João António Batista Tavares

Lic.

Encaminhamento de dados tolerante a intrusões para redes de sensores sem fios

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Henrique João Lopes Domingos, Prof. Dr., DI-FCT-UNL

Júri:

Presidente: Prof. Dr. António Ravara

Arguentes: Prof. Dr. Hugo Miranda

Vogal: Prof. Dr. Henrique Domingos

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2012

Resumo

As redes de sensores sem fios (RSSFs) constituem uma área de desenvolvimento de aplicações inovadoras, despertando um enorme interesse na comunidade de investigação. Estas redes são compostas por dispositivos sensores com capacidades limitadas de processamento, armazenamento e autonomia energética, comunicando entre si por ligações sem fios suportadas por comunicações por rádio frequência. As limitações desses dispositivos impõem diversos desafios ao desenvolvimento de *software* para estas redes.

Por outro lado, a estruturação de serviços de suporte à operação destas redes, bem como o suporte desses serviços em pilhas de protocolos adaptados às suas caraterísticas, continua na agenda de investigação. A concepção de serviços de segurança para a operação segura destas redes é um aspecto particularmente relevante, nomeadamente quando estas são utilizadas como redes auto-organizadas, operando em cenários de grande escala e sem supervisão humana. Neste tipo de ambiente de utilização, as RSSFs podem ser sujeitas a ataques às comunicações, bem como ataques por intrusão ao nível dos nós sensores.

O objectivo desta dissertação é desenvolver, implementar e testar um serviço de encaminhamento seguro e tolerante a intrusões para RSSFs de grande escala, funcionando de forma auto-organizada num ambiente de comunicação *multi-hop*. O sistema de encaminhamento adopta uma estratégia suportada na descoberta, seleção e organização de múltiplas rotas disjuntas e múltiplas estações base (ou nós agregadores), sendo as rotas formadas de forma proactiva durante o processo de auto-organização da rede, visando assegurar condições de resiliência para tolerância a intrusões.

Para o efeito, a solução baseia-se numa estrutura de rede sobreposta. Numa primeira camada, os nós sensores recolhem e transmitem dados pelas múltiplas rotas de encaminhamento criadas para as diferentes estações base. Numa segunda camada, as estações base, que possuem maiores capacidades de computação, comunicação e energia, funcionam como nós seguros de agregação e consenso de dados. Estes nós processam os dados encaminhados na primeira camada, tendo por base um protocolo de acordo bizantino

probabilístico sobre os dados recebidos, de forma tolerante a intrusões. Os dados resultantes do acordo podem então ser disponibilizados para serem finalmente processados por sistemas ou aplicações externos à rede.

Palavras-chave: redes de sensores sem fios, protocolos e serviços de encaminhamento, disseminação de dados, fiabilidade, segurança, tolerância a intrusões, consenso distribuído não-determinístico

Abstract

Wireless Sensor Networks (WSNs) provide a baseline to develop new and innovative applications, having become an area of great interest by the investigation community. These networks are composed by sensor nodes with limited computational, storage, wireless communication and energy resources. Sensor nodes communicate through radio frequency, usually using the standard IEEE 802.15.4. These limitations on sensor nodes impose various challenges to software development for WSNs.

Despite the interest in WSNs, the development of services to support these networks' operation, as well as the inclusion of those services in WSN adapted protocol stacks, is still in the research agenda. The development of security services for the secure operation of WSNs is particularly relevant, once these networks are used in unsupervised and large scale scenarios, needing to auto-organize the nodes. Considering this environment, WSNs are vulnerable to communication attacks and intrusion attacks on the sensor nodes.

The main purpose of this thesis is to develop, implement, test and evaluate a secure and intrusion tolerant routing service for large scale WSNs, following a disjoint routs multi-hop routing strategy to multiple base stations (or sink nodes), to assure resilience and intrusion tolerance properties. Routes are built during the network's self-organization process.

The solution is based on an overlay network vision. On the first layer, sensor nodes capture data and send it to the multiple base stations, through multiple routes. On the second layer, the base stations, nodes with greater computational, communication and energy capabilities than sensor nodes, are secure nodes that collect the data received from the first layer. Base stations process the received data using an intrusion tolerant probabilistic byzantine agreement protocol to assess the correctness of the received data. After being approved, the received data is delivered to the application level.

Keywords: wireless sensor networks, routing services and protocols, data dissemination, dependability, intrusion tolerance, non-deterministic distributed consensus

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

1.1 Motivação e Enquadramento

As redes de sensores sem fios (RSSF) tornaram-se numa área de grande interesse por parte da comunidade de investigação. Atendendo às suas características [39, 65, 73], estas redes são particularmente adequadas ao suporte de aplicações interessantes e inovadoras, entre as quais estão: monitorização de habitats naturais [49] e/ou de condições ambientais [9]; monitorização de indicadores biomédicos [53]; vigilância de instalações industriais críticas [29]; monitorização da operação de infra-estruturas de engenharia civil [36]; controlo e vigilância na monitorização e localização de pessoas e/ou mercadorias [25]; controlo de fenómenos naturais na área da vulcanologia e/ou sismologia [75]; aplicações militares de monitorização de campos de batalha e/ou de detecção de intrusões em zonas sob controlo militar [30].

No âmbito de algumas das aplicações apresentadas, as RSSFs operam sem supervisão humana, em condições de funcionamento como sistemas autónomos e como redes auto-organizadas. Nestes cenários de utilização, impõem-se requisitos de segurança aos diferentes níveis da estruturação da arquitectura de serviços de software para as RSSF. Estes requisitos estão associados à necessidade de suportar falhas ou ataques que ocorrem quer ao nível das comunicações sem fios, quer ao nível de falhas ou ataques por intrusão aos nós sensores.

Face aos requisitos de fiabilidade e segurança das comunicações, as soluções para as RSSF devem considerar no seu modelo de falhas e de segurança que as falhas e vulnerabilidades podem resultar de tipologias de ataques semelhantes às conceptualizadas pela Framework X.800 [71] ou definidas por um modelo de adversário do tipo Dolev-Yao [22].

Face aos requisitos impostos pela necessidade de defesa contra ataques por intrusão,

os anteriores modelos tornam-se limitados. Assim, os modelos de fiabilidade e segurança, devem estender-se a tipologias de ataques por intrusão envolvendo: a possível captura física de nós; o acesso a memória e armazenamento de dados nos sensores da rede; a modificação ou inserção de código malicioso nos nós capturados; ou a introdução de nós na rede, controlados pelo atacante. A tolerância face a ataques por intrusão torna-se particularmente mais complexa se admitirmos que o atacante é capaz de capturar e invadir um nó, e replicar rapidamente eventuais comportamentos maliciosos a outros nós da rede.

Os serviços de segurança para RSSFs requerem técnicas adequadas às características dos dispositivos que são utilizados como nós destas redes, atendendo às limitações desses dispositivos em relação a recursos computacionais e de comunicação e ao consumo energético. A necessidade destes serviços adaptados tem motivado a investigação de diferentes tipos de mecanismos, técnicas e serviços para níveis distintos da organização da pilha de serviços: serviços ao nível MAC ou de suporte básico de comunicação segura sem fios [54], onde os ataques têm como principais objectivos impedir que a RSSF funcione (*Denial of Service*) ou esgotar a bateria dos nós sensores; serviços ao nível rede que introduzem noções de segurança no processo de auto-organização topológica, no endereçamento e encaminhamento dos dados; e serviços ao nível das aplicações onde se introduzem noções de segurança específicas da aplicação.

O âmbito da presente dissertação encontra-se na segurança ao nível rede, onde se inserem os protocolos de encaminhamento de dados.

1.2 Problemática do Encaminhamento de Dados Tolerante a Intrusões em RSSFs

Grande parte da investigação em protocolos e sistemas de encaminhamento seguro tolerantes a intrusões para RSSFs, desenvolveu protocolos interessantes que incorporam contra-medidas face a determinadas tipologias de ataques ao encaminhamento de dados, Porém, muitas das soluções existentes são limitadas, nomeadamente na incorporação de mecanismos de tolerância a intrusões que abranjam, numa mesma solução, as dimensões de prevenção, detecção e recuperação dinâmica da rede, quando esta opera em cenários de grande escala, com operação não supervisionada e suporte de comunicação *multi-hop*. Por um lado torna-se particularmente interessante analisar as limitações de algumas das propostas de investigação, face à definição de modelos de adversário associados a este tipo de redes. Por outro lado, é preciso ter em conta que só se tornam realistas soluções que sejam adequadas ás características e capacidades de computação, processamento e limitações energéticas das RSSF.

No âmbito da presente dissertação interessam particularmente os protocolos que apostam na prevenção probabilística de intrusões, nomeadamente os protocolos definidos em

[20, 28, 46], e os protocolos de encaminhamento tolerantes a intrusões que incluem detecção e remoção de intrusos, nomeadamente os protocolos descritos em [42, 58].

Uma possibilidade de ultrapassar as anteriores limitações, consiste em estruturar as RSSFs para grande escala, conjugando o processamento interno da rede (encaminhamento primário de dados) com o processamento específico que pode ser realizado ao nível de estações base ou de nós especiais com maiores recursos computacionais. Nesta visão, as RSSFs são ambientes estruturados de redes sobrepostas, em que a primeira camada é responsável pelo processamento primário da rede e está associada aos serviços de auto-organização e encaminhamento *multi-hop* por rotas múltiplas e a segunda camada procede a computações de segurança e agregação de dados. Através de mecanismos e protocolos de consenso distribuído de dados recebidos pelas múltiplas rotas, as estações base fazem acordos de modo aos dados serem agregados com alta fiabilidade e posteriormente enviados em segurança para os sistemas de processamento e aplicações externas à rede. Na segunda camada é possível incluir também serviços de detecção e recuperação de intrusões existentes na primeira camada.

1.3 Problemática do Consenso Distribuído

De acordo com a anterior visão, vários nós de uma RSSF necessitarão de proceder a computações de consenso de valores com tolerância a falhas bizantinas, correspondentes aos dados encaminhados por múltiplas rotas.

Existem várias soluções apresentadas para o problema do consenso distribuído. Contudo, para o efeito pretendido, será necessário recorrer a mecanismos de consenso distribuído não-determinístico (devido aos problemas de comunicação nas RSSFs) e tolerante a intrusões. As soluções de consenso distribuído não-determinístico revelam-se mais apropriadas às características das RSSF, por admitirem que o sistema pode ser assíncrono. Existem diferentes propostas de solução para o problema do consenso distribuído não-determinístico, mas no âmbito desta dissertação serão mais focados os protocolos de consenso binário aleatórios [7, 11, 56, 55]. Estes protocolos serão discutidos na secção 3.3.

1.4 Objectivos e Contribuições

A presente dissertação visa propor, desenvolver, implementar e testar um serviço de encaminhamento seguro e tolerante a intrusões para RSSF de grande escala e ambiente de comunicação *multi-hop*. A solução proposta visa o encaminhamento resiliente de dados originados nos sensores por múltiplas rotas disjuntas formadas pela rede de sensores, tendo como destino diferentes estações base. As rotas são formadas de forma dinâmica e proactiva durante o processo de descoberta e auto-organização da rede, tendo como critério que um mesmo nó de encaminhamento não seja nó interior em mais do que uma rota estabelecida.

Os dados encaminhados para as estações base são aí submetidos a uma verificação

baseada num protocolo de consenso distribuído e tolerante a intrusões, sendo este protocolo executado ao nível das estações base utilizadas.

A solução preconizada pode ser vista como uma solução de rede sobreposta, com base em duas camadas do sistema de encaminhamento. A primeira camada é formada pela rede de sensores, sendo o encaminhamento suportado pelas múltiplas rotas estabelecidas até às estações base, sendo este encaminhamento suportado com base na pilha IEEE 802.15.4. Na segunda camada, as estações base agregam os dados recebidos e executam um protocolo de consenso distribuído tolerante a intrusões, sendo este suportado numa rede estabelecida pelas estações base. Nesta segunda camada, a comunicação entre as estações base pode ser assegurada por comunicações na pilha IEEE 802.15.4, rede local sem fios (ex., IEEE 802.11) ou mesmo, com base numa interligação via rede local com fios (ex., IEEE 802.3).

As principais contribuições são as seguintes:

- Desenvolvimento, implementação e teste, com base num ambiente de simulação para RSSFs de grande escala, de um protocolo de encaminhamento com prevenção de intrusões, usando múltiplas rotas disjuntas auto-organizadas (constituindo a primeira camada da visão de rede sobreposta).

- Desenvolvimento, implementação e integração, no protocolo de encaminhamento proposto, do nível de processamento de agregação e consenso dos dados por parte das estações base (constituindo o serviço da segunda camada da visão de rede sobreposta).

- Avaliação e validação da solução global proposta, com base num ambiente de simulação para redes de sensores sem fios de grande escala (permitindo simular redes da ordem de centenas de até dezenas de milhar de nós), com análise de métricas obtidas nas simulações dos seguintes indicadores:

  - latência do encaminhamento;
  - condições de conectividade e cobertura da rede;
  - indicadores de fiabilidade
  - métricas de consumo de energia;
  - indicadores relativos à correcção e execução do protocolo de consenso.

A solução apresentada na dissertação constitui uma proposta inovadora para um sistema de encaminhamento tolerante a intrusões para RSSF que integra mecanismos de prevenção de intrusões com encaminhamento resiliente multi-caminho e tolerância a intrusões com base em técnicas de consenso distribuído de dados. A solução global assegura um serviço de encaminhamento que defende a rede de ataques ao encaminhamento, incluindo ataques por intrusões que induzam processamento incorreto ou malicioso, quer ao nível dos nós de encaminhamento na RSSF, quer ao nível das estações base.

Um grande desafio que se colocava desde o início da fase de preparação da dissertação foi o facto de nenhum protocolo de consenso ter sido desenvolvido a pensar nas características das RSSFs. Os protocolos que mais se aproximam dos requisitos destas redes são os protocolos de consenso binário aleatório, que não são suficientes para realizar consensos de dados de maiores dimensões. Sabia-se que seria um desafio interessante, com resultados imprevisíveis por se tratar de uma solução inovadora.

Durante o processo de desenvolvimento dos protocolos de consenso, surgiu a ideia de incluir serviços de detecção e remoção de nós intrusos baseados na realização dos consensos. No entanto, por não se incluírem nos objectivos da tese, os serviços de detecção e remoção de intrusos são visto numa perspectiva de trabalho futuro.

1.5 Estrutura do Documento

As secções que se seguem neste documento estão organizados do seguinte modo:

- A secção 2 apresenta uma introdução às RSSFs, referindo as suas principais características e abordando os aspectos mais relevantes para a problemática tratada nesta dissertação.

- A secção 3 é dedicada à apresentação do estado da arte, cobrindo os aspectos do trabalho relacionado que estão directamente associados aos objectivos da dissertação, sendo esses aspectos: Os mecanismos de segurança e tolerância a intrusões utilizados nas RSSFs; os diferentes tipos de protocolos de encaminhamento seguro que existem para as RSSFs, assim como exemplos específicos de alguns desses protocolos; a problemática do consenso em RSSFs: o porquê de os protocolos de consenso determinístico não serem adequados às RSSFs e a apresentação de algumas soluções para o problema do consenso bizantino não-determinístico; apresentação de diferentes ambientes de simulação para RSSFs, cujas características podem servir para testar e avaliar a solução desenvolvida e outras existentes. Considerou-se relevante dedicar uma parte desta secção a mecanismos de detecção e remoção de intrusões, para apoiar a perspectiva de trabalho futuro.

- A secção 4 descreve o modelo do sistema ideal para o funcionamento da solução. A visão do modelo de sistema inclui: modelo de rede, modelos de adversário e de falhas e modelo de software. Dedica-se também uma sub-secção para fazer uma pequena descrição da arquitectura do simulador onde foi implementada e testada a solução.

- A secção 5 descreve o funcionamento da solução desenvolvida, o protocolo de encaminhamento MINSENS++. É descrito o funcionamento do protocolo nas suas diferentes fases. Esta secção é mais dedicada à primeira camada da visão de rede sobreposta, onde os dados são encaminhados.

- A secção 6 é dedicada à descrição dos protocolos de consenso de dados desenvolvidos. Esta secção inclui descrições de: (1) primitivas de comunicação utilizadas nos consensos, tendo uma delas, a difusão eco probabilística, sido desenvolvida no contexto desta dissertação; (2) protocolo de consenso binário Turquois, ligeiramente adaptado para melhor se adaptar aos requisitos das RSSFs; (3) protocolo de consenso multi-valor desenvolvido, uma versão fortemente adaptada de uma solução já existente; (4) módulos de consenso de dados e de rotas; (5) módulo de recuperação de atrasos, um componente importante na adaptação dos protocolos de consenso às RSSFs.

- A secção 7 descreve pormenores de implementação dos protocolos (encaminhamento e consenso). Também é referido um módulo de extensão ao simulador, que foi desenvolvido para avaliar os protocolos de consenso.

- A secção 8 contém os resultados da análise experimental realizada aos protocolos desenvolvidos.

- Na secção 9 apresentam-se as conclusões a que se chegou durante a elaboração da dissertação, referindo-se também algumas perspectivas de trabalho futuro.

2

Introdução às RSSFs

2.1 Redes de Sensores Sem Fios (RSSFs)

As redes de sensores sem fios (RSSF) são redes de comunicação por radiofrequência que utilizam a norma IEEE 802.15.4, ZigBee [5]. Estas redes são formadas por dispositivos (sensores) de baixo custo e de pequenas dimensões (da ordem de grandeza das dezenas de milímetros cúbicos às dezenas de centímetros cúbicos). Os sensores são dotados de capacidades computacionais, energéticas e de comunicação muito limitadas [35], podendo ser distribuídos ao longo de uma determinada área geográfica formando redes de comunicação mais ou menos densas, aproveitando o baixo custo dos dispositivos e a facilidade de instalação dos mesmos, sem custos operacionais relevantes. Para o efeito são particularmente importantes as características de auto-organização da rede e não necessidade de supervisão da operação [74].

Numa RSSF os sensores cooperam entre si para monitorizar eventos que resultam da interacção entre os dispositivos e o meio ambiente, medindo valores associados a diferentes tipos de fenómenos físicos. Na tecnologia actual, os sensores para RSSFs são pequenos dispositivos computorizados, baseados em substratos mais ou menos genéricos, sobre os quais poderão existir um ou mais sensores (propriamente ditos, que obtêm dados do ambiente). Aquele tipo de substrato envolve um microprocessador, um circuito de comunicação por rádio (IEEE 802.15.4) e um ou mais sensores com capacidades para processamento de sinal associado a determinado tipo de eventos externos: temperatura, som ou ruído, humidade, movimento, poluição ou níveis de concentração de certo tipo de substâncias, pressão, vibração ou luminosidade. Existem ainda sensores especializados para medir indicadores fisiológicos ou sinais vitais de saúde, por exemplo. Na figura 2.1 apresenta-se a arquitectura genérica dos nós sensores.

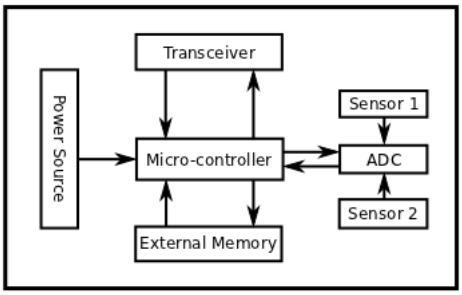


Figura 2.1: Arquitectura genérica de um nó sensor

Nas RSSFs os eventos são detectados e medidos em cada nó sensor, podendo também a informação obtida ser pré-processada localmente, sendo posteriormente difundidos pela rede através de outros sensores com base em topologias de organização *multi-hop* (podendo em alguns casos ser *single-hop* [67]). Os dados transmitidos podem ainda ser processados e agregados por outros sensores ao longo da rede, durante a transmissão e encaminhamento dos mesmos. Finalmente, a informação gerada pela rede é geralmente recolhida em nós de captura de dados (habitualmente designados por estações base ou nós *sink*), que poderão estar interligados a sistemas que executam aplicações para tratamento e análise de dados. Esta ligação poderá fazer-se com base em qualquer tipo de comunicação (ex.: redes locais com ou sem fios, através da Internet, etc.).

2.2 Pilha de serviços para suporte de aplicações em RSSFs

As arquitecturas de software para aplicações em RSSFs são estruturadas tendo em conta requisitos específicos de aplicações, sendo que em alguns casos os serviços correspondentes a diferentes camadas da pilha podem fundir-se, normalmente por motivos de eficiência. Pode-se, no entanto, ter um modelo genérico de referência, como o apresentado na figura 2.2. A camada de segurança tem um carácter mais transversal, podendo fornecer serviços a qualquer das outras camadas.

O foco desta dissertação é a camada de rede, assim como a camada de segurança no que diz respeito aos serviços que pode oferecer à camada de rede. Também serão vistas as camadas física e MAC, com o intuito de compreender os serviços que estas oferecem para suportar a camada de rede.

2.3 Camadas Física e MAC: A Norma IEEE 802.15.4

A norma IEEE 802.15.4 especifica as duas camadas de nível mais baixo da pilha de serviços: a camada física e a camada de controlo de acesso ao meio (MAC), que foram idealizadas tendo em consideração as baixas taxas de dados enviados/recebidos que caracterizam este tipo de redes, bem como as suas limitações a nível energético. O espaço de operação das redes 802.15.4 está bem delimitado – o raio de acção de um nó dessa rede

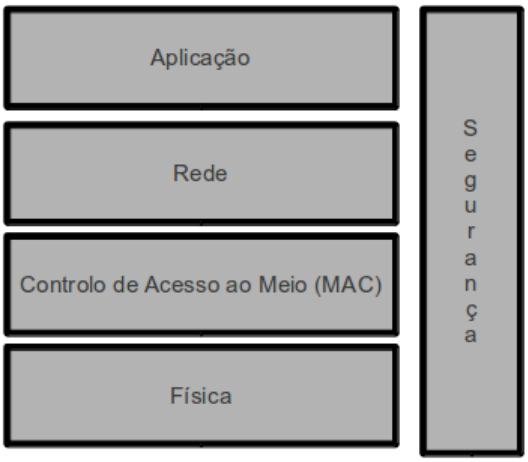


Figura 2.2: Exemplo de pilha de serviços para suporte a aplicações em RSSFs

é da ordem das dezenas de metros – pelo que estas redes são também conhecidas por WPAN (*Wireless Personal Area Network*).

Um dispositivo pertencente a uma rede 802.15.4 poderá ter um endereço IEEE de 64 bits ou um endereço mais curto de 16 bits.

A norma IEEE 802.15.4 é essencialmente um standard que tenta satisfazer um dos principais requisitos deste tipo de redes (as suas limitações energéticas), tentando assim obter custos de operação mínimos juntamente com uma simplicidade a nível tecnológico, sem que para isso seja necessário sacrificar a flexibilidade ou a usabilidade do sistema.

A camada física fornece uma interface entre a camada MAC e o canal físico de rádio. Esta camada gere o emissor/receptor dos sinais rádio e efectua a selecção de canais, para além de fornecer funções relacionadas com a energia e o sinal do dispositivo.

2.4 Camada de Rede

De forma a suportar a comunicação multi-hop, vários protocolos de encaminhamento foram propostos. Eles podem ser classificados segundo duas categorias: *table-driven* (orientados para a utilização de tabelas – protocolos pro-activos) ou *on-demand* (só actuam quando necessário – protocolos reactivos).

Nos protocolos *table-driven*, os nós trocam informação entre eles de forma constante de modo a manter as tabelas de encaminhamento sempre actualizadas, independentemente de existirem ou não pedidos de estabelecimento de comunicação. O DSDV (*Destination-Sequenced Distance-Vector*) [61] e o OLSR (*Optimized Link State Routing*) [17] são alguns exemplos de protocolos de encaminhamento *table-driven*. Este tipo de protocolos de encaminhamento envolve normalmente uma fase de descoberta de nós associada a processos de auto-organização da rede, uma fase de descoberta de rotas e selecção das rotas óptimas

ou adequadas para envio de eventos desde os nós sensores até às estações base, e uma fase de gestão e manutenção das tabelas de encaminhamento associadas às rotas utilizadas. A sobrecarga da gestão de tabelas de encaminhamento nestes protocolos compensa o encaminhamento de dados, sobretudo em aplicações em que valores de sensoriamento têm que ser enviados de forma regular ou permanente. Tal acontece por exemplo em aplicações de monitorização permanente de eventos ou aplicações de medida regular de métricas de fenómenos físicos medidos pelos sensores.

Para suporte de muitas aplicações baseadas na noção de disseminação não regular de eventos, os protocolos *on-demand* podem tornar-se mais vantajosos, pois os nós só trocam informação quando existem comunicações por estabelecer. Este facto faz com que não seja necessário manter tabelas de encaminhamento, podendo estes protocolos desencadear processos de selecção de rotas quando se torne necessário enviar dados pela rede, evitando a sobrecarga da gestão e manutenção de tabelas nos nós. O AODV (*Ad-hoc On-demand Distance Vector*) [62] é um exemplo de um protocolo *on-demand*. Existem também variantes que seguem uma política de *source-routing*, como é o caso do DSR (*Dynamic Source Routing*) [33].

Os protocolos *on-demand* são muito utilizados nas redes ad-hoc no geral, mas principalmente nas redes móveis devido, precisamente, à mobilidade dos nós, que dificulta a manutenção do estado das ligações. No caso das RSSFs, em que os sensores são estáticos e enviam regularmente os valores "sentidos", não se verifica a tendência de implementar protocolos de encaminhamento *on-demand*, uma vez que os processos de reorganização da rede, descoberta de rotas ou selecção e avaliação de novas rotas pode tornar-se desvantajoso face a gastos de energia, bem como podem induzir maior latência média na disseminação de eventos através da rede.

