nó, amostra recolhida*> cujos valores das amostras sejam os **x** maiores, onde **x** é parameterizável.

Listagem 4.7: Criação de uma Vizinhança Lógica.

```
public void start() {
  Template[] templates = {new LocationTmpl("SECTOR1"),
                          new TypeTmpl("AGGREGATOR")};

  TemplateSet tmplSet = new TemplateSet(templates, true);

  Pair<TLOpId_t, String> res = tl.createLN(SinkCfg.APP_ID, tmplSet);

  sect1AggregatorsLN_Op = res.getFirst();
  sect1SensorsLN_Id = res.getSecond();
}
```

Listagem 4.8: Instalação de reacção para receber instruções.

```
public void start() {
  AbstractField[] fields = {new ByteValue(MyAppCfg.TUPLE_COMMAND),
                            new ShortValue(CommonCfg.APP_ID),
                            new AnyString(),
                            new AnyByte(),
                            new AnyBlob()};

    Tuple executeCodeReact = new Tuple(fields, (byte)5, true);

    executeCodeReact_Op = tl.addReaction(CommonCfg.APP_ID, null,
                                         executeCodeReact, 0);
}
```

Listagem 4.9: Recepção de tuplos.

```
public void tupleReady(TLOpId_t opId, Tuple tuple) {
  if(dataReact_Op != null && TLOpId_t.isEqual(opId, dataReact_Op)) {
    short src = ((ShortValue)tuple.fields[3]).value();
    byte[] sample = ((BlobValue)tuple.fields[4]).value();

    addSample(src, sample);
  }
  else{ //... }
}
```

Listagem 4.10: Excerto de código de agregação.

```
class Aggregate{
  private LinkedList<ReceivedSample>
  getHigherValues(LinkedList<ReceivedSample> samples){ //... }

  public void aggregate(LinkedList<ReceivedSample> samples){
    LinkedList<ReceivedSample> higherValues = getHigherValues(samples);
    byte[] aggregatedSamples;

    try {
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      ObjectOutputStream oos = new ObjectOutputStream(baos);

      for(ReceivedSample next : higherValues){
        oos.writeShort( next.src );
        oos.writeObject( next.sample );
      }

      aggregatedSamples = baos.toByteArray();
      oos.close();
    } catch (IOException e) { e.printStackTrace(); return; }

    AbstractField[] fields = new AbstractField[6];
    fields[0] = new ByteValue(MyAppCfg.TUPLE_DATA);
    fields[1] = new ShortValue(CommonCfg.APP_ID);
    fields[2] = new StringValue(lnId);
    fields[3] = new ShortValue(app.nodeState().address());
    fields[4] = new ByteValue((byte)higherValues.size());
    fields[5] = new BlobValue(aggregatedSamples,
                              (short)aggregatedSamples.length);

    Tuple aggregatedData = new Tuple(fields, (byte)6, false);
    lastTuple = aggregatedData;
    app.getManager().out(CommonCfg.APP_ID, null, aggregatedData, 0);
  }
}
```

4.6 Mecanismos de segurança

Passam-se agora a apresentar, para cada uma das garantias de segurança apresentadas na secção 3.7, o respectivo mecanismo de segurança usado:

Com. Segura (confidencialidade, integridade, autenticidade) os dados são cifrados

com uma chave de rede que todos os nós conhecem, garantindo confidencialidade. É gerado um MIC (Message Integration Code) de 4 bytes sobre os dados e enviado juntamente com a mensagem, que uma vez testado permite verificar a integridade e autenticidade. A autenticidade é verificada não sobre um nó em particular, mas garante-se que o nó faz parte do grupo legítimo de nós que conhece a chave;

Com. Segura (não replicação) é usado um esquema de filtros de Bloom associado a um sistema de épocas. Como todos os sensores estão sincronizados neste sistema, as mensagens são cifradas juntamente com o número da época, o qual não é enviado com a mensagem. Posteriormente são feitas duas tentativas para decifrar, com a época actual e com a anterior, no nó receptor. Além disso, para este intervalo de duas épocas o filtro de Bloom permite excluir as mensagens replicadas. Garante-se que nenhuma mensagem é recebida duas vezes e que é recente;

Enc. Seguro (disponibilidade de rotas) o modo de funcionamento do protocolo SDD permite que os dados sejam retornados por várias rotas e os caminhos recuperados, sendo na presente implementação retornados por 3 rotas, que podem não ser disjuntas. Caso um nó verifique que outro não lhe está a enviar os dados à taxa pretendida, reforça outro disponível capaz de lhe enviar os dados. Isto permite que os dados sejam sempre retornados caso haja rotas descomprometidas disponíveis para tal;

Enc. Seguro (integridade, autenticidade e confidencialidade) garantias dadas apenas entre nós vizinhos. São geradas chaves partilhadas entre cada par de nós através dum *bootstrap* imediatamente após os nós serem instalados. Como as comunicações são locais e é necessário garantir-se bidireccionalidade para geração de um canal seguro, os ataques à comunicação a longas distâncias são evitados;

Enc. Seguro (autenticação da estação central) toda a rede possui uma chave partilhada a este nível, que serve para as operações gerais nesta camada, onde seja necessária segurança adicional. No entanto, esta chave não é refrescada nesta implementação. Relativamente ao protocolo SDD são enviados interesses para todos os nós a partir da estação central e geradas cadeias de autenticação sobre cada um através do protocolo μTesla, fazendo com que todas as mensagens sejam autenticadas. A primeira mensagem, no entanto, vai autenticada com a chave de rede pois o μTesla respectivo ainda não está em funcionamento. Como já foi dito, todos os nós podem também eles ter um papel de estação central, no sentido em que podem também eles enviar interesses para a rede;

Enc. Seguro (cifra de dados) no momento de envio da primeira mensagem de interesse, é enviada a chave para posterior cifra dos dados, que fica guardada apenas nos nós que subscrevam o interesse. Esta chave, nesta implementação, não é refrescada, usando-se sempre a mesma chave do início ao fim das execuções.

4.7 Considerações finais

Neste capítulo mostrou-se pormenorizadamente todos os componentes, módulos e acções que a pilha de serviços implementada suporta. Devido ao escasso tempo de implementação, não se perdeu muito tempo a analisar os melhores métodos de optimizar as estruturas de dados, redirigindo-se os esforços para a correcção da implementação e não para o máximo desempenho de processamento. É por isso aconselhável que se façam estas optimizações caso eventualmente se adapte e migre o código para sensores reais que corram máquinas virtuais *Java*, para maiores poupanças energéticas.

5

Testes e validação da plataforma

Este capítulo apresenta os testes realizados, resultados obtidos e sua análise que serviram de base para a validação do funcionamento da pilha implementada. Os testes foram realizados no ambiente de simulação descrito na secção 3.2. Os testes inserem-se em 4 classes: cobertura de rede, gastos energéticos, fiabilidade de entrega de dados e resiliência face a falhas.

São primeiro apresentados os vários parâmetros e condições de teste, seguidos dos testes, resultados e análises para cada uma das classes pela ordem apresentada. Os testes das duas últimas classes encontram-se numa mesma secção pois os testes foram efectuados em conjunto e as análises aos resultados encontram-se intimamente ligadas.

5.1 Parâmetros e condições de teste

Todos os testes foram realizados numa rede segundo uma área rectangular plana de 1200 x 480 m (0,576 km^2). Nessa área, os sensores foram colocados segundo uma distribuição topológica aleatória (de acordo com a quantidade especificada em cada teste). Foram utilizadas características de transmissão com raios de alcance a variar entre 15 e 25 metros e modelação MAC para comunicações IEEE 802.15.4, com simulação de colisões rádio. Usou-se como taxa máxima de transmissão de dados a conseguida pelos sensores *MicaZ* [Mic], capazes de enviar a 10.500B/s. Por último, para os testes de energia supôs-se que cada nó era capaz de armazenar o equivalente a duas pilhas AA alcalinas, que é o utilizado pelos sensores actuais mais conhecidos, como *Mica2*, *MicaZ*

ou *Sunspot*.

