Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Implementation of a Mobile Ad Hoc Network communication protocol for Human-Robot Search and Rescue Teams

José Santos Martins Pereira
jsmp@student.dei.uc.pt

Orientador:
Professor Doutor Filipe João Boavida Mendonça Machado de Araújo

Julho de 2013

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

A utilização de equipamentos móveis generalizou-se nos últimos anos. Os avanços nas capacidades de computação e comunicação aumentaram enormemente o espectro de aplicação destes dispositivos, que têm hoje os mais variados domínios de aplicação. Desde cenários militares, sempre uma força motriz de avanços científicos, passando por monitorização ambiental ou redes de viaturas, até cenários de emergência, a capacidade de constituir uma rede de forma dinâmica é cada vez mais uma exigência para a realização de trabalho de forma distribuída.

O presente trabalho propõe o protocolo de encaminhamento CHOPIN para redes móveis ad-hoc (MANET), uma implementação inspirada pelo protocolo OLSR e desenvolvido para dar resposta aos requisitos de cenários de cooperação de equipas humanas e de robôs em situações de emergência (Projecto CHOPIN). Neste contexto, protocolos actuais funcionam de forma centralizada, onde um Centro de Comando Operacional (CCO) coordena as operações no terreno. Actividades de mapeamento do cenário (comunicação de grupo), detecção de materiais perigosos, identificação de vítimas e pontos de interesse (comunicação para o CCO) são contributos possíveis dos agentes robóticos, reduzindo o perigo para os agentes humanos e adicionando uma vertente distribuída que pode ser extremamente útil na capacidade de resposta.

Foram realizados testes em que foi comparado o protocolo desenvolvido com o OLSR, sendo os resultados positivos. Uma vez que o OLSR inclui uma técnica de optimização de difusão de mensagens, o *MultiPoint Relaying* (MPR), ausente para já do CHOPIN, aquele necessita de enviar menos mensagens. Em contrapartida, o CHOPIN armazena muito menos estado da rede, sendo extremamente leve relativamente às exigências de memória. No que diz

respeito às métricas de desempenho mais importantes, como a velocidade de convergência da rede, latência e *throughput*, os protocolos apresentam resultados semelhantes, com ligeira vantagem para o CHOPIN.

Palavras-chave: Redes móveis Ad Hoc, MANET, Redes sem fios, Protocolos de Encaminhamento, Cenários de Emergência, Sistemas de Suporte

Abstract

Mobile devices have become ubiquitous in recent years. Advances in computing and communication capacity have greatly widened their scope of application. Ranging from military scenarios, to environmental monitoring, vehicular networks or search and rescue operations, the ability to deploy a dynamic communication platform on demand is not far from becoming commonplace.

The thesis proposes the CHOPIN routing protocol for mobile ad-hoc networks (MANET). The implementation is inspired by OLSR and was developed in response to the requirements of Human and robotic cooperation in search and rescue scenarios (Project CHOPIN). Current operational guidelines follow a centralized approach, where a Command Center (CC) coordinates activities on the field. Cooperative mapping (group communication), hazardous materials detection, victim localization and landmarks (communication towards the CC) are possible contributions from robotic agents, effectively reducing risk to human lives and allowing for distributed operation.

Tests where performed for comparison between the proposed protocol and OLSR, with positive results. Through the use of a flooding optimization technique, MultiPoint Relaying (MPR), OLSR operation results in less control overhead. On the other hand, CHOPIN maintains lesser state information, resulting in a lower memory footprint. In regard to some relevant metrics like convergence time, latency and throughput, the protocols behave on par, with CHOPIN slightly on the lead.

Keywords: Mobile Ad Hoc Networks, MANET, Wireless Networks, Routing Protocols, Emergency Scenarios, Support Systems

Índice

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

Acrónimos

MANET *Mobile Ad-Hoc Network*

CCO *Centro de Comando Operacional*

CHOPIN *Cooperation between Human and rObotic teams in catastroPhic INcidents*

OLSR *Optimized Link State Routing*

AODV *Ad hoc On-Demand Distance Vector*

MAODV *Multicast Ad hoc On-Demand Distance Vector*

DSDV *Destination-Sequenced Distance-Vector*

DSR *Dynamic Source Routing*

ZRP *Zone Routing Protocol*

MRSLAM *Multi-Robot Simultaneous Localization and Mapping*

TLV *Type Length Value*

ROS *Robot Operating System*

CSMA *Carrier Sense Multiple Access*

CSMA/CA *Carrier Sense Multiple Access with Collision Avoidance*

CSMA/CD *Carrier Sense Multiple Accces with Collision Detection*

WLAN *Wireless Local Area Network*

WSN *Wireless Sensor Networks*

GPS *Global Positioning System*

MPR *MultiPoint Relay*

RREQ *Route Request*

RREP *Route Reploy*

RERR *Route Error*

REP-ACK *Route Reply Acknowledgement*

LOS *Line of Sight*

IPC *Inter-Process Communication*

1 Introdução

Com o advento da Internet e, em anos recentes, com a disseminação de equipamentos móveis temos assistido a uma mudança de paradigma relativamente ao acesso à informação. Passou-se do puro consumo, passivo, de informação recolhida e trabalhada por entidades para a dicotomia produção/consumo de informação instantânea e sempre acessível.

O uso alargado de redes sem fios é uma realidade quotidiana. Exigências de acesso continuado a informação criam desafios interessantes na procura de uma experiência transparente. Com a proliferação de comunicações sem fios surgem novos domínios de aplicação, bem como novas oportunidades na reformulação de antigos. Uma área de interesse é a aplicação de equipamentos móveis de baixo custo no apoio em situações de salvamento e emergência.

O projecto CHOPIN (*Cooperation between Human and rObotic teams in catastroPhic INcidents*), no qual o presente estágio curricular se insere, é uma iniciativa conjunta entre o Instituto de Sistemas e Robótica da Universidade (ISR-UC), o Centro de Informática e Sistemas da Universidade de Coimbra (CISUC) e a Autoridade Nacional de Protecção Civil (ANPC), com a colaboração dos Bombeiros Sapadores de Coimbra. Tem como propósito o estudo da cooperação entre equipas de humanos e robôs no desenvolvimento de sistemas de suporte a situações de emergência em cenários urbanos.

Em situações de emergência e catástrofe existe a forte possibilidade de uma infra-estrutura de comunicação não estar prontamente disponível ou encontrar-se danificada. Torna-se então útil a capacidade da criação de uma rede de comunicação autónoma que possa servir de alternativa aos canais tradicionais, potenciando o desenvolvimento de soluções de suporte à intervenção dos agentes humanos.

O presente trabalho propõe o protocolo CHOPIN, uma implementação de um protocolo de comunicação para redes móveis ad-hoc como suporte à exploração da prova de conceito proposta pelo projecto CHOPIN. O Centro de Comando Operacional (CCO), uma entidade preponderante nos protocolos actuais de busca e salvamento, é um ponto de consolidação de informação onde é desenvolvida uma visão global do cenário. Mantém a sua importância no contexto de uma rede de comunicação assegurada pelos agentes robóticos. Os agentes robóticos adicionam uma capacidade distribuída, podendo actuar de forma autónoma e em paralelo para executar operações sem risco para os agentes humanos.

As operações da responsabilidade dos agentes robóticos inserem-se no contexto da investigação em várias vertentes do projecto CHOPIN. O protocolo implementado tem em consideração as necessidades específicas destas, nomeadamente o MRSLAM[1] – Localização e Mapeamento Simultâneos com Múltiplos Robôs, SmokeNav[2] – Navegação Robótica em Cenários de Reduzida Visibilidade e IBombeiro[3]. O requisito principal de cada uma das vertentes é a capacidade de comunicação com o CCO, e no caso do MRSLAM, comunicação em grupo, para troca e composição de mapas locais para posterior construção de uma mapa global.

[1] http://mrl.isr.uc.pt/?w=msc_proposals_information&ID=22
[2] http://mrl.isr.uc.pt/?w=msc_proposals_information&ID=21
[3] http://mrl.isr.uc.pt/?w=msc_proposals_information&ID=18

1.1 Objectivos

Este trabalho tem como objectivo principal a implementação de um protocolo que permita a comunicação e cooperação entre agentes robóticos e agentes humanos, constituindo uma rede móvel *ad-hoc* (MANET). Esta implementação deve corresponder às necessidades das várias vertentes do projecto CHOPIN, nomeadamente a utilização de equipamento de baixo custo, mais especificamente, placas de rede com a tecnologia Wi-Fi, capacidades de patrulha e mapeamento e detecção de materiais perigosos.

O segundo objectivo é a manutenção da transparência na comunicação de alto nível entre agentes através da plataforma *Robot Operating System* (ROS), que potencia o desenvolvimento de aplicações distribuídas para robôs através de mecanismos de publicação/subscrição e cliente/servidor.

Tendo sido um objectivo inicial a integração do protocolo como um módulo do ROS, constatou-se que seria melhor opção o funcionamento independente, dada a sua interacção com funções de baixo nível do sistema operativo.

Finalmente, pretende-se avaliar o protocolo implementado utilizando o protocolo OLSR como referência, através da realização de testes em situações reais.

1.2 Organização

O remanescente do presente documento encontra-se organizado da seguinte forma: o capítulo 2 aborda o estado da arte, fazendo referência às primeiras experiências em redes sem fios que originaram as tecnologias em uso actualmente. São apontadas as características distintivas entre redes com infra-estrutura e redes sem infra-estrutura e apresentado o funcionamento de alguns protocolos de encaminhamento utilizados em MANET. O capítulo 3 apresenta os requisitos e cenários relevantes para os objectivos propostos do projecto CHOPIN, seguindo-se no capítulo 4 a exposição da arquitectura adoptada. O

capítulo 5 aborda a implementação do protocolo CHOPIN. O capítulo 6 apresenta os testes realizados e discussão de resultados, e por fim no capítulo 7, são apresentadas as conclusões e considerações para trabalho futuro.

2 Estado de Arte

2.1 Hawaii

Viajemos até ao final dos anos 60. Local: Universidade do Hawaii, Honolulu, Hawaii. Dava-se o início de um estudo de utilização de ondas rádio para comunicação entre computadores liderado por Norman Abramson. O objectivo era permitir que campus remotos (alguns em ilhas distintas) pertencentes à Universidade, tivessem acesso aos recursos computacionais do campus principal.

Dois canais rádio na banda UHF foram utilizados, um pelo nó central para envio de pacotes para os nós remotos e outro partilhado pelos nós remotos para comunicação com o nó central. A partilha desde segundo canal podia dar origem a colisão de pacotes, uma vez que o seu envio por parte dos utilizadores remotos era feito sem qualquer sincronização. Este modo de operação ficou conhecido como *pure ALOHA* ou *Random Access Channel*. A resolução de colisões era feita através de retransmissão de pacotes perdidos de acordo com um intervalo aleatório.

O sistema ALOHA (ou ALOHANET) foi pioneiro no uso do conceito de *Packet Broadcasting* via rádio nos moldes referidos (Binder *et al.*, 1975). Este projecto teve um papel determinante no trabalho em *Carrier Sense Multiple Access* (CSMA), que viria posteriormente a dar origem a CSMA/CD (Ethernet) e CSMA/CA (Wi-Fi) (Abramson & Schwartz, 2009).

CSMA é um protocolo MAC (*Media Access Control*) que funciona de forma a que num meio físico partilhado um nó tente perceber se existe outro tráfego a circular antes de enviar a sua transmissão. Este protocolo deu origem ao *Carrier*

Sense Multiple Access with Collision Detection (CSMA/CD) responsável pela gestão da utilização do canal de comunicação partilhado utilizado originalmente na tecnologia Ethernet desenvolvida por Robert Metcalfe e David Boggs. A utilização de *switchs* e ligações *full-duplex* nas redes actuais eliminou a sua necessidade. A outra variante de CSMA, *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA), é utilizada na norma IEEE 802.11 (IEEE-SA, 2012) na especificação de *Wireless Local Area Networks* (WLAN).

Actualmente o Aloha Channel ainda é utilizado em redes de telecomunicações móveis e em sistemas de comunicações via satélite.

2.2 Ethernet e Wi-Fi

As duas tecnologias de rede com maior expressão actualmente são Ethernet (LAN) e Wi-Fi (IEEE 802.11, WLAN). É curioso constatar que estas tiveram uma força motivadora comum, um sistema de comunicação através de ondas rádio. Influenciado pelo sistema ALOHA, o funcionamento original da Ethernet era num ambiente *broadcast*, utilizando uma topologia BUS com ligações através de cabo coaxial.