Independentemente de ser reactivo ou pro-activo, um protocolo de encaminhamento pode descobrir as suas rotas de modo centralizado ou distribuído. Na descoberta centralizada os nós recorrem a estações base ou a outros nós especiais da rede para que estes procedam à descoberta das rotas, com base na informação da vizinhança de cada nó. No caso da descoberta distribuída cada nó é responsável por criar e manter a sua tabela de encaminhamento.

Algo que é comum a todos os protocolos de encaminhamento é poder-se considerar que são constituídos por três fases distintas: descoberta de rotas, estabelecimento de rotas e encaminhamento de dados. A fase de descoberta de rotas dá-se quando os nós enviam mensagens HELLO uns aos outros, para se darem a conhecer à vizinhança. O estabelecimento de rotas corresponde à fase em que os nós da rede decidem quais as rotas a utilizar. A fase de encaminhamento de dados dá-se quando os nós têm conhecimento das rotas e já estão a transmitir dados.

Os protocolos apresentados acima não têm qualquer tipo de segurança, sendo a sua única preocupação resolver o problema do encaminhamento. Na secção 3.2 abordam-se alguns protocolos seguros.

2.5 Organização e Operação das RSSFs

Uma RSSF forma um sistema distribuído capaz de capturar dados através da actividade autónoma e assíncrona dos sensores, com possível pré-processamento desses dados no nó que os registou. Os dados são depois transmitidos e processados de forma interna (ao nível dos nós intermédios) e enviados para os nós de agregação de dados (estações base ou nós *sink*).

Por norma, uma RSSF é formada por um grande número de nós sensores comuns e por um ou mais nós de agregação de dados. O modelo de comunicação consiste no encaminhamento *multi-hop* dos dados originados nos nós sensores, ao longo dos vários nós da rede até aos nós de agregação de dados.

Para os nós sensores das RSSFs, o custo energético da obtenção e do processamento dos dados pode ser de várias ordens de grandeza inferior ao custo energético da comunicação, tornando-se assim imperativo que seja feita alguma filtragem de dados ao longo da rede (de preferência a começar no nó origem) de modo a reduzir o consumo energético dos nós, o que leva a um maior tempo de vida da rede.

Um dos principais objectivos das RSSFs é permitir que estas sejam utilizadas em ambientes de grande escala, na ordem dos milhares de nós, sendo assim possível cobrir áreas geográficas enormes desde que se verifiquem determinadas propriedades de densidade e distribuição espacial, isto é, o número de nós tem de ser suficiente de forma a cobrir toda a área tendo em conta as limitações existentes ao nível das comunicações. Desta forma torna-se particularmente importante que as RSSFs se possam formar e gerir de forma autónoma, com base em determinados critérios de descoberta de nós e auto-organização da rede.

Os protocolos de encaminhamento têm diferentes modos de encaminhamento e organização dos nós e das rotas na rede. Existem 3 grandes grupos de protocolos que diferem nesta característica, que são os protocolos de encaminhamento geográfico, encaminhamento baseado em grupos (*cluster-based*) e encaminhamento plano.

Nos protocolos de encaminhamento geográfico assume-se que os nós conhecem a sua própria localização e a localização dos destinos para os quais vão enviar dados. Os nós não necessitam de guardar qualquer informação relativa a rotas, sendo que o local para onde os pacotes são enviados é determinado no momento da recepção dos mesmos e depende unicamente da informação geográfica do destino. Os exemplos mais significativos de protocolos de encaminhamento pertencentes a este grupo são os protocolos da família SIGF [76].

Os protocolos de encaminhamento baseado em grupos, também chamados de protocolos de encaminhamento hierárquico ou de encaminhamento de eficiência energética (*energy-efficient*), agregam os nós em grupos, sendo que cada grupo tem um representante, denominado de *cluster-head*. Os nós de cada grupo enviam informação para os representantes que por sua vez tratam de encaminhar os dados para o destino, que normalmente é uma estação base, podendo também ser outro nó da rede (nó "normal" ou

cluster-head). Podem existir um ou mais níveis na hierárquica de grupos, sendo que um representante de grupo pode ele próprio pertencer a um outro grupo que tem outro representante de nível superior. Alguns exemplos destes protocolos são: LEACH [31] e PEGASIS [44].

Os protocolos de encaminhamento plano não obrigam a qualquer tipo de estrutura para a rede, dando total liberdade a quem desenvolve um protocolo de definir o modo como os nós são organizados e as rotas são geradas. Dada esta liberdade, é natural que alguns protocolos optem por utilizar uma estrutura para organizar a rede e outros protocolos utilizem a rede sem uma estrutura específica. Tal como o nome indica, uma organização não estruturada não define regras no que diz respeito às rotas que são encontradas e à organização dos próprios nós da rede. Já no caso de uma organização estruturada, é necessário seguir regras de construção para a rede. Por exemplo, numa organização em árvore coloca-se o nó receptor (normalmente a estação base) na raiz e os restantes têm de estar organizados de modo a que cada nó tenha no mínimo um nó raiz acima dele. Um esquema deste tipo leva a um maior desgaste dos sensores dos níveis superiores da árvore. Exemplos destes protocolos são dados em detalhe na secção 3.2.

No âmbito desta dissertação são relevantes os protocolos de encaminhamento plano, visto estes se inserirem nos objectivos do trabalho a realizar.

2.6 Segurança em RSSFs

Devido às características das RSSFs, as técnicas, mecanismos e serviços de segurança utilizados vulgarmente nas redes convencionais, ou mesmo no âmbito das redes móveis ad-hoc em geral, não podem ser directamente aplicados às redes de sensores. Dadas as características físicas, a natureza das comunicações sem fios e os cenários de aplicação destas redes, os mecanismos de segurança têm de estar presentes em todos os níveis da pilha de serviços, como referido na secção 2.2.

As redes de sensores são muitas vezes distribuídas em locais sem vigilância, constituindo um maior risco devido à eventualidade de ataques físicos. A interacção destas redes com o meio envolvente aumenta também o nível de vulnerabilidade, pois os sensores estão expostos a todo o tipo de adversidades proporcionadas pelo ambiente em que estão inseridos.

Uma das características que se pretende que uma RSSF segura possua é que a degradação do funcionamento da rede seja, no máximo, directamente proporcional ao número de atacantes/ataques activos, característica a que se dá o nome de *graceful degradation*. É claro que, do ponto de vista de propriedades de segurança e fiabilidade, uma rede será tanto melhor quanto mais resistir a condições de falha ou de ataque, pelo que quanto mais imune for a rede à quantidade de nós afectados e quanto mais mantiver condições de cobertura ou conectividade global, melhor serviço prestará. Este critério é crucial para um serviço de encaminhamento seguro, pois este deve garantir a conectividade da rede,

propiciar que os dados possam ser entregues de forma fiável e, de preferência, garantir que a disseminação dos dados se faça em condições óptimas em relação a critérios como latência da comunicação e energia consumida. Para que isso aconteça é necessário também que os vários serviços de segurança, implementados em diferentes camadas da pilha de suporte, se complementem para tornar a rede mais segura, resistente e fiável, que possuam complexidade computacional e de comunicação adequada às limitações dos dispositivos usados e que garantam optimização da energia ao nível do processamento dos nós e do processamento na rede, no seu conjunto.

De modo a compreender o tipo de ameaças a que as RSSFs estão sujeitas é necessário definir o modelo de adversário. Tendo em conta as ameaças é também necessário definir os mecanismos de segurança a utilizar e em que camada da pilha de suporte de inserem. A presente dissertação foca-se na camada de rede, no entanto é importante compreender quais as garantias de segurança das camadas abaixo, nomeadamente a camada MAC.

2.6.1 Modelo de Adversário

Os modelos de adversário são importantes na medida em que permitem definir o possível comportamento de um atacante perante um determinado sistema. O modelo de adversário para as RSSFs está muito bem definido por Karlof-Wagner [35] e a descrição que se segue é baseada nessa definição.

Nas RSSFs considera-se que um adversário pode ser passivo ou activo. O adversário passivo tem características semelhantes às do modelo de adversário definido por Dolev-Yao [22], limitando-se a interceptar comunicações com o intuito de conhecer a informação que está a ser passada. O adversário activo tem como objectivo alterar o comportamento da rede, sendo que o seu comportamento abrange algumas das características dos modelos de adversário de Dolev-Yao e da Framework OSI X.800 [71], podendo o atacante ter comportamentos como alterar a informação transmitida numa comunicação, fazer-se passar por vários nós da rede, fazer *drop* de todas as mensagens que lhe chegam e que deveria enviar para outros nós, enviar informação de um nó da rede para outro que não pertence à sua vizinhança, etc.

Independentemente de ser activo ou passivo, o adversário pode ser externo ou interno. Um adversário externo actua de fora da rede, onde pode observar o tráfego ou bloquear as comunicações da rede por *jamming*. O adversário externo pode pertencer a uma de duas classes: *sensor-class* ou *laptop-class*. Um adversário *sensor-class* utiliza um ou vários sensores como os da rede para efectuar o ataque, o que indica que apenas consegue afectar a vizinhança de cada sensor. Um adversário *laptop-class* está equipado de um computador portátil, que tem maior poder de computação, memória, comunicação e energia que os nós da rede, o que faz com que este adversário possa chegar a grande parte da rede, potencialmente toda a rede, com capacidade de efectuar ataques mais sofisticados.

O adversário interno é o mais perigoso para as RSSFs e é também a razão pela qual

estas redes necessitam de serviços de segurança que as tornem tolerantes a intrusões. Assume-se que este adversário é capaz de capturar um ou vários nós da rede (quaisquer nós) e capturar toda a informação contida nele, incluindo histórico de comunicações e chaves criptográficas. Deste modo o nó continua a pertencer à rede, uma vez que possui as credenciais que lhe foram dadas, mas pode ter um comportamento especificado pelo adversário. Pode, portanto, ter o mesmo comportamento do atacante externo, mas em posição privilegiada por ter a confiança dos outros nós da rede. Ao alterar o comportamento normal de um nó, um adversário, que neste caso é um intruso, pode atacar seguindo o modelo bizantino.

2.6.2 Camada MAC

Alguns dos ataques descritos no modelo de adversário são dirigidos à camada MAC. Desses ataques, alguns seguem o protocolo MAC definido e outros não.

Nos ataques que seguem o protocolo MAC, o adversário age como um membro legitimo da rede. Ele pode atacar ao inundar a rede com pacotes de tamanho máximo, criando um ataque de *Denial of Service* que acelere o esgotamento da energia dos nós correctos da rede e reduza a largura de banda disponível. Outro ataque trata-se de configurar um nó para funcionar como se não estivesse a funcionar com bateria, o que faz com que o mecanismo CSMA/CA monopolize o acesso ao meio, por diminuir o tempo de pausa entre retransmissões no caso da detecção de colisões. Ambos os ataques descritos diminuem a taxa de entrega de pacotes [54].

Quando um adversário não segue o protocolo MAC, este pode fazer com que um nó não utilize correctamente o mecanismo CSMA/CA, de modo a que nó comunique quando já existem outras transmissões, o que provoca colisões e bloqueia todas as transmissões. Este atacante pode também reenviar ou alterar informação, assim como enviar informação falsa (como ACKs falsos, por exemplo).

2.6.3 Camada de Rede

A camada de rede nas RSSFs está relacionada com os modelos de encaminhamento e disseminação de dados, assim como com a organização da rede. Também são considerados mecanismos de distribuição de chaves criptográficas, no entanto no contexto da presente dissertação assume-se que está disponível um esquema seguro de distribuição de chaves.

Os primeiros protocolos de encaminhamento que surgiram assumem que todos os nós agem de forma correcta, não tendo quaisquer preocupações com a segurança. Os protocolos desenvolvidos mais recentemente já consideram o modelo de adversário para as RSSFs. Uma das principais preocupações é fazer com que os protocolos sejam tolerantes a intrusões, sendo que os protocolos têm uma de duas abordagens: prevenção ou detecção/correcção.

	Descoberta	Selecção	Estabelecimento
Informação de encaminhamento falsa		X	X
Encaminhamento selectivo			X
Ataque Sinkhole	X		
Ataque Sybil	X		
Wormhole	X		
HELLO flood	X		

Tabela 2.1: Relação entre os ataques às RSSFs e as fases dos protocolos de encaminhamento

Os protocolos preventivos utilizam mecanismos para funcionarem o mais correctamente possível na presença de intrusões. Os protocolos que apostam na detecção/correcção implementam detectores de intrusos e, alguns deles, corrigem a rede removendo os nós intrusos. Na secção 3.2 são vistos alguns exemplos de protocolos com estas características.

2.6.4 Ataques ao Encaminhamento nas RSSFs

De acordo com Karlof-Wagner [35], existem os seguintes ataques internos ao encaminhamento nas RSSFs: informação de encaminhamento falsa, encaminhamento selectivo (*selective forwarding*), ataques Sinkhole, ataques Sybil, wormholes e ataques HELLO flood.

Estes ataques podem ter dois modos de actuar distintos, sendo que uns ataques visam a informação trocada entre os nós da rede e outros pretendem alterar a topologia da própria rede.

Os ataques descritos em seguida, devido às suas características, são direccionados a diferentes fases dos protocolos de encaminhamento nas RSSFs. A tabela 2.1 mostra a relação entre os ataques e as fases dos protocolos. Esta relação ajuda a compreender em que fases e em que operações se devem implementar os mecanismos de defesa contra cada ataque específico.

2.6.4.1 Informação de encaminhamento falsa

A forma mais directa de atacar uma RSSF é tentar alterar os dados trocados entre os nós da rede. Ao alterar, reenviar ou enviar informação relativa ao encaminhamento, um atacante consegue criar partições na rede, criar ciclos de encaminhamento, etc.

2.6.4.2 Encaminhamento selectivo (*selective forwarding*)

As redes multi ponto (*multi-hop*) funcionam seguindo o principio de que todos os nós vão entregar correctamente os dados que passam por eles. Um nó que realize este ataque decide o que fazer com a informação que lhe chega. O caso mais simples é o nó agir como um buraco negro (*blackhole*), limitando-se a não enviar toda a informação que lhe chega. Um ataque do tipo *blackhole* corre o risco de ser detectado facilmente pelos vizinhos do nó atacante, por isso pode-se fazer algo mais sofisticado, fazendo com que o nó decida

se encaminha ou não informação com base numa probabilidade de envio, por exemplo. Este ataque só tende a funcionar em casos em que o atacante é um intruso, ou seja interno à rede, e esteja no meio de uma rota. Os ataques Sinkhole e Sybil são bons exemplos de como um atacante se pode introduzir na rede.

2.6.4.3 Ataque Sinkhole

O objectivo deste ataque é o de atrair o tráfego de uma determinada zona da rede para um único nó comprometido, ao qual se dá o nome de *sinkhole*. Para realizar este ataque, o atacante faz com que o nó comprometido se torne atraente para os outros nós, no que diz respeito ao protocolo de encaminhamento, i.e. se o protocolo utiliza métricas para seleccionar rotas óptimas, como a distância até à estações base, o nó oferece boas rotas de modo a entrar na grande maioria das rotas dos outros nós.

2.6.4.4 Ataque Sybil

No ataque Sybil [23] um nó comprometido apresenta várias identidades diferentes aos outros nós da rede. No caso de uma RSSF do tipo *multi-path* que crie duas rotas disjuntas entre um nó origem e uma BS, é possível com este ataque que um único nó comprometido se insira nas duas rotas em simultâneo, comprometendo todas as comunicações provenientes do nó origem.

2.6.4.5 Wormhole

O ataque *wormhole* [16] consiste em criar um túnel fazendo com que o tráfego desapareça de uma zona da rede e reapareça numa zona onde não era suposto passar. O que normalmente ocorre neste ataque é o atacante ter dois nós comprometidos em diferentes partes da rede que transmitam informação entre sim por um canal de comunicação que apenas está disponível para o atacante. Com esta técnica, o atacante pode criar um *sinkhole*, ao colocar um nó comprometido no meio da rede e outro junto a uma BS, fazendo com que o nó que está no meio da rede ofereça uma rota de um único salto até à BS.

2.6.4.6 Ataque HELLO flood

Os ataques HELLO *flood* são relativamente recentes e aproveitam-se do facto de grande parte dos protocolos de encaminhamento para RSSFs utilizarem mensagens de HELLO para que um nó se apresente à sua vizinhança. Quando um nó vizinho recebe uma mensagem HELLO assume que o emissor se encontra está a uma distância que permite que exista comunicação entre eles. Como intruso e se usufruir de um dispositivo com grande alcance de comunicação, um atacante pode utilizar um nó comprometido para enviar mensagens de HELLO para toda a rede, fazendo os nós acreditarem que esse nó é seu vizinho, deixando a rede num estado de confusão.

3

Trabalho Relacionado

3.1 Mecanismos de Defesa para RSSFs

De modo a proteger as RSSFs contra os possíveis ataques há a necessidade de implementar mecanismos de segurança na pilha de protocolos da rede. Como foi referido na secção anterior, todas as camadas de protocolos da rede estão sujeitas a ataques, sendo o foco da tese a camada de transporte, mais especificamente os protocolos de encaminhamento.

Como foi referido na secção dedicada aos ataques ao encaminhamento nas RSSFs, há ataques aos dados e ataques à topologia da rede. Cada uma destas categorias de ataques tem características muito diferentes e, por isso, os mecânismos de segurança utilizados em cada uma são diferentes. Passe-se então a uma visão dos mecânismos para cada uma destas categorias de ataques.

3.1.1 Defesas Contra Ataques aos Dados

Os ataques aos dados resumem-se à observação, replicação ou alteração dos mesmos. Uma boa maneira de impedir que os dados sejam vistos por atacantes é tornar os dados confidenciais, fazendo com que apenas os nós da rede interessados sejam capazes de ver o conteúdo de cada mensagem. A utilização de algoritmos criptográficos para cifrar mensagens é o mecanismo de defesa mais utilizado para obter confidencialidade. Devido às limitações dos sensores nas RSSFs não é viável utilizar mecanismos de cifra assimétrica (ou de chave pública), sendo, por isso, utilizada a cifra simétrica na grande maioria dos casos. No entanto tem sido investigada a hipótese de utilizar soluções de cifra assimétrica baseadas em hardware [27] ou com criptografia de curvas elípticas (ECC) [37, 45].

Para proteger os dados contra alterações e para autenticar o emissor dos mesmos,

utilizam-se mecanismos de integridade de dados, como assinaturas MAC (*Message Authentication Code*) e os seus derivados (CMAC, HMAC, VMAC, UMAC) ou autenticação de mensagens com chave privada, no caso de se utilizar criptografia assimétrica.

Perrig et al. desenvolveram um conjunto de protocolos adaptados às RSSFs, ao qual deram o nome de SPINS [63] que implementa mecanismos de defesa contra ataques aos dados, incluindo confidencialidade e autenticação de dados, e detecção e descarte de réplicas. Exemplos de outros protocolos que implementam estes serviços de segurança são Zigbee [5], TinySec [34] e MiniSec [48].

3.1.2 Serviços de Tolerância a Intrusões

Os ataques internos são feitos por um adversário que consegue invadir um ou mais nós na rede. Para preparar uma RSSF contra intrusões é necessário implementar mecanismos ou serviços de tolerância a intrusões. Um comportamento errado devido a intrusão pode ser considerado como uma falha bizantina [40], sendo por isso possível utilizar alguns dos mecanismos de defesa contra falhas bizantinas.

Um dos mecanismo de defesa mais utilizado para tratar destes casos é um clássico: redundância. Nas RSSFs considera-se normalmente redundância em duas dimensões diferentes: rotas e nós de agregação.

Os protocolos de encaminhamento *multi-path* utilizam redundância de rotas de cada nó origem até aos nós de destino, para poderem transmitir dados pelas diferentes rotas, esperando que alguma dessas rotas não contenha um intruso. Para reforçar a resiliência deste mecanismo, em muitos casos, utilizam-se múltiplas rotas disjuntas, para que um único intruso não esteja presente em diferentes rotas.

Poucos protocolos de encaminhamento para RSSFs utilizam redundância de estações base. Este mecanismo, juntamente com multiplas rotas, permite ter várias rotas para várias estações base, aumentando o número de opções que um nó tem ao transmitir dados. Quando um nó envia dados para 3 ou mais estações base é necessário que estas entrem em acordo em relação ao valor que foi recebido.

Karlof-Wagner [35] afirmam que ataques como *wormhole* e *sinkhole* devem ser tratados na fase de desenho de um protocolo de encaminhamento, referindo que um protocolo que utilize métricas para seleccionar melhores caminhos está mais susceptível a estes ataques.

Um modo de minimizar os danos causados por um intruso é fazer com que cada nó tenha um número limitado de vizinhos. Esta medida tem duas dimensões: (1) impedir que um intruso afecte muitos nós e impedir que nós correctos aceitem vários intrusos como vizinhos (esta dimensão é particularmente direccionada ao ataque *Sybil*).

3.1.3 Mecanismos de Detecção e Correcção de Intrusões

A tolerância a falhas é importante na medida em que permite que uma RSSF funcione correctamente mesmo na presença de intrusos. No entanto, para que a rede seja mais

fiável, é desejável que os intrusos sejam inexistentes ou em número reduzido. Uma abordagem que surgiu recentemente é a de detecção e remoção de intrusos, que consiste na implementação de mecanismos para detectar nós intrusos na rede. A estas implementações dá-se o nome de *Intrusion Detection System* (IDS).

Pouco trabalho foi desenvolvido na área de IDSs até ao momento, sendo uma das principais limitações das soluções propostas o facto de serem direccionadas a um ataque específico.

SAM (*Statistical Analysis Multipath scheme*) [70] é um esquema utilizado para detectar ataques *wormhole*. Neste esquema, se um nó está presente num grande número de rotas a rede entra em modo de suspeita.

Abu-Ghazaleh et al. [1] propõem um IDS, integrado num protocolo de encaminhamento por múltiplas rotas, baseado em valores de confiança associados a cada nó na tabela de encaminhamento, para detectar e remover nós intrusos que estejam a fazer o ataque de encaminhamento selectivo. Quando se observa que uma mensagem é entregue ao destino correcto e sem ter sido alterada incrementa-se o valor de confiança do emissor. Caso contrário o valor de confiança é decrementado. Quando o valor de confiança de um nó está abaixo de um determinado limite inferior, o nó em questão é removido da tabela de encaminhamento. O comportamento dos nós é observado pela sua vizinhança, que determina se as mensagens são ou não bem entregues.

Ramaswami e Upadhyaya [64] sugerem um IDS para detecção do ataque *blackhole* e remoção dos nós intrusos. No entanto, o processo de detecção do nó não é descrito.

Zhao e Delgado-Frias [78] sugerem também um IDS para detecção do ataque *blackhole*, com recurso a múltiplas rotas, mas apenas consegue detectar rotas que contêm intrusos. O nó envia a mesma mensagem por duas rotas, se apenas chegar uma ao destino assume-se que existe um intruso.

Na maioria dos IDS, é a vizinhança dos nós que detecta comportamentos errados, sendo por isso uma detecção feita de forma distribuída. Uma das abordagens utilizadas é um sistema de *watchdogs*, que consiste em ter vários nós monitores espalhados na rede que observam o comportamento dos outros nós [51, 18]. Outra abordagem é a utilização de um *neighborwood watch system*, no qual a vizinhança de um nó observa, guarda e troca informação sobre as comunicações desse nó. A grande diferença entre os dois sistemas de observação de vizinhança descritos, é que o primeiro utiliza nós monitores, o que sugere uma rede heterogénea, enquanto que o segundo utiliza nós "normais", podendo assim aplicar-se a uma rede homogénea.

Alguns protocolos de encaminhamento para RSSFs que implementam IDS no seu funcionamento são descritos na secção 3.2.

Na verdade, a inclusão de mecanismos de detecção de intrusões que podem ainda ser complementados com mecanismos de reconfiguração ou recuperação da rede face a intrusões, é uma área que só mais recentemente tem sido explorada ao nível de serviços de encaminhamento para redes de sensores. O principal desafio em conseguirem suportar-se protocolos de encaminhamento tolerantes a intrusões que conjuguem as dimensões de

prevenção, detecção e recuperação prende-se com a complexidade deste tipo de mecanismos. Esta complexidade é particularmente notável em redes de grande escala, com encaminhamento *multi-hop* e sem nós especiais de supervisão (que possam ser imunes a intrusões e permitam instituir componentes para computações confiáveis associadas à existência de uma base de confiança). Por um lado, a complexidade introduzida por esse tipo de soluções pode ser irrealista ou incompatível dadas as limitações de processamento, comunicação e energia dos nós. Por outro lado, mecanismos de detecção, descarte ou recuperação de nós afectados por falhas ou ataques por intrusão podem introduzir desvantagens importantes, onerando o desempenho da rede e o balanço energético da rede.

Apesar de não fazer parte do foco da presente dissertação, esta secção sobre IDSs permite ver que há poucas soluções, sendo que a maior parte se baseia em soluções distribuídas e são executadas pelos nós sensores. A ideia que surgiu e que é discutida com pouco detalhe na secção de trabalho futuro pretende apresentar um serviço de deteção e recuperação de intrusões executado pelas estações base, baseado nos protocolos de consenso, o que contraria a tendência das soluções referidas nesta secção.