Esclarecem-se agora todos os outros parâmetros, medidas e conceitos utilizados nos testes:

Throughput ou taxa de transmissão em B/s. Define o número máximo de bytes que podem ser transmitidos pelos sensores num segundo;

Tentativa de *bootstrap* corresponde a uma execução do protocolo de *bootstrap*, onde os sensores estabelecem canais seguros com os seus vizinhos directos. Para várias tentativas, cada uma inicia-se com um *delay* de 1s relativamente à anterior;

Duração do *bootstrap* duração de todo o processo de *bootstrap* desde o momento em que a primeira mensagem é enviada para a rede, até ao momento em que a última é recebida;

Ligações totais na rede corresponde ao total de canais unidireccionais não seguros entre os nós da rede;

Ligações seguras na rede corresponde ao total de canais unidireccionais seguros por um segredo partilhado apenas por cada par de nós ao nível da camada de encaminhamento;

Número de colisões corresponde ao número total de mensagens que foram enviadas com sucesso pelos nós da rede, mas que chegaram corrompidas a um nó e foram por isso descartadas;

Número de retransmissões corresponde ao número total de adiamentos de envios de mensagens pelos nós da rede, após ser verificado que o canal de transmissão continha ruído que impossibilitava a sua transmissão;

Registo de uma VL mensagem enviada por *flooding* para toda a rede com as especificações de uma VL e gravada na memória de cada sensor que a recebe. Nos testes cada VL possui uma área, de onde as amostras de temperatura são recolhidas. Podem também ser especificadas as taxas de recolha de amostras;

Agregador de uma VL nó situado junto às fontes de dados, que recolhe e agrega as amostras retornadas pelos nós de uma VL;

Nós participantes do encaminhamento todos os nós que, pelo menos por uma vez, receberam e encaminharam dados reais pertencentes a um fluxo. Não são englobados os nós que enviaram para esse fluxo mensagens de dados exploratórios.

5.2 Testes de cobertura de rede

São a seguir apresentados os testes que medem o impacto do número de tentativas de *bootstrap*, da densidade de nós e da taxa de transmissão de mensagens no estabelecimento de um maior número de canais seguros, aumentando assim a cobertura da rede.

5.2.1 Variação do número de tentativas de *bootstrap*

Com este teste procurou-se medir o impacto do número de tentativas de *bootstrap* na cobertura final dos nós na rede.

Método

As constantes de teste foram as seguintes: 3750 nós e *throughput* máximo de 1 KB/s. Variou-se o número de tentativas de *bootstrap* em 1, 2, 3, 4 e 5 tentativas. Contou-se o número de ligações totais e o número efectivo de ligações seguras. Contaram-se também os números de colisões e de retransmissões e a duração do processo.

Prova de teste

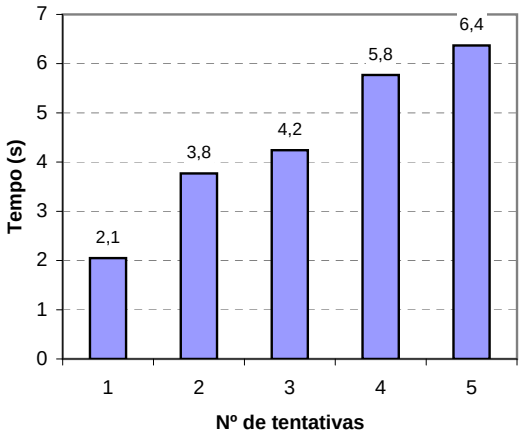
Aumentando-se o número de tentativas de *bootstrap* faz com que haja uma maior probabilidade de um nó fazer chegar com sucesso uma mensagem de *Hello* ou *Reply* aos seus vizinhos. No entanto há que verificar se esse aumento não se traduz num aumento abusivo do número de colisões.

Resultados

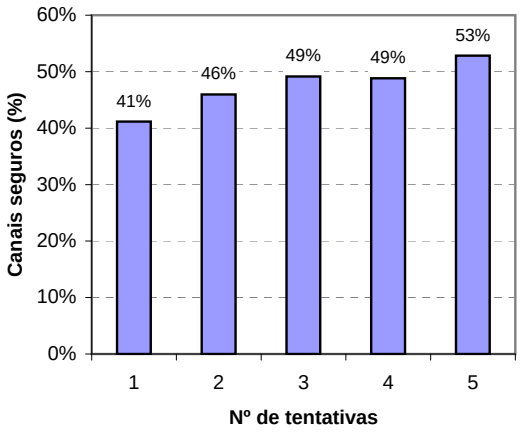
Os resultados podem ser vistos na figura 5.1. Foi medida a duração do *bootstrap*, calculada a percentagem de canais seguros resultantes, a média de colisões e retransmissões e a média de ligações seguras por nó.

Conclusões

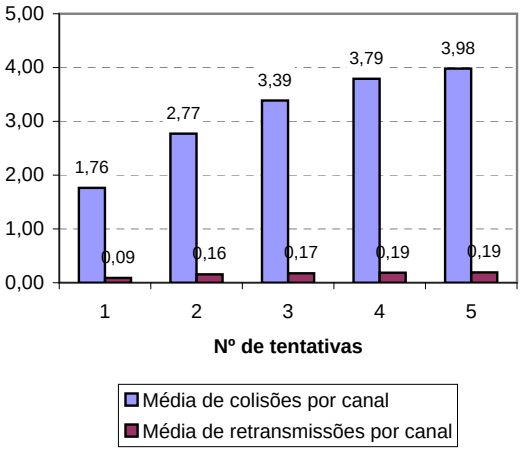
Analisando a figura 5.1(a) verifica-se que o acréscimo de duração do processo para 5 tentativas é praticamente desprezável, resultando apenas no aumento de cerca de 4 segundos, face aos 2 iniciais. Isto deve-se ao facto do *throughput* ser elevado e as mensagens serem rapidamente despachadas.



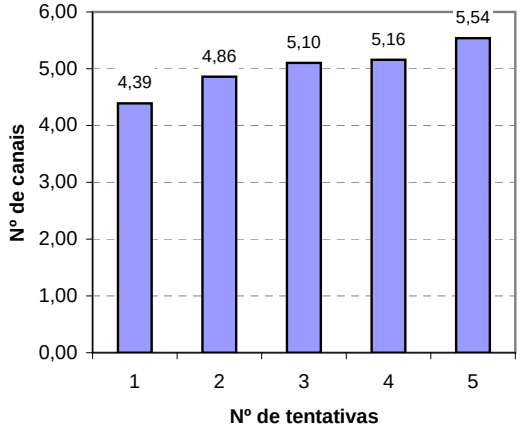
(a) Duração do bootstrap.



(b) Percentagem de canais seguros.



(c) Média de colisões e retransmissões por canal.



(d) Média de ligações seguras por nó.

Figura 5.1: Resultados do bootstrap variando o número de tentativas.

Na figura 5.1(b) nota-se que a percentagem de canais seguros aumentou consideravelmente para 5 tentativas cerca de 12%. Isto deve-se, como se previa, a que mais mensagens foram recebidas com sucesso e mais canais gerados.

Na figura 5.1(c) nota-se que, apesar de aproximadamente 5 vezes mais mensagens circularem na rede, apenas se registou um acréscimo de cerca de 2 colisões por canal. Isto deve-se a que os canais permaneceram livres grande parte do tempo pois as mensagens de bootstrap são muito pequenas (aproximadamente 40 bytes), o que se pode confirmar olhando para o número de retransmissões que permaneceu praticamente igual.

Na figura 5.1(d) regista-se um aumento em média de mais de um vizinho por cada nó, devendo-se isto ao aumento do número de canais seguros.

Conclui-se portanto que seria viável aumentar o número de tentativas para 5, pois o ganho de 1 vizinho por nó sobrepõe-se ao facto de o número de colisões aumentar para pouco mais do dobro. Como o *bootstrap* é um processo de apenas alguns segundos e os pacotes muito pequenos, a energia gasta no envio dos mesmos e possíveis retransmissões é praticamente nula.

5.2.2 Variação da densidade de nós

Com este teste procurou-se medir o impacto da densidade de nós na cobertura final dos nós na rede.

Método

As constantes de teste foram as seguintes: *throughput* máximo de 1 KB/s e uma única tentativa de *bootstrap*. Variou-se o número de nós em 2500, 3750 e 5000 nós. Contou-se o número de ligações totais e o número efectivo de ligações seguras. Contou-se também o número de colisões, retransmissões e duração do processo.

Além do teste à fase de *bootstrap*, foi efectuado na mesma execução um teste à cobertura efectiva da rede, onde o nó sink após a fase inicial registou uma VL em toda a rede. No final contou-se o número de nós que efectivamente receberam esse registo.