Uma rede com fios é caracterizada por equipamentos fixos interligados por cablagem, no caso das LAN actuais, utilizando cabo de par traçado ou fibra óptica. O seu meio de transporte físico permite grande qualidade de transmissão e elevada largura de banda. A sua evolução afastou-se do ambiente *broadcast* para uma gestão de largura de banda e colisões através de Ethernet Switches para segmentação de rede e isolamento de domínios de colisão.

O ambiente *broadcast* é característico das redes Wi-Fi. Devido às particularidades físicas deste meio de transmissão não é possível a definição precisa de áreas de cobertura. As características de propagação de sinal são dinâmicas e imprevisíveis. Pequenas variações de posicionamento ou direcção podem ter um impacto profundo nas condições de sinal.

As normas IEEE 802.11b e IEEE 802.11g, que especificam a tecnologia Wi-Fi actual, utilizam ondas rádio para transmissão de pacotes na banda ISM (Industrial, Scientific, Medical) na gama 2.4GHz. No âmbito das WLAN, a especificação 802.11 define dois modos de operação: redes baseadas numa infra-estrutura e redes sem infra-estrutura ou ad-hoc (Chlamtac *et al.*, 2003).

No primeiro modo de operação, equipamentos móveis estão dependentes de um ponto de acesso fixo e da sua capacidade de alcance para ligação à rede. A interacção típica apresenta estes equipamentos como terminais, uma vez que a sua acção acontece na fronteira da rede, à distância de um salto (*hop*). São clientes na medida em que desempenham um papel de consumidor, através da interligação com uma infra-estrutura para acesso a determinado recurso.

No segundo modo de operação a rede é composta por nós móveis que se auto-organizam e cooperam para executar as responsabilidades de funcionamento sem necessidade de uma infra-estrutura pré existente. Surgem então oportunidades de aplicação em ambientes dinâmicos e com características particulares nas quais o uso de uma infra-estrutura fixa não é possível ou desejável.

2.3 Mobile Ad-hoc Networks

Uma rede móvel ad-hoc é constituída por nós móveis formando um sistema autónomo sem necessidade de uma infra-estrutura ou administração.

Devido às limitações inerentes ao meio de transmissão, nomeadamente as suas características de propagação e susceptibilidade a interferências, um factor importante para a comunicação rádio é a linha de visão (LOS - *line-of-sight*). Dadas as características das redes MANET é comum os membros da rede não possuírem *LOS* para todos os outros. Este condicionamento origina o funcionamento *multi-hop* onde um pacote poderá ter que atravessar nós intermédios de forma a atingir o seu destino.

Numa rede deste tipo todos os nós possuem capacidade de movimento e de ligação de forma dinâmica e arbitrária. Adicionalmente constroem e mantêm

informação de encaminhamento, funcionando como *routers*. Em redes Wi-Fi comuns a comunicação é *single-hop*, ou seja o equipamento móvel encontra-se a um "salto" de distância da estação de acesso. As redes MANET, dadas as suas características de organização dinâmica e autónoma, compõem uma topologia variável *multi-hop* onde é possível a entrada e saída de membros.

Uma série de áreas de aplicação têm sido identificadas onde a troca de informação é determinante, que podem beneficiar das capacidades das redes sem fios, em particular da versatilidade das redes móveis Ad-Hoc.

Military environments	Civilian environments	Commercial
- Automated battlefield - Special operations - Homeland defense - Soldiers, tanks, plants	- Disaster Recovery (flood, fire, earthquakes etc) - Law enforcement (crowd control) - Search and rescue in remote areas - Environment monitoring (sensors) - Space/planet exploration - Boats, small aircraft - Sports stadiums - Taxi cab network	- Sport events, festivals, conventions - Patient monitoring - Ad hoc collaborative computing (Bluetooth) - Sensors on cars (car navigation safety) - Vehicle to vehicle communications - Video games at amusement parks

Tabela 1 – Aplicações para MANET (Royer & Toh, 1999)

2.4 Redes de Sensores

Existe uma gama especial de redes móveis ad-hoc denominada Wireless Sensor Networks (WSN). Ainda que considerações semelhantes aproximem as MANET das redes de sensores, as exigências acentuadas destas últimas apresentam desafios ainda maiores em termos de mobilidade, consumo de energia e qualidade de serviço.

As redes de sensores, comparativamente com as MANET, são constituídas por nós com capacidades mais reduzidas de energia, menor capacidade de computação, alcance e transmissão. A densidade da rede é normalmente mais elevada e o seu paradigma de funcionamento passa pela recolha de informação por parte de nós sensores e posterior envio para uma unidade central para processamento, normalmente denominada *sink* (Akyildiz *et al.*, 2002).

2.5 Protocolos de Encaminhamento

Um protocolo de encaminhamento é um mecanismo de manutenção de informação para encaminhamento de mensagens numa rede. É seu objectivo obter e manter a informação necessária que permita que um membro da rede obtenha um caminho para o destino desejado.

Podemos identificar dois grandes tipos de protocolos de encaminhamento: *Link State* e *Distance Vector*. Protocolos *Link State* mantêm informação de conectividade para com os vizinhos, ou seja, estado das ligações ou segmentos aos quais está ligado. Em protocolos *Distance Vector* cada membro da rede mantém uma tabela de distâncias para os destinos na rede alcançáveis através dos seus vizinhos.

2.6 Protocolos Ad-Hoc

Um protocolo de encaminhamento para uma rede móvel Ad-Hoc necessita de ter em consideração uma série de factores que não se colocam habitualmente para uma rede fixa. Nestas circunstâncias surge como necessidade a utilização de protocolos de encaminhamento dinâmicos com capacidades de adaptação a alterações nas condições da rede.

Um dos métodos de classificação de protocolos de encaminhamento é através do seu comportamento na aquisição e manutenção da informação de encaminhamento. Desta forma três grandes grupos de protocolos podem ser identificados para redes Ad-Hoc: proactivos, reactivos e híbridos. Uma outra forma de classificação é através da organização da rede subjacente ao protocolo em funcionamento. Alguns autores apresentam a seguinte divisão:

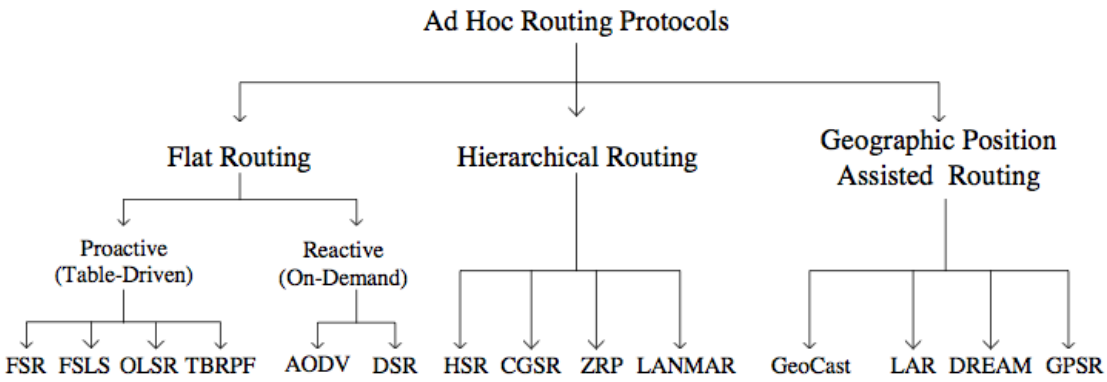


Figura 2.1 – Protocolos Ad-Hoc (Hong *et al.*, 2002)

Os protocolos proactivos caracterizam-se pela sua acção contínua de manutenção de informação de encaminhamento. O seu funcionamento passa pela constante comunicação entre os nós constituintes de uma rede, de forma a possuir um mapa da rede a todos os momentos. São também chamados *Table-Driven* pela sua utilização e actualização de tabelas em cada nó, onde mantém caminhos para todos os outros. Estes caminhos são mantidos de igual forma sem ter em conta a sua frequência de utilização ou necessidade. Este modo de

funcionamento incorre num considerável *overhead* de mensagens de controlo devido aos periódicos anúncios de estado.

Os protocolos proactivos são menos tolerantes a alterações topológicas, no sentido em que uma alteração na morfologia da rede implica a actualização de todas as tabelas.

Os Protocolos Reactivos, também denominados *On-Demand* (ou *Source-Initiated*), por outro lado constroem rotas apenas quando necessário.

O seu funcionamento geral envolve três fases: *Route Request, Route Reply, Route Maintenance*. O nó de origem emite um pedido (*Route Request*) para a rede que vai sendo reencaminhado pelos outros nós até alcançar o destino. Uma vez encontrado o destino, este nó envia um *Route Reply* de volta para a origem. A rota encontrada é mantida enquanto for necessária (*Route Maintenance*). A não aquisição de uma rota resulta na criação de um *Route Error* destinado ao nó de origem.

Os protocolos reactivos são particularmente adequados para redes com características de alta mobilidade, logo com topologia variável e habitualmente com restrições energéticas. A redução considerável de mensagens de controlo é um factor importante na baixa da pegada energética. É de notar no entanto que existe um *atraso* associado à aquisição inicial de uma rota, contrariamente aos protocolos pró-activos onde essa informação se encontra disponível de imediato. Alguns protocolos enviam o primeiro pacote de dados juntamente com *Route Request/Route Discovery* para minorar este efeito.

Os protocolos híbridos que combinam as duas abordagens anteriores tentam utilizar as melhores propriedades de cada um, sendo no entanto mais complexos em termos de comportamento e manutenção.

Uma nota importante para os protocolos Geográficos, que fazem uso de informação de posicionamento em algumas decisões do processo de encaminhamento. A sua utilização foca a optimização de recursos, no sentido em que através de informação geográfica é possível dirigir mensagens para determinada zona, sem necessidade de intervenção de outras áreas da rede.

O acesso a dispositivos com capacidade GPS torna estes protocolos apetecíveis. No entanto a tecnologia GPS é apropriada para operação em exteriores. Para cenários de operação em interiores, novos desafios surgem para utilização de informação de localização em decisões de encaminhamento.

Na subsecção seguinte são analisados alguns dos protocolos utilizados em redes móveis ad-hoc. A sua escolha deveu-se fundamentalmente à sua frequente referência na literatura disponível.

2.6.1 Protocolos Proactivos

Optimized Link State Routing

OLSR (Clausen & Jacquet, 2003) é um protocolo reactivo que opera de forma semelhante aos protocolos *link state* clássicos, onde a informação de controlo é inundada pela rede. Utiliza uma técnica denominada *MultiPoint Relaying*(MPR) para preservar largura de banda e optimizar o processo de difusão de informação de controlo.

Cada nó calcula o seu conjunto de MPRs a partir da sua vizinhança. Um conjunto de *MultiPoint relays* (MPRs) é escolhido de forma que todos os vizinhos *2-hop* consigam ser alcançados através dele e constitui o menor grupo de vizinhos *1-hop* que apresenta esta característica. Assim, ao reduzir o número de nós envolvidos no processo de encaminhamento de mensagens de controlo o processo de difusão é optimizado.

Podem ser identificados três tipos de mensagens definidos pelo OLSR: HELLO, TC (Topology Control) e MID (Multiple Interface Declaration).

As mensagens HELLO são utilizadas para detecção de vizinhança e cálculo do conjunto MPR. O seu conteúdo identifica os endereços de interfaces dos nós vizinhos e o estado das ligações, bem como o conjunto MPR do nó. Uma mensagem HELLO tem um tempo de vida de *1-hop* (TTL=1), significando que apenas alcança a vizinhança local. Estas mensagens não são reencaminhadas em qualquer circunstância.

As mensagens TC distribuem informação topológica da rede, calculada de acordo com a informação recolhida das mensagens HELLO. Apenas os nós eleitos como MPR geram e difundem mensagens deste tipo.

Finalmente, as mensagens MID são utilizadas para declaração de múltiplas interfaces num determinado nó.