3.2 Protocolos de Encaminhamento Plano para RSSFs

Como foi referido anteriormente, no decorrer da presente dissertação pretende-se desenvolver um protocolo de encaminhamento tolerante a intrusões para RSSFs e determinou-se à partida que esse protocolo deve ser plano no que diz respeito à organização dos nós na rede.

Foram desenvolvidos no passado vários protocolos de encaminhamento para RSSFs, no entanto apenas recentemente começou a surgir a preocupação da tolerância a intrusões e, por isso, ainda não existem muitas soluções. Nesta secção são vistas as contribuições mais significativas nesta área, sendo que algumas delas serviram de base ou inspiração para o protocolo MINSENS++, desenvolvido no âmbito desta dissertação.

3.2.1 INSENS

O INSENS (*INtrusion-tolerant routing protocol for wireless SEnsor NetworkS*) [20] é um protocolo de encaminhamento seguro, tolerante a intrusões, que utiliza múltiplas rotas disjuntas para ultrapassar nós comprometidos. Uma das principais preocupações do protocolo é impedir que um único nó consiga afectar o funcionamento geral da rede. Para tal, o INSENS oferece defesas contra ataques de DoS que pretendam inundar a rede e ataques ao encaminhamento que tenham como intenção propagar dados alterados.

Este protocolo tira partido da assimetria nos nós para tratar do problema do consumo energético. A única estação base, como tem um maior poder computacional e energético que os sensores, agrega os dados enviados por todos os sensores e tem o trabalho de

calcular as tabelas de encaminhamento de todos os sensores da rede. A estação base partilha uma chave de cifra simétrica com cada sensor da rede, para que a comunicação seja segura.

Para prevenir que um único nó consiga afectar toda a rede, as comunicações têm algumas limitações. Apenas a estação base tem capacidade de comunicar por *broadcast* (para toda a rede), sendo as mensagens autenticadas para evitar *tampering*. As comunicações *unicast* são sempre dirigidas à estação base, para evitar que seja lançado um ataque de DoS contra um nó específico.

O protocolo funciona em três rondas:

- ***Route Request:*** a estação base inunda a rede com um pedido para que cada sensor se dê a conhecer aos seus vizinhos.
- ***Route Feedback:*** os sensores respondem à estação base enviando a sua topologia local. A mensagem é enviada pelo caminho inverso pelo qual chegou ao sensor.
- ***Routing Table Propagation:*** A estação base constrói as tabelas de encaminhamento para cada sensor e envia cada uma para o sensor correspondente.

3.2.2 MINSENS

O protocolo MINSENS [28] surge como uma melhoria do INSENS, onde se considera a existência de múltiplas estações base, ou seja, utiliza-se redundância de nós de agregação para aumentar a fiabilidade e balancear o gasto energético da RSSF.

Cada estação base funciona como uma estação base do INSENS ao criar a sua árvore de disseminação. Assim sendo, cada nó sensor tem B tabelas de encaminhamento, sendo B o número de estações base no sistema. Para que um nó consiga perceber o próximo destino de uma mensagem, atribui-se um identificador único a cada rota, que é incluído nas mensagens transmitidas.

No MINSENS todas as rotas de todas as estações base são disjuntas, i.e. cada nó é nó interno apenas de uma rota em todo o sistema. Esta opção garante que um intruso afecta apenas uma rota no sistema. Cada estação base calcula rotas disjuntas apenas para si, não havendo garantias de que as suas rotas são completamente disjuntas das rotas das outras estações base. Por isso, depois da descoberta individual das rotas, as estações base trocam as rotas geradas e entram em acordo sobre que rotas cada uma vai utilizar, de modo a que todas sejam disjuntas.

O protocolo é muito flexível no que diz respeito ao encaminhamento. Um nó pode enviar dados apenas para uma estação base e por um só caminho aleatório ou enviar por vários caminhos. Do mesmo modo que pode enviar dados apenas para uma ou para várias estações base. No caso em que são enviados dados para várias estações base é necessário que estas entrem em consenso em relação ao valor recebido.

Pode observar-se que o MINSENS baseia-se muito em consensos para funcionar correctamente. Na secção 3.3 é aprofundado o tema do consenso distribuído em RSSFs.

3.2.3 Clean-Slate

O protocolo Clean-Slate [58] foi desenvolvido com 3 objectivos em mente: prevenção, detecção/correcção e resiliência.

No inicio do funcionamento do protocolo, existe uma autoridade na rede que atribui a cada sensor um identificador certificado e um conjunto de desafios aleatórios. Os sensores iniciam, então, um processo seguro de descoberta de vizinhos, em que cada sensor se dá a conhecer aos seus vizinhos directos. Este processo previne a entrada de novos sensores intrusos no futuro. Depois de estabelecida a vizinhança, inicia-se um algoritmo de agrupamento recursivo, que utiliza um serviço de difusão fiável (*reliable broadcast*) para trocar mensagens e é atribuído um endereço de rede único a cada sensor.

A prevenção é conseguida com a implementação de mecanismos de criptografia para proteger os dados e obtém-se resiliência através da utilização de múltiplas rotas.

O Clean-Slate faz detecção de nós duplicados e de mensagens alteradas de dois modos diferentes. No primeiro caso, depois do algoritmo de agrupamento recursivo, os sensores enviam o seu identificador e endereço de rede para os vizinhos, para se iniciar um algoritmo de detecção de réplicas [59]. No segundo caso, é utilizada uma *Grouping Verification Tree* (GVT), baseada em *Hash Trees* (também conhecidas como *Merkle hash trees* [52]).

Para remover os intrusos utiliza-se o mecanismo de recuperação *Honeybee*, que define que, quando um sensor detecta e reporta outro sensor intruso, tanto o sensor que reporta como o sensor intruso são removidos da rede. Remove-se ambos para evitar que sensores maliciosos acusem sensores correctos de serem intrusos.

3.2.4 H-SPREAD

O H-SPREAD [46] é um protocolo de encaminhamento com múltiplas rotas disjuntas para uma estação base, que utiliza um *threshold secret sharing scheme* para enviar mensagens pelas múltiplas rotas, de modo a aumentar a fiabilidade da rede.

Um *threshold secret sharing scheme* consiste em dividir um segredo (*secret*) em várias partes, chamadas de *shares*. Mais concretamente, é aplicado um algoritmo $A(T, N)$ ao *secret* que gera N *shares*. Para alguém conseguir reconstruir o *secret* necessita de ter, pelo menos, T *shares*.

A descoberta das múltiplas rotas disjuntas é feita de forma distribuída por todos os nós da rede, com recurso a um algoritmo de *branch-aware flooding* para calcular uma árvore de cobertura para a rede, seguido de outro algoritmo que cria novas ligações para gerar as múltiplas rotas.

Durante a fase em que os nós tratam do encaminhamento de mensagens, quando um sensor pretende enviar uma mensagem para a estação base, executa o algoritmo de *threshold sharing scheme* para gerar os *shares* da mensagem. Os vários *shares* são enviados pelas múltiplas rotas para a estação base. Deste modo, para que uma mensagem seja comprometida por um adversário é necessário que este comprometa, no mínimo, T rotas

do nó origem para a estação base.

3.2.5 SeRINS

O SeRINS (*Secure alternate path Routing IN Sensor networks*) [42] é um protocolo de encaminhamento com múltiplas rotas para uma estação base, em que os nós da rede estão estruturados numa árvore com a estação base como raiz e com nós que têm múltiplos nós pais. O protocolo faz detecção e eliminação de nós que enviem informação de encaminhamento errada e utiliza as múltiplas rotas para prevenir outras intrusões.

O processo de geração de rotas é iniciado pela estação base, que faz *broadcast* de um *route update* a anunciar que está a distância (em nº de *hops* até à estação base) 0. Qualquer nó, ao receber uma mensagem de *route update*, se não tem pai principal adiciona o nó como pai principal, se tem pai principal e a distância anunciada na mensagem é igual ou inferior à distância do pai principal adiciona o nó como pai normal.

Os nós da rede encaminham sempre as mensagens para um dos pais (principal ou normal), escolhido aleatoriamente. Esta medida serve para desviar de possíveis atacantes, inserindo não-determinismo no encaminhamento.

Ao utilizar a distância como métrica para definir os pais, o protocolo poderia ser facilmente violado se um intruso anunciasse uma distância falsa. Para isso foi implementado um sistema de detecção de intrusos que alteram a informação do encaminhamento, baseado na informação dada pelos vizinhos, sendo que a decisão final é da estação base. Quando se detecta um intruso, todos os nós da rede que o tenham como pai removem-no da lista de pais.

3.2.6 Análise Comparativa

Os protocolos de encaminhamento apresentados têm comportamentos diferentes no que diz respeito ao modo como abordam os ataques. Todos os protocolos referidos estão protegidos contra ataques externos, i.e. visualização do conteúdo das mensagens, alteração de mensagens e reenvio de mensagens. No caso particular do Clean-Slate, em relação à alteração de mensagens, o protocolo protege as suas comunicações de adversários externos e tem um mecanismo de detecção/remoção de intrusos que alterem mensagens durante o seu encaminhamento.

A tabela 3.1 mostra as defesas dos protocolos contra ataques realizados por intrusos. O valor **Prob.** indica que a defesa que o protocolo tem contra o ataque é probabilística. **Remoção** indica que o protocolo tem mecanismos para detectar e remover os intrusos que estejam a realizar o ataque. No caso do ataque *Sinkhole*, alguns protocolos têm o valor **N/A**, basicamente porque por não utilizarem métricas para o estabelecimento das rotas, é impossível para um nó mostrar-se mais atractivo aos outros e fazer um ataque *Sinkhole*.

Os únicos protocolos de encaminhamento para RSSFs encontrados, que têm características de tolerância a intrusões e incluem mecanismos de detecção e remoção de nós

	Informação de encaminhamento falsa	Encaminhamento selectivo	Sybil	Sinkhole	Worm hole	HELLO flood
INSENS	✓	Prob.	✓	N/A		✓
MINSENS	✓	Prob.	✓	N/A		✓
Clean-Slate	✓	Prob.	Remoção	N/A		✓
H-SPREAD	✓	Prob.		N/A		
SeRINS	Remoção	Prob.		Remoção		

Tabela 3.1: Defesas dos protocolos de encaminhamento contra os ataques internos

intrusos foram o Clean-Slate e o SeRINS. Mesmo assim, os dois protocolos são muito diferentes neste mecanismos, sendo que nem servem para defender os mesmos ataques. São, no entanto, interessantes para estudar diferentes mecanismos de detecção/remoção de intrusos.

Todos os protocolos referidos acima utilizam múltiplas rotas para as estações base, o que os faz ter defesas probabilísticas contra ataques de encaminhamento selectivo, mas todos eles o fazem de maneira diferente, tanto no modo como são descobertas as rotas como no próprio encaminhamento dos dados.

A solução desenvolvida baseia-se no protocolo MINSENS, onde são inseridos protocolos de consenso probabilístico adequados às características das estações base e que permitem estender o modelo de adversário às estações base, uma vez que os próprios protocolos de consenso são tolerantes a falhas e intrusões.

A solução de detecção e correcção de intrusos idealizada poderia inserir-se no protocolo MINSENS++, uma vez que se baseia nos resultados dos protocolos de consenso.

3.3 Consenso Distribuído

Os protocolos de acordo têm um papel importante em sistemas distribuídos. Essencialmente estes protocolos permitem que um grupo de processos entrem em acordo em relação a qualquer tipo de informação [60]. O modo mais comum de encapsular o problema de chegar a um acordo é através da definição de consenso. O problema de consenso consiste num conjunto de processos em que cada processo propõe um valor e todos têm de decidir um valor comum, que tenha sido proposto por um dos processos.

Apesar da sua definição simples, a solução para o problema do consenso em sistemas distribuídos tolerantes a falhas está longe de ser trivial. Estes sistemas devem funcionar correctamente mesmo na presença de componentes do sistema que estejam sujeitos a falhas. É quando se considera a presença de falhas que o problema do consenso se torna mais complexo. Dependendo do modelo do sistema, é bem possível que o problema do consenso não tenha uma solução determinística. Para que um protocolo de consenso que tolere falhas tem de respeitar as seguintes propriedades:

- **Terminação:** Todos os processos correctos eventualmente decidem um valor.
- **Validade:** Um processo apenas pode decidir um valor que tenha sido proposto

previamente.

- **Acordo:** Todos os processos correctos decidem o mesmo valor.

Fazem parte do modelo de sistema os modelos de falhas e modelos de sincronia. O modelo de falhas define os tipos de falhas que podem ocorrer no sistema, podendo essas falhas ser falhas por omissão ou falhas bizantinas. O modelo de sincronia define os intervalos de tempo requeridos para executar as tarefas do sistema.

Nos dois extremos dos modelos de sincronia estão um sistema síncrono e um sistema assíncrono. Num sistema síncrono assume-se que todas as operações podem demorar um determinado intervalo de tempo a serem executadas. Já no modelo assíncrono assume-se que todas as operações têm um tempo de execução indeterminado. Desenvolver um protocolo de consenso assumindo um sistema assíncrono é uma vantagem do ponto de vista de segurança e tolerância a falhas.

Na secção 3.3.1 apresenta-se um resultado de impossibilidade que prova não ser possível desenvolver uma solução determinística para o problema do consenso na presença de pelo menos uma falha, o que implica que não existem soluções determinísticas para sistemas com um modelo assíncrono. As RSSFs têm um modelo assíncrono, devido ao carácter faltoso da comunicação dos sensores. Por esse mesmo motivo, torna-se imperativo utilizar soluções de consenso não-determinístico numa RSSF.

O protocolo de encaminhamento proposto nesta tese necessita de realizar consensos de valores entre nós da rede e, por isso, serão analisados protocolos não-determinísticos de consenso distribuído, com propriedades de tolerância a falhas bizantinas.

3.3.1 Resultados de Impossibilidade

Como foi referido anteriormente, os sistemas assíncronos estão limitados por um resultado de impossibilidade, ao qual usualmente se dá o nome de FLP *impossibility* [26]. Na sua essência, o resultado atesta que o consenso não pode ser resolvido deterministicamente em sistemas assíncronos no caso de um único processo falhar. Intuitivamente o resultado vem do facto de, dada uma configuração assíncrona, é impossível para um processo que ainda não recebeu uma mensagem de que está à espera saber se o outro processo falhou ou simplesmente está lento. Por ser impossível identificar um processo que falhou, os processos correctos podem ficar à espera de mensagens por tempo indeterminado.

Existe também um resultado de impossibilidade para sistemas síncronos, que é a impossibilidade de Santoro e Widmayer [66]. O resultado desta impossibilidade atesta que num sistema síncrono é impossível realizar consenso determinístico se n-1 ou mais mensagens podem ser perdidas, por ronda de comunicação, num sistema com n processos.

3.3.2 Consenso Não-Determinístico

Desde que a impossibilidade de FLP foi publicada foi feita muito trabalho para tentar contornar o resultado da impossibilidade. Essencialmente as soluções propostas baseiam-se em extensões ao modelo base que incluem: Aleatoriedade, mecanismos mais fortes de sincronização, detectores de falhas e suposições temporais adicionais.

3.3.2.1 Modelos de Aleatoriedade

Os modelos aleatórios introduzem terminação probabilística ao problema do consenso. Se se estender a propriedade de terminação para que o consenso termine apenas com probabilidade 1, a impossibilidade de FLP não se aplica se a probabilidade colectiva de existir um processo que falha for 0.

Existem dois modos de introduzir aleatoriedade em modelos. Um deles é assumindo que o próprio modelo base já é aleatório, isto é, que as operações acontecem de forma probabilística. A este modelo deu-se o nome de *randomized scheduling*, originalmente proposto por Bracha e Toueg [8], e não se tornou uma solução muito utilizada por fazer suposições em relação ao modelo base que não são aplicáveis na prática [3]. Outro modo, o mais utilizado, é inserir aleatoriedade nos processos através da utilização de algoritmos aleatórios.

Nos algoritmos aleatórios os processos têm disponíveis operações de *coin-flip* que devolvem valores binários aleatórios, de acordo com uma dada distribuição de probabilidades.

As garantias que os algoritmos aleatórios oferecem no que diz respeito à segurança depende do tipo de adversário que se considera. Quando um adversário tem acesso à informação de *coin-toss* anteriores, consegue calcular os próximos valores.

No desenvolvimento de algoritmos de consenso aleatório com *coin-flips* foram adoptadas duas metodologias diferentes: *Local Coin-toss Protocol* (LCP) e *Shared Coin-toss Protocol* (SCP). Os modos de operação são muito diferentes nos dois casos. Os protocolos LCP (e.g. [6, 7]) são mais simples mas espera-se que terminem num número exponencial de rondas de execução, enquanto que os protocolos SCP (e.g. [10, 11]) utilizam esquemas criptográficos sofisticados, normalmente baseados em cifra assimétrica, e terminam num número constante de rondas.

Os algoritmos referidos nesta secção apenas resolvem o problema do consenso binário, o que pode não ser suficiente para algumas aplicações, que necessitem de realizar consensos sobre um valor estruturado ou até um conjunto de valores estruturados. Na secção 3.3.4 é dado um exemplos de uma pilha de protocolos que trata o problema do consenso de valores estruturados e conjuntos de valores.

3.3.2.2 Suposições Temporais Adicionais

A impossibilidade de FLP pode ser contornada com a utilização de limites para o tempo de transmissão de mensagens no sistema. Dolev, Dwork e Stockmeyer foram os grandes

pioneiros desta solução ao apresentarem o impacto de se adicionar sincronia limitada a sistemas de troca de mensagens [21]. Mais tarde Dwork, Lynch e Stockmeyer introduzem o modelo de sincronia parcial, tendo também desenvolvido fórmulas para o cálculo dos limites temporais bizantinos, dados o número de processos faltosos que se pretende considerar e alguns valores de limites temporais conhecidos pelos processos [24, 4].

3.3.2.3 Detectores de Falhas

Com esta abordagem considera-se que existem mecanismos no sistema que notificam os processos participantes de que um outro processo falhou. Um exemplo já foi referido anteriormente, que são os limites superiores para a espera de uma mensagem. Alguns detectores de falhas podem ser vistos como não-confiáveis, uma vez que podem identificar um processo correcto como faltoso e vice-versa. O desenvolvimento deste tipo de detectores foi iniciado por Chandra e Toureg, que definem as condições mínimas que um detector de falhas não-confiável deve satisfazer para permitir consenso [14, 13].

3.3.2.4 Primitivas Fortes de Comunicação

Nestes modelos utilizam-se primitivas fortes de memória partilhada para estender ou apoiar as operações básicas de leitura e escrita de dados. Loui e Abu-Amara mostraram ser possível realizar consenso com uma ou mais possíveis falhas com a utilização de bits de controlo [47]. Outra grande corrente de trabalho feito nesta área foi iniciada por Herlihy, que definiu uma hierarquia de objectos com memória partilhada para resolver o consenso [32].

3.3.3 Consenso Tolerante a Falhas Bizantinas

Dado o facto de o modelo de adversário considerado para as RSSFs incluir a possibilidade de existir intrusos na rede deve-se assumir que alguns dos participantes de um consenso possam apresentar um comportamento errado. Daí vem a importância de considerar soluções de consenso tolerantes a falhas bizantinas.

As soluções para consenso distribuído tolerante a falhas bizantinas estão limitadas pelas conclusões tiradas em [40], de onde se obtém que, num sistema com N nós é possível ter $f = \frac{(N-1)}{3}$, sendo f o número máximo de nós com comportamentos maliciosos.

O PBFT [12] é um algoritmo determinístico de replicação que dá garantias de safety e liveness até $f = \frac{(N-1)}{3}$, sendo f o número de réplicas bizantinas toleradas. Na sua versão original, o algoritmo é composto por uma réplica principal, que atende os pedidos dos clientes e é responsável por iniciar o processo de sincronização das réplicas, e por réplicas secundárias que se limitam a guardar a informação redundante. O processo de sincronização das réplicas inicia-se quando se suspeita que a réplica principal pode estar comprometida. A suspeita de falhas na réplica principal é levantada pelo cliente que faz os pedidos.

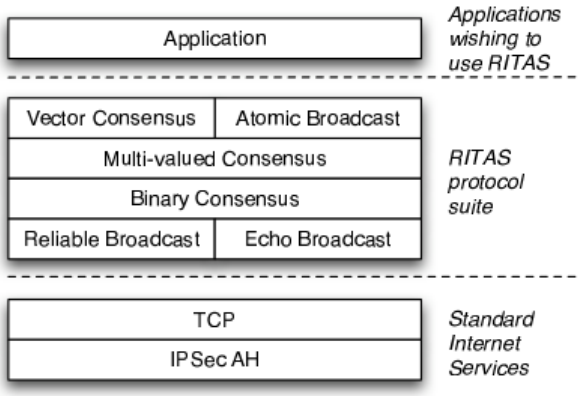


Figura 3.1: Pilha de Protocolos RITAS

Paxos é uma família de protocolos para resolver o problema do consenso em sistemas com processos não-fiáveis. Existem vários tipos de algoritmos Paxos, destinados a resolver diferentes problemas, como reduzir o número de mensagens ou o tempo de execução. Existe um algoritmo Paxos que considera a possibilidade de alguns dos participantes do algoritmo terem falhas bizantinas, ao qual se dá o nome de Paxos Bizantino. Nesta versão do Paxos existe um coordenador que recebe pedidos dos clientes e inicia o processo de consenso. O consenso é feito entre as várias réplicas que contêm os dados que se pretende retornar ao cliente.

3.3.4 Consenso Não-Determinístico e Tolerante a Falhas Bizantinas

Muito devido às características das redes sem fios ad-hoc no geral, tem havido um esforço em desenvolver soluções para resolver o problema do consenso não-determinístico e tolerantes a falhas bizantinas. Nesta secção são dados dois exemplos de pilhas de protocolos que pretendem resolver este problema.

A RITAS (*Randomized Intrusion-Tolerant Asynchronous Services*) [56] é uma pilha de protocolos (c.f.3.1), todos eles assíncronos, que utiliza algoritmos aleatórios para resolver o problema do consenso binário. Um dos principais objectivos desta implementação é mostrar que os protocolos aleatórios, nomeadamente os LCP, são eficientes a resolver o problema do consenso em LAN's e WAN's. A camada de TCP é utilizada para garantir fiabilidade e a de IPSec para garantir integridade dos dados transmitidos. As camadas entre o TCP e a Aplicação têm como objectivo resolver o problema do consenso, com resiliência óptima para $f = \frac{(N-1)}{3}$ processos com falhas bizantinas. A camada de consenso binário utiliza um protocolo aleatório baseado no protocolo desenvolvido por Bracha [7], ou seja, um LCP.

O Turquois [55] é um protocolo para resolver o problema do consenso distribuído assíncrono e binário (baseado em LCP), em que apenas é necessário que k de n nós entrem e consenso sobre um valor. Assume-se que os nós participantes estão à distância de uma

ligação, para aproveitar a comunicação natural por broadcast nas redes sem fios, onde o custo do envio de uma mensagem para todos os nós da vizinhança é igual ao custo de enviar para apenas um nó. Assume-se também que o sistema não oferece nenhum mecanismo de comunicação fiável (e.g. TCP), para se aproximar das características das redes ad-hoc sem fios. Os resultados obtidos na avaliação do Turquois são muito prometedores, quando comparados com os de outras soluções existentes.

3.3.5 Análise Crítica

As RSSFs têm um modelo de comunicação assíncrono e estão sujeitas a intrusões, pelo que se torna necessário que, para realizar consenso numa RSSFs, se utilize uma solução de consenso que consiga adaptar-se a estas características. As soluções de consenso não-determinístico tolerantes a intrusões são as que mais se adequam ao modelo de sistema considerado, sendo as soluções vistas a RITAS e a Turquois.

A RITAS não se adequa totalmente às RSSFs por necessitar de TCP, o que não é prático de utilizar nestas redes devido à natureza das suas comunicações. Por esse motivo, a solução mais indicada para incluir num protocolo de encaminhamento tolerante a intrusões para RSSFs é a Turquois, por ser assíncrona e não assumir qualquer tipo de fiabilidade na comunicação. O único problema é necessitar de que os nós participantes estejam a uma ligação de distancia, que é resolvido com a utilização de uma rede sobreposta, o que pode prejudicar a performance do protocolo.

Assumindo que o sistema sobre o qual os participantes no consenso comunicam tem um modelo de comunicação síncrono, a solução escolhida é a do Paxos bizantino.

Todas as soluções apresentadas assumem que cada participante do protocolo de consenso conhece todos os outros. Caso se pretenda realizar o consenso sem conhecimento dos participantes pode integrar-se o protocolo de consenso no BFT-CUP (*Byzantine Fault-Tolerant Consensus with Unknown Participants*) [2].

Este tipo de soluções induzirá necessariamente maior complexidade e exigência de recursos computacionais na rede. A implementação deste tipo de mecanismos em redes planas de sensores de baixo custo poderá ser um exercício irrealista, pelo menos ao nível da tecnologia existente e mesmo emergente. Uma possibilidade é conceber este tipo de mecanismos em nós especiais da rede, dotados de mais recursos computacionais. Outra possibilidade será implementar estes mecanismos como mecanismos de processamento complementar ao nível de estações base. Tal permitirá abordar a concepção de novos serviços de tolerância a intrusões adaptáveis a ambientes de redes de sensores sem fios, combinando serviços de encaminhamento seguro com prevenção face a intrusões, a formas de agregação segura de dados naqueles nós especiais, mesmo considerando modelos de adversário que incluam ataques por intrusão a esses mesmos nós. Esta hipótese vem na linha dos objectivos e contribuições da presente dissertação.