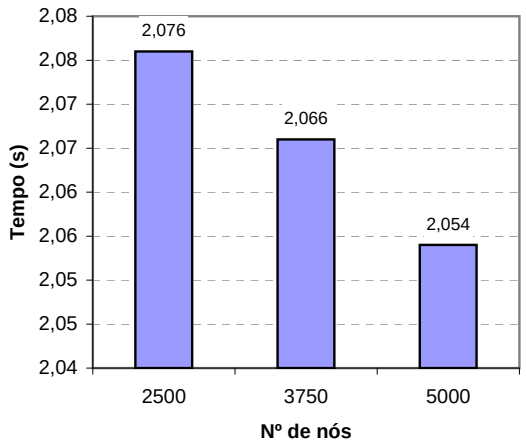
Para cada número de nós foram efectuadas 5 execuções e os resultados obtidos através da sua média aritmética simples.

Prova de teste

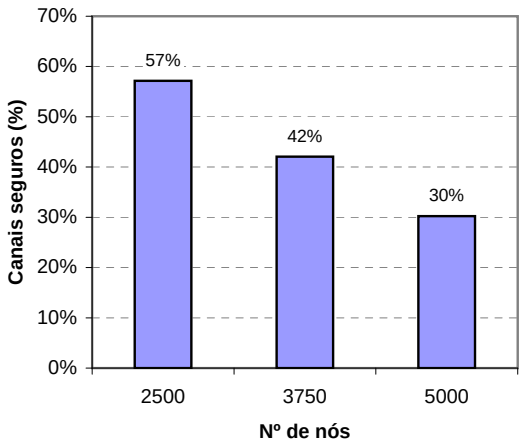
Aumentando-se a densidade de nós contribui para que mais vizinhos estejam ao alcance de qualquer nó. Há assim maiores probabilidades de se formarem mais canais seguros. Há, no entanto, que verificar até que ponto esse aumento no número de vizinhos faz com que os canais de comunicação fiquem cheios de tráfego e comecem a gerar enormes quantidades de colisões.

Resultados

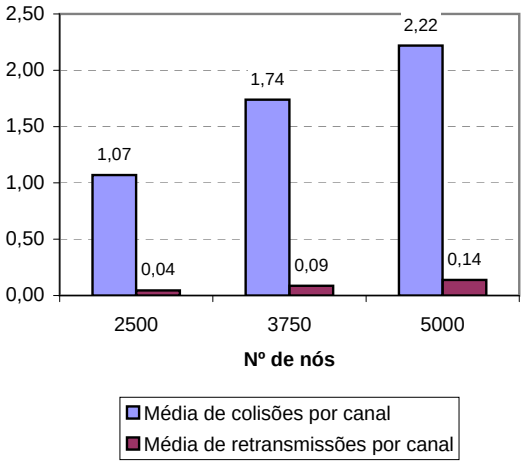
Os resultados podem ser vistos na figura 5.2. Foi medida a duração do *bootstrap*, calculada a percentagem de canais seguros resultantes, a média de colisões e retransmissões, a média de ligações seguras por nó e a percentagem de nós cobertos pelo nó *sink*.



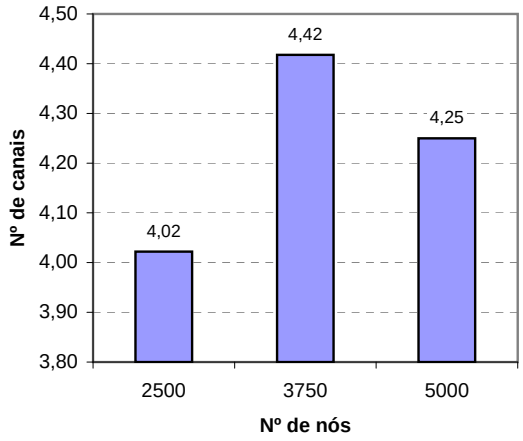
(a) Duração do bootstrap.



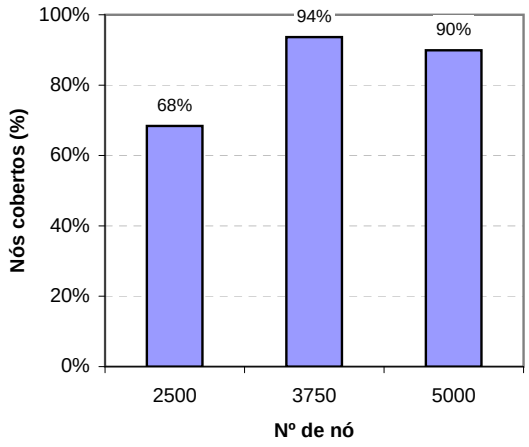
(b) Percentagem de canais seguros.



(c) Média de colisões e retransmissões por canal.



(d) Média de ligações seguras por nó.



(e) Percentagem de nós cobertos pelo nó sink.

Figura 5.2: Resultados do bootstrap variando a densidade de nós.

Conclusões

Na figura 5.2(a) nota-se que, com o aumento da densidade, o tempo de *bootstrap* diminui muito ligeiramente. Isto deve-se a que, desde o início, devido aos canais estarem mais congestionados, houve um aumento no número de colisões, aumentando a perca de mensagens, quebrando o protocolo entre parte dos vizinhos.

A figura 5.2(b) vem comprovar a ideia anterior, notando-se uma drástica diminuição de percentagem de canais seguros com o aumento da densidade, o que não quer dizer que a rede não esteja melhor coberta, como se verá adiante.

A figura 5.2(c) mostra que o aumento da densidade para o dobro, se traduz no dobro das colisões, o que seria de esperar. O número de retransmissões aumentou dez vezes, devido aos canais estarem menos tempo livres.

Na figura 5.2(d) verifica-se que 3750 nós é o número ideal para a área do teste, registando um maior número de vizinhos por nó. Para 2500 os nós estão muito afastados e existem poucos vizinhos e para 5000 geram-se demasiadas colisões, o que se traduz em muito menos processos de *bootstrap* efectuados com sucesso.

A figura 5.2(e) vem comprovar a ideia anterior, notando-se que o nó *sink* consegue alcançar uma maior percentagem de nós caso o número total seja 3750.

Conclui-se, portanto, que, para a área em que foi efectuado o teste, 2500 nós é pouca densidade, gerando-se partições na rede e fazendo com que as mensagens não cheguem a grande parte dos mesmos. Pelo contrário, 5000 são demasiados, gerando-se um aumento na quantidade de colisões, devido aos nós estarem muito próximos, fazendo com que muitas mensagens se percam. O ideal são portanto 3750 nós.

5.2.3 Variação da taxa de transmissão

Com este teste procurou-se medir o impacto do *throughput* de envio de mensagens na cobertura final dos nós na rede.

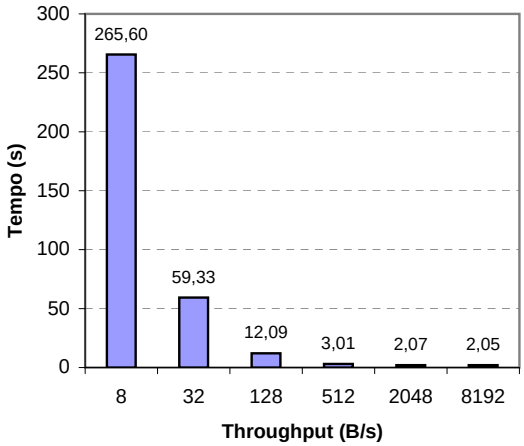
Método

As constantes de teste foram as seguintes: 5000 nós e uma única tentativa de *bootstrap*. Variou-se o *throughput* máximo em 8.192B/s, 2.048B/s, 512B/s, 128B/s, 32B/s e 8B/s. Contou-se o número de ligações totais e o número efectivo de ligações seguras. Contou-se também o número de colisões, retransmissões e duração do processo.

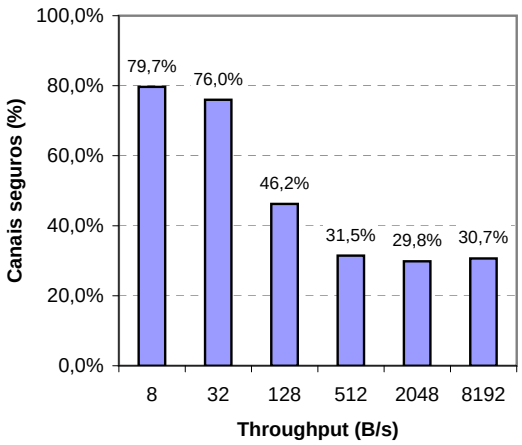
Prova de teste

Variando-se o *throughput* de envio de dados faz com que mais ou menos dados circulem nos canais simultaneamente, aumentando ou diminuindo a probabilidade de acontecerem colisões. O que é preciso verificar é até que ponto a variação desse parâmetro afecta consideravelmente o número de canais seguros estabelecidos.