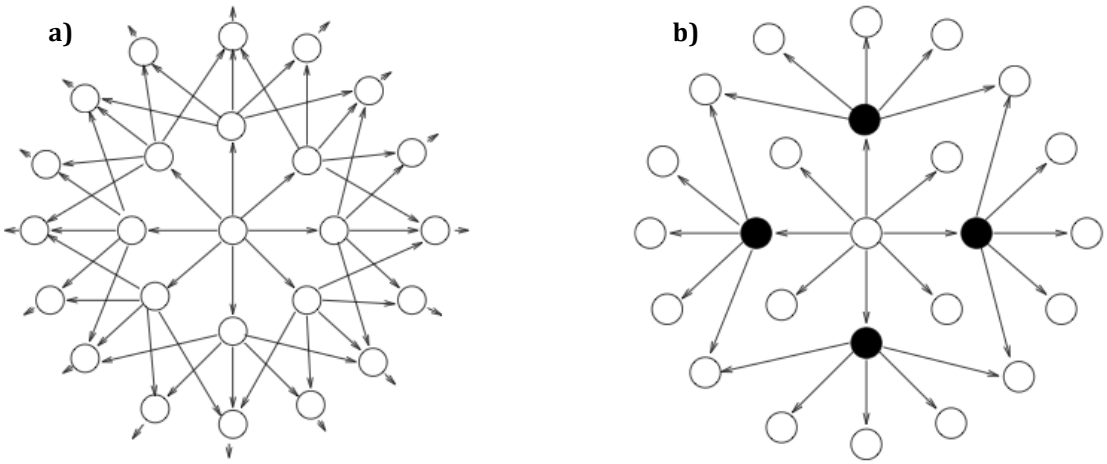


Figura 2.2 – **a)** *Difusão tradicional* **b)** Difusão MPR

A Figura 2.2 mostra dois tipos de difusão: tradicional e difusão MPR. Nós assinalados a preto identificam MPRs.

Destination-Sequenced Distance-Vector

Este protocolo, de forma semelhante aos protocolos proactivos para redes fixas, mantém para cada nó uma tabela com todos os destinos possíveis na rede. O DSDV (Perkins & Bhagwat, 1994), baseado no algoritmo Bellman–Ford, faz uso de números de sequência de forma a identificar rotas obsoletas e a prevenir ciclos.

De forma a manter a consistência de informação, mensagens de controlo com informação de encaminhamento circulam de forma periódica.

Este modo de operação tem alguns inconvenientes para redes sem fios, que possuem largura de banda limitada, dada a frequente circulação de informação. Para minorar este facto as mensagens de actualização de rotas possuem dois modos de operação: o chamado *full dump* e o modo *incremental*. No modo *full dump* é transmitida toda a informação disponível, podendo ser necessário a sua distribuição em vários pacotes. O modo *incremental* apenas actualiza a informação que mudou desde o último *full dump*, contribuindo de alguma forma para reduzir a frequência e quantidade de mensagens de controlo.

As decisões de encaminhamento são tomadas baseando-se numa métrica de número de sequência mais alto e menor número de saltos.

2.6.2 Protocolos Reactivos

Ad-Hoc On-Demand Distance Vector

O protocolo AODV (Perkins *et al.*, 2003), como especificado no *Request for Comments* 3561, funciona sobre UDP , no porto 654. Define quatro tipos de mensagens:

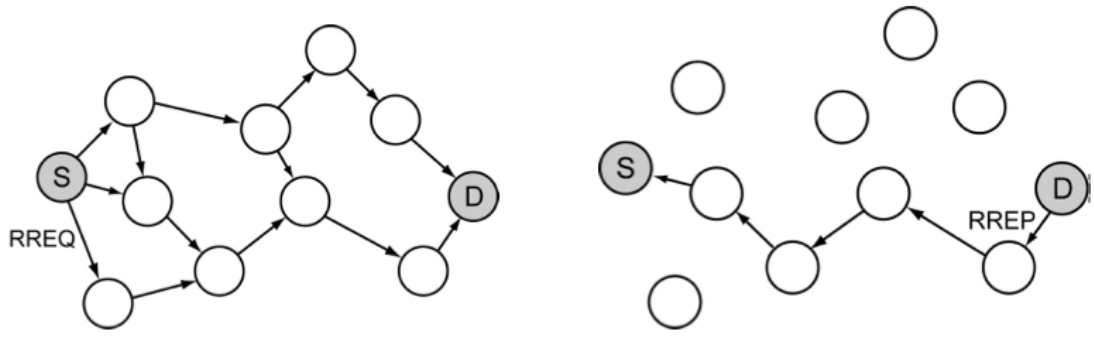
- Route Request (RREQ)
- Route Reply (RREP)
- Route Error (RERR)
- Route Reply Acknowledgment (RREP-ACK)

Tratando-se de um protocolo reactivo, o AODV apenas entra em funcionamento quando existe a necessidade de obtenção de um caminho. O processo começa com o envio de uma mensagem RREQ no modo *broadcast*. Um nó ao receber essa mensagem verifica se possui na sua tabela de routing um caminho para o destino desejado. Em caso afirmativo, é originado um RREP de volta para a origem com informação desse caminho. De outra forma o RREQ é propagado aos vizinhos do nó que executarão semelhante processo.

Uma parte fundamental do funcionamento do AODV é a manutenção de números de sequência, também denominados *Destination Sequence Numbers* e a sua gestão por parte de cada nó. Através deste mecanismo são prevenidos ciclos de encaminhamento e o problema de *counting to infinity* associado com protocolos *Distance Vector*.

Para além deste número de sequência, cada nó é responsável por manter um *broadcast* ID que é incrementado a cada nova RREQ, que juntamente com o IP a identifica univocamente. Uma mensagem RREQ inclui, entre outros campos, o IP de destino e de origem do pacote, bem como números de sequência associados a cada um deles.

Uma estratégia denominada *expanding ring search* é utilizada para minorar os efeitos de difusão de mensagens na rede. O nó originador poderá retransmitir o RREQ a áreas sucessivamente maiores através de incrementos do campo *Time-to-Live* (TTL) do pacote.



(a) Source node S initiates the path discovery process.

(b) A RREP packet is sent back to the source.

Figura 2.3 - Processo aquisição de rota AODV (Jörg, 2003)

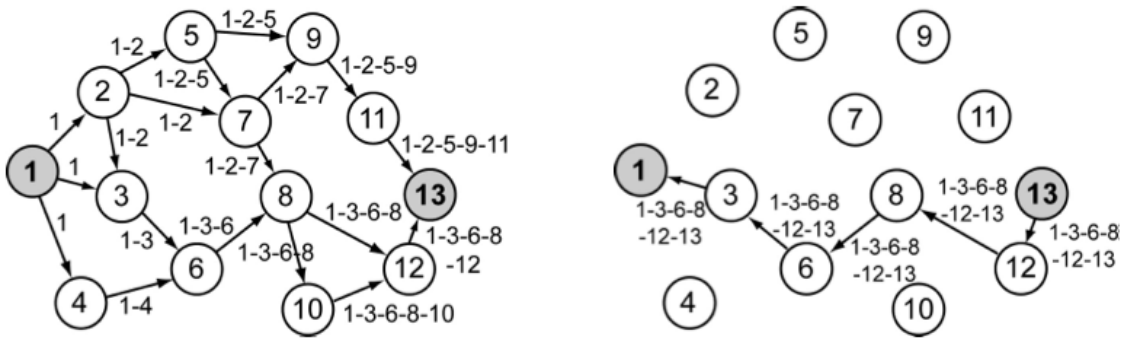
Cada nó mantém uma tabela com informação relativa a caminhos activos. O estado de um caminho pode ser monitorizado através de mensagens HELLO que os nós enviam para os seus vizinhos. Quando é detectada uma falha uma mensagem RERR é utilizada para notificar outros nós do problema. Este processo é tornado possível através da manutenção da chamada *precursor list* que indica que vizinhos poderão utilizar o nó que originou o RERR como caminho para alcançar um destino entretanto tornado inválido.

O protocolo AODV é originalmente um protocolo *unicast*. A especificação não aborda o processamento de mensagens *multicast* ainda que os cabeçalhos das mensagens AODV contenham campos que apontem para a sua possível utilização. O MAODV é uma extensão ao protocolo original onde o tratamento de mensagens *multicast* é abordado.

Dynamic Source Routing

Distingue-se principalmente do AODV pela utilização de *source routing* (Johnson *et al.*, 2007), em que cada pacote transporta no cabeçalho o caminho completo percorrido. Utiliza também um mecanismo de *caching* de rotas que viabiliza a utilização de múltiplos caminhos e actualização de rotas sem participação activa no processo de descoberta, através do modo promíscuo.

Apresenta dois mecanismos principais de funcionamento: *Route Discovery* e *Route Maintenance*. Um nó ao aperceber-se da necessidade de enviar um pacote começa por consultar a sua cache. Se um caminho estiver disponível este é adicionado ao pacote e enviado. Caso contrário uma mensagem *Route request* é difundida. Quando um nó intermédio recebe esta mensagem, verifica a sua cache por um caminho para o destino. Se não encontrar, adiciona o seu endereço e envia para os seus vizinhos. No caso de um caminho estar disponível este é concatenado com o caminho já existente e enviado em percurso inverso para o nó de origem.



(a) Building of the route record. (b) Propagation of the route reply.

Figura 2.4 - Processo aquisição de rota DSR (Jörg, 2003)

2.6.3 Protocolos Híbridos

Zone Routing Protocol

O objectivo deste protocolo é aproveitar as vantagens dos protocolos reactivos e proactivos. O seu funcionamento básico passa por considerar que cada nó possui uma zona centrada em si próprio com um raio definido em termos de *hops*. Os nós abrangidos pela zona fazem uso de protocolos proactivos enquanto que os que se encontram fora do raio definido utilizam protocolos reactivos, para comunicação entre zonas.

Considere-se uma zona com raio 2. Dentro desta, é utilizado um *IntrAzone Routing Protocol* (IARP) proactivo que potencialmente pode ser qualquer protocolo proactivo. Fora da zona recorre-se ao *IntErzone Routing Protocol* (IERP) reactivo que faz uso de pacotes RREQ e RREP à semelhança dos protocolos on-demand.

Figura 2.5 - Zona centrada no nó A com raio 2 (Jörg, 2003)

Numa determinada zona, o IARP garante que existe sempre caminho para os seus nós constituintes. Quando determinado nó não tem informação de como alcançar determinado destino é uma indicação de que esse destino provavelmente se encontrará fora da zona.

Aquando da necessidade de comunicação entre zonas entra em funcionamento o Bordercast Resolution Protocol (BRP) que permite o envio de RREQ para os nós constituintes da fronteira da zona. Estes nós reenviam o pacote para outros nós fronteira até que seja alcançada uma zona que tem como membro o nó de destino.

Trata-se de um protocolo interessante pois tenta fazer uso do melhor dos pró-activos e reactivos. Através do uso de IARP permite o conhecimento constante das ligações dentro da zona, podendo o seu raio de acção ser parametrizado através de *hop count*, desta forma restringindo a circulação constante de mensagens de controlo inerentes dos protocolos pró-activos. Por outro lado é eliminado o *delay* inicial de aquisição de rotas dos protocolos reactivos.

Com o uso de IERP obtém-se a comunicação entre zonas *on-demand*, ou seja, apenas quando o nó de origem e de destino pertencem a zonas distintas. Ao verificar-se esta situação os nós fronteiriços propagam RREQ entre zonas. Um nó

fronteira apenas propaga mensagens RREQ a outros nós fronteira, uma vez que devido ao IARP estes tem conhecimento de toda a informação de encaminhamento da sua zona. Com este mecanismo o *broadcast* de mensagens RREQ fica circunscrito aos nós na fronteira das zonas (Perkins & Bhagwat, 1994).

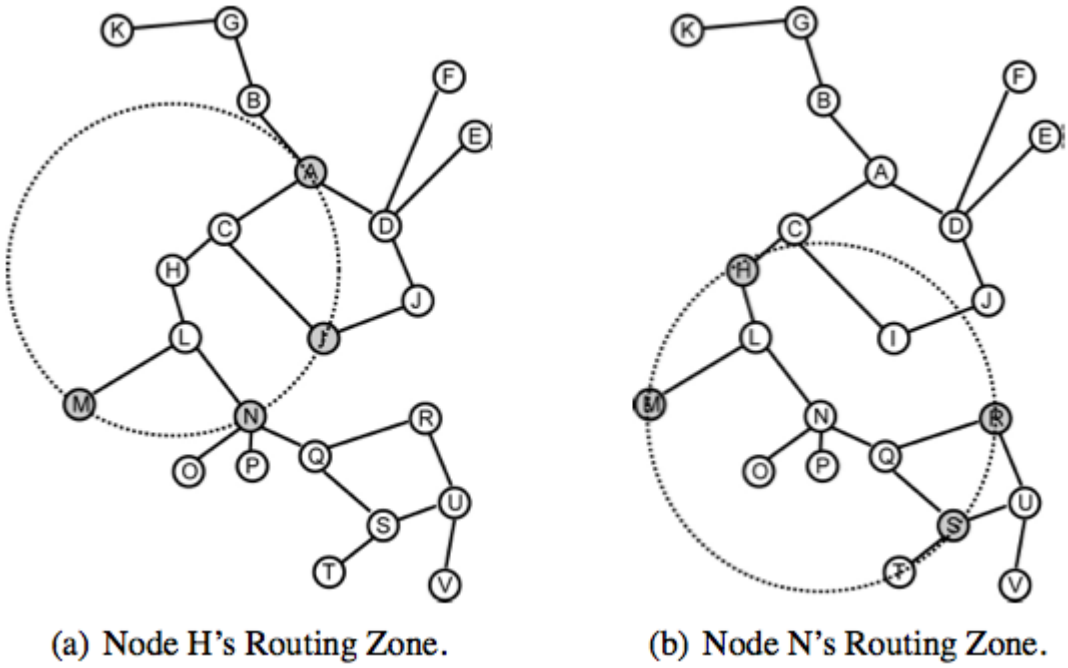


Figura 2.6 - Processo de aquisição de rota ZRP. Nó A pretende comunicar com nó V (Jörg, 2003)

3 Requisitos

O projecto CHOPIN (***C****ooperation between* ***H****uman and r****O****botic teams in catastro****P****hic* ***In****cidents*) é uma iniciativa conjunta entre o Instituto de Sistemas e Robótica da Universidade (ISR-UC), o Centro de Informática e Sistemas da Universidade de Coimbra (CISUC) e a Autoridade Nacional de Protecção Civil (ANPC), com a colaboração dos Bombeiros Sapadores de Coimbra.