3.4 Simulação de RSSFs

A partir do momento em que se projectam aplicações críticas a funcionar sobre RSSFs é importante ter algum modo de testar e verificar o comportamento do software desenvolvido para estas redes antes de ser implementado nos dispositivos reais. Nestas circunstâncias, o desenvolvimento de plataformas de simulação de RSSFs torna-se mais importante, para que o software introduzido nos sensores seja o mais fiável possível.

Para que uma plataforma de simulação seja o mais realista possível deve conter:

- **Emulador:** deve fazer com que os nós virtuais tenham um comportamento semelhante ao dos nós reais (ex.: comunicação rádio, sensores, código executável, etc.).

- **Simulador:** Permite construir diferentes configurações da rede para testar o seu funcionamento e avaliar certos indicadores como a latência, o consumo de energia, etc.

Nas próximas secções são vistos alguns exemplos de plataformas de simulação de RSSFs, fazendo-se uma análise crítica de todos, de modo a escolher a que mais se adequa para testar e avaliar a solução proposta.

3.4.1 TOSSIM/PowerTOSSIM

O TOSSIM [43] é um simulador exclusivamente para sensores equipados com o sistema operativo TinyOS. Este simulador compila o mesmo código que os sensores físicos podem correr e emula um tipo limitado de hardware. Este simulador tem muitas limitações, como o emulador ser demasiado simplista, o que leva a resultados muito diferentes quando se compara uma distribuição real com uma simulação; ou o facto de apenas suportar RSSFs simétricas, pois todos os nós do simulador têm de correr o mesmo código.

O PowerTOSSIM [68] surge como uma extensão ao TOSSIM, onde se introduzem mecanismos de emulação de consumo de energia nos nós. São feitas medições do consumo de energia para cada dispositivo de hardware dentro do sensor (CPU, sensores, comunicação rádio, etc.). Esta é uma extensão importante visto que uma das principais características das RSSFs é terem dispositivos muito limitados a nível energético.

3.4.2 Freemote

O Freemote [50] é uma plataforma de emulação de RSSFs baseada em Java, utilizada para desenvolver software para RSSFs. Esta plataforma permite emular nós que correm código Java e que incorporam versões optimizadas de JVMs para sensores. A plataforma divide a arquitectura do software em 3 camadas distintas: Aplicação, Encaminhamento e Comunicação, Hardware. Os nós reais podem ser quaisquer dispositivos baseados na interface de comunicação standard IEEE802.15.4 (ex.: MICAz, JMotes, Tmote Sky).

3.4.3 Avrora

O *AVR Simulation and Analysis FrameworkPlatform* é um projecto de investigação do qual surgiu o simulador Avrora [72]. É utilizado como um conjunto de ferramentas de simulação e análise para programas escritos para o micro-controlador AVR produzido pelos sensores Mica2. Este simulador oferece uma implementação quase completa e realista do hardware dos sensores Mica2.
O AvroraZ [19] surge como uma extensão ao Avrora e permite emular sensores com o micro-controlador AVR e comunicação rádio pela norma IEEE802.15.4 (ex.: Crossbow's MicaZ). O objectivo desta plataforma é fornecer uma emulação precisa da comunicação rádio, sem que se façam alterações no código dos nós reais para os do simulador.

3.4.4 VMNet

O VMNet [77] é um emulador de RSSFs que tem como objectivo fornecer avaliação de performance realista para aplicações de RSSFs.Neste simulador uma RSSFs é emulada como uma VMN (Virtual Mote Network). O código desenvolvido para os sensores reais pode ser corrido no emulador, sendo que este emula o funcionamento dos componentes de hardware do dispositivo real e faz uma avaliação realista do tempo de resposta e consumo de energia. Actualmente os únicos dispositivos suportados pelo VMNet são os Crossbow Mica2.

3.4.5 NS-2 e NS-3

Os simulador NS2 e NS3 (*Network Simulator*) [57] são simuladores de eventos discretos mais focados para apoiar a investigação na área de redes de computadores. É provavelmente o simulador mais utilizado que suporta simulação de RSSFs. Inclui um grande número de protocolos, geradores de tráfego e ferramentas para simular vários protocolos de encaminhamento sobre redes com e sem fios, locais ou por satélite. O seu principal foco é a simulação do modelo ISO/OSI, incluindo a geração aleatória de fenómenos no nível de hardware e modelos de consumo de energia. Para as RSSFs inclui modelos de simulação de sensores, bateria, pilhas de protocolos para sensores e tem ferramentas de geração de estatísticas relativas ao funcionamento da rede.

A grande desvantagem destes simuladores é o facto de o seu nível de detalhe na simulação tornar praticamente impossível simular redes muito grandes (até aos 1000 nós [38]).

3.4.6 SENSE

SENSE [15] é um simulador específico para RSSFs. Oferece modelos de bateria e das camadas de encaminhamento e de aplicação. A implementação de comunicação rádio é limitada à norma IEEE 802.11. Actualmente o SENSE é capaz de suportar redes com

cerca de 5.000 nós, mas este número pode baixar para 500, dependendo dos padrões de comunicação utilizados.

3.4.7 JProwler

O JProwler [41] é um simulador de eventos discretos, implementado em Java, que simula a transmissão, propagação e recepção rádio, incluindo colisões, e a operação da camada MAC. As definições de rádio são plugins, o que oferece uma grande extensibilidade ao simulador. O simulador de eventos discretos pode ser configurado para funcionar no modo determinístico, de maneira a reproduzir resultados replicáveis, ou no modo probabilistico que simula a natureza indeterminística dos canais de comunicação e dos protocolos de comunicação de baixo nível dos motes.

3.4.8 WiSeNet

O WiSeNet [69] é um simulador baseado no Jprowler, que acrescenta um mecanismo de simulação de ataques a RSSFs. O seu principal foco no desenho foi facilitar a criação e configuração de topologias de redes.

Alguns dos pontos interessantes deste simulador são o facto de ter uma interface gráfica para visualizar e configurar a rede, com informação geral da rede e individual para cada nó, um gerador de topologias de rede que permite gerar topologias aleatórias, em grid ou estruturadas, e contém também módulos para extracção de diversas métricas como o consumo de energia, a latência, fiabilidade, etc.

3.4.9 Análise Crítica

De modo a avaliar com rigor protocolo de encaminhamento desenvolvido nesta tese é necessário tirar conclusões sobre o seu comportamento enquanto está a ser atacado. Por ser o único simulador que permite introduzir facilmente ataques à rede e também por providenciar informação sobre métricas que são importantes para avaliar o funcionamento da rede (latência, consumo de energia, fiabilidade, etc.), o simulador WiSeNet foi o escolhido para ser a plataforma de teste, de onde se retiraram os dados que permitiram avaliar o funcionamento do protocolo desenvolvido.

4

Modelo de Sistema

O MINSENS++ é um protocolo de encaminhamento tolerante a falhas e intrusões desenvolvido atendendo às características das redes de sensores sem fios auto-organizáveis, constituídas por nós autónomos, geograficamente distribuídos aleatoriamente em ambientes de campo aberto. Os nós têm as características e limitações descritas na secção 2.1 e têm como função monitorizar, pré-processar e encaminhar eventos (normalmente associados a medidas de condições de fenómenos físicos ou ambientais).

4.1 Modelo de Rede

O protocolo foi concebido para redes de média e grande escala, suportando um número de nós que pode ir de algumas centenas até às dezenas de milhar (de 100 a 15000 sensores). Os nós formam uma rede *multi-hop*, na qual são transmitidos e encaminhados os dados até nós especiais de processamento e armazenamento, denominados de nós agregadores ou estações base. Por sua vez, os nós especiais estão integrados, através de *gateways* dedicados, com outros ambientes de redes (por exemplo redes locais IEEE 802.3 ou IEEE 802.11, com suporte de pilha TCP/IP) ou com aplicações de gestão dos dados capturados, que executam em sistemas finais.

A topologia da rede deve ser plana, sendo a operação da rede suportada na suite 802.15.4 em modo sem coordenação (ad-hoc). A rede é vista como um grafo que deve assegurar, após organização da rede, conectividade e cobertura de modo a que todos os nós possam enviar eventos para as estações base e receber comandos provenientes das mesmas.

O MINSENS++ tira partido da utilização de múltiplas estações base no encaminhamento. As estações base têm de ter a capacidade de comunicar entre si, podendo esta

comunicação ser como no resto da rede (IEEE 802.15.4) ou através de meios de comunicação próprios para o efeito (IEEE 802.11, por exemplo). Em ambos os casos assume-se que a comunicação é assíncrona e não necessariamente fiável (possuindo uma natureza *best effort* ao nível do envio e receção de mensagens). Consideram-se dois possíveis cenários de encaminhamento entre estações base, de modo a diminuir o risco de falhas na comunicação:

- O encaminhamento entre estações base é one-hop
- Se o encaminhamento é multi-hop, é necessário que a rede, sobre a qual o encaminhamento se realiza, tenha condições de tolerância a intrusões.

4.2 Modelos de Adversário e de Falhas

Considera-se um modelo de falhas assíncronas e independentes, quer nos nós internos da rede como nas estações base. De acordo com o modelo de falhas, consideram-se falhas arbitrárias ou bizantinas, que são toleradas pelas soluções desenvolvidas.

No que diz respeito ao modelo de adversário, assume-se que existe a hipótese de captura física e intrusão de todos os nós da rede (sensores e/ou estações base), com capacidade de introdução de comportamento malicioso e possível captura de informação de segurança (incluindo chaves e segredos criptográficos). A capacidade deste tipo de ataques por intrusão viabiliza a introdução de comportamentos desviantes da especificação do protocolo que, na prática, equivalem à introdução pro-ativa, pelo atacante, de falhas bizantinas no sistema.

Não se considera, no entanto, nas hipóteses do adversário eventuais ataques do tipo DoS, nomeadamente realizados por exploração de comportamento malicioso no processamento da comunicação IEEE 802.15.4 (como se encontra descrito na secção 2.6).

4.3 Modelo de Software

O modelo de software utilizado é o referido na secção 2.2. Na camada física considera-se que os nós são do tipo MicaMotes, modelo de sensores suportado no simulador WiSeNet. Na camada MAC conta-se com a interface de comunicação 802.15.4, com CSMA/CA, sendo que é necessário que esta camada ofereça serviços de comunicação segura e estabelecimento seguro e tolerante a intrusões de chaves criptográficas.

A camada em que nos focamos mais é a camada de rede, onde se insere o protocolo de encaminhamento, o MINSENS++. Esta camada oferece um serviço de encaminhamento tolerante a falhas e intrusões, com auto-organização dos nós e descoberta de rotas.

Por fim há a camada de aplicação, onde é possível desenvolver software que utilize as primitivas oferecidas pela camada de rede para disseminar informação.

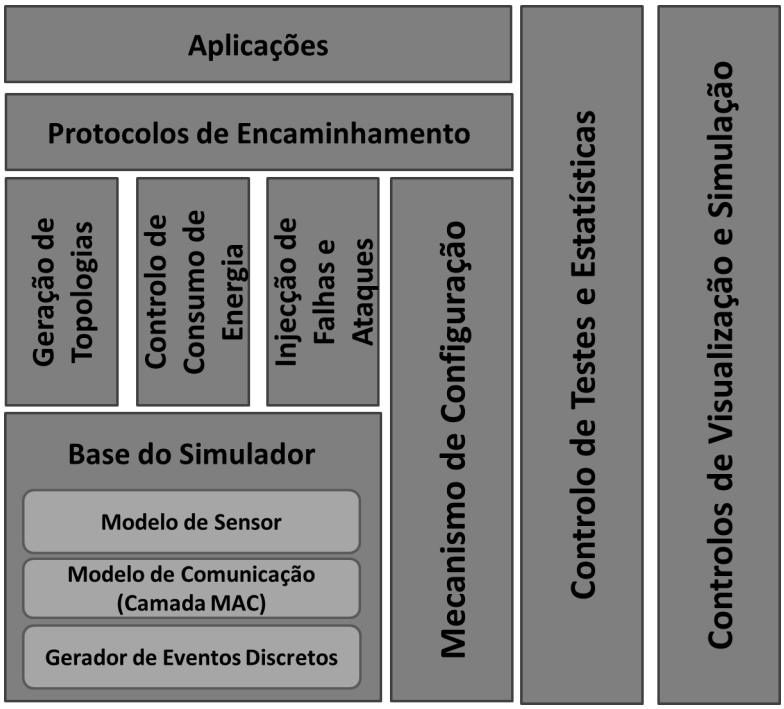


Figura 4.1: Arquitetura do simulador WiSeNet

4.4 WiSeNet: Arquitetura do Simulador

O simulador escolhido para implementar e testar o MINSENS++, o WiSeNet, segue uma arquitetura própria, que se apresenta na figura 4.1, onde se representam os principais serviços oferecidos pelo simulador.

A camada que representa a Base do Simulador é uma extensão ao simulador Jprowler. Esta camada contém os modelos base de sensores de RSSFs, tais como os modelos de comunicação e de MAC.

O Mecanismo de Configuração permite definir, guardar e reutilizar a informação de configuração do simulador e das simulações, tais como os parâmetros base do simulador, topologias de rede ou parâmetros dos testes realizados. As configurações são guardadas em ficheiros XML, o que permite que se repitam simulações exatamente com as mesmas condições.

O módulo de Geração de Topologias oferece uma interface gráfica que permite definir topologias de RSSFs. A interface permite criar rapidamente uma topologia aleatória dentro de um determinado espaço e com um dado número de nós, ou uma topologia em grelha, em que cada nó está separado dos seus vizinhos por um determinado espaço, definido pelo utilizador. Para além da colocação dos nós, este módulo permite definir que nós se pretende que atuem como estações base na simulação.

O Controlo de Consumo de Energia é um dos módulos mais importantes do simulador, uma vez que permite fazer a avaliação do consumo energético do funcionamento

do protocolo de encaminhamento utilizado, de uma forma transparente ao utilizador. A avaliação do consumo energético é de extrema importância, sendo a energia um dos recursos mais limitados nas RSSFs e o que define o tempo de vida útil da rede.

O módulo de Injeção de Falhas e Ataques permite introduzir nos nós pretendidos um comportamento diferente do observado nos restantes nós da rede. Esta capacidade do simulador permite fazer avaliações dos protocolos de encaminhamento na presença de nós intrusos.

Na camada de Protocolos de Encaminhamento é onde se insere a implementação realizada do MINSENS++, tal como dos protocolos INSENS e MINSENS. Nesta camada insere-se qualquer protocolo de encaminhamento que se pretenda desenvolver para ser testado neste simulador.

A camada de Aplicação é onde se inserem as aplicações a testar no simulador, sobre os protocolos de encaminhamento desenvolvidos.

O módulo de Controlo de Testes e Estatísticas trata da realização dos testes aos protocolos de encaminhamento e da recolha dos dados relevantes para a avaliação dos protocolos. Os testes podem ser realizados em condições normais na rede ou na presença de nós sujeitos a falhas e/ou intrusões. Este módulo recolhe dados que permite revelar dados estatísticos referentes à cobertura da rede, fiabilidade na entrega de mensagens e latência desde a origem até ao destino das mensagens.

O módulo de Controlo de Visualização e Simulação oferece toda a interface visual para que o utilizador possa criar as suas simulações intuitivamente. Facilita a introdução dos parâmetros da simulação assim como dos testes. Permite visualizar os resultados dos testes, uns em formato numérico e outros em gráfico.

5

MINSENS++

O MINSENS++ é um protocolo de encaminhamento para RSSFs *multi-hop*, que utiliza múltiplas rotas disjuntas de cada nó para múltiplas estações base ou nós agregadores. A rede é auto-organizável e suportam-se redes de grande escala.

Como já foi referido, o MINSENS++ é uma extensão ao protocolo MINSENS, e os seus objetivos são: aumentar a fiabilidade da rede ao introduzir um mecanismo de consenso dos dados recebidos, a ser realizado pelas estações base; estender o modelo de falhas e de adversário às estações base, sendo para isso necessário utilizar protocolos de consenso adaptados a RSSFs e com propriedades de tolerância a falhas e intrusões.

O protocolo MINSENS++ passa por três fases: (1) descoberta de rotas; (2) estabelecimento de rotas; (3) encaminhamento de dados.

Na fase de descoberta de rotas cada nó trata de se dar a conhecer à sua vizinhança, ao mesmo tempo que toma conhecimento dos nós seus vizinhos. Após algum tempo de reconhecimento de vizinhança, cada nó transmite a informação da sua vizinhança para as estações base. Por fim, cada estação base calcula múltiplas rotas disjuntas desde cada nó até si, com base na informação da vizinhança fornecida pelos nós. Depois de geradas as rotas, inicia-se a fase de estabelecimento de rotas.

A fase de estabelecimento de rotas inicia-se quando as estações base transmitem a cada nó da rede a sua tabela de encaminhamento. As tabelas de encaminhamento são enviadas por ordem de distância do nó à estação base. Depois de todos os nós receberem as suas tabelas de encaminhamento, a rede está preparada para a disseminação de dados, iniciando-se assim a fase de encaminhamento de dados.

Na fase de encaminhamento de dados os nós emissores enviam dados para as estações base. De modo a verificar que não existem erros nos dados transmitidos, todas as estações base para onde uma mensagem é enviada têm de consensualizar os dados

recebidos na mensagem. É nesta fase que se insere a maior contribuição da presente dissertação, o protocolo de consenso MVC-WSN (ver secção 6).

5.1 Descoberta de Rotas

No inicio do protocolo, cada estação base b_i faz *broadcast* de uma mensagem de *route request*, com o formato $< RREQ, bsId >$, que se propaga pela rede por epidemia. Ao receber a mensagem $< RREQ, bsId >$ um nó n_i verifica se já recebeu essa mensagem. Se a mensagem é nova, o emissor é adicionado ao conjunto de vizinhos V_i, e é definido como "pai", de modo a transmitir a informação da vizinhança para a estação base, com identificador único $bsId$, pelo caminho inverso pelo qual a mensagem de $RREQ$ chegou. Caso a mensagem seja repetida, apenas se adiciona o emissor a V_i. Algum tempo após a receção da mensagem, n_i envia para a estação base a sua informação de vizinhança, pelo caminho inverso, como já foi referido. É enviada pelo nó n_i, para a estação base, uma mensagem de *route feedback* do tipo $< FDBK, i, neighbors >$.

As estações base acumulam as informações de vizinhança dos nós de modo a ter informação suficiente para calcular as rotas. Em cada estação base são geradas rotas disjuntas ao estilo do INSENS, no entanto é necessário que todas as rotas geradas por todas as estações base sejam disjuntas. Por esse motivo realiza-se um consenso de rotas entre todas as estações base, tal como descrito na secção 6.6. Ao serem aceites, as rotas são guardadas na tabela de encaminhamento que será enviada para o nó de origem da rota.

5.2 Estabelecimento de Rotas

Quando as tabelas de encaminhamento estão preparadas nas estações base é necessário distribuí-las para os respetivos nós. A distribuição é feita por ordem de distância da estação base ao nó. É feito deste modo para que sejam utilizadas as tabelas de encaminhamento dos nós mais próximos das estações base, para encaminhar as tabelas de encaminhamento para os nós mais distantes. A estação base manda uma mensagem de *route update* do tipo $< RUPD, bsId, nodeId, routingTable >$, em que $bsId$ e $nodeId$ são os identificadores únicos da estação base e do nó destino, respetivamente, e $routingTable$ é a tabela de encaminhamento com as rotas disjuntas com origem no nó $nodeId$ e destino na estação base $bsId$. A tabela de encaminhamento é cifrada com a chave simétrica partilhada com o nó destino da mensagem, de modo a garantir que apenas esse nó possa aceder aos seus conteúdos.

Um nó guarda na sua tabela de encaminhamento final tabelas de encaminhamento para todas as estações base. Ao receber uma mensagem de *route update*, um nó adiciona a tabela de encaminhamento, para a estação base emissora da mensagem, à sua tabela de encaminhamento final. Caso a mensagem não lhe seja destinada, o nó verifica se tem na sua tabela de encaminhamento um caminho até à estação base emissora da mensagem,

proveniente do nó destino da mensagem. Se tem, a mensagem é retransmitida para continuar o seu caminho. O comportamento do nó pode ser observado no protocolo 1.

Depois de ter a sua tabela de encaminhamento completa, i.e, com tabelas de encaminhamento para todas as estações base, um nó declara-se como estável e pronto para tratar do encaminhamento de dados na rede.

Input: o identificador do nó $nodeId_i$

$routingTable_i \leftarrow \emptyset$

when $m =< RUPD, bsId, nodeId, routingTable >$ *is received* **do**
  if $nodeId = nodeId_i$ **then**
    $routingTable_i \leftarrow routingTable_i \cup routingTable$;
  else
    if $routingTable_i$ *contains route to* $nodeId$ **then**
      broadcast(m);
    end
  end
end

Protocolo 1: Protocolo de tratamento de mensagens RUPD no nó $nodeId_i$ da rede

5.3 Encaminhamento de Dados

No MINSENS++ divide-se o encaminhamento de dados em duas fases distintas: (1) encaminhamento normal entre os nós da rede; (2) consenso dos dados nas estações base. A primeira fase trata de encaminhar os dados, pelas rotas estabelecidas, desde o nó origem até às estações base de destino. Na segunda fase, cada estação base deve tratar da receção das várias réplicas da mesma mensagem por diferentes rotas e depois realizar o protocolo de consenso sobre os dados recebidos.

5.3.1 Encaminhamento na Rede

Os nós emissores capturam dados e transmitem-nos para as estações base. A estrutura geral das mensagens de dados é $< DATA, id, s, d, routes, r, data >$, sendo id o identificador único da mensagem, s o identificador único do nó de origem, d o identificador da estação base de destino, $routes$ o conjunto de rotas que a mensagem deve seguir, r a rota que a própria mensagem está a percorrer e d os dados cifrados. As mensagens são autenticadas por MAC, que serve também de controlo de integridade do conteúdo.

Existem cinco modos de encaminhamento no MINSENS++: (1) encaminhamento por apenas uma rota aleatória; (2) encaminhamento por apenas uma rota num sistema de *round-robin* (uma rota de cada vez); (3) encaminhamento por k rotas aleatórias; (4) encaminhamento por k rotas aleatórias, balanceadas por estação base; (5) encaminhamento

por todas as rotas.

Ao encaminhar dados apenas por uma rota, os dados chegam a uma única estação base. Se os dados são encaminhados por várias rotas, pode acontecer uma de duas situações: (1) são encaminhados por várias rotas para a mesma estação base; (2) são encaminhados por múltiplas rotas para diferentes estações base, sendo possível encaminhar por uma ou várias rotas para a mesma estação base.

O encaminhamento de dados (ver o protocolo 2) inicia-se quando um nó emissor está preparado para enviar uma nova mensagem m, fazendo *broadcast* da mesma. Um nó que recebe a mensagem verifica se tem na sua tabela de encaminhamento uma das rotas pelas quais a mensagem deve ser encaminhada e, se tal se verifica, a mensagem é retransmitida. De modo a reduzir o número de mensagens em circulação, cada nó tem um registo M_i das mensagens que já passaram por si, sendo que uma mensagem só é retransmitida se for nova. Quando uma mensagem chega a uma estação base inicia-se a próxima fase do encaminhamento, descrita na secção seguinte.

Input: o identificador do nó $nodeId_i$; a tabela de encaminhamento rt_i; o modo de encaminhamento rs_i

when *has data ready to send* **do**
- $msgId \leftarrow$ get new unique message identifier;
- $routes \leftarrow$ get routes for rs_i from rt_i;
- **foreach** $r : r \in routes$ **do**
  - $d \leftarrow$ destination of r;
  - broadcast($< DATA, msgId, nodeId_i, d, routes, r, data >$);
- **end**

end

when $m =< DATA, id, s, d, routes, r, data >$ *is received* **do**
- **if** $nodeId_i = d$ **then**
  - handle reception of m;
- **else**
  - **if** $\exists route \in rt_i : route$ *identifier* $= r$ **then**
    - broadcast(m);
  - **end**
- **end**

end

Protocolo 2: Protocolo de encaminhamento de mensagens de dados na rede

5.3.2 Encaminhamento de Dados nas Estações Base

O tratamento que as estações base dão às mensagens recebidas depende do modo como foram encaminhadas na rede. Nos casos em que uma mensagem é encaminhada apenas

por uma rota para uma estação base, a estação base em questão valida a mensagem e, se for válida, entrega-a à aplicação. Aqui a validação passa pelo controlo de autenticação e integridade da mensagem. Caso uma mensagem tenha sido encaminhada por várias rotas, o tratamento é mais complexo e depende do número de rotas e estações base envolvidas no encaminhamento.

Quando uma mensagem chega a uma estação base bs_i, proveniente de um número r de rotas ($r > 1$), bs_i guarda as diferentes réplicas das mensagens em R_i. Quando R_i contém mais de $\frac{r}{2}$ mensagens com o mesmo valor nos dados, a mensagem é considerada válida. A partir deste momento há duas possibilidades: (1) a mensagem é destinada apenas a bs_i e, nesse caso, depois de validada é entregue à aplicação; (2) a mensagem foi enviada para diferentes estações base, por isso é necessário realizar o consenso dos dados validados.

O processo de consenso de dados está descrito em pormenor na secção 6.5. O importante a reter nesta fase é que, se as estações base entram em acordo em relação a um valor dos dados da mensagem, a mensagem é considerada válida e entregue à aplicação, em todas as estações base corretas. Caso não se chegue a consenso, a mensagem é descartada.