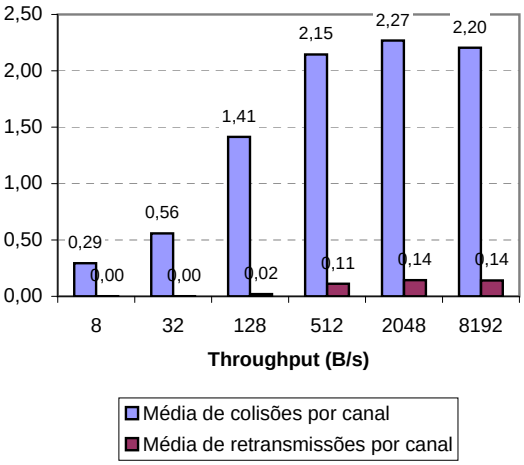
Resultados



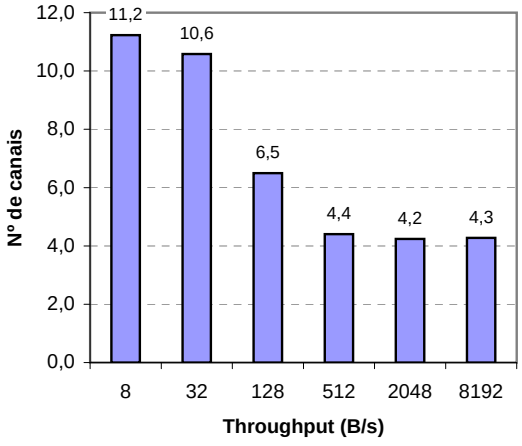
(a) Duração do bootstrap.



(b) Percentagem de canais seguros.



(c) Média de colisões e retransmissões por canal.



(d) Média de ligações seguras por nó.

Figura 5.3: Resultados do bootstrap variando o *throughput* máximo de envio de dados.

Os resultados podem ser vistos na figura 5.3. Foi medida a duração do *bootstrap*,

calculada a percentagem de canais seguros resultantes, a média de colisões e retransmissões e a média de ligações seguras por nó.

Conclusões

Na figura 5.3(a), como era de esperar, nota-se um enorme aumento do tempo de *bootstrap* para os *throughputs* mais baixos de 32B/s e 8 B/s. Como as mensagens nesta fase possuem cerca de 40 bytes, têm de aguardar imenso tempo antes de serem transmitidas.

A figura 5.3(b) vem comprovar as expectativas, onde os *throughputs* mais baixos conseguem percentagens de canais seguros a tender para o óptimo, começando-se a ver melhorias significativas nos 128B/s.

A figura 5.3(c) mostra o aumento do número de colisões com o aumento do *throughput*, sendo que a partir dos 512B/s não se notam alterações significativas.

Na figura 5.3(d) vemos o número médio de ligações a aumentar drasticamente com a diminuição do *throughput*, conseguindo uma média de 11 vizinhos por nó em 8B/s. De 512B/s para cima não se notam alterações.

Após analisar os resultados, conclui-se que o *throughput* mínimo será sempre o óptimo para o processo de *bootstrap*. No entanto, por serem impraticáveis devido aos seus altos tempos de execução, excluem-se os throughputs de 8B/s e 32 B/s, sendo que 128B/s é o que apresenta resultados gerais mais interessantes na linha das duas ideias anteriores. Os *throughputs* superiores a esse valor não trazem nenhuma vantagem pois a percentagem de canais seguros é muito baixa.

5.3 Testes de gastos energéticos

São a seguir apresentados os testes que medem a energia dispendida num processo de agregação, podendo com isso prever-se o tempo de vida de um nó nesse mesmo processo, e a contribuição de cada camada da pilha para os gastos energéticos totais.

5.3.1 Energia dispendida no processo de agregação

Com este teste procurou-se medir o impacto do número de nós pertencentes a uma VL nos gastos energéticos de uma operação de agregação num nó agregador, de modo a calcular o tempo aproximado de vida do mesmo.

Método

As constantes de teste foram as seguintes: 3750 nós, *throughput* máximo de 256B/s e taxas de recolhas de amostras de 20 em 20 segundos. Variou-se a área da VL para 100x100m, 100x200m e 100x300m. Comparou-se a energia gasta pelo nó agregador durante 1h de funcionamento.

Para cada uma das áreas efectuaram-se três testes, tendo sido os resultados obtidos através da sua média aritmética simples.

Prova de teste

Uma operação de agregação numa VL está dependente que os nós pertencentes à mesma retornem dados para um nó agregador, que por sua vez as encaminha normalmente para um nó sink. Ora, medindo no nó agregador a energia das mensagens recebidas e enviadas, consegue-se prever o tempo aproximado de vida que esse nó agregador tem. Não se sabe, no entanto, se essa energia gasta é proporcional ao número de nós da VL, pois pode haver mensagens que se percam quando uma VL tem demasiados nós, devido a colisões.

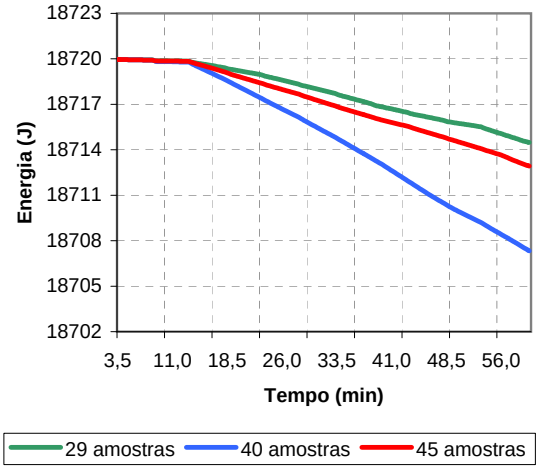
Resultados

Os resultados podem ser vistos na figura 5.4. Foi medida a energia gasta para três quantidades de nós para cada VL com os tamanhos 100x100m, 100x200m e 100x300m, assim como a média das três áreas.

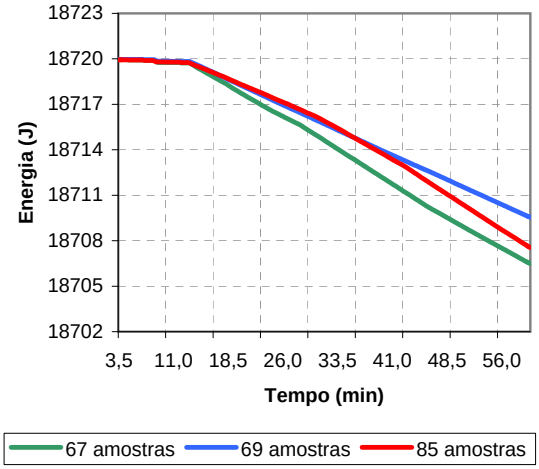
Conclusões

Analisando a figura 5.4(a) pode-se ver que um maior número de nós de uma VL não está intimamente ligado a uma maior quantidade de energia gasta, pois com 45 nós há muito menos energia gasta do que com 40 nós. Isto pode-se dever a várias razões, sendo que a perca de mensagens com amostras pode ser uma delas. O facto do nó agregador ter ficado numa periferia da VL também pode contribuir, pois intercepta menos mensagens nos canais do que se ficasse no seu centro. O número de vizinhos também influencia, pois podem enviar-lhe mensagens duplicadas que chegaram por caminhos distintos.

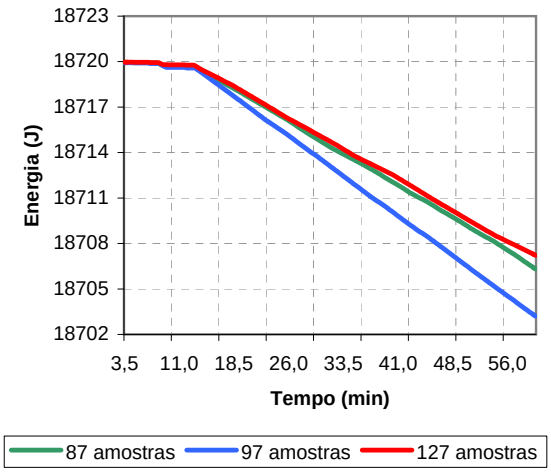
As figuras 5.4(b) e 5.4(c) também reflectem a ideia anterior, sendo o caso mais evidente o da energia gasta a agregar amostras de 127 nós ser bastante menor que de 97



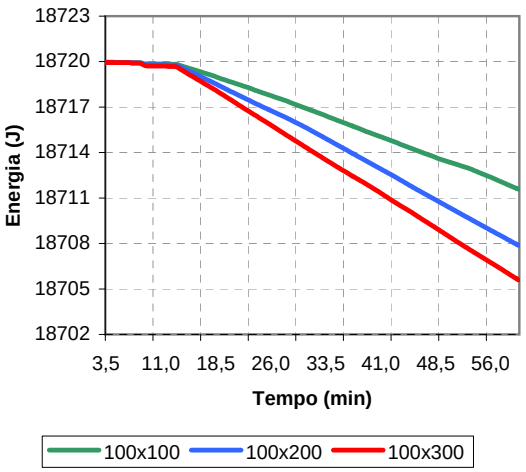
(a) Gastos energéticos com área da VL 100x100m.