É um projecto financiado pela Fundação para a Ciência e Tecnologia e tem como propósito o estudo da cooperação entre equipas de humanos e robôs no desenvolvimento de sistemas de suporte a situações de emergência em cenários urbanos. É constituído por uma equipa interdisciplinar que integra investigação em áreas diversas como cooperação em sistemas multi-robô, mapeamento, detecção de materiais perigosos, algoritmos de patrulhamento ou redes móveis ad-hoc.

Para o caso de estudo proposto, o projecto considera dois cenários de teste: ocorrência de fogo numa garagem de grande dimensão e fuga de gases tóxicos e substâncias radioactivas numa instalação.

No contexto de cenários de busca e salvamento, e de acordo com protocolos em vigor, existe um Centro de Comando que coordena as operações no terreno. Isto significa que de forma geral a actuação dos agentes está dependente da comunicação com esta entidade, ainda que exista, dentro de determinados parâmetros, espaço para actuação autónoma.

Equipas multi-robô são apresentadas como um possível contributo para a melhoria da eficiência e eficácia de operações em cenários de emergência. São apresentados como veículos para um funcionamento distribuído com capacidade de reduzir o risco para agentes humanos e automatizar a recolha de informação pertinente para o decorrer da operação.

De acordo com os cenários e objectivos do projecto CHOPIN verifica-se que não é realista a substituição do elemento humano na intervenção em situações de emergência. Assim, o foco é voltado para a cooperação, de forma a complementar e, se possível, melhorar os processos aplicados actualmente. Mantendo a estrutura actual de um Centro de Comando e equipas de agentes humanos, é introduzido um novo actor para potenciar o conceito de colaboração distribuída.

Desta forma, é possível identificar três actores que intervêm no processo de troca de informação:

- CCO
- Agentes Robóticos
- Humanos

Para que este novo actor possa ser útil impõe-se a constituição de um mecanismo de comunicação que permita, por um lado, fazer chegar a informação ao CCO e por outro que os agentes robóticos possam realizar operações distribuídas de cooperação. É da responsabilidade destes constituir e manter uma infra-estrutura dinâmica de comunicação.

Como mencionado, o CCO mantém o seu papel preponderante nos processos de intervenção. A adição de um novo actor não altera este facto. O CCO continua a ser o ponto de convergência nesta nova estrutura de comunicação, pois será o destino da maioria da informação recolhida, sobre a qual terá de actuar.

As equipas de robôs vão integrar equipas de humanos em tarefas de exploração e patrulhamento, sendo responsáveis pelo mapeamento e recolha de dados do ambiente. Estes agentes cooperam para construir e partilhar mapas parciais do seu meio envolvente, que vão contribuir para a construção de um mapa global no CCO.

Os agentes Humanos irão seguir os protocolos da Protecção Civil existentes, e aconselhados pelo CCO, poderão actuar na informação recolhida pelos agentes robóticos.

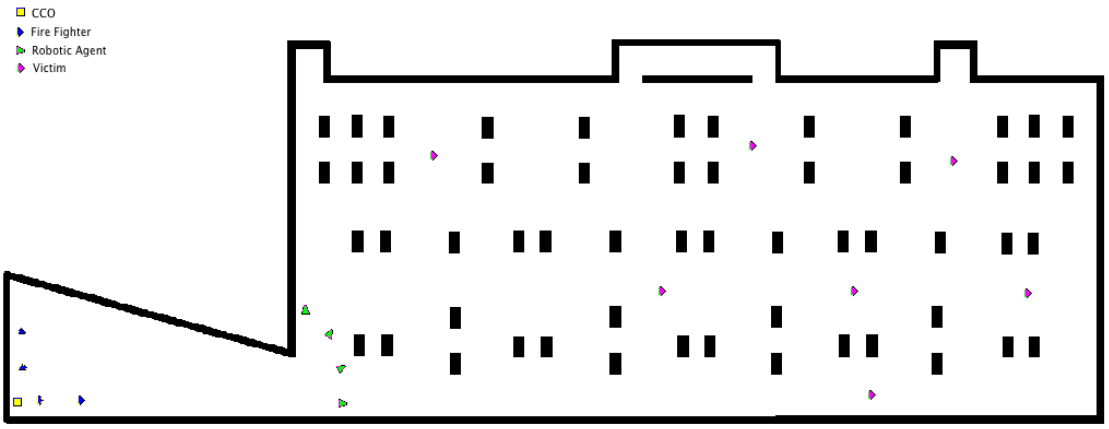


Figura 3.1 - Exemplo de cenário CHOPIN: fogo numa garagem

Na Figura 3.1 é mostrado um exemplo de um cenário CHOPIN, que inclui agentes robóticos. É possível identificar no canto inferior esquerdo a localização do CCO, com agentes humanos. Os agentes robóticos seguem na frente para reconhecimento e detecção avançada de pontos de interesse e vítimas. As vítimas encontram-se espalhadas pelo cenário que é composto por rectângulos pretos, representando as colunas da estrutura do edifício. Este cenário é utilizado em simulações de desempenho dos agentes robóticos na ocorrência de diversos focos de incêndio.

Do ponto de vista da rede todos os actores são nós. No entanto, do ponto de vista do fluxo de informação não desempenham todos o mesmo papel. O CCO é particularmente relevante devido ao seu carácter autoritário. Os agentes Humanos, em determinadas situações têm alguma autonomia. No entanto, de maneira geral, actuam sob indicações do *Comando*. Conclui-se que a maioria da informação viaja na direcção do CCO, com origem nos outros actores, havendo também comunicação no sentido inverso na forma de comandos e orientações.

As equipas de robôs, de acordo com as suas responsabilidades de patrulhamento, mapeamento e recolha de dados sensoriais, têm necessidade de comunicar com elementos da mesma equipa. Ainda que nada impeça que um membro de outro grupo possa reencaminhar mensagens de determinado nó, não se identifica a necessidade de comunicação arbitrária ponto a ponto.

De acordo com as considerações anteriores, são identificados os seguintes padrões de comunicação:

1. Todos os nós tem responsabilidades de encaminhamento

Em cenários de emergência é possível que não exista uma infra-estrutura que possa ser utilizada. A natureza *ad-hoc* da rede requer que todos os membros partilhem responsabilidades de encaminhamento de forma a assegurar a conectividade e comunicação *multi-hop*, uma vez que não existe um ponto de acesso . É espectável que os nós mais próximos do CCO tenham responsabilidades acrescidas uma vez que farão parte da maioria dos caminhos.

2. O CCO comunica com nós individuais e grupos de nós

Como referido, o CCO tem especial relevância uma vez que terá que actuar sobre a informação recolhida pelos Humanos e equipas de robôs. Devido a esta característica o CCO funciona como um mediador nas comunicações.

3. Nós individuais comunicam com o CCO

Grande parte da informação trocada na rede terá como destino o CCO. Os agentes robóticos irão enviar mapas parciais que serão compostos num mapa global do cenário. Dentro do grupo dados de mapeamento serão enviados ao CCO pelo membro que se verificar estar mais próximo. Outros tipos de informação que se espera que circulem na rede serão dados de sensores e pontos de interesse.

4. Nós individuais comunicam com membros do grupo

Para além da cooperação entre nós para manutenção de conectividade e comunicação *multi-hop*, alguns possuem responsabilidades de mapeamento.

Desta forma a comunicação de grupo é maioritariamente direccionada para troca e fusão de mapas parciais e manutenção de vizinhança.

5. Nós sabem o grupo a que pertencem

Quando existe o *deployment* das equipas de Humanos e robôs, cada membro sabe à partida a que grupo pertence. Isto permite comunicação dentro do grupo sem a necessidade de um sistema de identificação para mapeamento de nós e equipas. Adicionalmente é assumido que nós pertencentes ao mesmo grupo se encontram espacialmente próximos e que quaisquer dois nós da mesma equipa podem comunicar através de caminhos nunca exteriores à própria equipa.

Apesar da extensa investigação na área de redes móveis ad-hoc e redes de sensores, verificou-se que as soluções actuais se revelam muito genéricas, não se adequando às necessidades específicas dos cenários em causa.

Por outro lado se é considerado que há uma aproximação aos requisitos das redes de sensores, os protocolos desenvolvidos para estas focam-se principalmente em optimização da utilização de recursos, dadas as características típicas dos equipamentos.

Ainda que os protocolos existentes possam ser soluções válidas para responder aos cenários do projecto CHOPIN, optou-se pela exploração das suas necessidades específicas, resultando numa implementação mais simples conceptualmente e com capacidade de extensão.

4 Arquitectura

O encaminhamento em redes móveis ad-hoc é um domínio extremamente interessante e complexo. Os sistemas operativos actuais não estão nativamente adaptados aos modelos e requisitos de protocolos ad-hoc. Esta realidade obriga a soluções que muitas vezes envolvem programação de sistemas a baixo nível, podendo resultar em sistemas instáveis e implementações não portáveis.

Tradicionalmente, a funcionalidade de encaminhamento encontra-se dividida em duas partes (Kawadia *et al.*, 2002). Uma reside no espaço do utilizador e é responsável por manutenção de informação de encaminhamento e decisões no cálculo de caminhos. A outra, presente no espaço do *kernel*, encaminha pacotes de acordo com um processo baseado em manutenção de tabelas.

O facto de um protocolo ad-hoc ser reactivo ou proactivo tem implicações relevantes ao nível da implementação e interacção com o sistema operativo.

Um protocolo ad-hoc proactivo tem um modo de operação semelhante aos protocolos para redes fixas. A tabela de encaminhamento do *kernel* é povoada por um programa, que funciona como um serviço do sistema operativo, e troca informação com a vizinhança de rede para computação de rotas.

Protocolos ad-hoc reactivos, por seu lado, apresentam desafios adicionais. Devido ao facto de determinadas rotas não se encontrarem prontamente disponíveis, é necessário que a funcionalidade de encaminhamento do *kernel* não descarte pacotes de forma indevida. Alterações de baixo-nível ao sistema operativo são necessárias para tratar pedidos de rotas, havendo a necessidade de armazenamento de pacotes durante o processo de aquisição de uma nova rota e até que a tabela do *kernel* seja actualizada.

As abordagens referidas apresentam vantagens e desvantagens. Por um lado o processamento ao nível do espaço do utilizador simplifica a

implementação e não adiciona complexidade ao nível do *kernel*. Por outro lado cria a necessidade de comunicação inter-contexto – espaço utilizador para espaço privilegiado – que poderá ser indesejada. O processamento ao nível do espaço privilegiado elimina a necessidade dessa interacção, resultando potencialmente num melhor desempenho. No entanto a complexidade acrescida ao nível do núcleo do sistema bem como em termos de implementação e manutenção poderá não ser bem vinda.

4.1 O Protocolo CHOPIN

O funcionamento do protocolo CHOPIN é inspirado no protocolo OLSR. À semelhança deste, trata-se de um protocolo proactivo, que faz uso de mensagens periódicas para recolher e propagar informação sobre o estado da rede. A informação de estado é armazenada em estruturas de dados e tem uma determinada validade, que é verificada através do recurso a eventos temporizados. Também de forma semelhante ao funcionamento do protocolo OLSR, foi tomada a opção da concepção de um serviço que executa ao nível do espaço do utilizador, com alguma interacção com o espaço privilegiado. Esta interacção dá-se através de chamadas de sistema IPC (*Inter-Process Communication*), mais especificamente através da família de *sockets* NETLINK, que é utilizada para troca de informação de rede. As chamadas de sistema são utilizadas para modificação da tabela de encaminhamento IP do sistema operativo.