Os dados da mensagem são cifrados pelo protocolo, por isso, antes de serem entregues à aplicação, os dados da mensagem são decifrados e entregues em claro.

Input: identificador da estação base bs_i; tabela de encaminhamento da estação base rt_i; gestor de réplicas R_i

when $m =< DATA, id, s, routes, r, d >$ *is received* **do**
  if m *is valid* $\wedge r \in rt_i$ **then**
    $R_i \leftarrow R_i \cup m$;
    if R_i *has enough replicas* **then**
      $participants \leftarrow$ get target base stations from R_i;
      $msg \leftarrow$ get replica with most received value from R_i;
      if $participants$ *size* $= 1$ **then**
        deliver msg to the application;
      else
        start consensus for msg;
      end
    end
  end
end

Protocolo 3: Protocolo de receção de mensagens de dados na estação base bs_i

6

MVC-WSN: Protocolo de Consenso Multi-Valor

Decidiu-se dar o nome de MVC-WSN ao protocolo de consenso multi-valor, probabilístico e assíncrono desenvolvido durante a presente dissertação. O protocolo pretende ser tolerante a falhas e intrusões, na presença de até f processos "maliciosos", sendo $f < \frac{n}{3}$. Por ser assíncrono assume-se que as comunicações entre processos estão sujeitas a falhas. O consenso multi-valor consiste em realizar acordos sobre valores de qualquer tipo, para além de booleano (o problema de acordos de valores booleanos é resolvido por protocolos de consenso binário), desde que esse valor seja convertível para um *array* de bytes (devido à transmissão dos dados pela rede).

- **Terminação:** Eventualmente k processos decidem um valor.
- **Acordo:** Todos os processos corretos decidem o mesmo valor.
- **Validade:** Se todos os processos corretos propõem v, então os processos corretos que decidem, decidem v.

Este protocolo é, na verdade, constituído por uma pilha de protocolos que contém: PEB (*Probabilistic Echo Broadcast*), Turquois (protocolo de consenso binário aleatório) e um protocolo de consenso multi-valor (*multi-valued consensus*) baseado no protocolo de consenso multi-valor da pilha RITAS, referido em 3.3.4. Aos protocolos Turquois e *multi-valued consensus* foi adicionado um mecanismo de recuperação de nós atrasados. Todos estes componentes serão apresentados em detalhe nas próximas secções, sendo que a ordem pela qual surgem está relacionada com o grau de independência em relação aos outros

protocolos.

6.1 PEB: Difusão Eco Probabilística

Verificou-se durante a idealização do protocolo MVC-WSN que a camada de consenso multi-valor que se pretendia utilizar necessitava de ter disponíveis duas primitivas de comunicação *broadcast*: *Echo Broadcast* e *Reliable Broadcast*. Ambas as primitivas, cujas funcionalidades serão explicadas de seguida, exigem que todos os processos envolvidos no *broadcast* sejam corretos, algo que vai contra a propriedade de tolerância a falhas e intrusões que se pretende. Para além disso, são primitivas que utilizam TCP e IPSec na comunicação, protocolos não disponível nas RSSFs.

Por esse motivo, teve de se pensar numa solução que permitisse a existência de falhas e intrusões nos processos e também falhas de comunicação. A solução que surgiu foi o PEB. Antes de passar à especificação do protocolo PEB, faremos uma descrição breve dos protocolos *Echo Broadcast* e *Reliable Broadcast*.

6.1.1 Difusão Eco

A primitiva de *echo broadcast* garante que **todos os processos corretos que entregam uma mensagem, entregam a mesma mensagem**. O protocolo inicia com o emissor a fazer *broadcast* de uma mensagem do tipo $(INIT, m)$ para todos os processos, sendo m a mensagem a entregar. Quando um processo recebe esta mensagem, envia por *broadcast* a mensagem $(ECHO, m)$. Cada processo espera até receber $\frac{n+f}{2}$ mensagens $(ECHO, m)$ para entregar m.

6.1.2 Difusão Fiável

Esta primitiva surge como uma extensão ao *echo broadcast*. É mais forte, mas também menos eficiente e garante uma nova propriedade: **se o emissor é correto então a mensagem é entregue por todos os processos corretos**.

As duas primeiras rondas de comunicação são semelhantes às do *echo broadcast*. O emissor faz *broadcast* de uma mensagem $(INIT, m)$. Ao receber essa mensagem, o processo envia uma mensagem $(ECHO, m)$ para todos os processos. De seguida espera até receber mais de $\frac{n+f}{2}$ mensagens $(ECHO, m)$ ou $f + 1$ mensagens $(READY, m)$, sendo que ao verificar-se uma destas condições o processo faz *broadcast* de uma mensagem $(READY, m)$. Finalmente, o processo espera até receber $2f + 1$ mensagens $(READY, m)$ para entregar m.

6.1.3 PEB: Descrição do Protocolo

As primitivas de *echo broadcast* e *reliable broadcast* necessitam de TCP/IP na entrega das mensagens, algo que não é viável em RSSFs. Por esse motivo, foi necessário desenvolver um meio de emissão de mensagens adequado às características das RSSFs, com propriedades probabilísticas na entrega, o *probabilistic echo broadcast*. O protocolo garante que k processos corretos que eventualmente entregam uma mensagem, entregam a mesma mensagem.

Input: a mensagem a enviar m; o período entre transmissões sucessivas, em milissegundos, p; o número máximo de transmissões max
Output: valor booleano que define se o protocolo foi ou não concluído $finished$

$received \leftarrow \emptyset$;
$sends \leftarrow 0$;

Task Sender:
while $sends < max$ **do**
  broadcast($< INIT, m >$);
  $sends \leftarrow sends + 1$;
  sleep(p);
end

Task Receiver:
when $m_i =< t_i, msg_i >$ *is received* **do**
  if t_i *is* $INIT$ **then**
    broadcast($< ECHO, msg_i >$);
  end
  else if t_i *is* $ECHO$ **then**
    $received \leftarrow received \cup \{m_i : mi$ is valid $\}$;
    if $|\{< t, msg > \in received : t \text{ is } ECHO\}| > \frac{n+f)}{2}$ **then**
      deliver(msg);
    end
  end
end

Protocolo 4: Protocolo PEB

A solução encontrada baseia-se no protocolo *echo broadcast*, tendo sido introduzido um broadcaster periódico de mensagens. O *broadcaster* periódico recebe como parâmetros de entrada (m, p, max). A função deste componente é enviar periodicamente a mensagem m para todos os processos, sendo que o período entre sucessivas transmissões é definido por p (em milissegundos). Para limitar o número de envios do *broadcaster*, m só é enviado até max vezes.

O PEB (ver Protocolo 4) é baseado no protocolo *echo broadcast*, contém as mesmas rondas de comunicação, estando as grandes diferenças ao nível da emissão de mensagens

e requisitos para o funcionamento do protocolo. Assume-se ainda assim que as mensagens estão autenticadas e possuem controlo de integridade. No caso do MINSENS++ estas propriedades são asseguradas pelo encaminhamento de mensagens na rede. Caso as mensagens sejam transmitidas por canais de comunicação exteriores à rede, a confidencialidade, integridade e autenticação da mensagem devem ser asseguradas pela aplicação que utiliza o PEB.

6.2 Recuperação de Nós Atrasados

Devido ao modelo de comunicação assíncrona das RSSFs, é natural que a mesma mensagem ao ser transmitida para diferentes estações base, chegue a cada um dos seus destinos com atrasos temporais significativos, o que impede que todos os processos do consenso iniciem o processo de consenso ao mesmo tempo.

Quando se chegou à fase de testes dos protocolos de consenso tal como estavam idealizados, verificou-se que a percentagem de processos de consenso terminados por todos os participantes do consenso era muito reduzida, na ordem dos 40% de consensos concluídos por todos os processos, nos casos em que se utilizam cinco estações base. Este resultado não foi satisfatório, no entanto foi possível observar que o problema devia-se ao facto de os processos que entravam no consenso, quando este já estava a decorrer entre outros participantes, não conseguirem receber as mensagens das primeiras fases do consenso, sendo assim impossível participarem de forma ativa no consenso. É possível que um nó chegue atrasado, uma vez que os protocolos de consenso apenas necessitam de $\frac{n+f}{2}$ nós para iniciarem.

De modo a resolver o problema dos nós atrasados, desenvolveram-se mecanismos de recuperação de atrasados. Para tal, cada participante no consenso guarda as mensagens enviadas, em cada fase do protocolo, no conjunto L_i. Quando um processo recebe uma mensagem com o valor da fase inferior à sua fase atual, este reenvia a mensagem correspondente à fase da mensagem recebida, de maneira a que o nó que enviou essa mensagem possa recuperar e acompanhar o consenso (ver algoritmo 5).

As mensagens dos protocolos de consenso passam a ter um campo adicional, um valor booleano *isResend*, que define se a mensagem é um reenvio ou não. Uma mensagem só é reenviada se for em resposta a uma mensagem não-reenviada ou nova, para assegurar que os protocolos não re-enviam mensagens indefinidamente.

$L_i \leftarrow \emptyset$;

when m *is sent* **do**
  $L_i \leftarrow L_i \cup m$;
end
when m *is received* **do**
  if m *is not re-send* $\wedge m$ *phase* $<$ *current phase* **then**
    broadcast($msg \leftarrow \{msg \in L_i : msg$ phase $= m$ phase$\}$);
  end
end

Protocolo 5: Funcionamento Genérico de um Recuperador de Atrasos

6.3 Turquois

O Turquois é um protocolo de consenso binário aleatório, que permite que k processos de um total de n chegue a consenso de um valor binário $v \in \{0, 1\}$ (ver Protocolo 6). A correção do protocolo é garantida desde que o número de processos Bizantinos esteja limitado por $f < \frac{n}{3}$.

Input: valor booleano com a proposta inicial, $proposal_i \in \{0, 1\}$; o período entre transmissões sucessivas, em milissegundos, p; o número máximo de transmissões max
Output: valor booleano com o valor decidido após consenso, $decision_i \in \{0, 1\}$

$\phi_i \leftarrow 1$;
$v_i \leftarrow proposal_i$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;
$sends \leftarrow 0$;

Task Sender:
while $sends < max$ **do**
  broadcast($< i, \phi_i, v_i, status_i >$);
  $sends \leftarrow sends + 1$;
  sleep(p);
end

Task Receiver:
funcionamento é igual ao do Turquois original

when ϕ_i *changes* **do**
  $sends \leftarrow 0$;
end

Protocolo 6: Turquois

Cada processo tem um estado interno composto por três variáveis: (1) a fase $\phi_i \geq 1$; (2) o valor proposto $v_i \in \{0, 1\}$, (3) o estado da decisão $status_i \in \{decided, undecided\}$. Os processos iniciam o protocolo com $\phi_i = 1, status_i = undecided$ e o valor inicial v_i

define-se pelo valor dado como *input* do protocolo, $proposal_i$.

O protocolo funciona em ciclos de três fases, CONVERGE, LOCK e DECIDE. Um processo está em cada uma destas fases quando se verifica, respetivamente, $\phi_i \bmod 3 = 1$, $\phi_i \bmod 3 = 2$, $\phi_i \bmod 3 = 0$. Na fase CONVERGE os processos tentam convergir para o valor proposto mais observado entre os participantes. Na fase LOCK os processos assumem um valor $v \in \{0, 1\}$ ou $\perp$ caso o processo não seja capaz de decidir. Por fim, na fase DECIDE, os processos tentam decidir o valor que assumiram na fase anterior. Caso não se chegue a uma decisão, cada processo calcula um valor aleatório para iniciar um novo ciclo. A escolha aleatória de novos valores dá garantias de que eventualmente os processos irão decidir um valor.

As tarefas *Sender* e *Receiver* correm em paralelo. O *Sender* do Turquois original inspirou o *Broadcaster* do protocolo PEB (ver secção 6.1.3). No entanto, para aproximar o *Sender* do *Broadcaster*, foram feitas duas alterações: (1) a introdução de p que define o tempo em milissegundos entre envios sucessivos de mensagens; (2) a introdução de um limite de mensagens enviadas por fase, max. O *Receiver* é ativado quando uma mensagem é recebida. Todas as mensagens passam por um processo de validação, de modo a descartar mensagens que possam ter sido enviadas por processos Bizantinos. Nesse sentido, a validação tenta garantir que a mensagem foi enviada por um processo correto. As mensagens válidas são guardadas no conjunto V_i.

Existem dois fatores que provocam uma mudança de estado num processo:

1. Receber uma mensagem válida cujo valor de fase ϕ é maior que ϕ_i
2. V_i ter mais de $\frac{n+f}{2}$ mensagens com fase igual a ϕ_i

No primeiro caso, o estado de p_i passa a ser igual ao da mensagem. Existe uma situação excecional, que acontece quando p_i está na fase CONVERGE e v foi obtido aleatoriamente (através de lançamento de moeda), e que provoca um lançamento de moeda em p_i para determinar v_i.

O segundo caso é mais complexo e depende da fase atual de p_i. Na fase CONVERGE, o valor de v_i passa a ser igual ao valor encontrado na maioria das mensagens recebidas nessa mesma fase. Na fase LOCK v_i pode ser alterado de duas maneiras: (1) Se V_i contém mais de $\frac{n+f}{2}$ mensagens do tipo $< *, \phi, v, * >$, t.q. $\phi = \phi_i$, $v_i = v$ (2) caso não se verifique (1), v_i passa ter o valor $\perp$. Por fim, na fase DECIDE, $status_i$ passa a $decided$ se existe mais que $\frac{n+f}{2}$ mensagens do tipo $< *, \phi, v, * >$, t.q. $\phi = \phi_i$ e $v \neq \perp$. Caso contrário o valor de v_i vai ser dado pelo lançamento de uma moeda. Independentemente do que acontece, a fase actual, ϕ_i, é sempre incrementada uma unidade.

Em relação à validação das mensagens recebidas, são realizados dois tipos de validação: validação de autenticidade e validação semântica.

A validação de autenticidade trata-se de confirmar que uma mensagem foi realmente enviada pelo seu emissor declarado. No Turquois original esta validação é feita através da partilha de matrizes de segredos com todas as combinações *fase X valor proposto* possíveis. Uma vez que o MINSENS++ obriga os nós a partilharem chaves simétricas, o

mecanismo de autenticação de mensagens do Turquois foi substituído por autenticação de mensagens MAC, que é igualmente eficaz e evita que se realize uma ronda de disseminação segura de matrizes de segredos.

A validação semântica das mensagens assegura que os valores de estado transmitidos nas mensagens (ϕ, v, $status$) estão de acordo com o funcionamento correto do protocolo. Esta validação baseia-se no estado do processo p_i e nas mensagens válidas em V_i. É também possível fazer a validação semântica de forma explícita, enviando juntamente com a mensagem todas as mensagens que justifiquem os seus valores. Todas as mensagens enviadas em anexo são também validadas. O modo de validação semântica explícita adiciona uma grande carga de informação a ser transmitida, no entanto a sua utilização não é obrigatória.

Cada valor de estado enviado numa mensagem é avaliado separadamente, do ponto de vista semântico. No entanto, uma mensagem só é considerada válida se as três variáveis de estado forem válidas. Apenas as mensagens com valor de fase 1 não são sujeitas a validação. A validação semântica das variáveis de estado é explicada de seguida:

- **Valor de Fase (ϕ):** O valor ϕ de uma mensagem requer mais de $\frac{n+f}{2}$ mensagens em V_i com o formato $< *, \phi - 1, *, * >$ para ser considerado válido
- **Valor de Proposta (v):** O valor de proposta depende do valor de fase da mensagem.
  - *Mensagem com fase CONVERGE:* A validade de v neste caso depende se o valor foi obtido de forma determinística ou aleatória. Se foi de forma determinística, é necessário que V_i contenha mais de $\frac{n+f}{2}$ mensangens do tipo $< *, \phi-2, v, * >$. Se foi obtido aleatoriamente é necessário que V_i tenha mais de $\frac{n+f}{2}$ mensagens do tipo $< *, \phi - 1, \perp, * >$
  - *Mensagem com fase LOCK:* O valor v é válido se V_i contém mais de $\frac{\frac{n+f}{2}}{2}$ mensagens do tipo $< *, \phi - 1, v, * >$
  - *Mensagem com fase DECIDE:* Se $v \in \{0, 1\}$ então é necessário que V_i contenha mais de $\frac{n+f}{2}$ mensagens do tipo $< *, \phi - 1, v, * >$. No caso de v ser $\perp$ é necessário que V_i contenha mais de $\frac{\frac{n+f}{2}}{2}$ mensagens do tipo $< *, \phi - 1, 0, * >$ e $\frac{\frac{n+f}{2}}{2}$ mensagens do tipo $< *, \phi - 1, 1, * >$.
- **Valor de estado($status$):** Todas as mensagens com $\phi \leq 3$ têm de ter necessariamente $status = undecided$. Para as restantes mensagens, $status = decided$ necessita que V_i contenha mais de $\frac{n+f}{2}$ mensagens do tipo $< *, \phi, v, * >$, em que $\phi \text{mod} 3 = 0$. Uma mensagem com $status = undecided$ necessita que V_i contenha mais de $\frac{\frac{n+f}{2}}{2}$ mensagens do tipo $< *, \phi', 0, * >$ e $\frac{\frac{n+f}{2}}{2}$ mensagens do tipo $< *, \phi', 1, * >$, em que ϕ' é o maior valor que garanta $\phi' \text{mod} 3 = 2$, sendo inferior a ϕ

Até agora foi descrito o protocolo Turquois original, no qual se introduziu o mecanismo de recuperação de nós atrasados para atender às necessidades das RSSFs.

6.4 MVC

O *Multi-Valued Consensus*, ou MVC, permite que vários processos realizem consenso sobre um valor de tamanho arbitrário, $v \in \bigvee$. O valor decidido ou é um dos valores propostos ou um valor predefinido $\perp \ni \bigvee$.

O protocolo original recorre a camadas de *reliable broadcast* e *echo broadcast*, no entanto essas camadas não são viáveis em RSSFs, como já foi referido. Por esse motivo, foram substituídas pelo PEB nesta versão. Para além do PEB, o MVC requer também uma camada de consenso binário, sendo que o protocolo de consenso binário utilizado nesta implementação é o Turquois. O MVC inclui também o mecanismo de recuperação de nós atrasados, para fazer face às necessidades das RSSFs.

O protocolo (ver protocolo 7) inicia quando cada processo p_i dá a conhecer o seu valor proposto v_i, através da primitiva de *probabilistic echo broadcast*. A mensagem enviada tem o formato $< INIT, i, v_i >$. Cada processo espera até receber $n - f$ mensagens $INIT$ e guarda os valores recebidos em V_i. Se um processo recebe pelo menos $n - 2f$ mensagens com o mesmo valor v, faz um *probabilistic echo broadcast* da mensagem $< VECT, i, v >$. Caso não seja possível decidir v, é enviado o valor $\perp$. No protocolo original é enviado um vector de valores que justificam o valor escolhido, isto é, os valores recebidos na fase $INIT$. Uma vez que se realiza o PEB no envio de mensagens assume-se que os processos têm as mesmas mensagens da fase $INIT$ e, por isso, não é enviado o vector de valores de justificação, de modo a reduzir o tamanho das mensagens trocadas. No passo seguinte do protocolo o processo espera até receber $n - f$ mensagens $VECT$. Se foram recebidas $n - 2f$ mensagens $VECT$ com o mesmo valor proposto, então inicia-se a execução do consenso binário propondo o valor inicial 1. Caso não seja possível decidir um valor v, inicia-se o consenso binário com o valor inicial 0. Caso o resultado do consenso binário for 1 aceita-se o valor v. Se o resultado for 0 o valor decidido é $\perp$.

Input: valor com a proposta inicial, $proposal_i \in \bigvee$; identificador único do nó i
Output: valor decidido após consenso, $decision_i \in \bigvee$ ou $\perp$ se nenhum valor for decidido

$v_i \leftarrow proposal_i$;
$phase_i \leftarrow INIT$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;

Initialization:
probabilistic echo broadcast($< INIT, i, v_i >$);

Task Receiver:
when $m =< msgPhase, j, v_j >$ *is received* **do**
  $V_i \leftarrow V_i \cup \{m : m \text{ is valid }\}$;
  if $phase_i = INIT \wedge |\{< INIT, *, * >\in V_i\}| > n - f$ **then**
    if $|\{< INIT, *, v >\in V_i\}| > n - 2f$ **then**
      $v_i \leftarrow v$;
    else
      $v_i \leftarrow \perp$;
    end
    $phase_i \leftarrow VECT$;
    probabilistic echo broadcast($< VECT, i, v_i >$);
  else if $phase_i = VECT \wedge |\{< VECT, *, * >\}| > n - f$ **then**
    if $|\{< VECT, *, v >\}| > n - 2f$ **then**
      $binResult \leftarrow$ binary consensus(1);
    else
      $binResult \leftarrow$ binary consensus(0);
    end
    if $binResult = 1$ **then**
      $decision_i \leftarrow v$;
    else
      $decision_i \leftarrow \perp$;
    end
    $status_i \leftarrow decided$;
  end
end

Protocolo 7: MVC

6.5 Consenso de Dados

O consenso de dados trata de toda a preparação dos processos para iniciarem um consenso multi-valor. É necessário que mais de $\frac{n}{2}$ processos estejam preparados para iniciar o consenso de dados. Considera-se que um processo p_i está preparado quando tem ao seu dispor o valor v_i sobre o qual pretende realizar o consenso.

No MINSENS++, um processo está pronto para o consenso quando uma estação base recebe uma mensagem proveniente da rede. Quando isso acontece é enviada, por PEB, uma mensagem CREQ (*consensus request*) com o formato $< CREQ, i, msgId >$ para as estações base para onde a mensagem foi enviada. O valor $msgId$ contém o identificador único da mensagem recebida, de modo a distinguir as diferentes instâncias de consenso sobre diferentes mensagens que possam acontecer ao mesmo tempo. As mensagens CREQ recebidas por p_i são guardadas em V_i.

Quando um processo recebe mais de $\frac{n}{2}$ mensagens de CREQ para o mesmo $msgId$ dá-se inicio ao consenso multi-valor, neste caso o protocolo MVC, com o valor da proposta inicial a serem os dados da mensagem $msgId$.

6.6 Consenso de Rotas

Como foi referido anteriormente, no MINSENS++ todas as rotas são disjuntas, sendo que, uma vez que existem múltiplas estações base, é necessário que estas entrem em acordo em relação às rotas que vão utilizar.

Foi desenvolvido um módulo de consenso de rotas (ver Protocolo 8) que, dada uma rota gerada numa estação base, determina, através de um protocolo de consenso com as outras estações base, se a rota pode ou não ser utilizada. Este acordo tem de ser total, isto é, todas as estações base têm de concordar com a rota para que esta seja aceite. O protocolo é inspirado no PEB.

Após a geração de uma nova rota r_x numa estação base bs_i, dá-se início ao processo de consenso da rota. Inicialmente, bs_i envia uma mensagem com o formato $< ROUTE, bs_i, r_x >$ para as restantes estações base, utilizando o mecanismo de retransmissão periódica de mensagens, como no protocolo PEB. Ao receber uma mensagem $ROUTE$, uma estação base verifica se a rota recebida é válida, isto é, se é disjunta de todas as que já foram geradas até ao momento e faz *broadcast* de uma mensagem $< RECHO, bs_j, r_x, dec >$, em que dec é um valor booleano que contém a decisão tomada, *true* se a rota é válida ou *false* caso contrário. As estações base registam as mensagens $RECHO$ recebidas e, se não existirem votos contra, a rota é dada como aceite e, consequentemente, adicionada ao conjunto de rotas aceites.

Input: rota recém-gerada, r_x; identificador único da estação base bs_i; conjunto de rotas aceites na estação base, $routes_i$, conjunto com os identificadores das estações base participantes no consenso, $bsIds$

Output: valor booleano, $decision_x$, que contém $true$ se a rota for aceite e $false$ caso contrário

$echoes_i \leftarrow m =< RECHO, bs_i, r_x, true >$;

Initialization:
periodic broadcast($< ROUTE, bs_i, r_x >$);

Task Receiver:
when $m =< RECHO, bs_j, r_x, dec >$ *is received* **do**
  if $bs_j \in bsIds \wedge m \notin echoes_i$ **then**
    if *dec* **then**
      $echoes_i \leftarrow echoes_i \cup m$;
      if $echoes_i.size = bsIds.size$ **then**
        $routes_i \leftarrow routes_i \cup r_x$;
      end
    else
      r_x is discarded;
    end
  end
end

Protocolo 8: Consenso de Rotas

7

Implementação

Nesta secção descrevem-se os pormenores e especificidades da implementação dos protocolos referidos nas secções 5 e 6. Para além do MINSENS++ e do conjunto de protocolos de consenso desenvolvidos (MVC-WSN), foram implementados os protocolos INSENS e MINSENS, visto servirem de base para o trabalho desenvolvido.

O simulador WiSeNet foi desenvolvido em Java, sendo Java a linguagem na qual os protocolos de encaminhamento têm de ser implementados. O simulador oferece boas ferramentas de avaliação do desempenho de protocolos de encaminhamento, como as avaliações transparentes (para o utilizador) de fiabilidade, latência, cobertura da rede e gastos energéticos do protocolo. No entanto, para avaliar o funcionamento dos protocolos de consenso dentro da rede, foi necessário desenvolver um novo componente de avaliação capaz de recolher dados relativos aos consensos e produzir indicadores relevantes para a avaliação dos protocolos.