(b) Gastos energéticos com área da VL 100x200m.



(c) Gastos energéticos com área da VL 100x300m.



(d) Média dos resultados anteriores para as três áreas.

Figura 5.4: Resultados dos gastos energéticos num nó agregador variando o tamanho da VL.

nós. Muito provavelmente bastantes mensagens se perderam antes de chegarem ao agregador.

As médias dos resultados anteriores, vistas na figura 5.4(d), mostram uma quantidade gasta de energia proporcional ao número de nós, mas os testes pecam por ser em baixo número, não sendo possível assim comprovar essa ideia.

Calculando agora o tempo médio de vida do nó agregador, tendo por base a média dos resultados da área 100x300m, verifica-se que o nó começou o processo de agregação aos 15m, tendo gasto nos 45m seguintes cerca de 15J. Calculando, através duma regra matemática conhecida, o tempo que leva a descarregar os 18720J iniciais, obtemos um tempo aproximado de vida para o nó de 39 dias. Esse tempo de vida pode ser estendido diminuindo a frequência de recolha de amostras, ou então usando técnicas com mais do que um nó agregador, dividindo o trabalho em fatias de tempo. Por exemplo, para prolongar por 365 dias o tempo de vida de uma operação de agregação, combinando a técnica mencionada, seriam necessários cerca de 10 nós agregadores, o que é bastante praticável.

5.3.2 Energia dispendida pelas camadas da pilha

Com este teste procurou-se medir a contribuição em termos de gastos energéticos dos serviços de segurança de cada camada da pilha para os gastos totais.

Método

As constantes de teste foram as seguintes: 3750 nós e *throughput* máximo de 256B/s. Criou-se uma VL com área 100x100m e recolheram-se resultados dum nó agregador de 20 em 20 segundos no nó *sink*. Mediram-se no nó *sink*, para essa execução, os gastos energéticos da responsabilidade de cada camada da pilha, durante uma 1 hora.

Todas as operações de comunicação de *overheads* de segurança, tal como mensagens especiais de segurança, *bootstrap* e *MICs* das mensagens, e de operações sobre as mensagens de cifra/decifra de dados foram contabilizadas na camada responsável pela sua geração.

Prova de teste

Medindo a contribuição dos serviços de segurança de cada camada da pilha para os gastos energéticos totais, permite verificar se o *overhead* comparado com os gastos totais tem um peso bastante significativo.

Resultados

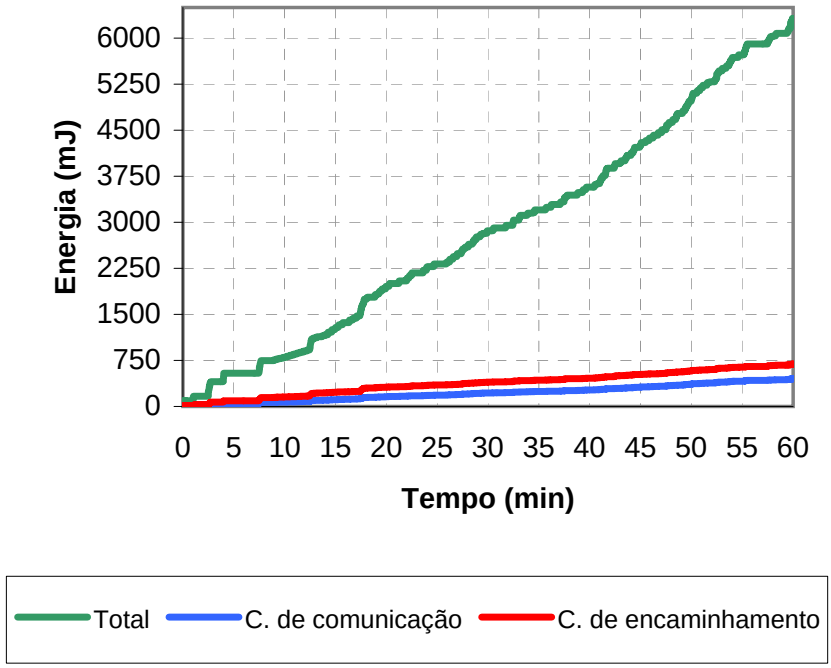


Figura 5.5: Comparação da energia dispendida pelos serviços de segurança das camadas de comunicação e encaminhamento face à energia total gasta no nó *sink*.

Os resultados podem ser vistos na figura 5.5. Foi medida a energia gasta nas camadas de comunicação e encaminhamento, assim como a energia total gasta no nó *sink*.

Conclusões

Analisando a figura 5.5, verifica-se um aumento constante das energias dispendidas, o que antes de mais prova a ausência de anomalias no retorno de dados agregados para o nó *sink*, sendo feito a um ritmo constante, apesar de se notar algumas irregularidades na linha de energia total, sinónimo de perca de algumas mensagens.

Em termos energéticos nota-se que a camada de encaminhamento consome cerca de 10% do total gasto e a camada de comunicação cerca de 7%. Estes resultados, não sendo bastante elevados, são significativos, mas a garantia de um elevado grau de segurança na pilha só se consegue em troca de consumos adicionais de energia.

5.4 Testes de fiabilidade e resiliência

Apresentam-se de seguida os testes que permitem medir a fiabilidade e resiliência do protocolo de encaminhamento. Note-se que ambos os testes foram efectuados em conjunto e, por estarem intimamente ligados, são os dois apresentados numa única secção.

5.4.1 Fiabilidade e resiliência no encaminhamento

Com este teste procurou-se medir a variação na taxa de mensagens recebidas e, ao mesmo tempo, na quantidade de nós participantes no encaminhamento, face a falhas dos nós encaminhadores, variação da taxa de refrescamento de interesse e de dados exploratórios.

Método

As constantes de teste foram as seguintes: 3750 nós e *throughput* máximo de 256B/s. Criou-se uma VL com área 100x100m e retornaram-se resultados dum nó agregador de 30 em 30 segundos para o nó *sink*. Efectuaram-se testes variando as mensagens de refrescamento de interesse e de dados exploratórios, ambas simultaneamente para 1 e 2 minutos. Para cada um dos valores anteriores, variaram-se ainda as falhas nos nós encaminhadores para acontecerem de 10 em 10 segundos e de 30 em 30 segundos, medindo-se também uma situação sem falhas. A simulação destas falhas consistiu no desligar do rádio de comunicação. Contabilizou-se para cada combinação o número de mensagens enviadas e recebidas assim como o número de nós encaminhadores participantes no encaminhamento de dados reais.

Prova de teste

Injectando-se falhas nos nós encaminhadores faz com que os vizinhos de um nó que falhou se readaptem e passem a encaminhar por outros caminhos. Isto deve-se ao algoritmo adaptativo de encaminhamento de dados do SDD. Mudanças no tempo de refrescamento de interesses, conseguido através de revogações de chaves de autenticação de interesses, e de dados exploratórios faz com que os nós se reorganizem mais ou menos rapidamente para o correcto encaminhamento dos dados.

Medindo-se a taxa de mensagens recebidas ao longo do tempo, consegue-se verificar se muitas mensagens foram perdidas e se essa perda é constante ou se se mantém igual.

Além disso a variação ao longo do tempo do número de nós encaminhadores permite verificar se houve crescimentos abruptos nesse número, situação onde o nó que

falhou muito provavelmente pertencia a uma rota principal e os dados tiveram de se encaminhar por outra distante, ou então se esse número aumentou ligeiramente, caso onde a rede se auto-organizou localmente e manteve praticamente as mesmas rotas.

Resultados

Os resultados podem ser vistos na figura 5.6. Foi medido o número de mensagens recebidas e variação do número de nós encaminhadores, face a falhas de nós e tempo de refrescamento de interesses e de dados exploratórios.