De notar que o CHOPIN não interfere com quaisquer mensagens de outros serviços. Os serviços da camada aplicacional, nomeadamente as aplicações ROS, utilizam TCP/IP para comunicar e, devem tomar os procedimentos necessários para gerir o envio e recepção de mensagens. A possibilidade de entrega de mensagens a determinado destino depende da presença desse destino na tabela de encaminhamento do *kernel* (responsabilidade do protocolo de encaminhamento) e das funcionalidades de encaminhamento da camada IP (da responsabilidade do sistema operativo).

As considerações apresentadas têm em conta a utilização de placas de rede Wireless 802.11 e o modelo comummente denominado TCP/IP. Apesar do nome, este modelo identifica uma família de protocolos, entre eles o protocolo de transporte UDP. O âmbito da aplicação adequa-se à utilização do protocolo de transporte UDP, dadas as suas características de melhor esforço e ausência de conexão. Dois exemplos de protocolos que utilizam UDP são o DNS e o DHCP.

O CHOPIN recebe mensagens UDP no porto 269 (Chakeres, 2009) e a difusão de mensagens é feita através de *flooding* tradicional, utilizando o endereço de *broadcast* da rede. São definidos dois tipos de mensagens: *Base Station Beacon* e *Node Beacon*. O primeiro é originado num nó definido como *Base Station*, que, no contexto do projecto CHOPIN tem maior relevância, dada a sua importância e influência no decorrer da missão e na difusão de informação. A função deste tipo de mensagem é dar a conhecer aos membros da rede como alcançar este nó. O segundo tipo, *Node Beacon*, é originado por todos os outros nós. Esta mensagem tem duas funcionalidades: dar a conhecer à *Base Station* a topologia de rede e construir a vizinhança de cada nó.

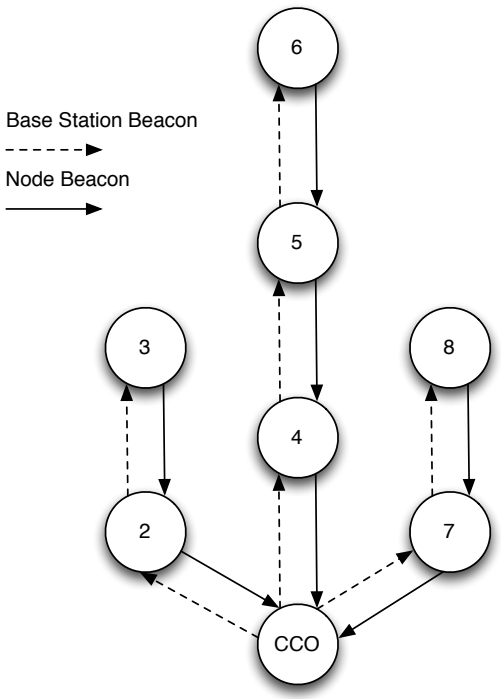


Figura 4.1 – Fluxo de mensagens do protocolo CHOPIN

As mensagens definidas implementam uma parte do RFC5444 (Clausen *et al.*, 2009), que define um formato genérico de pacotes e mensagens para redes móveis ad-hoc. Esta especificação foi inspirada pelo formato de mensagens utilizado originalmente pelo OLSR e foi adoptada pela versão mais recentes deste, o OLSRv2 (Perkins *et al.*, 2013), bem como pelo AODVv2 (Clausen *et al.*, 2013), ambos ainda em desenvolvimento.

O CCO mantém a visão global da rede. Este nó possui caminhos para todos os outros. Cada nó tem um caminho para o CCO bem como informação da sua vizinhança, que deverá constituir o seu grupo. Ao nível dos nós não existe a necessidade de comunicação arbitrária ponto a ponto, sendo que não são mantidos caminhos para todos os outros nós. Este facto não impede a comunicação entre nós de diferentes grupos se a proximidade o permitir.

É assumido que a identificação de nós e grupos se encontra pré definida e é conhecida a nível aplicacional. Desta forma, de acordo com a informação mantida pelo protocolo as aplicações poderão tomar decisões em termos de gestão de grupo e comunicação intra-grupo.

4.2 A Plataforma ROS

Os agentes robóticos tiram partido das capacidades da plataforma ROS para criar aplicações distribuídas de patrulhamento, mapeamento e recolha de dados do ambiente. A comunicação faz-se maioritariamente através de tópicos, num modelo publicação/subscrição.

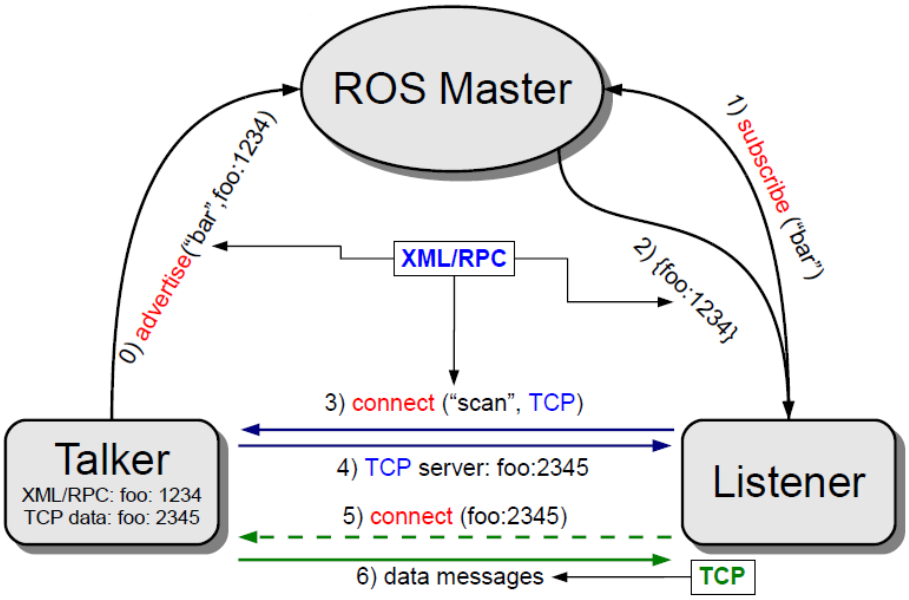


Figura 4.2 – Exemplo de funcionamento de um tópico ROS (Barraquand & Negre, 2012)

Um tópico em ROS é um mecanismo assíncrono de troca de mensagens entre processos. Na Figura 4.2 o processo Talker regista um tópico no ROS Master e o processo Listener faz a subscrição. De notar que este procedimento é assíncrono e independente, de forma que a produção e consumo de informação estão dissociados. Para além da gestão de tópicos e serviços, o ROS Master permite que processos ROS tenham conhecimento uns dos outros. A partir deste momento a comunicação faz-se directamente entre processos, por TCP.

A Figura 4.2 identifica como vários processos no mesmo agente robótico comunicam. Por exemplo, como é que o processo que controla o algoritmo de patrulhamento comunica com o processo responsável pelo deslocamento. Interessa no entanto perceber como os agentes robóticos trocam informação entre si.

A criação de tópicos em sistemas de registo (ROS Master) remotos é feita através do pacote *foreign relay* (Gassend, 2012). Este pacote permite que, determinado agente, ao publicar num tópico local, terá esse tópico replicado num outro agente. Para facilitar a comunicação distribuída recorre-se ao pacote *wifi_comm* (Cabrita & Sousa, 2011), desenvolvido no Instituto de Sistemas e Robótica. Através de informação de vizinhança e encaminhamento, o *wifi_comm*

faz uso da funcionalidade disponibilizada pelo *foreign_relay* para criar tópicos remotos noutros agentes espalhados pela rede.

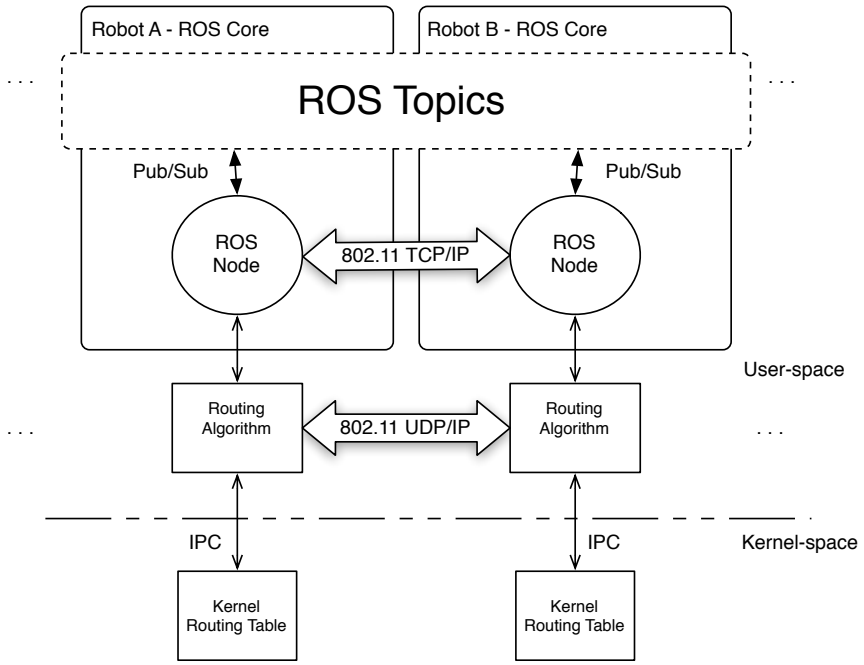


Figura 4.3 – Arquitectura de nós CHOPIN

A Figura 4.3 mostra a arquitectura geral dos agentes robóticos a fazer uso da plataforma ROS para comunicação distribuída. O campo "*ROS Topics*" representa o grupo de tópicos partilhados entre os nós da rede, através do mecanismo referido anteriormente. O protocolo de encaminhamento permite o conhecimento da vizinhança e estrutura de rede necessárias à criação remota de tópicos.

Uma vez que a comunicação através da plataforma ROS é feita por TCP/IP, interessa compreender o caminho de uma mensagem na camada protocolar. Ao nível do TCP, de acordo com a necessidade, determinada mensagem pode ser dividida em várias partes para envio. Cada uma destas partes é denominada por segmento, ao qual é adicionado um cabeçalho TCP, que entre outros campos mantém um "número de sequência". Este número de sequência é utilizado para

garantir que a ordem dos segmentos é mantida e no caso de extravio seja possível requisitar uma retransmissão do conteúdo em falta. De seguida o TCP passa cada um dos segmentos para a camada IP.

Ao nível do IP, a única preocupação deste protocolo é fazer chegar um datagrama ao seu destino. Para encaminhar o datagrama e permitir que sistemas intermédios façam o reencaminhamento, o IP adiciona o seu próprio cabeçalho. No cabeçalho encontram-se, entre outros, o endereço de origem, o endereço de destino e o número do protocolo utilizado, neste caso 6, para o TCP. Como existem vários protocolos que podem fazer uso de IP, esta informação torna-se necessária.

Por outro lado, a maioria dos sistemas faz uso de Ethernet. A este nível, não existe o conceito de endereço IP. A Ethernet tem o seu próprio formato de endereço, MAC *address*, que identifica uma interface (*NIC*) em determinado equipamento. O mapeamento de um endereço IP para um endereço MAC é feito através do ARP (*Address Resolution Protocol*) no caso do IPv4 e NDP (*Neighbor Discovery Protocol*) no caso do IPv6. No caso de 802.11 existe ainda um nível adicional de encapsulamento antes do pacote seguir para a camada física.

No destino, dá-se o processo inverso. É feita uma correspondência com o endereço MAC, é removido o cabeçalho IP e verificado o tipo de protocolo, que no caso do exemplo referido é o TCP. O TCP por sua vez, combina os segmentos necessários para obter a mensagem original, que é passada para o nível aplicacional.

5 Implementação

O protocolo CHOPIN é uma aplicação *multi-threaded* que funciona de forma distribuída. É constituído por duas *threads* principais - uma responsável pela leitura e processamento de mensagens e a outra encarregue do envio. Implementa um formato genérico de mensagens para redes móveis ad-hoc – RFC 5444 – e pretende ser paralelamente um sistema para fácil experimentação e futuro desenvolvimento. Através de um ficheiro de configuração é possível definir alguns parâmetros, entre eles, a interface onde o protocolo irá correr, o endereço IP, o endereço de *broadcast* da rede ou o porto. São também definidos alguns valores que influenciam o comportamento do protocolo, como a frequência de mensagens de controlo ou o tempo de validade de uma ligação.