7.1 Camada de Rede no WiSeNet

O simulador WiSeNet oferece uma API para a camada de rede, onde se inserem os protocolos de encaminhamento. A funcionalidade base e API desta camada estão presentes na classe abstrata *RoutingLayer*.

A classe *RoutingLayer* implementa, nos seus métodos privados, os mecanismos que permitem fazer a avaliação dos protocolos de encaminhamento de modo transparente para o utilizador. Para além disso, esses mesmos métodos tratam da entrega de eventos de envio e receção de mensagens à base do simulador, assim como da gestão de comportamento de nós maliciosos. Para além dos métodos privados, a classe contém alguns

métodos abstratos a serem implementados em cada implementação de um protocolo específico, sendo que esses métodos constituem a API publica da camada de rede. A tabela 7.1 contém uma descrição da API publica e da funcionalidade de cada um dos métodos.

Quando se está a desenvolver um protocolo de encaminhamento é necessário implementar todos os métodos descritos, visto representarem as principais funcionalidades e responsabilidades que um protocolo de encaminhamento deve ter.

Para além da classe *RoutingLayer*, o simulador oferece também uma classe abstrata *Message*, classe que todas as classes que representam mensagens devem estender, e um utilitário *Timer*, que permite realizar tarefas periódicas.

Método	Descrição
void onReceiveMessage(Object message)	A implementação deste método deve descrever o funcionamento de um protocolo quando recebe uma mensagem vinda da camada MAC, normalmente mensagens enviadas por outros nós na rede. A mensagem é representada pelo objeto *message*.
void onSendMessage(Object message, Application app)	Este método representa a receção de uma mensagem vinda da camada de aplicação. O normal é que, depois de algum tratamento especial dado à mensagem (e.g. encriptação, assinatura digital, etc.), esta mensagem seja encaminhada para outros nós. Os objetos *message* e *app* representam a mensagem enviada e a aplicação que a enviou, respetivamente.
void onRouteMessage(Object message)	Quando é utilizada a primitiva *send* na camada de rede para fazer *broadcast* da mensagem *message*, antes de ser enviada, a mensagem pode ser afetada neste método. Aqui é onde se espera que o protocolo trate do encaminhamento da mensagem, isto é, defina o seu destino e, possivelmente, a rota que ela deve seguir.
void sendMessageDone()	Este método serve para adicionar algum comportamento que o protocolo deva ter logo após o *broadcast* correto de uma mensagem.
Message encapsulateMessage(Message m)	De certo modo, este método não representa nenhuma funcionalidade chave nem do simulador, nem de um protocolo de encaminhamento. No entanto, deve ser utilizado para encapsular uma mensagem vinda da aplicação numa mensagem específica do protocolo.
void onStartUp()	Quando uma simulação é iniciada e os nós começam a funcionar, este método é chamado e deve conter o comportamento de inicialização e preparação do protocolo.
void onStable()	Normalmente um nó passa a ser considerado estável, do ponto de vista do encaminhamento, quando está pronto a encaminhar dados. Neste método é possível introduzir comportamentos que o protocolo deve ter quando um nó passa a ser estável.
void startupAttacks()	O WiSeNet permite simular ataques a nós da rede. Esses ataques têm de ser implementados para serem utilizados. Neste método podem-se definir quais os ataques a que, os nós que correm o protocolo de encaminhamento, podem ser sujeitos.
void initAttacks()	Este método define o comportamento do nó quando um ataque é inicializado.

Tabela 7.1: API pública da camada de rede do WiSeNet

7.2 INSENS

A implementação do protocolo INSENS era necessária, uma vez que serve de base ao protocolo MINSENS e, consequentemente ao MINSENS++.

A implementação do protocolo INSENS introduz-se na camada de rede da pilha de *software*, como seria de esperar. O seu principal componente é a classe *INSENSRoutingLayer*, que estende *RoutingLayer* e, por isso, implementa a funcionalidade base do INSENS, tanto dos nós sensores como da estação base. No caso da estação base existe também um controlador, *BaseStationController*, que trata das funções específicas da estação base, como calcular rotas e criar tabelas de encaminhamento para os nós.

As mensagens trocadas pelo protocolo são do tipo *INSENSMessage*, que contém um *payload* que pode ter um de quatro tipos, todos eles descritos no funcionamento do MINSENS++ (secção 5): (1) *RREQPayload* que representa as mensagens de *route request*; (2) *FDBKPayload*, que representa as mensagens de *route feedback*; (3) *RUPDPayload*, representante das mensangens de *route update*; (4) *DATAPayload*, que representa as mensagens de dados, i.e. que transportam os dados na rede.

Quando uma aplicação envia uma mensagem para a camada de rede, a mensagem é guardada num *buffer* de mensagens, representado pelo objeto *List<Message> messagesQueue*, que é do tipo *LinkedList<Message>*. Na inicialização do protocolo é criado um *Timer* que transmite periodicamente a mensagem mais antiga, que se encontra em *messagesQueue*, para a camada MAC, que trata de fazer *broadcast* da mensagem.

Ao receber uma mensagem, o nó verifica qual o tipo do seu conteúdo, de modo a decidir o tratamento que faz à mensagem. Como foi referido na secção 5, dependendo do tipo de mensagem o comportamento do protocolo é diferente (exemplo no código 7.1).

O protocolo inicia-se na única estação base do protocolo, que envia uma mensagem de *route request*. Depois de enviada a mensagem é iniciado um *Timer* que define o tempo que a estação base vai esperar, pelas informações de vizinhança dos nós da rede, até iniciar o cálculo das rotas. Quando o *Timer* termina, a estação base inicia o cálculo das múltiplas rotas disjuntas de cada nó para a estação base, tal como está descrito em [20].

Depois da estação base calcular as rotas que vão ser utilizadas no encaminhamento de mensagens, organiza-as em tabelas de encaminhamento que serão entregues a cada nó. A tabela de encaminhamento é representada pela classe *INSENSForwardingTable*, que contém o identificador único do nó a que pertence como a origem, o identificador único da estação base como destino e um conjunto de *RoutingTableEntry*, representado pelo objeto *Set<RoutingTableEntry>* do tipo *HashSet<RoutingTableEntry>*. Cada *RoutingTableEntry* representa uma rota, tendo a informação de quem é o destino e a origem da rota e o do nó que antecede do próprio nó na rota. A classe *INSENSForwardingTable* tem o método boolean *haveRoute(short source, short destination, short immediate)* que verifica se existe uma rota com a origem, o destino e o nó antecessor, passados como argumento. Os nós sensores utilizam o resultado deste método para decidirem se continuam a encaminhar uma mensagem, caso tenham uma rota, ou se descartam a mensagem, caso não exista uma

```
1  private void processFDBKMessage(INSENSMessage m) throws INSENSException {
2    FDBKPayload payload = new FDBKPayload(m.getPayload());
3    if (Arrays.equals(myRoundMAC, payload.parent_mac)) {
4      if (getNode().isSinkNode()) { // if node is a base station
5        getBaseStationController().addFeedbackMessages(payload);
6        INSENSFunctions.decryptData(getNode(), payload.neighborInfo, null);
7      } else {
8        byte[] new_payload = modifiedParentMAC(payload);
9        send(new INSENSMessage(new_payload))
10     }
11   }
12 }
13
14 private void processDATAMessage(INSENSMessage m) {
15   if (isStable()) {
16     if (!itsForMe(m)){routeMessage(m);}
17     else {
18       DATAPayload payload = new DATAPayload(m.getPayload());
19       getBaseStationController().receiveReplica(payload);
20     }
21   }
22 }
```

Código 7.1: Exemplos de tratamento de mensagens no INSENSRoutingLayer

rota para os dados recebidos.

Ao receber a tabela de encaminhamento, um nó declara-se como estável e está pronto para produzir dados e encaminhar dados pela rede. O simulador oferece duas possibilidades diferentes no que diz respeito à criação de mensagens: (1) uma aplicação desenvolvida gera e envia as mensagens; (2) os testes do simulador geram mensagens de teste nos nós sensores.

Já na fase de encaminhamento, quando uma mensagem chega à estação base e esta verifica que a mensagem é válida, se a mensagem só foi encaminhada por uma rota é aceite e entregue à aplicação, mas se foi enviada por múltiplas rotas a mensagem passa para o controlador *BaseStationController*, que por sua vez passa a mensagem para o seu gestor de réplicas *ReplicaManager*.

A classe *ReplicaManager* (ver código 7.2) faz a gestão das várias réplicas de mensagens recebidas. Tem um mapa de réplicas do tipo *HashMap<Long,ReplicaInfo>*, em que a chave é o identificador único da mensagem e o valor associado é do tipo *ReplicaInfo*, que contém toda a informação necessária para se saber se a estação base já tem todas as réplicas de que precisa.

O gestor de réplicas oferece três métodos simples: (1) *boolean addReplica(DATAPayload dp)* que recebe uma mensagem de dados e adiciona-a à estrutura de réplicas; (2) *boolean hasEnoughReplicas(long msgId)* que verifica se já foram recebidas replicas suficientes para aceitar a mensagem com identificador *msgId*; (3) *DATAPayload getMostReceivedReplica(long msgId)* que devolve uma das réplicas da mensagem que contém o valor que foi mais recebido, ou *null* se ainda não foram recebidas réplicas suficientes. A classe *ReplicaInfo* está associada apenas a uma mensagem e contém a informação das suas réplicas

```
1  public class ReplicaManager {
2    private HashMap<Long, ReplicaInfo> replicas;
3
4    public ReplicaManager(){
5      replicas = new HashMap<Long, ReplicaInfo>();
6    }
7
8    public boolean addReplica(DATAPayload dp){
9      ReplicaInfo ri = replicas.get(dp.originalMsgId);
10     if(ri == null){
11       ri = new ReplicaInfo(dp.routeIds);
12       replicas.put(dp.originalMsgId, ri);
13     }
14     return ri.addReplica(dp);
15   }
16
17   public boolean hasEnoughReplicas(long msgId){
18     if(replicas.containsKey(msgId))
19       return replicas.get(msgId).hasEnoughReplicas();
20     return false;
21   }
22
23   public DATAPayload getMostReceivedReplica(long msgId){
24     if(replicas.containsKey(msgId))
25       return replicas.get(msgId).getOneReplica();
26     return null;
27   }
28 }
```

Código 7.2: Classe ReplicaManager

recebidas. Em cada mensagem de dados está o conjunto de rotas pelas quais a mensagem foi encaminhada, por isso é possível saber o número total de réplicas a receber. A classe tem duas estruturas do tipo *Set<Integer>*, uma onde guarda os identificadores de todas as rotas por onde devem chegar mensagens e outra onde se guardam os identificadores das rotas por onde as mensagens já chegaram. Todas as réplicas recebidas são guardadas, para ser possível determinar qual o valor mais recebido. Esta medida serve para eliminar réplicas que possam ter sido alteradas por nós maliciosos.

Quando o gestor de réplicas tem réplicas suficientes para aceitar a mensagem, a mensagem é aceite e entregue à aplicação, o que conclui o funcionamento do protocolo.

7.3 MINSENS

Como já foi referido, o MINSENS é uma extensão ao INSENS. O funcionamento é muito similar, sendo que o MINSENS utiliza múltiplas estações base em que cada uma delas é uma instância do INSENS. As principais alterações acontecem no encaminhamento de mensagens de dados, construção de tabelas de encaminhamento e na introdução de consenso para aceitação de dados e rotas.

Para o consenso, o MINSENS assume que as estações base comunicam entre si por

meios externos à rede e sem um protocolo específico. Na implementação do protocolo esta característica é concretizada através da entrega direta da informação, por chamada direta a métodos de cada estação base. Cada gestor *BaseStationController* tem referências para os objetos que representam as outras estações base, de modo a poder chamar diretamente os métodos que permitem proceder ao processamento do consenso.

No calculo de rotas foi adicionado um passo à funcionalidade anterior, do protocolo INSENS. Quando uma estação base termina a geração de uma nova rota envia-a para todas as outras estações base, que decidem, dependendo das rotas que já foram aceites, se a rota pode ser aceite ou não. Não se aceitam rotas não-disjuntas, i.e. nenhum nó pode ser nó intermédio de quaisquer duas rotas diferentes. Caso uma rota seja aceite, as estações base registam-na no seu conjunto de rotas aceites.

Em relação às tabelas de encaminhamento, foi necessário criar uma nova classe *MINSENSForwardingTable*, que contém um objeto *HashMap<Short,INSENSForwardingTable> ftables* que faz o mapeamento entre identificador de estação base e a tabela de encaminhamento para as rotas que se destinam a essa estação base. Os métodos para verificação de rotas são semelhantes aos referidos na implementação da tabela de encaminhamento do INSENS, acrescentando-se o passo de obter a *INSENSForwardingTable* referente à estação base para onde a mensagem está a ser encaminhada. Esta nova tabela de encaminhamento tem métodos que facilitam a obtenção de rotas dependendo do modo como o encaminhamento será feito (exemplos no código 7.3).

```
1  public SimpleEntry<Short,Integer> getRandomRoute(){
2    Random r = new Random();
3        int bsPos = r.nextInt(fTables.size());
4        SimpleEntry<Short,Integer> result = null;
5
6        for(Entry<Short, INSENSForwardingTable> e: fTables.entrySet()){
7      if(bsPos == 0){
8            Integer routeId = e.getValue().getRandomRoute();
9            result = new SimpleEntry<Short, Integer>(e.getKey(), routeId);
10         }
11         else{bsPos--;}
12       }
13       return result;
14 }
15
16 public Hashtable<Short,Set<Integer>> getAllRoutes(){
17       Hashtable<Short,Set<Integer>> routes = new Hashtable<Short,Set<Integer>>();
18       for(INSENSForwardingTable ft: fTables.values() ){
19         routes.put(ft.getBsId(),ft.getRouteIdsToBS());
20       }
21       return routes;
22 }
```

Código 7.3: Exemplos de obtenção de rotas na classe MINSENSForwardingTable

O modo como o encaminhamento deve ser feito é definido no código, definindo a constante *int FORWARDING_TYPE* para um dos valores definidos para os tipos de encaminhamento possíveis (ver código 7.4). Quando um nó decide enviar uma mensagem, pede à sua tabela de encaminhamento as rotas correspondentes ao tipo de encaminhamento a ser feito. Os identificadores únicos das rotas devolvidas pela tabela de encaminhamento são inseridos na mensagem de dados a enviar. O protocolo de encaminhamento é semelhante ao INSENS a partir deste ponto: os nós que recebem uma mensagem de dados para encaminhar verificam se são ponto de passagem de alguma das rotas presentes na mensagem e, se for, continua a encaminhar a mensagem, caso contrário descarta-a.

```
1 public static final int K = 3;
2 public static final int K_RANDOM_ROUTES = 0;
3 public static final int K_BALANCED_ROUTES = 1;
4 public static final int ONE_RANDOM_ROUTE = 2;
5 public static final int ONE_ROUND_ROBIN = 3;
6 public static final int ALL = 4;
7
8 public static final int FORWARDING_TYPE = 4;
```

Código 7.4: Selecção dos diferentes tipos de encaminhamento

A última alteração é na receção de mensagens de dados, em que é necessário verificar se as rotas por onde a mensagem viajou pertencem todas à estação base que recebe a mensagem ou não. Se todas as rotas pertencem à estação base recetora, o processo de receção é igual ao descrito para o INSENS. A diferença está quando algumas rotas são para outras estações base, o que torna necessária a realização de consenso de dados. O processamento é igual ao das mensagens apenas destinadas a uma estação base, mas no final ao invés de a mensagem ser imediatamente entregue à aplicação, passa primeiro pelo consenso e apenas se for aceite no consenso é entregue à aplicação. O consenso acontece aqui do mesmo modo que no consenso de rotas, por chamada direta dos métodos das outras estações base.

7.4 MINSENS++

A implementação do MINSENS++ dá seguimento ao que foi desenvolvido para o MINSENS. A grande diferença está nos protocolos de consenso, onde a comunicação deixa de acontecer por chamada direta de métodos das outras estações base e passa a ser feita através da passagem de mensagens entre as estações base, e são utilizados os protocolos de consenso descritos na secção 6.

À semelhança do que foi feito em relação à gestão de réplicas, foi implementado um gestor de instâncias de consenso, *ConsensusManager*. Este gestor guarda as instâncias de consenso iniciadas num mapa do tipo *HashMap<Long,ConsensusInstance>* e permite obter

uma instância ou todas (ver código 7.5).

```
1  public class ConsensusManager {
2    private HashMap<Long, ConsensusInstance> consensusInstances;
3
4    public ConsensusManager(){
5      consensusInstances = new HashMap<Long, ConsensusInstance>();
6    }
7
8    public boolean addConsensusInstance(ConsensusInstance ci){
9      if(!consensusInstances.containsKey(ci.getMsgId())){
10       consensusInstances.put(ci.getMsgId(), ci);
11       return true;
12     }
13     return false;
14   }
15
16   public boolean consensusInstanceExists(long msgId){
17     return consensusInstances.containsKey(msgId);
18   }
19
20   public ConsensusInstance getConsensusInstance(long msgId){
21     return consensusInstances.get(msgId);
22   }
23
24   public Collection<ConsensusInstance> getConsensusInstances(){
25     return consensusInstances.values();
26   }
27 }
```

Código 7.5: Classe ConsensusManager

A classe *ConsensusInstance* faz parte do componente de consenso de dados, explicado na próxima secção.

7.5 Consenso de Dados

A classe *ConsensusInstance* representa uma única instância de consenso de dados numa estação base. Trata da preparação das estações base para o inicio do processo de consenso e da entrega das mensagens de consenso que chegam à estação base.

Uma instância de consenso pode estar em uma de três fases distintas: (1) fase de preparação, que é a fase inicial e onde se prepara o processo para iniciar o consenso; (2) fase de consenso, durante o período de tempo em que o consenso está a ser realizado; (3) fase final, depois de o consenso ter terminado e quando é possível determinar o resultado final do consenso.

Quando um objeto do tipo *ConsensusInstance* é inicializado, é iniciado o processo de

preparação do consenso, descrito em 6.5. O envio e receção de mensagens de consenso, sejam mensagem $CREQ$ ou dos protocolos de consenso, acontece entre o controlador da estação base, *BaseStationController*, e a instância de consenso, *ConsensusInstance*. Quando uma estação base recebe uma mensagem de consenso, tem acesso aos objetos que representam instâncias de consenso. No entanto, isso não se verifica na comunicação em sentido inverso, pois a instância de consenso não possui referências para o controlador da estação base. Por esse motivo foi implementado uma hierarquia de classes destinadas à mediação da comunicação entre os dois intervenientes. Essa hierarquia, na implementação atual, é composta pela interface *CommunicationChannel* e pela classe *MINSENSChannel* (ver código 7.6). A classe *MINSENSChannel* implementa a mediação com a o controlador da estação base. Foi criada a interface *CommunicationChannel* para ser possível implementar outros métodos de comunicação, como comunicação TCP ou UDP, sem ter de alterar muitas linhas de código (na verdade apenas é necessário alterar a inicialização das variáveis do tipo *CommunicationChannel*).

Depois do *probabilistic echo broadcast* de mensagens $CREQ$, que corre num thread à parte da execução normal de *ConsensusIntance*, estar terminado conclui-se que é possível iniciar o protocolo de consenso MVC e, consequentemente, o protocolo Turquois, ambos vistos em detalhe nas próximas secções.

```
1  public interface CommunicationChannel {
2    public void broadcastConsensusMsg(Message m, Set<Short> destinations);
3    public void broadcastCREQMessage(byte[] creqPayloadBytes) throws IOException;
4  }
5
6  public class MINSENSChannel implements CommunicationChannel {
7    private MINSENSRoutingLayer rl;
8    private short myNodeId;
9    private long dataMsgId;
10
11   public MINSENSChannel(MINSENSRoutingLayer rl, short myNodeId, long dataMsgId)
         {
12     this.rl = rl;
13     this.myNodeId = myNodeId;
14     this.dataMsgId = dataMsgId;
15   }
16
17   public synchronized void broadcastConsensusMsg(Message m, Set<Short>
         destinations){
18     byte msgType = -1;
19     byte[] data = null;
20     try{
21       if(m instanceof TurquoisMessage){
22         TurquoisMessage msg = (TurquoisMessage) m;
23         msgType = ConsensusMsg.TURQUOIS_MSG;
24         data = msg.toByteArray();
25       }
26       else if(m instanceof MVConsensusMsg){
27         MVConsensusMsg msg = (MVConsensusMsg) m;
28         msgType = ConsensusMsg.MVCONSENSUS_MSG;
29         data = msg.toByteArray();
30       }
31       ConsensusMsg consMsg = new ConsensusMsg((Short) m.getSourceId(),
             destinations, myNodeId, dataMsgId, msgType, data);
32       rl.sendConsensusMessage(consMsg);
33     } catch (Exception e) {e.printStackTrace();}
34   }
35
36   public synchronized void broadcastCREQMessage(byte[] creqPayloadBytes) throws
         IOException{
37     rl.sendCREQMessage(creqPayloadBytes);
38   }
39 }
```

Código 7.6: Hierarquia de mediadores de comunicação CommunicationChannel e MINSENSChannel

7.6 Turquois

A implementação do Turquois é uma adaptação quase direta, para Java, do protocolo 6. Por isso, são focadas nesta secção as estruturas de dados utilizadas. É importante referir que esta implementação do Turquois utiliza *CommunicationChannel* para mediar a comunicação com a estação base.

Para guardar os participantes no consenso utiliza-se um conjunto com os identificadores de todos os participantes, do tipo *Set<Short>*. Para além do conjunto de participantes, faz-se o mapeamento entre o identificador do participante e a chave simétrica partilhada com esse participante, numa estrutura do tipo *HashMap<Short,byte[]>*.

De modo a implementar o conjunto de mensagens válidas aceites foi desenvolvida uma nova classe, *ValidMessagesList*. Depois de uma mensagem ser validada pelo protocolo, é adicionada ao conjunto de mensagens válidas. Nesta fase passa ainda por uma validação de modo a verificar se é uma mensagem nova ou não. Este conjunto tem métodos que facilitam a obtenção de dados relativos às mensagens recebidas, necessários para o Turquois (ver Tabela 7.2).

As tarefas de broadcaster e recetor de mensagens são executadas por *threads* distintas, de modo a não ser afetado o funcionamento normal do Turquois.

Método	Descrição
List<TurquoisMessage> getMsgsInPhase(int phase)	Dado um valor para a fase, retorna uma lista com todas as mensagens recebidas nessa fase ou uma lista vazia caso a fase não tenha sido iniciada pelo protocolo.
int getPhaseCounter(int phase)	Retorna o número de mensagens recebidas na fase $phase$.
int getValuePerPhaseCounter(int phase, int value)	Retorna o número de mensagens recebidas na fase $phase$, e cujo valor seja $value$.
int getMostReceivedValueInPhase(int phase)	Retorna o valor mais recebido na fase $phase$ ou -1 caso nenhum valor possa ser obtido.

Tabela 7.2: API pública da classe ValidMessagesList

7.7 MVC

Tal como no Turquois, a implementação do MVC é uma adaptação quase direta do protocolo 7. Mais uma vez, os pormenores mais importantes que se tiram desta implementação são as estruturas de dados utilizadas.

Para guardar as mensagens recebidas foi criada a estrutura *ReceivedMessagesList*, que para além de guardar as mensagens oferece métodos que facilitam a obtenção de dados relevantes para o protocolo MVC (ver Tabela 7.3)

Na classe *ReceivedMessagesList* a contagem de valores recebidos nas mensagens é feita com recurso a uma estrutura do tipo *Vector<HashMap<ByteArrayWrapper,Integer>*. O *Vector* ajuda a identificar e aceder facilmente os valores da fase do protocolo, enquanto que o *HashMap<ByteArrayWrapper,Integer>* (ver código 7.7) permite fazer o mapeamento entre valor recebido e o seu respetivo contador. Foi necessário criar a classe *ByteArrayWrapper*, que encapsula a tipo nativo *byte[]*. Esta decisão foi tomada porque a obtenção do *hashcode* de uma variável do tipo *byte[]* não depende do seu conteúdo, mas sim do identificador do objeto na JVM. Isso constitui um problema na utilização do tipo como chave de um *HashMap*. A classe *ByteArrayWrapper* resolve esse problema, permitindo obter um *hashcode* que depende unicamente do conteúdo de um *byte[]* e permite também que a comparação entre dois objetos do tipo *ByteArrayWrapper* tenha em conta o conteúdo dos *byte[]* que encapsulam.

O funcionamento do protocolo PEB, utilizado na emissão de mensagens, e a receção de mensagens acontecem em *threads* distintas, separadas do funcionamento do MVC. Quando chega a fase do MVC em que se recorre ao protocolo Turquois para realizar o consenso binário, a execução dos dois protocolos acontece separadamente, e é criado uma *thread* do tipo *BinConsensusResultTask*, que verifica se o consenso binário (Turquois, neste caso) já pode devolver um resultado e, se pode, obtém esse resultado, de modo ao protocolo MVC poder decidir o seu próprio valor de consenso.

Método	Descrição
int getValueCount(byte[] value, short phase)	Retorna o número de mensagens recebidas, que contêm o valor $value$, na fase $phase$ do protocolo.
int getNumMessagesInPhase(short phase)	Retorna o número total de mensagens recebidas na fase $phase$ do protocolo.
boolean allMessagesWithSameValue(short phase)	Verifica se todas as mensagens recebidas na fase $phase$ têm o mesmo valor.
byte[] getMostReceivedValueInPhase(short phase)	Retorna o valor mais recebido nas mensagens da fase $phase$.