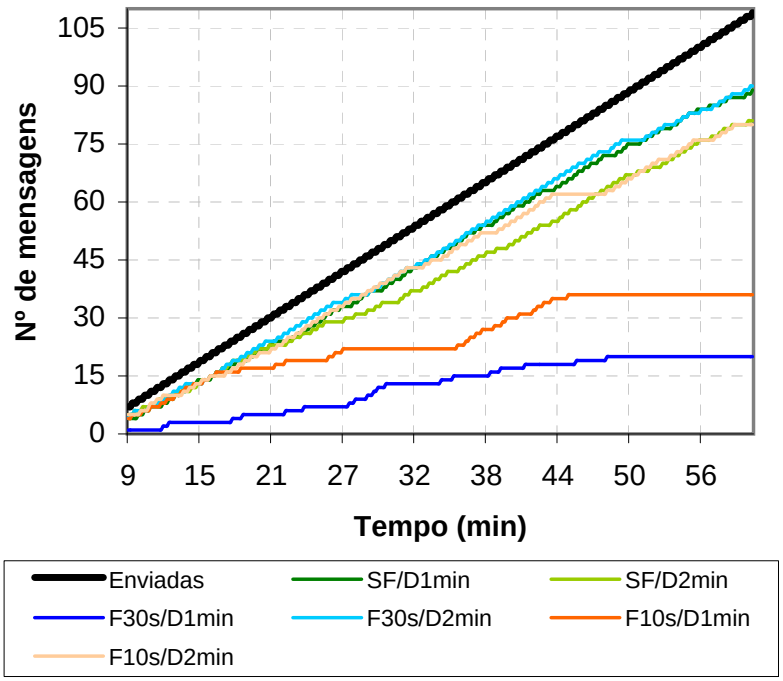
Em ambas as figuras interprete-se na legenda as situações 'SF' como 'Sem Falhas', 'Fxs' como 'Falhas de x em x segundos' e 'Dxmin' como 'Refrescamentos de interesses e envio de mensagens de dados exploratórios de x em x min'.

Conclusões

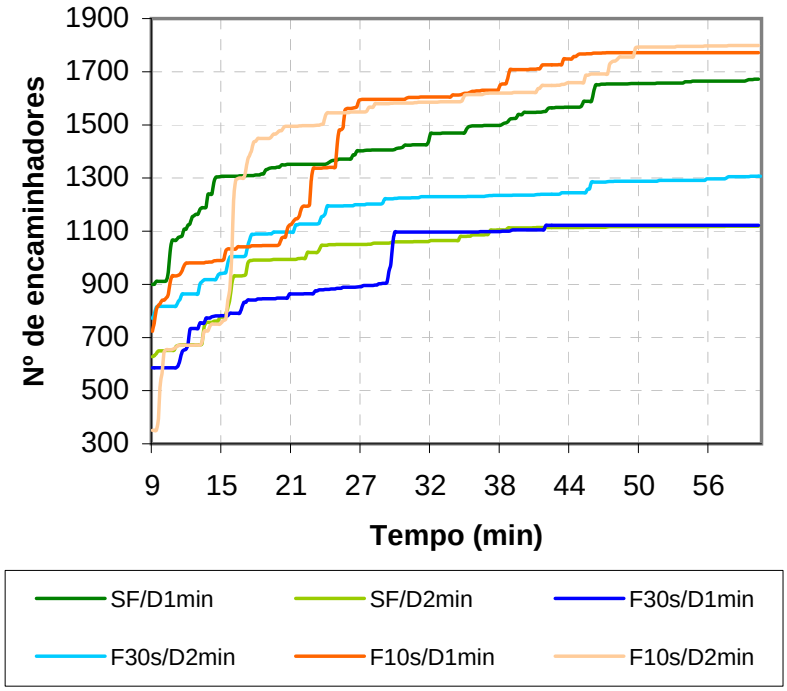
Analisando primeiro os casos onde não houve falhas (linhas a verde), observa-se uma constante recepção de mensagens ao longo do tempo, divergindo ligeiramente em relação ao total de mensagens enviadas, tal como era esperado. Nota-se também uma estagnação do número de nós encaminhadores para uma revogação de 2min, sendo este o caso esperado, mas um incremento neste número para uma revogação de 1min. Isto deve-se possivelmente a que no primeiro caso as rotas se mantiveram sempre as mesmas ao longo do tempo, sendo que no segundo caso os dados foram reencaminhados por melhores rotas entretanto descobertas. Isto vem provar um não determinismo em termos de rotas, dificultando a vida a um atacante.

Já para os casos com falhas de 30s (linhas a azul), observam-se situações distintas. Com revogações de 1min desde cedo se notou uma baixa taxa de entrega de mensagens, pois o número de nós encaminhadores começou muito baixo, devendo-se isso a que muito provavelmente desde cedo falharam nós fulcrais que ligavam partições numa zona da rede, deixando pouquíssimas rotas válidas e, com menos rotas de retorno de dados, mais mensagens foram perdidas. Já com revogações de 2min, a rede manteve-se bastante estável, estagnando o aumento do número de nós encaminhadores aos cerca de 25min e entregando uma alta taxa de mensagens. Aliás, foi a mais alta de todas.

Nos casos com falhas de 10s (linhas a cor-de-laranja), observa-se em ambos um crescimento exagerado do número de nós encaminhadores ao longo do tempo, sendo que com revogações de 1min a taxa de mensagens entregues foi sempre demasiado baixa, possivelmente devido à mesma razão das falhas de 30s com revogações de 1min. Já a taxa de entrega de mensagens para revogações de 2 min foi sempre muito alta.



(a) Variação da taxa de entrega de mensagens.



(b) Variação do número de nós encaminhadores.

Figura 5.6: Variação da taxa de entrega de mensagens e número de nós encaminhadores, face à frequência de falhas de nós encaminhadores e taxa de refrescamento de interesses e dados exploratórios.

Pelos resultados nota-se, ao contrário do esperado, que as revogações de 1min apresentam piores resultados em termos de fiabilidade. Isto deve-se a que possivelmente os fluxos de retorno de dados colidiam mais vezes com as revogações e mensagens de dados exploratórios enviadas mais frequentemente, perdendo-se assim mais vezes. No entanto, pode apenas ter sido uma coicidência de se terem formado poucos caminhos de retorno de dados, ou muitos caminhos coicidentes, em que falharam nós de rotas sectoriais importantes da rede. É portanto um assunto que teria de ser estudado em mais detalhe.

6

Conclusões e trabalho futuro

Neste capítulo são apresentadas as principais conclusões da presente dissertação, bem como os aspectos em aberto para trabalho futuro.

6.1 Conclusões

As RSSF estão a despoletar o interesse da comunidade científica e de muitas empresas, pois estas podem solucionar muitos problemas que surgem quando existe a necessidade de monitorizar em larga escala, por um longo período, por exemplo, um terreno de difícil acesso ou um campo de batalha, onde não seja possível efectuar-se qualquer tipo de manutenção da rede. Apesar dos fracos recursos dos nós dispositivos, estas redes são capazes de comunicar e se auto organizar, sendo que com técnicas adequadas de sincronização podem prolongar a sua vida útil à máxima eficiência por muitos meses, ou até anos.

O problema surge quando estas redes executam sem qualquer tipo de segurança e onde um atacante pode interceptar mensagens, forjar dados, ou mesmo aceder fisicamente aos nós roubando os seus segredos. As redes necessitam assim de conter serviços de segurança e protocolos de encaminhamento de dados adequados que combatam estas situações.

Além dos problemas de segurança é, hoje em dia, ainda muito difícil endereçar e

agrupar nós da rede logicamente que obedeçam a um certo estado, de forma a responderem a um conjunto de pedidos. Mais problemático ainda é o facto de existirem poucas propostas seguras e eficientes para efectuar pré-processamento de dados e agregações junto às fontes de dados, diminuindo assim a quantidade de dados totais retornados à estação central e, consequentemente, baixando a energia total gasta pelos nós encaminhadores.

Na génese deste trabalho esteve o desenho, implementação e validação de uma plataforma, na forma de pilha de serviços, de forma a resolver os problemas mencionados anteriormente. Mais concretamente, foram combinadas as melhores soluções existentes hoje em dia em termos de *comunicação segura*, *encaminhamento seguro*, *pré-processamento de dados* e *manipulação e endereçamento de nós por grupos lógicos* ou *vizinhanças lógicas*. Respectivamente, foram combinados os sistemas Minisec [LMPG07], Secure Directed Diffusion [HYWLZ06] e TeenyLime [CMMP06].

Através da pilha implementada, a maioria dos ataques existentes hoje em dia sobre as RSSF são evitados e o esforço no desenvolvimento de aplicações reduzido, pois todos os serviços principais estão contidos na plataforma e são acedidos através duma API.