Neste capítulo são apresentadas as estruturas de dados utilizadas e o formato de mensagens definido, bem como os principais algoritmos responsáveis pelo funcionamento do protocolo.

5.1 Estruturas de dados

O protocolo mantém informação de estado em três estruturas principais. As estruturas definidas são DUPLICATE-SET, LINK-SET e ROUTING-TABLE. Dado que não existe a necessidade de pesquisa ordenada nestas estruturas optou-se pela utilização de *hashtables*, com tempo de acesso O(1).

Para além das três estruturas que mantém informação do funcionamento do protocolo, uma quarta estrutura, OUT-BUFFER, armazena mensagens que esperam para ser escritas para socket.

Duplicate Set

Esta estrutura armazena as mensagens recebidas de forma a evitar processamento de mensagens duplicadas. Para manter esta informação não é necessário armazenar a mensagem completa, basta alguns campos do seu cabeçalho.

orig-addr	msg-type	seq-num	exp-time

Tabela 2 - Estrutura de uma entrada na *hashtable* DUPLICATE-SET

A chave que identifica cada entrada é constituída pelos três primeiros campos apresentados.

Link Set

O OLSR utiliza uma estrutura semelhante para descrever os links entre interfaces locais e interfaces remotas, especificamente as interfaces dos nós vizinhos. O CHOPIN por seu lado mantém informação dos links para os vizinhos e adicionalmente para a Base Station. Esta informação é também utilizada para popular a tabela de encaminhamento do protocolo.

local-addr	neighbor-addr	l-time

Tabela 3 - Estrutura de uma entrada na *hashtable* LINK-SET

Routing Table

Esta estrutura contém a informação que indica um determinado destino e qual o nó seguinte através do qual é possível alcançar esse destino. Através desta informação a tabela de encaminhamento do Kernel pode ser modificada para reflectir os caminhos disponíveis.

destination	next-hop	hop-count

Tabela 4 - Estrutura de uma entrada na *hashtable* ROUTING-TABLE

Out Buffer

Esta é a estrutura é uma fila, com semântica FIFO, onde as mensagens originadas no nó aguardam a escrita para socket. Trata-se de uma estrutura *thread-safe* onde a *thread* leitora coloca mensagens depois de processadas e consideradas para encaminhamento. Por outro lado, as mensagens periódicas geradas pelo protocolo também são colocadas nesta estrutura, por outra *thread,* que é responsável por um temporizador. A *thread* escritora é informada através de um semáforo quando existe conteúdo disponível na fila para escrita.

5.2 Mensagens

As mensagens implementam uma parte do RFC5444, que define um formato genérico de pacotes e mensagens para redes móveis ad-hoc. De forma simplificada, este documento especifica um formato de pacotes, mensagens, blocos de endereços e de atributos. Os atributos são definidos através de uma codificação tipo-tamanho-valor, ou *type-length-value* (TLV) e podem ser associados aos componentes anteriores.

A adopção do RFC5444 apresenta diversas vantagens, nomeadamente a facilidade de processamento, a extensibilidade e a capacidade de reduzir a quantidade e tamanho de pacotes através da inclusão de várias mensagens e uma representação compacta de endereços. A sua concepção está vocacionada também para ser possível tomar determinadas decisões sem ter necessidade de processar toda a mensagem, através da informação presente no cabeçalho.

A estrutura básica de um pacote que obedece à especificação RFC5444 é mostrada de seguida:

```
<packet> := <pkt-header>
            <message>*
```

Figura 5.1 – Estrutura de um pacote RFC5444

Um pacote é então definido por uma cabeçalho e zero ou mais mensagens. O cabeçalho de um pacote pode ser constituído por até quatro campos: *version*, *pkt-flags*, *pkt-seq-num* e *tlv-block*. Os dois últimos campos podem não estar presentes:

```
<pkt-header> := <version>
                <pkt-flags>
                <pkt-seq-num>?
                <tlv-block>?
```

Figura 5.2 – Estrutura de um cabeçalho de pacote RFC5444

Por seu lado uma mensagem é composta por um cabeçalho, um bloco TLV e zero ou mais blocos de endereços acompanhados por blocos TLV. Na presente implementação não foi feito uso de blocos de endereço pelo que não serão aprofundados. Uma mensagem é então constituída por:

```
<message> := <msg-header>
             <tlv-block>
             (<addr-block><tlv-block>)*
```

Figura 5.3 – Estrutura de uma mensagem RFC5444

O cabeçalho da mensagem possui quatro campos mandatários e quatro campos opcionais. Os campos opcionais claramente identificam parâmetros de extrema utilidade no contexto das redes móveis ad-hoc e que podem ser utilizados para tomar decisões de processamento e encaminhamento, sem a necessidade de inspecção adicional da mensagem.

```
<msg-header> := <msg-type>
                <msg-flags>
                <msg-addr-length>
                <msg-size>
                <msg-orig-addr>?
                <msg-hop-limit>?
                <msg-hop-count>?
                <msg-seq-num>?
```

Figura 5.4 – Estrutura do cabeçalho de uma mensagem RFC5444

Finalmente, resta definir o conceito de TLV e bloco de TLVs. Um bloco agrega zero ou mais TLVs associados a um pacote, mensagem ou bloco de endereços. Na implementação actual, apenas a mensagem possui um *tlv-block* associado. O primeiro campo indica o tamanho dos *tlvs* incluídos no bloco.

O TLV adoptado utiliza o campo referente ao tipo - *tlv-type* -, e os parâmetros especificados em *tlv-flags* indicam como interpretar os campos seguintes. No caso em questão indicam que o TLV é composto por um campo *length* e um campo *value*.

```
<tlv-block> := <tlvs-length>
               <tlv>*

<tlv> := <tlv-type>
         <tlv-flags>
         <tlv-type-ext>?
         (<index-start><index-stop>?)?
         (<length><value>?)?
```

Figura 5.5 – Estrutura de um bloco TLV e TLV RFC5444

5.2.1 Tipos de Mensagens

Uma vez adoptado o formato de mensagem segue-se a definição. O protocolo OLSR define entre outras, a mensagem HELLO e a mensagem TC. As mensagens definidas para o protocolo CHOPIN, *Base Station Beacon* e *Node Beacon* apresentam alguma funcionalidade semelhante.

As mensagens HELLO do protocolo OLSR servem o propósito de manutenção da vizinhança, transportam a lista dos seus vizinhos e não podem ser reencaminhadas. A mensagem *Base Station Beacon* funciona como uma mensagem HELLO que é reencaminhada e que em termos de conteúdo, apenas transporta a identificação do nó mais recente por onde passou.

Por outro lado, as mensagens TC do OLSR são utilizadas para propagar informação topológica da rede pelos nós eleitos como MPRs. A mensagem *Node Beacon* funciona como uma mensagem HELLO+TC, onde é utilizada para construção da vizinhança de um nó e acumula o caminho percorrido até alcançar a *Base Station*.

O protocolo transporta como atributo associado a uma mensagem um inteiro ou uma lista de inteiros, representando endereços IPv4.

Base Station Beacon

Como visto anteriormente existe um nó na rede com particular relevância, o que corresponde ao Centro de Comando Operacional. Dada a sua especificidade torna-se útil que, no contexto da rede, este nó tenha conhecimento da topologia. Neste contexto também, denominamos este nó de *Base Station*.

Esta mensagem tem um periodicidade de dois segundos e o no seu bloco TLV transporta um único atributo indicando inicialmente o endereço IP da *Base Station*. À medida que a mensagem é difundida pela rede e atravessa nós intermédios o conteúdo desse TLV é modificado para o IP do nó que foi transitado e que reencaminha a mensagem. O campo <*msg-orig-addr*> nunca é alterado e indica sempre a origem da mensagem.

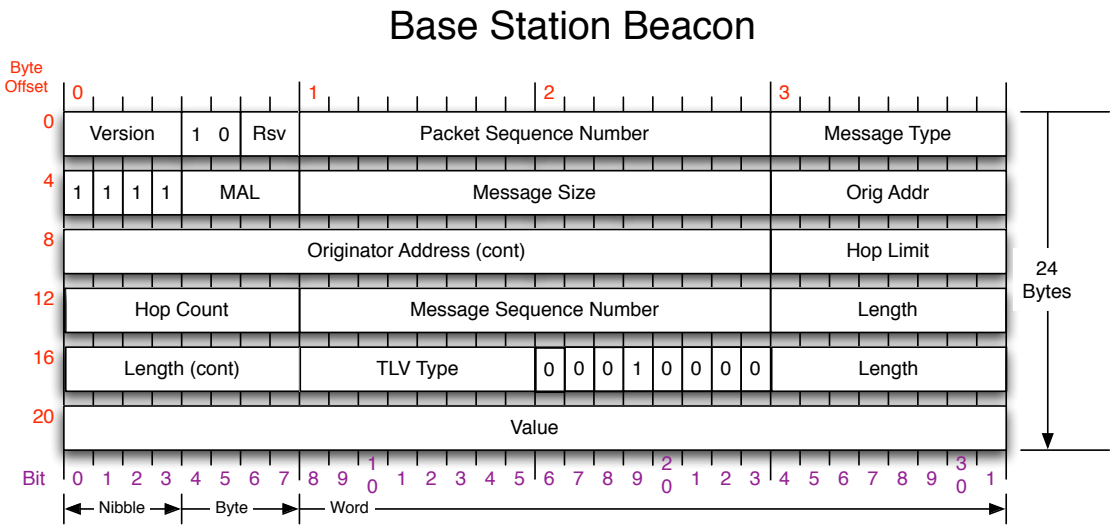


Figura 5.6 – Estrutura de uma mensagem Base Station Beacon de acordo com a especificação RFC5444[4]

Ao atravessar um nó intermédio a mensagem sofre as seguintes alterações:

1. valor do TLV é alterado para IP do nó actual
2. *hop-count* é incrementado
3. *hop-limit* é decrementado

Cada nó intermédio adiciona ou actualiza a entrada referente à *Base Station* na tabela de encaminhamento, indicando o valor do TLV como o próximo *hop*. Desta forma, esta mensagem apenas transporta a informação necessária para que cada nó saiba como alcançar a *Base Station*, ou seja, o nó seguinte no percurso.

[4] Estrutura adaptada de http://www.troyjessup.com/headers/

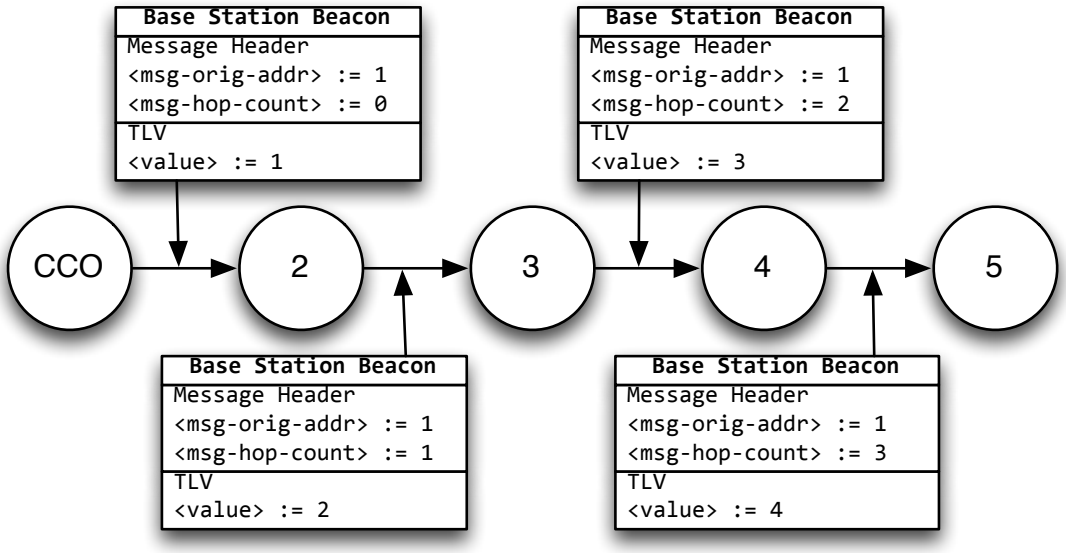


Figura 5.7 – Exemplo do conteúdo de uma mensagem *Base Station Beacon* durante o seu percurso

Node Beacon

A mensagem *Node Beacon* transporta inicialmente o endereço do nó onde foi originada. Ao longo do seu percurso de difusão vai acumulando os endereços dos nós intermédios. Ao alcançar a *Base Station*, a mensagem contém o caminho completo até ao nó de origem. Desta forma a topologia da rede é conhecida ao nível da Base Station, que representa o CCO nos cenários de emergência.