Tabela 7.3: API pública da classe ReceivedMessagesList

```
1  public class ByteArrayWrapper {
2    private byte[] data;
3
4    public ByteArrayWrapper(byte[] b){data = b;}
5
6    public byte[] getData() {return data;}
7
8    public boolean equals(Object other){
9      return Arrays.equals(data, ((ByteArrayWrapper) other).getData());
10   }
11
12   public int hashCode(){return Arrays.hashCode(data);}
13 }
```

Código 7.7: Classe ByteArrayWrapper

7.8 Extensões ao Modelo de Avaliação do Simulador

Os mecanismos de avaliação e os indicadores obtidos pelo simulador são muito importantes para avaliar um protocolo de encaminhamento. No entanto, nenhum destes indicadores (latência, cobertura, energia e fiabilidade) permite fazer a avaliação dos protocolos de consenso. Por esse motivo, foi necessário desenvolver um novo componente de avaliação capaz de recolher e produzir informação relativa aos protocolos de consenso, concretizado na classe *StatsManager*.

A classe *StatsManager* foi implementada com recurso ao padrão de desenho *Singleton*, para garantir que apenas existe uma instância da classe em todo o sistema, o que permite que as diferentes estações base registem os seus respetivos dados no mesmo local. Por os mecanismos de consenso não serem uma característica de RSSFs, os mesmos não são considerados pelo WiSeNet, o que não permite fazer avaliações sobre o consenso com o mesmo grau de transparência que se realizam as restantes avaliações disponíveis no simulador. Por esse motivo, foram inseridos na classe *BaseStationController* os mecanismos que permitem fazer o registo de eventos relevantes para a avaliação do consenso.

Este novo módulo de avaliação de consenso permite registar as mensagens enviadas pelos nós sensores, as mensagens que dão origem ao início de processos de consenso e as mensagens para as quais os respetivos processos de consenso terminam com sucesso. O tempo em que cada um desses eventos acontece é registado para avaliar a percentagem de tempo ocupada por cada fase do encaminhamento de uma mensagem. Com os dados recebidos é possível determinar a percentagem de mensagens que chegam efetivamente às estações base e dessas, a percentagem que dá origem a processos de consenso e, por fim, a percentagem de consensos que terminam face aos que são iniciados. Para cada um dos indicadores referidos podem-se obter valores para apenas uma estação base, mais que k estações base ou todas as estações base.

De modo a apoiar o armazenamento de toda a informação foram desenvolvidas três classes distintas: *ReliabilityStats*, *CREQStats* e *TimeStats*.

- ***ReliabilityStats*:** apoia o registo de mensagens recebidas pela aplicação, nas estações base.

- ***CREQStats*:** tem uma função semelhante à da classe *ReliabilityStats*, mas apoia o registo da informação relativa à fiabilidade do processo de consenso. Permite obter a percentagem de consensos iniciados, tendo como base o número de mensagens que chegaram às estações base vindas da rede, e permite também obter a percentagem de *consensus requests* terminados com base no número de *consensus requests* iniciados. Os consensos terminados são registados em *ReliabilityStats*, como mensagens entregues.

- ***TimeStats*:** apoia o registo temporal dos eventos de envio e receção de mensagens e do inicio e final do processo de consenso. Estes dados permitem obter o tempo médio necessário em cada fase do encaminhamento de uma mensagem (encaminhamento na rede e consenso).

Na classe *StatsManager* faz-se o mapeamento entre o identificador de cada mensagem e os objetos *ReliabilityStats*, *CREQStats* e *TimeStats* correspondentes. *StatsManager* oferece uma interface pública que permite registar os eventos e obter os indicadores pretendidos, relativos à fiabilidade e a latência temporal da entrega de mensagens (ver Tabela 7.4).

Método	Descrição
boolean addSentMsg(long msgId)	Regista o envio da mensagem com identificador *msgId*.
void addReceived(long msgId, Set<Short> participants, short nodeId, boolean consensus, long consensusTime)	Regista a receção da mensagem com identificador *msgId*, quando esta é entregue à aplicação. O argumento *consensus* indica se a mensagem foi entregue depois de consenso ou não, pelo nó com identificador *nodeId*. Caso tenha existido consenso, *participants* deve conter os identificadores de todos os participantes no consenso e *consensusTime* o tempo de execução do consenso.
boolean addCreqStarted(long msgId, short nodeId, Set<Short> participants, long routingTime)	Regista o início do processo de consenso da mensagem com identificador *msgId*, que coincide com a sua chegada à estação base com identificador *nodeId*. *routingTime* deve ser o tempo de transmissão da mensagem desde a sua origem até que chega à estação base. *participants* contém os identificadores das estações base que participam no consenso.
boolean addCreqFinished(long msgId, short nodeId)	Regista o final do processo de pedido de consenso para a mensagem com identificador *msgId* na estação base com identificador *nodeId*.
int getNumReceivedMessages()	Retorna o número total de mensagens entregues à aplicação.
int getNumReceivedByAllMessages()	Retorna o número de mensagens que foram entregues à aplicação, por todos os nós a que a mensagem era destinada.
double getOneCopyReliabilityPercentage()	Retorna a percentagem de mensagens que foram entregues à aplicação em pelo menos um dos nós destino.
double getAboveThresholdReliabilityPercentage()	Retorna a percentagem de mensagens que foram entregues à aplicação por mais de $n - f$ nós, sendo n o número de nós destino da mensagem e f o número de falhas que se pretende tolerar.
double getAllCopiesReliabilityPercentage()	Retorna a percentagem de mensagens que foram entregues em todos os nós para onde eram destinadas.
String getMeanRoutingAndConsensusTimes()	Retorna uma String com o formato "$rt(rt\%)/ct(ct\%)$", em que rt e ct correspondem ao tempo médio de encaminhamento na rede e tempo médio de consenso, respetivamente. $rt\%$ e $ct\%$ são as percentagens correspondentes aos respectivos tempos.

Tabela 7.4: API pública da classe StatsManager

8

Avaliação Experimental

A avaliação experimental da solução desenvolvida, o MINSENS++, permite obter resultados que ajudem na formulação de conclusões relativas ao funcionamento correto e esperado do protocolo. Visto o MINSENS++ ser uma versão melhorada do MINSENS, pretende-se confrontar os dois protocolos principalmente nas áreas em que se espera que o MINSENS++ tenha resultados diferentes do MINSENS. Indicadores como a latência, cobertura e consumo de energia vão ter resultados muito semelhantes para os dois protocolos, uma vez que o modo de encaminhamento do MINSENS não sofreu alterações. Esperam-se resultados diferentes na fiabilidade da entrega de mensagens e na resiliência a ataques, sendo que as alterações esperadas são uma descida na fiabilidade, devido à introdução da fase de consenso, e um aumento da resiliência, agora que o protocolo está mais preparado para resistir a ataques na rede e até nas estações base. Também foram incluídos resultados do protocolo INSENS para os mesmos indicadores referidos.

Uma vez que os protocolos de consenso foram o principal foco da presente dissertação, também foram realizadas análises experimentais sobre os resultados obtidos especificamente por estes protocolos, no que diz respeito à taxa de conclusão do processo de consenso em pelo menos $2f$ nós e a mesma taxa para os processos terminados em todos os nós. Para maximizar a utilização dos protocolos de consenso, em todos os testes efetuados o protocolo MINSENS++ faz o encaminhamento de mensagens por todas as rotas. Assim todas as estações base podem participar no consenso, o que permite fazer a avaliação do mesmo com um maior número de participantes.

Os testes foram realizados no simulador WiSeNet, versão r329, suportado pelo Java, com o JRE na versão jre7 para sistemas de 64 bits. O sistema operativo instalado é o Windows 7 Home Premium 64-bits ©, com o Service Pack 1. Os componentes relevantes do sistema de hardware são um processador Intel ® Core ™ i7-2670QM 2.2GHz e memória

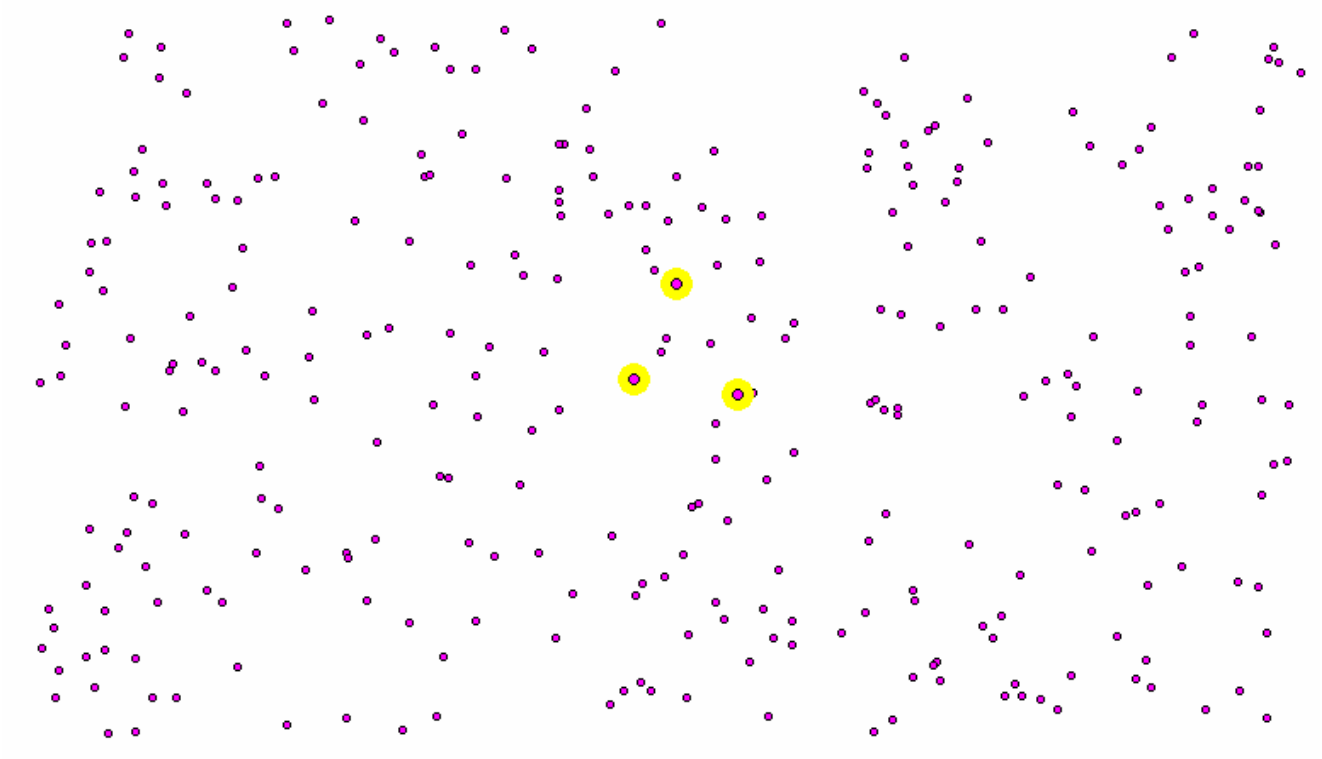


Figura 8.1: Topologia com 300 nós, 3 estações base

RAM 6GB DDR3.

Realizaram-se testes em três diferentes topologias de rede, com diferentes números de nós (300/500/1000 nós), geradas aleatóriamente pelo simulador (para exemplos de topologias ver figuras 8.1 e 8.2). Para os protocolos MINSENS e MINSENS++ definiram-se três estações base para a topologia de 300 nós e cinco estações base para as restantes, enquanto que o protocolo INSENS funciona apenas com uma estação base para todas as topologias. O objetivo seria ter cerca de 1% de nós na rede como estações base para os protocolos INSENS e MINSENS, no entanto, a restrição de as estações base terem de estar a um *hop* de distância entre si não permite ter um grande número de estações base, porque a proximidade entre elas afeta o número de rotas que conseguem estabelecer.

Para a obtenção dos resultados foi calculada a média dos valores obtidos em 10 execuções do protocolo, para os resultados obtidos serem fiáveis. Em cada uma das execuções escolheram-se 20% dos nós da rede aleatoriamente para serem emissores de dados, emitindo uma única mensagem cada um.

Cada uma das próximas secções é dedicada a um dos referidos indicadores (latência, cobertura, consumo de energia, fiabilidade e resiliência), com uma secção adicional dedicada unicamente aos dados recolhidos relativos aos protocolos de consenso, visto estes terem sido o principal foco da presente dissertação.

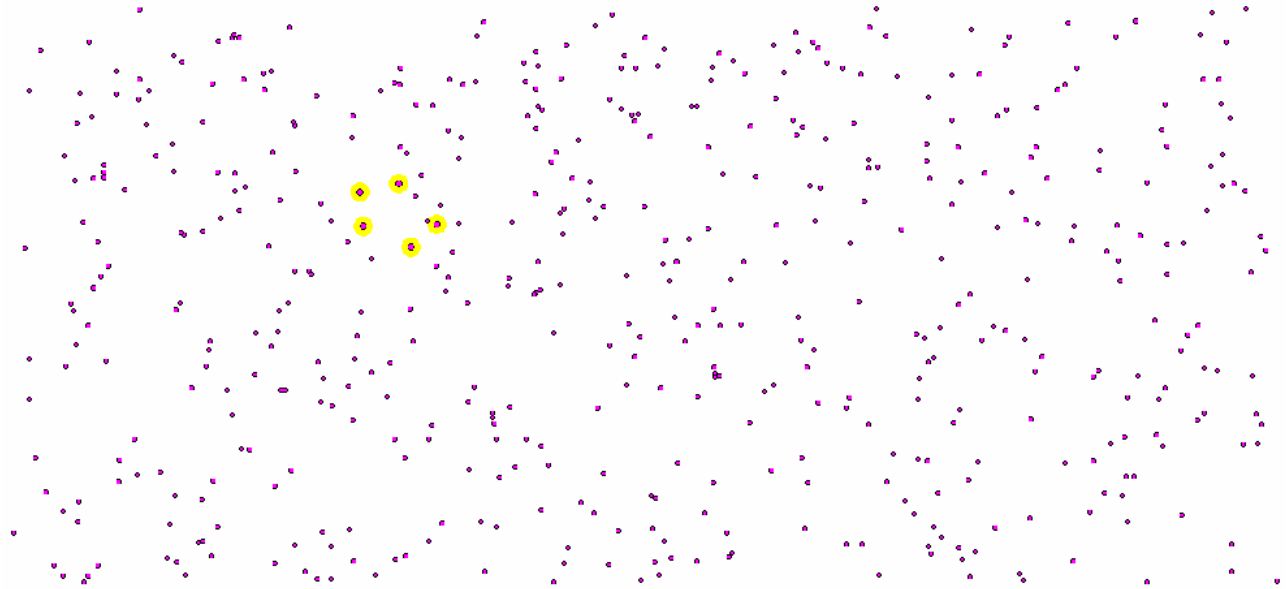


Figura 8.2: Topologia com 500 nós, 5 estações base

8.1 Cobertura

Os valores obtidos para definir a cobertura da rede representam a percentagem de nós emissores que possuem rotas para as estações base.

Nos resultados obtidos (ver figura 8.3) observa-se que os protocolos MINSENS e MINSENS++ cobrem uma maior percentagem dos nós emissores que o protocolo INSENS. Esta observação deve provavelmente ao facto de os nós que estão mais longe das estação base, no INSENS, não conseguirem fazer com que as suas mensagens com informação de vizinhança cheguem em condições à estação base, o que faz com que a estação base não calcule rotas para esses nós. Nos casos de MINSENS e MINSENS++ a existência de várias estações base aumenta a probabilidade de as mensagens de um nó chegarem, pelo menos, a uma delas, o que explica os valores mais elevados de cobertura para estes nós.

Os resultados da cobertura têm influência em praticamente todos os outros indicadores, uma vez que podem limitar o número de nós capazes de produzir e encaminhar informação.

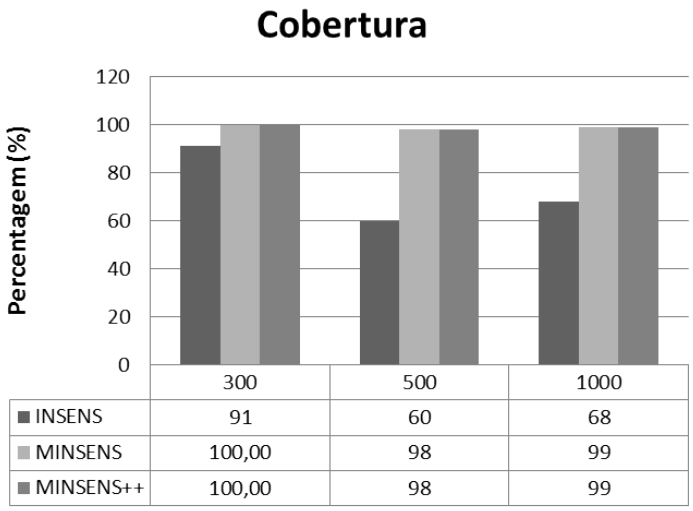
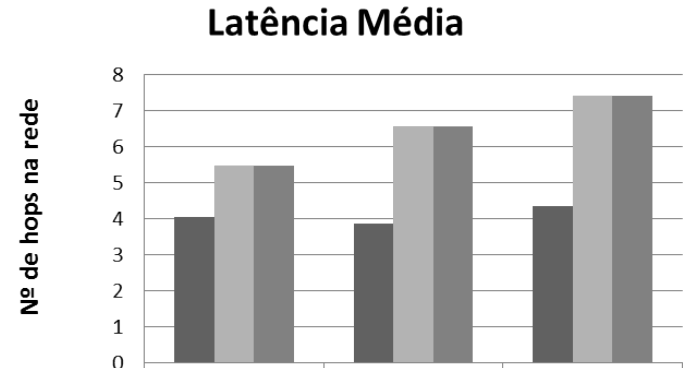


Figura 8.3: Resultados de cobertura

8.2 Latência

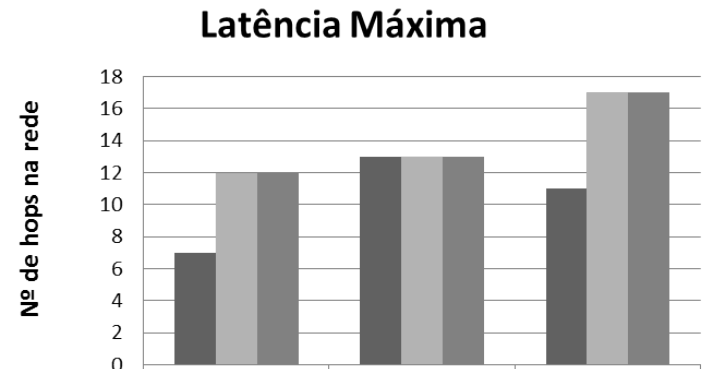
A valor de latência obtido corresponde ao número de nós por onde uma mensagem passa desde a sua origem até ao destino. Como foi referido anteriormente, os valores obtidos para este indicador, para os protocolos MINSENS e MINSENS++ são semelhantes. A latência foi avaliada em duas dimensões diferentes: a latência média e a latência máxima.

Os valores obtidos para as latências médias e máximas estão representados nas figuras 8.4 e 8.5, respetivamente. É possível observar que, no geral, os protocolos MINSENS e MINSENS++ conseguem valores mais elevados na latência máxima do que o protocolo INSENS em cenários com topologias de rede iguais. No protocolo INSENS observa-se que os nós mais distantes da estação base não recebem as suas tabelas de encaminhamento, sendo por isso considerados nós instáveis para encaminhamento, o que justifica também os valores mais baixos nos números de hops registados. A latência temporal do encaminhamento de mensagens foi medida de maneira a poder avaliar o peso de cada fase do encaminhamento (encaminhamento nos sensores e consenso nas estações base). Verificou-se que cerca de 99% do tempo de encaminhamento é passado durante o consenso, algo que seria de esperar, uma vez que as estações base têm de aguardar que existam estações base suficientes para iniciar o consenso de uma mensagem, porque existem os *broadcasters* periódicos nos processos de consenso, que podem provocar uma maior demora se o período entre retransmissões for extenso e por estarem a ser utilizados canais de comunicação como os que se observam na rede (IEEE 802.15.4). O tempo passado no consenso pode reduzir se a comunicação fosse implementada numa rede local (ex. *ethernet*) com taxas de transferência mais elevadas, como são os casos das redes IEEE 802.11 e IEEE 802.3, o que significaria um aumento na taxa de transmissão de mensagens



	300	500	1000
INSENS	4,048	3,874	4,348
MINSENS	5,47	6,576	7,417
MINSENS++	5,47	6,576	7,417

Figura 8.4: Valores de latência média, por mensagem, em número de hops



	300	500	1000
INSENS	7	13	11
MINSENS	12	13	17
MINSENS++	12	13	17

Figura 8.5: Resultados de latência máxima, em número de hops

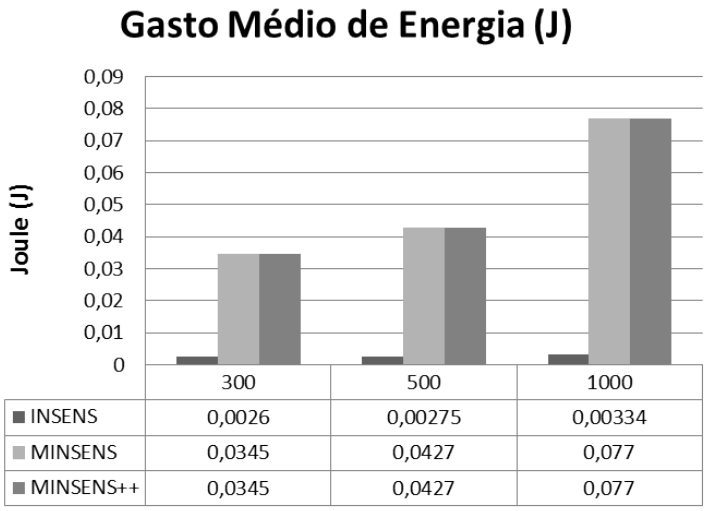
na ordem de 10^3 a 10^5 vezes. Pretende-se concretizar estas observações em trabalho futuro.

Os resultados de cobertura observados podem estar relacionados com a latência, pois uma vez que o INSENS não tem capacidade de chegar a nós mais distantes, esses nós não são cobertos pelo protocolo.

8.3 Consumo de Energia

Na avaliação do consumo de energia tem-se em conta a energia gasta em média por cada nó sensor da rede, na fase de encaminhamento de dados. Assume-se que as estações base têm uma fonte de energia constante e, por isso, não entram nas contas de consumo energético (este aspeto será abordado nas perspetivas de trabalho futuro).

Pode-se observar pelos valores obtidos (ver figura 8.6) que os nós quando utilizam os protocolos MINSENS e MINSENS++ gastam mais energia do que quando utilizam o protocolo INSENS. Estes resultados podem estar relacionados com a falta de cobertura do INSENS, pois, uma vez que tem menos nós cobertos, é produzida menos informação pelos nós e há menos dados a transmitir na rede. Para além disso, é natural que os protocolos MINSENS e MINSENS++ apresentem, mesmo assim, valores mais elevados para os consumos médios de energia, pois existem múltiplas estações base e os nós têm múltiplas rotas para as mesmas, o que faz com que a informação transmitida por um nó seja encaminhada por várias rotas, passando por vários nós, o que provoca um maior desgaste energético por nó.



	300	500	1000
INSENS	0,0026	0,00275	0,00334
MINSENS	0,0345	0,0427	0,077
MINSENS++	0,0345	0,0427	0,077

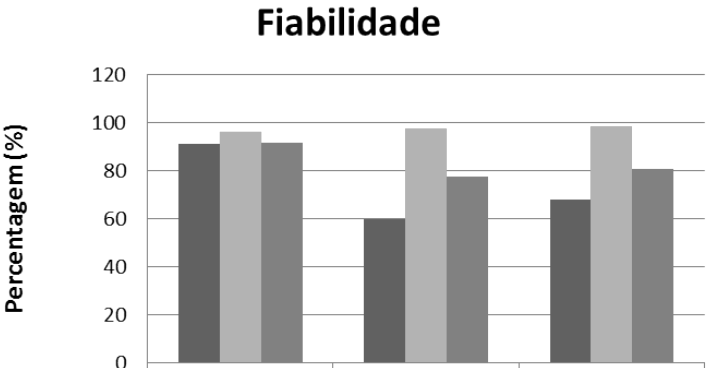
Figura 8.6: Resultados de consumo de energia médio por nó

8.4 Fiabilidade

A fiabilidade representa a percentagem de mensagens de dados entregues nas estações base, tendo em conta o número total de mensagens enviadas pelos nós sensores. A fiabilidade foi medida em duas dimensões diferentes: (1) a fiabilidade da 1ª camada da rede, composta pelos nós sensores, onde contam como entregues as mensagens que chegam até à sua estação base de destino; (2) a fiabilidade real, onde se tem em consideração as duas camadas da rede, i.e., o encaminhamento nos sensores e o consenso nas estações base, considerando-se apenas entregue uma mensagem que seja aprovado pelo processo de consenso.