Os testes à plataforma mostram baixos custos energéticos resultantes da adição dos serviços de segurança, uma alta fiabilidade na entrega de mensagens, conseguida através de replicação de rotas, e boa resiliência face a falhas dos nós encaminhadores. Além disso, os testes à cobertura da rede permitiram encontrar valores óptimos do número de sensores instalados num terreno, sem criar muitas partições na rede.

6.2 Trabalho futuro

Como trabalho futuro ficam em aberto vários testes que podem ser feitos, entre os quais testes de latência e fiabilidade das operações com diferentes taxas de congestionamento dos canais. Além disso, pode ser testado o impacto do envio replicado de mensagens com operações para os sensores, de forma a aumentar a taxa de recepção de instruções.

Pode ainda ser melhorado o algoritmo de encaminhamento de dados para permitir que sejam criadas múltiplas rotas de retorno de dados disjuntas, o que neste momento não está garantido. A possibilidade dos sensores recusarem encaminhar dados com uma certa probabilidade, dependendo da sua carga de bateria, também é um aspecto em aberto.

Na linha do sistema, tal e qual como foi testado, podem ser criadas novas aplicações que, por exemplo, façam uso duma sincronização de nós agregadores, ou dos nós temperatura, para efectuarem leituras à vez, poupando-se assim energia.

Podem ainda ser implementados e testados novos mecanismos MAC de gestão do canal, de forma a diminuir o número de colisões e aumentar a fiabilidade do envio de operações ou retorno de dados.

O simulador de energia pode ainda ser refinado para passar a medir operações relativamente pesadas, tais como captura dos dados dos sensores de um nó, operações de *Hash* ou iterações de listas.

Pode também ser criado um simulador de leituras das amostras dos sensores dos nós que, por exemplo, leia amostras de temperatura dum ficheiro resultado da simulação de um incêndio numa zona da rede. Com estes tipos de simulações podem ser efectuados testes antes da instalação real duma rede.

Finalmente o sistema, o qual foi implementado em *JDK1.6*, pode agora ser migrado para sensores reais que possuam, por exemplo, uma máquina virtual *Java JDK1.3*, de forma a realizar outro tipo de testes mais precisos. Note-se que ao longo do trabalho se teve o cuidado de se usar estruturas simples para facilitar essa tarefa.

Bibliografia

[AFN08] H. Alzaid, E. Foo, e J. G. Nieto. Secure data aggregation in wireless sensor network: a survey. In *AISC '08: Proceedings of the sixth Australasian conference on Information security*, pág. 93–105, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[Ama09] Pedro Amaral. Um sistema de encaminhamento seguro e resistente a intrusões para redes de sensores sem fios. Tese de Mestrado, Departamento de Informática FCT/UNL, 2009.

[ASSC02] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, e E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, Aug 2002.

[BJJ06] Misic V. B., Fung J., e Misic J. Mac layer attacks in 802.15.4 sensor networks. In Yang Xiao, editor, *Security in Sensor Networks*, chapter 12. CRC Press, 2006.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[CFP+06] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, e S. Masri. Monitoring civil structures with a wireless sensor network. *Internet Computing, IEEE*, 10(2):26–34, March-April 2006.

[CMMP06] P. Costa, L. Mottola, A. L. Murphy, e G. P. Picco. Teenylime: transiently shared tuple space middleware for wireless sensor networks. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*, pág. 43–48, New York, NY, USA, 2006. ACM.

[CW09] Jared Cordasco e Susanne Wetzel. An attacker model for manet routing security. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pág. 87–94, New York, NY, USA, 2009. ACM.

[DDHV03] W. Du, J. Deng, Y.S. Han, e P.K. Varshney. A witness-based approach for data fusion assurance in wireless sensor networks. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 3, pág. 1435–1439, Dec. 2003.

[DY83] D. Dolev e A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, Mar 1983.

[FRL05] C. L. Fok, G. C. Roman, e C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pág. 653–662, Washington, DC, USA, 2005. IEEE Computer Society.

[Gel85] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[HPJ05] Y.C. Hu, A. Perrig, e D. B. Johnson. Ariadne: a secure on-demand routing protocol for ad hoc networks. *Wirel. Netw.*, 11(1-2):21–38, 2005.

[HYWLZ06] H. Hao Yang, S.H.Y. Wong, S. Lu, e L. Zhang. Secure diffusion for wireless sensor networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pág. 1–10, Oct. 2006.

[IGE00] C. Intanagonwiwat, R. Govindan, e D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pág. 56–67, New York, NY, USA, 2000. ACM.

[Jan02] M. Jansson. Context shadow: An infrastructure for context aware computing. Third workshop on Artificial Intelligence in Mobile Systems (AIMS), 2002.

[jPr] jprowler. `http://w3.isis.vanderbilt.edu/projects/nest/jprowler/index.html`.

[KSW04] C. Karlof, N. Sastry, e D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pág. 162–175, New York, NY, USA, 2004. ACM.

[KW03] C. Karlof e D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pág. 113–127, May 2003.

[LMPG07] M. Luk, G. Mezzour, A. Perrig, e V. Gligor. Minisec: A secure sensor network communication architecture. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pág. 479–488, April 2007.

[LSP82] Leslie Lamport, Robert Shostak, e Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[MCP+02] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, e J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pág. 88–97, New York, NY, USA, 2002. ACM.

[MFHH03] S. Madden, M. Franklin, J. Hellerstein, e W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pág. 491–502, New York, NY, USA, 2003. ACM.

[Mic] Micaz. http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148:micaz.

[MP06] L. Mottola e G. P. Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *DCOSS*, pág. 150–168, 2006.

[MPR06] A. L. Murphy, G. P. Picco, e G. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[PR99] C.E. Perkins e E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pág. 90–100, Feb 1999.

[PSP03] B. Przydatek, D. Song, e A. Perrig. Sia: secure information aggregation in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pág. 255–265, New York, NY, USA, 2003. ACM.

[PST+02] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, e D. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, 2002.

[SCV+06] P. Sikka, P. Corke, P. Valencia, C. Crossman, D. Swain, e G. Bishop-Hurley. Wireless ad hoc sensor and actuator networks on the farm. In *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, pág. 492–499, 0-0 2006.

[Ski] Skipjack. http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.

[SMZ07] K. Sohraby, D. Minoli, e T. F. Znati. *Wireless sensor networks*. John Wiley and Sons, Inc., 2007.

[Vig] Vigilnet. http://www.cs.virginia.edu/wsn/vigilnet.

[WM04] M. Welsh e G. Mainland. Programming sensor networks using abstract regions. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pág. 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[WSBC04] K. Whitehouse, C. Sharp, E. Brewer, e D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pág. 99–110, New York, NY, USA, 2004. ACM.

[YY06] Wang Y. e Tseng Y. Attacks and defenses of routing mechanisms in adhocand sensor networks. In Yang Xiao, editor, *Security in Sensor Networks*, chapter 12. CRC Press, 2006.

A

Anexos

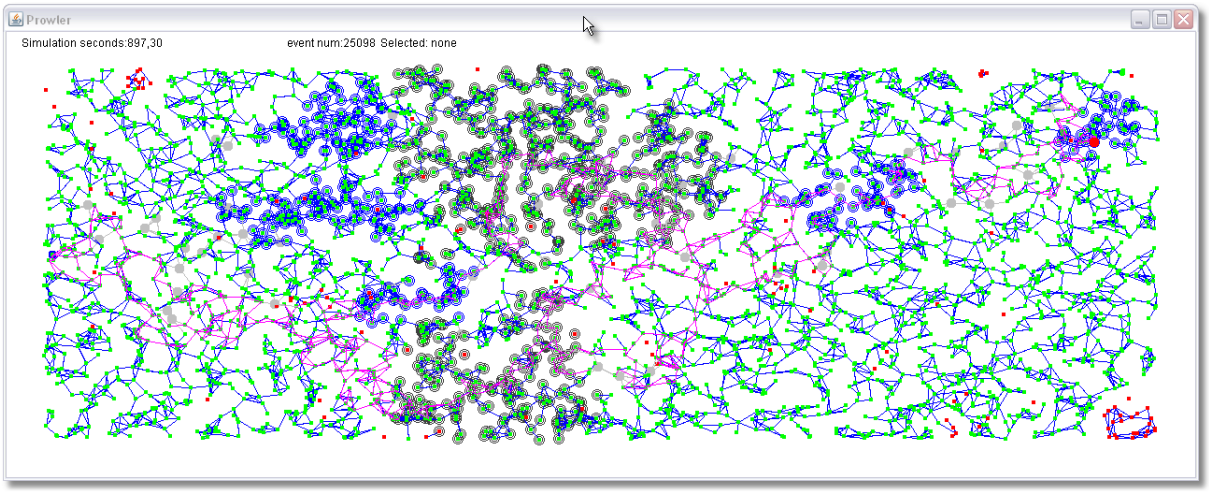


Figura A.1: Janela da área de instalação dos nós sensores. Podem-se ver, por exemplo, os vários tipos de mensagens trocadas (círculos coloridos), ligações seguras entre os nós (linhas a azul), caminho de retorno de dados (linhas a cor-de-rosa) e nós isolados da rede (pontos a vermelho).