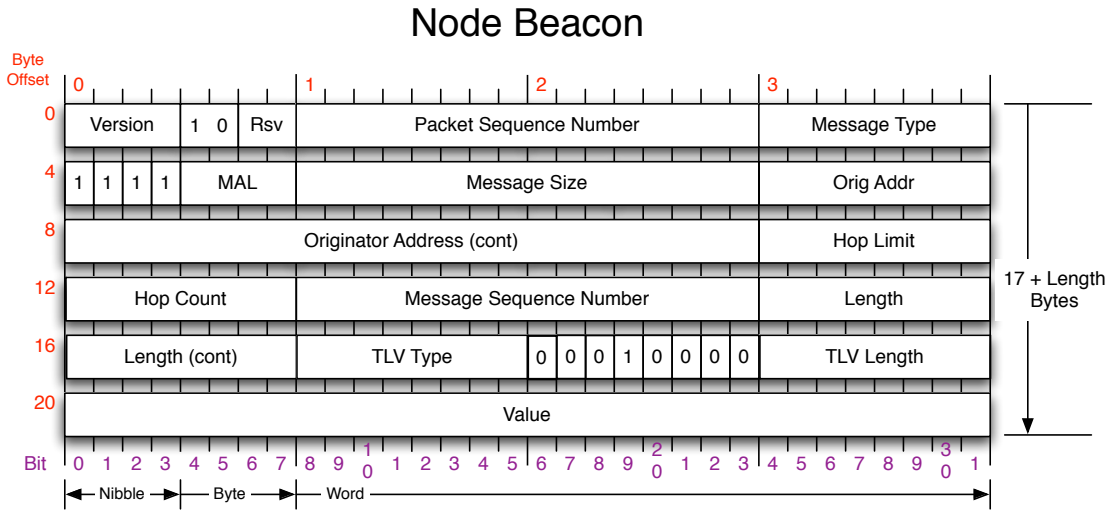


Figura 5.8 – Estrutura de uma mensagem Node Beacon de acordo com a especificação RFC5444[5]

[5] Estrutura adaptada de http://www.troyjessup.com/headers/

Ao atravessar um nó intermédio a mensagem sofre as seguintes alterações:

1. ao valor do TLV é adicionado o endereço IP do nó actual
2. *hop-count* é incrementado
3. *hop-limit* é decrementado

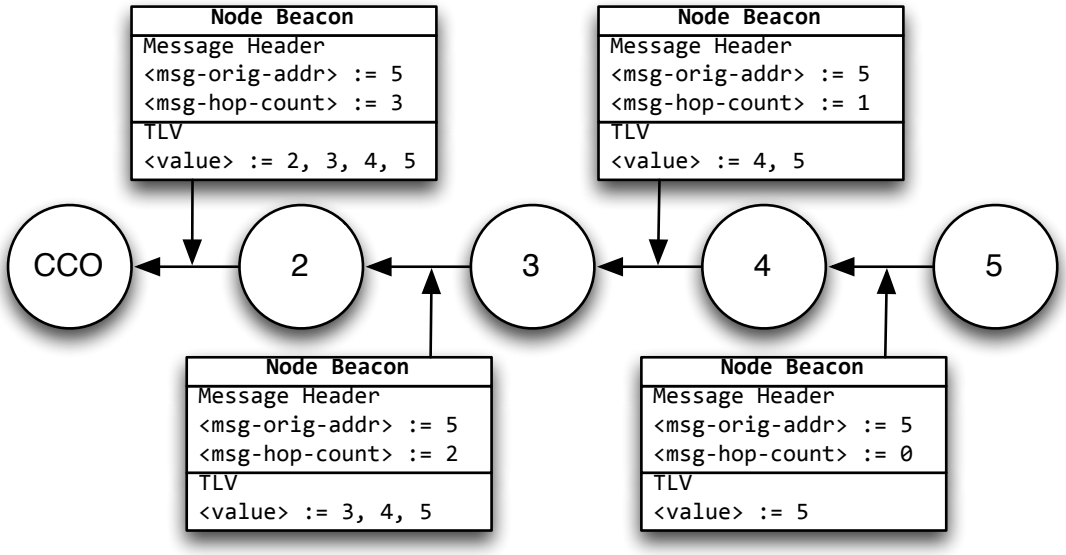


Figura 5.9 - Exemplo do conteúdo de uma mensagem *Node Beacon* durante o seu percurso

5.3 Processamento

Aquando da chegada de uma mensagem esta passa por um procedimento de validação. As primeiras validações são efectuadas apenas com a informação presente no cabeçalho da mensagem, sendo suficiente para descartar a maioria das mensagens que não devem ser consideradas para posterior processamento.

1. Se o tipo de mensagem não se encontra definido, a mensagem deve ser descartada.
2. Se o tempo de vida é igual a zero, a mensagem deve ser descartada.

3. Se o número máximo de *hops* atingir 255, a mensagem deve ser descartada.
4. Se a mensagem foi enviada pelo nó actual, ou seja, o endereço IP do originador da mensagem é o mesmo do nó actual, esta deve ser descartada.
5. Se existir uma entrada na tabela de duplicados com o mesmo endereço de origem, tipo de mensagem e número de sequência, a mensagem já foi processada e deve ser descartada.
6. Se o conteúdo do bloco TLV é inválido a mensagem deve ser descartada. Este conteúdo é inválido quando não contém um endereço IP bem formado, ou uma lista de endereços IP bem formados.
7. De outra forma, a mensagem deve ser considerada para processamento.

```
0:  Receive Message
1:  if Message Type is not valid then discard
2:  else if Hop Limit == zero then discard
3:  else if Hop count == 255 then discard
4:  else if Message Origin Address == Host Address then discard
5:  else if Duplicate Set contains Message then discard
6:  else if TLV Block is not valid then discard
7:  else Process Message
```

Figura 5.10 – Algoritmo de validação de mensagem CHOPIN

Quando uma mensagem é considerada para processamento dá-se a sua análise e as estruturas devidas são actualizadas.

Para começar, a tabela LINK-SET é actualizada. Se já existe uma entrada na tabela para o nó que originou a mensagem, o seu tempo de validade é estendido; de outra forma, é criada uma nova entrada com os campos apresentados na Tabela 3.

```
0:  Menssage for processing
1:  if Exists Link entry for Message then
2:    Update expiration time
3:  else
4:    Create new Link entry
```

Figura 5.11 – Algoritmo de inserção na tabela LINK-SET

De forma semelhante é adicionada uma entrada na tabela DUPLICATE-SET com a estrutura apresentada na Tabela 2.

De seguida a mensagem pode ser reencaminhada. Se a mensagem for do tipo *Base Station Beacon*, o TLV é substituído pelo endereço IP do nó actual, o *hop-count* é incrementado, o *hop-limit* é decrementado e a mensagem é colocada no *buffer* de saída. Se a mensagem for do tipo *Node Beacon*, ao TLV é adicionado o endereço IP do nó actual sendo de seguida observado o mesmo comportamento até colocação no *buffer* de saída OUT-BUFFER.

De notar que se existir um nó assinalado como *Base Station*, apesar de manter as mesmas estruturas que todos os outros, não faz qualquer encaminhamento de mensagens.

5.4 Temporizadores

Para efeitos do funcionamento do protocolo são definidos três temporizadores. Estes são responsáveis pelo envio periódico de mensagens e pela verificação do tempo de validade da informação presente nas tabelas mantidas. A periodicidade destes mecanismos pode ser especificada facilmente através do ficheiro de configuração *.config*[6].

São definidos três temporizadores, identificados da seguinte forma:

[6] Um exemplo do ficheiro .config pode ser consultado no final do Apêndice B

1. *check-duplicate-holding*
2. *check-link-set-validity*
3. *new-beacon*

Tendo em conta que o OLSR é um protocolo com diversos anos de desenvolvimento e amplamente testado, foram adoptados os seus valores propostos para intervalos de emissão e tempos de validade. Foram utilizados os seguintes valores:

- *REFRESH_INTERVAL* = 2 segundos
- *DUP_HOLD_TIME* = 30 segundos
- *NEIGHB_HOLD_TIME* = 3 x *REFRESH_INTERVAL*

```
0:  // Beacon Timer
1:  Generate beacon
2:  Put in out buffer
3:  Signal semaphore
4:  // DUPLICATE-SET Timer
5:  for entry in DUPLICATE-SET do
6:  if validity time is expired then
7:    Remove entry
8:  endif
9:  // LINK-SET Timer
10: for entry in LINK-SET do
11:   if validity time is expired then
12:     Remove entry from LINK-SET
13:     Remove entry from ROUTING-TABLE
14: endif
```

Figura 5.12 – Algoritmos dos temporizadores do protocolo CHOPIN

5.5 Comunicação Inter-Processos

Como referido, o protocolo CHOPIN, como a maioria dos protocolos denominados protocolos de encaminhamento, na verdade não é responsável pelo actual encaminhamento de pacotes. O seu trabalho é manter informação e estruturas através das quais seja possível retirar considerações relativamente à

rede e ao seu estado. O passo seguinte é passar essa informação para o sistema operativo, o qual é responsável pela encaminhamento efectivo de pacotes, mais especificamente a camada IP.

O protocolo CHOPIN, depois da validação de uma mensagem recebida e a sua disponibilização para processamento, como referido, actualiza as estruturas relevantes.

Depois de actualização do LINK-SET e posteriormente da tabela ROUTING-TABLE dá-se a comunicação da informação ao sistema operativo. Como referido no capítulo 4, esta comunicação é feita através de chamadas de sistema, mais especificamente *Inter-Process Communication*. É utilizada a família de *sockets* NETLINK que permite consultar e editar informação de rede do sistema. Esta família de *sockets* disponibiliza vários protocolos, através dos quais é possível aceder a diferentes componentes do *kernel*.

Para comunicação com a tabela de encaminhamento é utilizado o protocolo NETLINK_ROUTE, através dos tipos de mensagem RTM_NEWROUTE e RTM_DELROUTE. Como o nome indica são utilizadas, respectivamente, para adicionar e remover entradas na tabela de encaminhamento do *kernel*.

```
sock = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE)
```

Figura 5.13 – criação de um socket NETLINK para manipulação da tabela de encaminhamento

Esta funcionalidade é desenvolvida em C e compilada como biblioteca partilhada, sendo depois utilizada pelo protocolo CHOPIN na manutenção da tabela de encaminhamento do sistema operativo. O código completo encontra-se disponível no Apêndice C.

```
CFLAGS := -fPIC -g -Wall -Werror
CC := gcc
NAME := kernel_routes
all: lib
$(NAME): $(NAME).o
	$(CC) $(CFLAGS) $^ -o $@
lib: lib$(NAME).so
lib$(NAME).so: $(NAME).o
	$(CC) -shared -Wl,-soname,lib$(NAME).so $^ -o $@
clean:
	$(RM) *.o *.so* $(NAME)
```

Figura 5.14 – *Makefile* para a biblioteca partilhada *kernel_routes*

A Figura 5.15 mostra a visão global do algoritmo do protocolo CHOPIN.