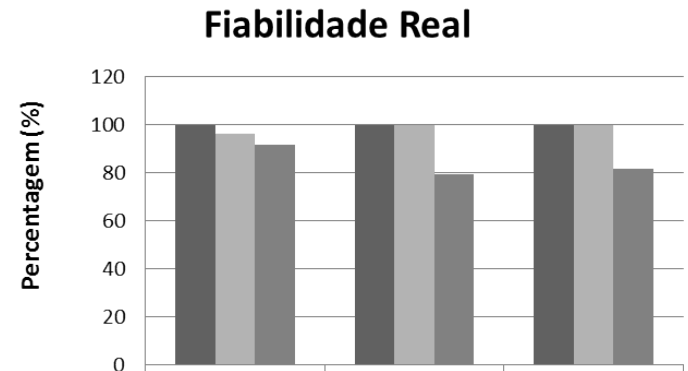
Este é o primeiro indicador para o qual os três protocolos apresentam valores diferentes. Devido à introdução do protocolo de consenso no processo de aceitação de mensagens do MINSENS++, esperava-se que o valor da fiabilidade pode-se ser afetado, uma vez que por existirem mais fases de comunicação, entre o envio da mensagem e a sua aceitação, a probabilidade de se perderem mensagens aumenta. Os valores obtidos para a fiabilidade (ver figura 8.7) confirmam as suspeitas que existiam, tendo-se observado uma descida nos valores de fiabilidade do MINSENS++, em relação ao MINSENS.

Para avaliar corretamente os valores obtidos para a fiabilidade é necessário ter em conta que a fiabilidade máxima está limitada pelo valor da cobertura que se observa na mesma situação, uma vez que é de todo impossível que um nó não coberto pelo protocolo seja capaz de entregar mensagens às estações base. Por esse mesmo motivo é possível observar, com maior clareza, os valores da fiabilidade na figura 8.8. Aqui observa-se bem que o MINSENS++, apesar de ter uma boa percentagem de mensagens entregues, está um pouco abaixo dos outros dois protocolos, o que se justifica pelo aumento da complexidade e de transmissão de mensagens imposto pela presença do protocolo de consenso



	300	500	1000
INSENS	91	60	68
MINSENS	96,33	97,70	98,62
MINSENS++	91,69	77,73	80,80

Figura 8.7: Resultados de fiabilidade



	300	500	1000
INSENS	100	100	100
MINSENS	96,326	99,694	99,616
MINSENS++	91,685	79,316	81,620

Figura 8.8: Resultados de fiabilidade real

no MINSENS++.

8.5 Resiliência

Foi feita uma avaliação comparativa entre os protocolos para observação da fiabilidade dos mesmos face à presença de 10% de nós intrusos que fazem o ataque *Blackhole*. Considera-se, como critério, a análise comparativa de mensagens que chegam à estação base, antes de passar pelo processo de consenso (o que poderia interferir negativamente na obtenção dos resultados uma vez que esse processamento não está presente no caso do protocolo INSENS). Os resultados estão representados na figura 8.9 e mostram, como seria esperado, que a fiabilidade diminui na presença de nós atacantes. Mas também é possível observar que os protocolos MINSENS e MINSENS++ são mais resistentes a estes ataques, uma vez que o decréscimo na fiabilidade é bastante inferior ao apresentado pelo INSENS.

Deve notar-se que considerar 10% de nós intrusos neste teste é um cenário de ataque bastante agressivo. Por exemplo, numa rede de 1000 nós, tal significa que um atacante teria conseguido provocar intrusões em 100 nós. Por outro lado, este ataque é feito por selecção aleatória de nós, o que pode fazer com que um número significativo dos nós seleccionados estejam próximos das estações base.

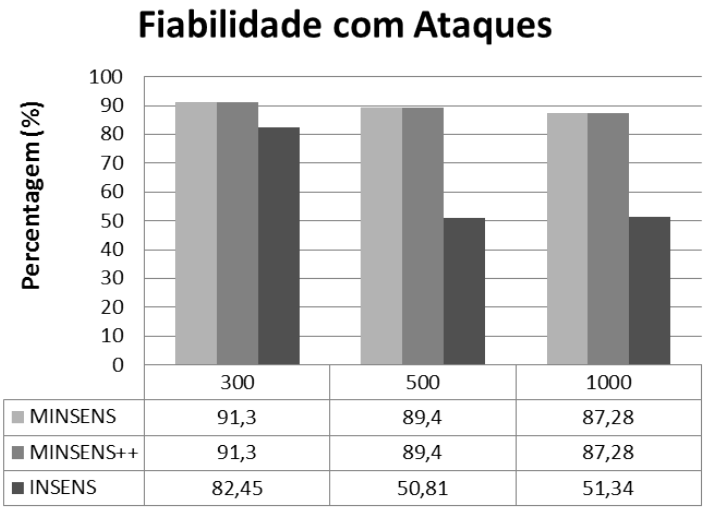


Figura 8.9: Resultados de fiabilidade com 10% de nós intrusos

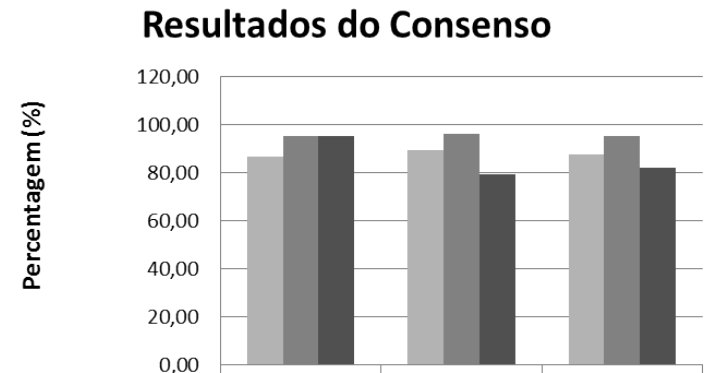
8.6 Resultados de Consenso

Apesar de os valores obtidos para a fiabilidade no MINSENS++ darem pistas do comportamento do protocolo de consenso, não é possível obter resultados concretos sobre o protocolo apenas com a fiabilidade. Por isso foi desenvolvido o módulo que recolhe dados do protocolo de consenso, sendo assim possível fazer uma análise mais detalhada.

Há vários dados relevantes sobre o consenso que necessitam ser observados para tirar conclusões. Esses dados são: (1) a percentagem de pedidos de consenso iniciados, tendo em conta o número de mensagens enviadas pelos nós; (2) a percentagem de processos de consenso multi-valor iniciados face ao número de pedidos de consensos; (3) a percentagem de processos de consenso multi-valor terminados face ao número dos processos iniciados. Para estes dados observam-se os valores obtidos por, pelo menos, $2f$ estações base, e por todas as estações base.

As percentagens de pedidos de consenso iniciados são iguais aos valores obtidos para a fiabilidade do MINSENS, uma vez que até aí o processo de encaminhamento não foi alterado. As restantes percentagens estão representadas na figura 8.10. A percentagem de consensos terminados por $> 2f$ e por todos é relativa ao número de consensos iniciados. É importante relembrar que na topologia de 300 nós apenas existem três estações base, o que significa que, sendo $f = \frac{n}{3}$, então $2f = 2$, logo o resultado para $> 2f$ consensos terminados é igual ao resultado dos consensos terminados por todos. Já no caso das topologias de 500 e 1000 nós existem cinco estações base, o que provoca a diferença nos resultados.

A diferença dos resultados obtidos para $> 2f$ consensos terminados e todos terminados, nas topologias com cinco estações base, pode dever-se a um destes motivos ou até



	300	500	1000
Iniciados (Todos)	86,52	89,54	87,44
Terminados (>2f)	95,18	96,36	95,50
Terminados (Todos)	95,18	79,56	81,93

Figura 8.10: Resultados de consensos

aos dois: (1) perda de mensagens devido à proximidade das estações base e à grande intensidade de comunicação provocada pelo consenso; (2) recuperadores de nós atrasados pouco eficientes.

9

Conclusões

A presente dissertação aborda a problemática da concepção de sistemas de encaminhamento seguro, com tolerância a intrusões, para redes de sensores sem fios. Na dissertação propõe-se um serviço de encaminhamento seguro e tolerante a intrusões para RSSF de grande escala e ambiente de comunicação *multi-hop*, serviço associado ao protocolo designado por MINSENS++.

O protocolo MINSENS++ suporta encaminhamento resiliente de dados originados em sensores, através de múltiplas rotas disjuntas formadas pela rede de sensores, tendo como destino diferentes estações base, nas quais essas rotas convergem.

As rotas disjuntas são descobertas, selecionadas e estabelecidas de forma dinâmica e proactiva, durante o próprio processo de descoberta e auto-organização da rede. A formação destas rotas é feita de forma coordenada pelas estações base, de forma que um mesmo nó de encaminhamento não seja nó interior em mais do que uma rota estabelecida.

Com base nas múltiplas rotas estabelecidas, os dados podem ser encaminhados desde cada nó origem com a flexibilidade julgada apropriada para as aplicações finais, de forma a optimizar o balanço entre consumo de energia e segurança.

Para efeitos de maximizar o suporte de segurança e tolerância a intrusões, a ideia é enviar os dados em cada caso pelas múltiplas rotas disjuntas estabelecidas.

O protocolo MINSENS++ integra um mecanismo de consenso distribuído de dados tolerante a intrusões ao nível das estações base. Com base neste mecanismo, implementado por um protocolo de consenso designado por MVC-WSN que concretiza um protocolo de consenso distribuído probabilístico tolerante a falhas bizantinas (ou intrusões provocadas por adversários bizantinos), os dados encaminhados pela rede para as estações base são submetidos a uma verificação de consenso, antes de serem disponibilizados

para sistemas ou aplicações externos à rede.

A solução apresentada possui uma abordagem de rede sobreposta, com base em duas camadas do sistema de encaminhamento:

- A primeira camada, formada pela rede de sensores propriamente dita, suporta o encaminhamento de dados pelas múltiplas rotas estabelecidas até às estações base, sendo este encaminhamento suportado com base em dispositivos que implementam a pilha IEEE 802.15.4.

- Na segunda camada, as estações base agregam os dados recebidos e executam um protocolo de consenso distribuído tolerante a intrusões, sendo este suportado numa rede estabelecida pelas estações base. Esta rede pode ser concretizada por uma rede IEEE 802.15.4, uma rede local sem fios (ex., IEEE 802.11) ou uma interligação das estações base por uma rede local com fios (ex., IEEE 802.3).

Para se avaliar o sistema de encaminhamento proposto, os protocolos que o constituem foram concretizados e avaliados num simulador para redes de sensores sem fios de grande escala: o simulador WiSeNet. Neste ambiente de simulação, foram testadas redes com 300, 500 e 1000 nós, analisando-se os seguintes indicadores obtidos:

- latência do encaminhamento;

- condições de conectividade e cobertura da rede;

- indicadores de fiabilidade;

- métricas de consumo de energia;

- indicadores relativos à correção e execução do protocolo de consenso.

Os resultados obtidos mostram que a solução proposta é viável, apresenta vantagens face a soluções anteriores (nomeadamente o protocolo INSENS e o protocolo MINSENS) e mostra-se como uma aproximação que melhora significativamente a capacidade de resistência da rede face a ataques por intrusão.

Os resultados obtidos são também prometedores em relação à viabilidade de se utilizarem protocolos de consenso distribuído de características probabilísticas em RSSFs, com as necessárias adaptações e usando redes heterogéneas formadas por nós sensores de baixo custo (com limitações de processamento , memória e energia) e dispositivos que possam ser usados como nós de agregação e verificação de dados, quer atuando como estações base, quer atuando como nós intermédios de processamento de segurança no interior da rede.

9.1 Aspectos em Aberto e Direções de Trabalho Futuro

A partir dos resultados alcançados que se mostraram mais interessantes, é possível destacar alguns aspectos em aberto bem como identificar perspetivas de direções de trabalho futuro, tendo em vista melhorar o funcionamento dos protocolos, tanto o protocolo MINSENS++ como o protocolo MVC-WSN. Também poderiam ser feitas algumas melhorias ao simulador WiSeNet para permitir fazer algumas avaliações que atualmente não são viáveis mas que poderão ser úteis para uma avaliação mais exaustiva dos protocolos propostos.

9.1.1 Melhorias ao Nível do Simulador WiSeNet

O simulador não permite obter dados da fase de organização da rede (descoberta e estabelecimento de rotas), dados esses que seriam interessantes, por exemplo, para determinar o consumo de energia associado ao processo de organização. Também seria interessante obter valores de fiabilidade para esta fase, o que permitiria verificar melhor o comportamento do protocolo a partir das condições iniciais de organização da rede.

No ambiente de simulação, as estações base são definidas como nós semelhantes aos restantes. No entanto, não é possível simular intrusões ao nível dos nós definidos para atuarem como estações base. Essa incapacidade impediu a avaliação experimental do protocolo de consenso na presença de intrusos, um dos objetivos inicialmente previstos. O processo de seleção de nós intrusos deveria pois ser melhorado, de modo a tornar possível escolher qualquer nó como nó atacante (incluindo os nós que atuam como estações base).

9.1.2 Melhorias ao Nível do Protocolo MINSENS++

Sendo possível definir uma estação base como nó intruso no ambiente do simulador, seria interessante comprovar o funcionamento do protocolo MINSENS++ quando uma ou mais estações base são sujeitas a ataques por intrusão. Deste modo seria possível avaliar o potencial do protocolo de consenso inserido no protocolo de encaminhamento, verificando na prática as garantias de consenso probabilístico face a intrusões e nas condições de simulação do suporte de comunicação.

Um problema do modelo de sistema que foi concretizado é o facto de as estações base estarem a um *hop* de distância. Esta concretização apresenta problemas a níveis distintos: (1) devido à necessidade de as rotas serem disjuntas, estando as estações base próximas, existe grande possibilidade de se esgotarem rapidamente todas as rotas possíveis, uma vez que os nós mais próximos das estações base apenas podem ser interiores numa única rota; (2) como as estações base estão próximas, se um atacante consegue ter acesso a uma das estações base poderá eventualmente ter acesso às restantes; (3) devido à proximidade

das estações base, as rotas formadas tendem a convergir numa mesma área da rede, o que aumenta a possibilidade de colisões e falhas de comunicação nessa zona, sendo também contraproducente do ponto de vista da distribuição da carga de processamento no processo de encaminhamento, além de que essa convergência provoca um maior consumo energético na região da rede próxima das estações base.

Para resolver o problema é necessário afastar as estações base, mas para isso é necessário assegurar que continuam a conseguir comunicar entre si a um *hop* de distância, ou conceber um novo algoritmo de consenso tolerante a intrusões numa rede com características de comunicação *multi-hop*. Esta segunda possibilidade é mais complexa e envolve a investigação e proposta de novos algoritmos de consenso distribuído, com base em comunicações sem fios em redes *multi-hop*. A primeira solução é mais simples e mais imediatamente viável. É possível assumir que as estações base têm meios de comunicação de longo alcance (por exemplo suportadas em redes locais do tipo IEEE 802.3 ou IEEE 802.11, implementando a pilha TCP/IP. Desde que no simulador seja possível concretizar este ambiente, poderá voltar a simular-se experimentalmente o sistema de encaminhamento proposto neste novo cenário.

Uma melhoria muito interessante que se poderá fazer ao MINSENS++ é a introdução de mecanismos de detecção e remoção de nós intrusos, como complemento à tolerância a intrusões já conseguida pelo protocolo. Esta melhoria permitia que um intruso apenas causasse problemas na rede durante um período limitado de tempo. Uma possível solução pode ser baseada na detecção de nós intrusos, realizada pelas estações base. Tal como foi concebido e está implementado, o protocolo MINSENS++ permite facilmente identificar rotas suspeitas. Quando uma mensagem é enviada por várias rotas é possível saber que existe pelo menos uma rota por onde os dados não chegaram ou chegaram alterados. Mas este tipo de detecção é muito abrangente e eliminar toda uma rota pode prejudicar muito o funcionamento da rede, dado que se descartam todos os nós dessa rota. Uma solução mais elaborada passaria por uma alteração na construção de rotas, sendo que as rotas passariam a ser apenas parcialmente disjuntas, i.e. várias rotas poderiam ter um número pequeno de nós em comum. Este modelo de construção de rotas permitia que um único nó intruso prejudicasse várias rotas para diferentes estações base, o que tem como consequência que várias estações base conseguem detetar rotas suspeitas e verificarem se existe algum nó comum às várias rotas suspeitas, descobrindo assim potenciais intrusos. Esta solução prevê uma alteração nas rotas, o que poderia prejudicar o restante funcionamento do protocolo, sendo necessário fazer uma reavaliação do novo protocolo face aos restantes indicadores estudados.

9.1.3 Melhorias ao Nível do Protocolo MVC-WSN

O protocolo de consenso multi-valor desenvolvido apenas foi testado com cinco estações base participantes. Esta avaliação poderá ser alargada para se analisar o funcionamento

do protocolo quando existe um grande número de participantes.

O protocolo PEB foi desenvolvido propositadamente para ser introduzido no protocolo MVC. No entanto nunca foi testado e avaliado separadamente. Tal seria interessante para obter dados concretos sobre o protocolo e identificar separadamente os factores que contribuem para os indicadores obtidos, em diferentes condições de funcionamento, por exemplo: funcionamento com variações no número de participantes; funcionamento com variação no número máximo de retransmissões das mensagens; ou funcionamento com variações no período entre sucessivas retransmissões. Tal é particularmente importante dado que o processamento envolvido é significativamente importante no impacto de latência e nas condições probabilísticas de completamento do consenso.

Os mecanismos de recuperação de nós atrasados foram desenvolvidos com a intenção de resolver um problema prático, sendo uma solução concebida a partir dos resultados observados nas simulações. Uma análise mais cuidadosa deveria ser feita de forma a melhorar a especificação, concepção e implementação de mecanismos de coordenação para recuperação de nós atrasados, de forma a optimizar o processamento do consenso de dados.

Bibliografia

[1] N. Abu-Ghazaleh, K.-D. Kang, and K. Liu. Towards resilient geographic routing in wsns. In *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, Q2SWinet '05, pages 71–78, 2005.

[2] E. A. Alchieri, A. N. Bessani, J. Silva Fraga, and F. Greve. Byzantine consensus with unknown participants. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, OPODIS '08, pages 22–40, 2008.

[3] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2–3):165–175, Sept. 2003.

[4] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41:122–152, January 1994.

[5] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, and Y. F. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Comput. Commun.*, 30:1655–1695, May 2007.

[6] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, PODC '83, pages 27–30, 1983.

[7] G. Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, pages 154–162, 1984.

[8] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32:824–840, 1985.

[9] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3:38–45, January 2004.

[10] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography (extended abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 123–132, 2000.

[11] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 42–51, 1993.

[12] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Third Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, Louisiana, Feb. 1999. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.

[13] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43:685–722, July 1996.

[14] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43:225–267, March 1996.

[15] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and K. Szymanski. Chapter 1 sense : A sensor network simulator. *Components*, pages 249–267, 2004.

[16] Y. chun Hu, A. Perrig, and D. B. Johnson. Wormhole detection in wireless ad hoc networks. Technical report, 2002.

[17] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). Rfc, RFC Editor, 2003.

[18] A. P. R. da Silva, M. H. T. Martins, B. P. S. Rocha, A. A. F. Loureiro, L. B. Ruiz, and H. C. Wong. Decentralized intrusion detection in wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Quality of Service and Security in Wireless and Mobile Networks (Q2SWINET'05)*, pages 16–23. ACM Press, 2005.

[19] R. de Paz Alberola and D. Pesch. Avroraz: extending avrora with an ieee 802.15.4 compliant radio chip model. In *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '08, pages 43–50, New York, NY, USA, 2008. ACM.

[20] J. Deng, R. Han, and S. Mishra. INSENS: Intrusion-tolerant routing for wireless sensor networks. *Computer Communications*, 29(2):216–230, Jan. 2006.

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

[22] D. Dolev and A. C. Yao. On the security of public key protocols. Technical report, Stanford, CA, USA, 1981.

[23] J. R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, 2002.

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

[25] L. Evers, M. J. J. Bijl, M. Marin-perianu, R. Marin-perianu, and P. J. M. Havinga. Wireless sensor networks and beyond: A case study on transport and logistics. In *In International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005*, pages 1381–3625, 2005.

[26] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[27] G. Gaubatz, J.-P. Kaps, E. Ozturk, and B. Sunar. State of the art in ultra-low power public key cryptography for wireless sensor networks. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, PERCOMW '05, pages 146–150, 2005.

[28] A. Guerreiro. Intrusion tolerant routing protocols for wireless sensor networks. Master's thesis, Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Sept. 2011.

[29] V. C. Gungor and G. P. Hancke. Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Transactions on Industrial Electronics*, 56(10):4258–4265, Oct. 2009.

[30] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2:1–38, February 2006.

[31] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, pages 8020–, 2000.

[32] M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13:124–149, 1993.

[33] D. B. Johnson, D. A. Maltz, and J. Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. In *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pages 139–172, 2001.

[34] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 162–175, 2004.

[35] C. Karlof and 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*, pages 113–127, 2003.

[36] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 254–263. ACM, 2007.

[37] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[38] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer. Shawn: A new approach to simulating wireless sensor networks. *CoRR*, abs/cs/0502003, 2005.

[39] M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen. A survey of application distribution in wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.*, 2005:774–788, October 2005.

[40] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982.

[41] A. Ledeczi. Jprowler. `http://www.escherinstitute.org/Plone/frameworks/nes/tools/prowler`.

[42] S.-B. Lee and Y.-H. Choi. A secure alternate path routing in sensor networks. *Comput. Commun.*, 30:153–165, December 2006.

[43] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM.

[44] S. Lindsey and C. S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 3, pages 3–1125–3–1130 vol.3, 2002.

[45] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 245–256, 2008.

[46] W. Lou and Y. Kwon. H-spread: A hybrid multipath scheme for secure and reliable data collection in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 55:1320–1330, July 2006.

[47] M. C. Loui and H. H. Abu-Amara. *Memory requirements for agreement among unreliable asynchronous processes*, volume 4, pages 163–183. JAI press, 1987.

[48] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488, New York, NY, USA, 2007. ACM Press.

[49] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97. ACM, 2002.

[50] T. Maret, R. Kummer, P. Kropf, and J.-F. Wagen. Freemote emulator: a lightweight and visual java emulator for wsn. In *Proceedings of the 6th international conference on Wired/wireless internet communications*, WWIC'08, pages 92–103, Berlin, Heidelberg, 2008. Springer-Verlag.

[51] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 255–265, 2000.

[52] R. C. Merkle. *Protocols for Public Key Cryptosystems*, pages 122–134. IEEE Computer Society Press, 1980.

[53] A. Milenković, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Comput. Commun.*, 29:2521–2533, August 2006.

[54] V. B. Misic, J. Fung, and J. V. Misic. Mac layer security of 802.15.4-compliant networks. In *MASS*. IEEE, 2005.

[55] H. Moniz, N. F. Neves, and M. Correia. Turquois: Byzantine consensus in wireless ad hoc networks. In *DSN*, pages 537–546. IEEE, 2010.

[56] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Ritas: Services for randomized intrusion tolerance. *IEEE Trans. Dependable Secur. Comput.*, 8:122–136, January 2011.

[57] nsnam. Ns-3. http://www.nsnam.org/.

[58] B. Parno, M. Luk, E. Gaustad, and A. Perrig. Secure sensor network routing: a clean-slate approach. In *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, pages 11:1–11:13, 2006.

[59] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 49–63, 2005.

[60] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27:228–234, April 1980.

[61] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 234–244, 1994.

[62] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *IN PROCEEDINGS OF THE 2ND IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS*, pages 90–100, 1997.

[63] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8:521–534, September 2002.

[64] S. S. Ramaswami and S. Upadhyaya. Smart handling of colluding black hole attacks in manets and wireless sensor networks using multipath routing. *2006 IEEE Information Assurance Workshop*, pages 253–260, 2006.

[65] A. Rito. Redes de sensores sem fios. Master's thesis, Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2006.

[66] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313, London, UK, 1989. Springer-Verlag.

[67] D. Schmidt, M. Berning, and N. Wehn. Error correction in single-hop wireless sensor networks: a case study. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1296–1301, 2009.

[68] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *In Sensys*, pages 188–200. ACM Press, 2004.

[69] P. Silva. Wisenet - wireless sensor networks simulator. `http://code.google.com/p/wisenet/`.

[70] N. Song, L. Qian, and X. Li. Wormhole attacks detection in wireless ad hoc networks: A statistical analysis approach. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 17 - Volume 18*, IPDPS '05, 2005.

[71] I. Telegraph and T. C. Committee. *CCITT Recommendation X.800: Data Communication Networks: Open Systems Interconnection (OSI); Security, Structure and Applications : Security Architecture for Open Systems Interconnection for CCITT Applications*. International Telecommunication Union, 1991.

[72] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[73] L. Vanzago. Overview on wireless sensor networks. Master's thesis, Dipartiamento di Scienze del'Informazione, Università degli Studi di Milano, 2006.

[74] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary. Wireless sensor network security: A survey," in book chapter of security. In *in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds*, pages 0–849. CRC Press, 2007.

[75] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10:18–25, March 2006.

[76] A. D. Wood, L. Fang, J. A. Stankovic, and T. He. Sigf: a family of configurable, secure routing protocols for wireless sensor networks. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, SASN '06, pages 35–48, 2006.

[77] H. Wu, S. Member, Q. Luo, P. Zheng, and L. M. Ni. Vmnet: Realistic emulation of wireless sensor networks. Technical report, 2005.

[78] L. Zhao and J. G. Delgado-Frias. Multipath routing based secure data transmission in ad hoc networks. In *Proceedings of the 2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 17–23, 2006.