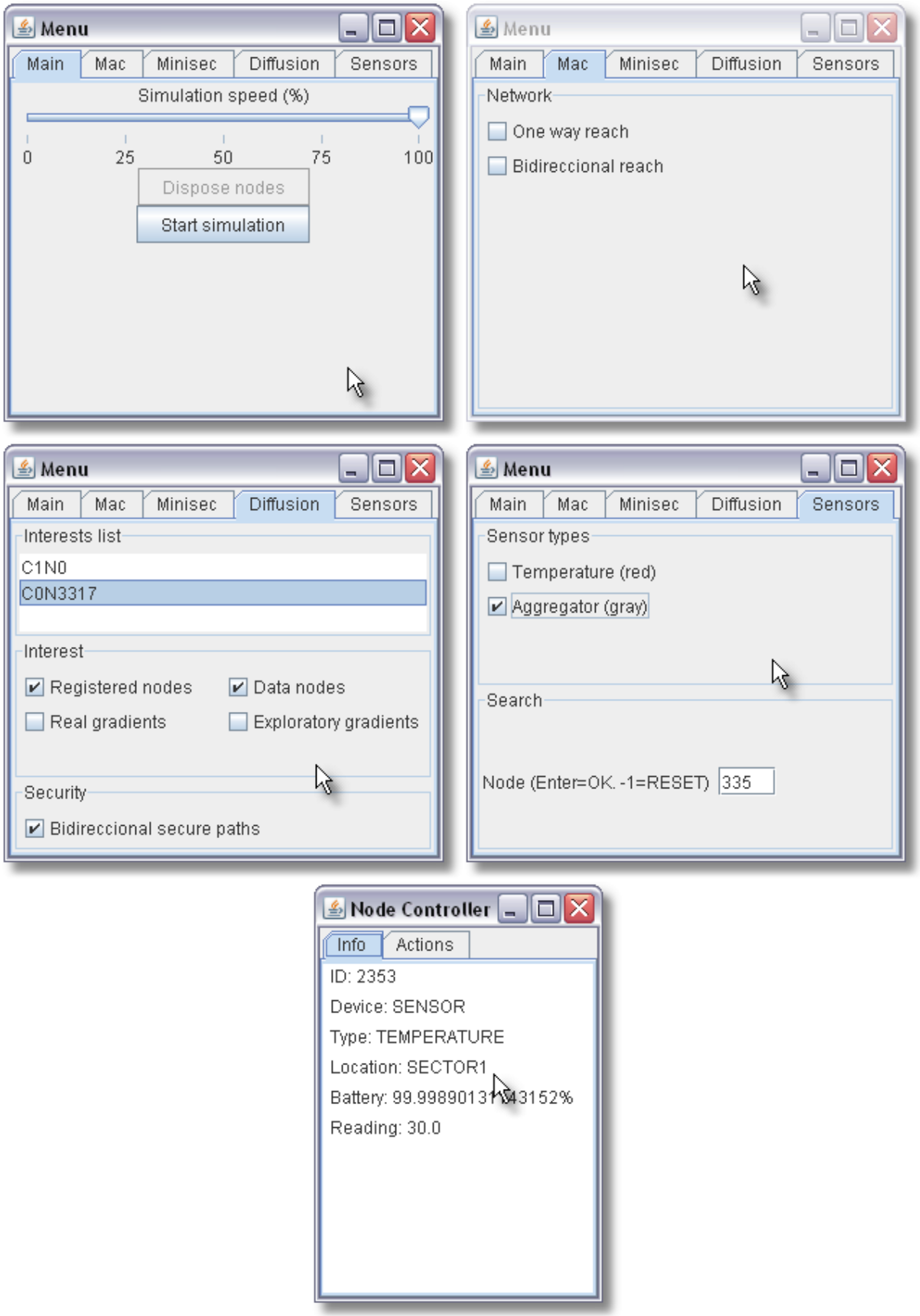


Figura A.2: Janelas de menus da interface gráfica.

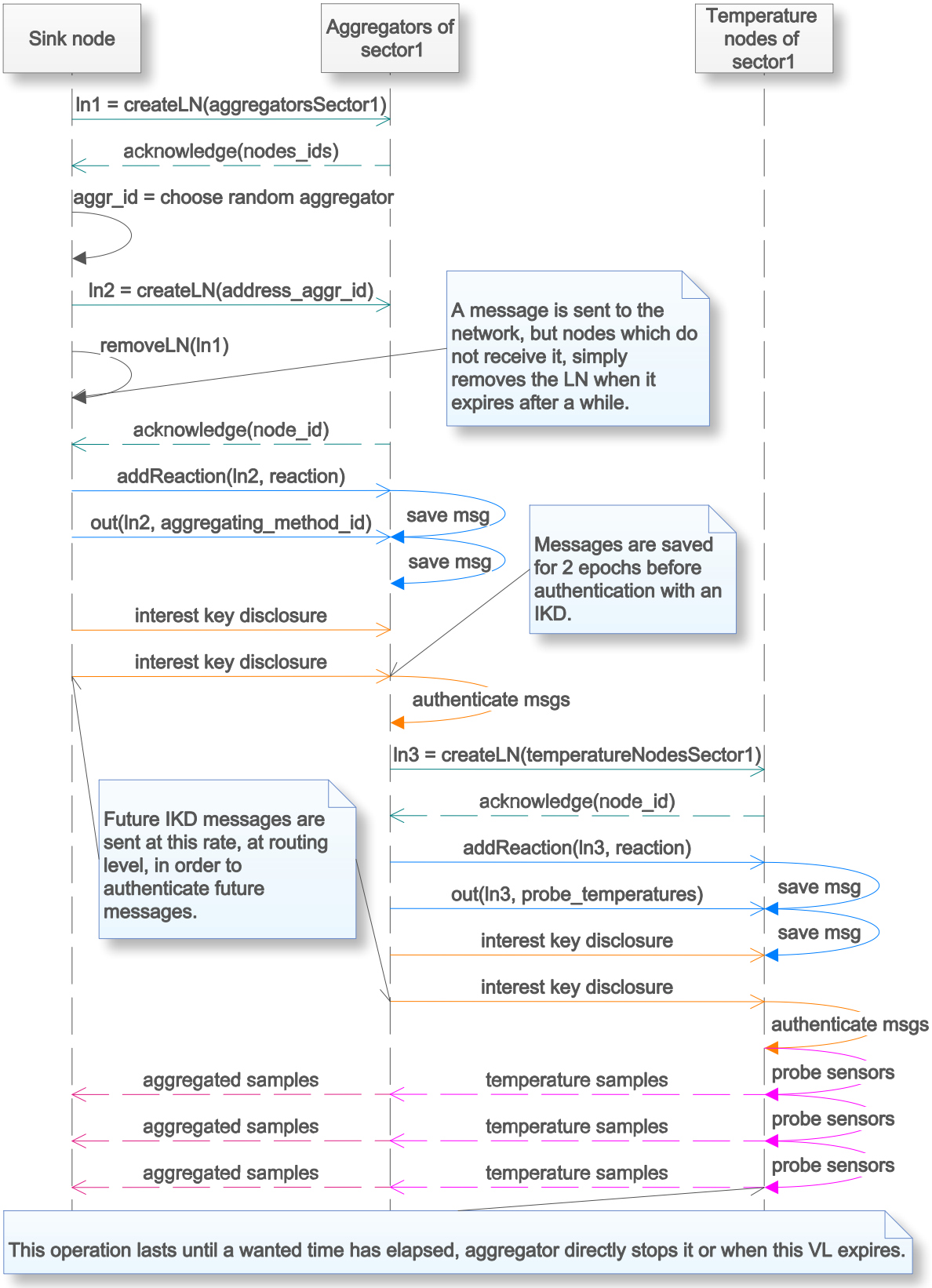


Figura A.3: Sequência de acções de uma agregação periódica de temperaturas.