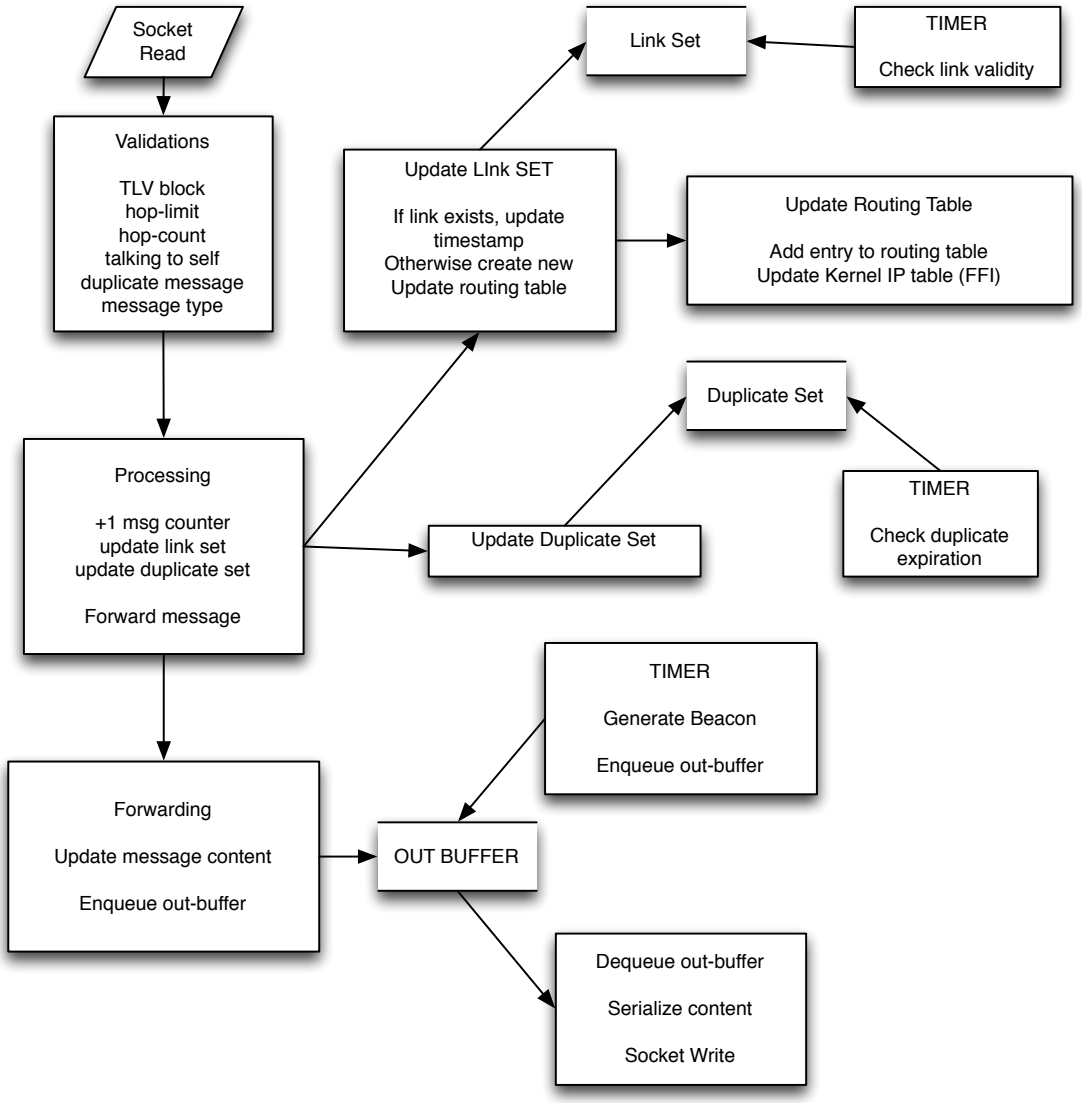


Figura 5.15 – Algoritmo do protocolo CHOPIN

5.6 Configuração

O protocolo CHOPIN foi desenvolvido utilizando a linguagem Common Lisp, mais especificamente a implementação Steel Bank Common Lisp[7]. SBCL é uma das implementações Common Lisp *open source* mais activas e robustas existentes.

Seguem-se os passos necessários para instalar e executar o protocolo CHOPIN, no sistema operativo Ubuntu, a correr o Linux Kernel 3.9.

Um dos membros da rede deve criar uma rede em modo *ad-hoc* para outros se poderem associar. Supõe-se uma configuração estática de rede, com endereços IP privados.

5.6.1 Compilação do SBCL

```
# sudo apt-get install sbcl // instalar sbcl do Ubuntu
// Download SBCL >= 1.1.6
sbcl-1.1.6 # sh make.sh // compilar 1.1.6 com versão previamente instalada
sbcl-1.1.6 # sudo apt-get remove sbcl // uma vez compilada nova versão, remover a antiga
sbcl-1.1.6 # sh install.sh // instalar versao 1.1.6 acabada de compilar
```

5.6.2 Instalação do Quicklisp

Quicklisp é um gestor de bibliotecas para Common Lisp. A sua instalação facilita o acesso a um largo número de bibliotecas e disponibiliza facilidades no desenvolvimento e execução de aplicações Common Lisp.

```
wget –O http://beta.quicklisp.org/quicklisp.lisp
sbcl --load quicklisp.lisp
```

5.6.3 Instalação do Protocolo CHOPIN

```
cd ~/quicklisp/local-projects/
```

[7] http://www.sbcl.org/

```
unzip chopin-routing.zip
cd chopin-routing
cd utils/
./setup.sh <interface>
// escolher opção 3 “Build .config file”
// escolher opção 4 “Build Shared Libs”
```

5.6.4 Execução do Protocolo CHOPIN

```
cd ~/quicklisp/local-projects/chopin-routing/
sudo ./run-daemon.sh
```

6 Testes

Fora de um ambiente de simulação, o teste de redes móveis ad-hoc apresenta bastantes desafios. A simulação permite definir uma dimensão de aplicabilidade muito elevada. Questões de disponibilidade de hardware e até recursos humanos fazem com que as dimensões de um teste real sejam relativamente modestas em comparação. Se por um lado a simulação permite um maior controlo das variáveis da experiência, por outro lado não permite atingir os níveis não determinísticos dos testes empíricos reais (Cavin *et al.*, 2002).

Os cenários de teste foram pensados de forma a testar o funcionamento básico do protocolo bem como a ir de encontro a situações plausíveis no contexto do projecto CHOPIN. O protocolo OLSR foi eleito como termo de comparação, uma vez que serviu de inspiração no desenvolvimento.

Os testes efectuados medem entre outros parâmetros, o tempo de convergência da rede e o débito alcançado, sendo as topologias consideradas simuladas através de pequenas ferramentas desenvolvidas.

6.1 Geração de topologia

Com recurso a um simples programa e uma lista de endereços MAC das máquinas intervenientes foram geradas as topologias a testar. Com recurso à aplicação *Iptables*, foram manipuladas algumas regras de processamento de pacotes na camada de rede de forma a simular a ausência de comunicação em determinadas direcções.

```
(build-topology 1
                (node-spec 5 6)
                (node-spec 6 5 7)
                (node-spec 7 6 2)
                (node-spec 2 7 3)
                (node-spec 3 2 10)
                (node-spec 10 3))
```

Figura 6.1 – Exemplo de construção de topologia para teste

Através do código da Figura 6.1 é gerado um ficheiro com o mapeamento entre cada nó e o endereço MAC de cada um dos seus nós directamente alcançáveis. Este ficheiro é processado e são criadas regras *Iptables* para permitir comunicação com os endereços indicados. Quaisquer outras comunicações são ignoradas.

```
1. sudo iptables -t mangle $2 PREROUTING $idx -m mac --mac-source $1 -j ACCEPT;
2. sudo iptables -t mangle -A PREROUTING -i $IFACE -j DROP;
```

Figura 6.2 – Regras *Iptables* utilizadas

Na regra número 1 da Figura 6.2, a variável *$2* indica a remoção ou adição de uma regra e pode tomar os valores *-D* ou *-I*. A variável *$idx* refere-se ao número da regra e a variável *$1* indica o endereço MAC ao qual a regra será aplicada. A regra número 2 ignora todos os pacotes destinados à interface identificada pela variável *$IFACE*.

A ordem das regras é importante. Desta forma as regras *ACCEPT* surgem primeiro, seguidas da regra *DROP* que ignora tudo o resto.

```
# Test 1
# .5 <-> .6 <-> .7 <-> .2 <-> .3 <-> .10

# 192.168.0.5 (44:6d:57:c5:2f:40)
44:6d:57:c5:4f:e3

# 192.168.0.6 (44:6d:57:c5:4f:e3)
44:6d:57:c5:2f:40
44:6d:57:b5:4d:db

# 192.168.0.7 (44:6d:57:b5:4d:db)
44:6d:57:c5:4f:e3
78:92:9c:87:26:b6

# 192.168.0.2 (78:92:9c:87:26:b6)
44:6d:57:b5:4d:db
78:92:9c:87:0b:2e

# 192.168.0.3 (78:92:9c:87:0b:2e)
78:92:9c:87:26:b6
00:c0:a8:d6:21:77

# 192.168.0.10 (00:c0:a8:d6:21:77)
78:92:9c:87:0b:2e
```

Figura 6.3 – Ficheiro gerado pelo código da Figura 6.1

6.2 Convergência

O intuito deste teste é verificar, após a inicialização do algoritmo em cada nó, quão rapidamente a rede estabiliza. De acordo com as restrições de comunicação criadas para geração da topologia pretendeu-se verificar quão rapidamente os protocolos estabilizam a informação nas suas estruturas de dados.

Para este efeito, uma vez as regras *Iptables* definidas é colocado o primeiro nó a enviar pacotes ICMP para o nó mais afastado, o nó 4, esperando uma resposta. Estes pacotes são enviados através do programa *ping*, disponível em qualquer distribuição Linux. Nesta fase não deverá obter qualquer resposta.

O CCO apenas consegue comunicar com o seu vizinho directo, o nó 1, devendo-se este facto às permissões em efeito e à sua proximidade.

Ao dar-se o arranque dos protocolos é iniciada a troca de mensagens de controlo e as estruturas de informação começam a ser preenchidas. O momento em que o nó inicial começa a receber respostas ao seu pedido ICMP será aquele em que a rede convergiu, e um caminho *multi-hop* foi estabelecido até ao nó de destino.

Para construção de caminhos *multi-hop* é necessário um mínimo de três nós. A experiência foi iniciada com três nós, sendo efectuadas cinco medições para cada um dos protocolos. O processo foi repetido para um número crescente de nós, atingindo no máximo sete.

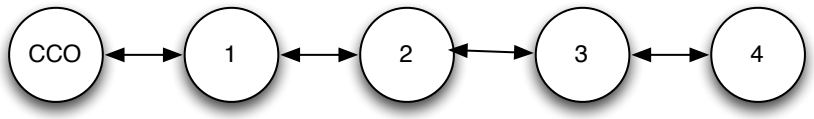


Figura 6.4 – Cenário de Teste 1

Resultados

	3	4	5	6	7
CHOPIN	0:00:11	0:00:12	0:00:13	0:00:13	0:00:12
OLSR	0:00:15	0:00:18	0:00:22	0:00:22	0:00:24

Tabela 5 - Tempo de convergência inicial, para 3 - 7 nós, em segundos

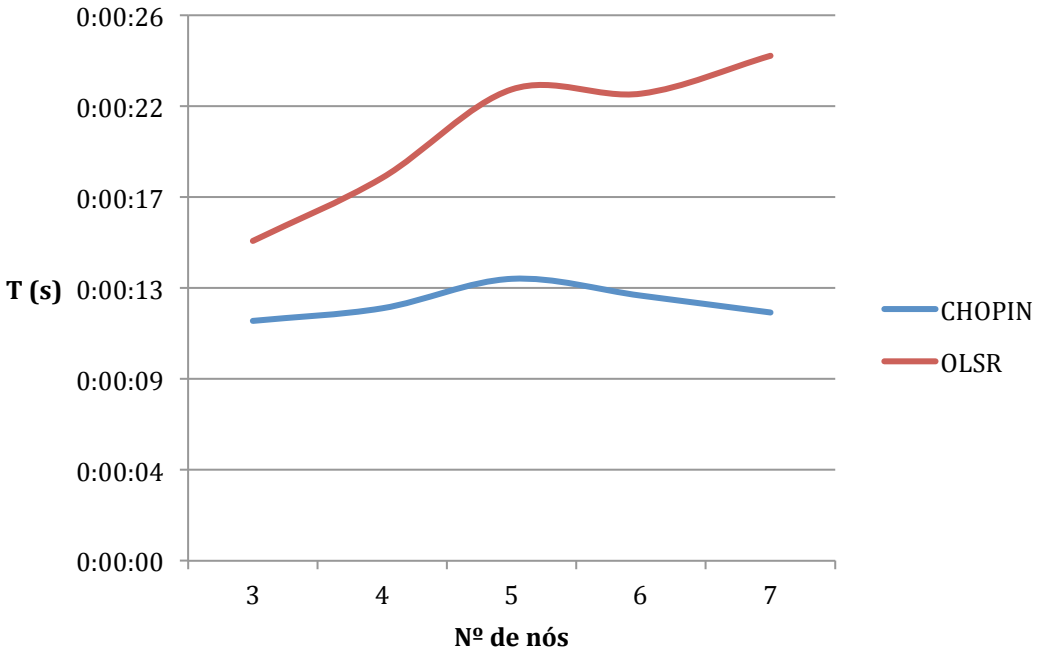


Gráfico 1 - Tempo de convergência inicial, para 3 - 7 nós

A diferença de resultados é devida principalmente à simplicidade do protocolo CHOPIN em comparação com a relativa complexidade do OLSR. Por um lado, o protocolo CHOPIN mantém três estruturas de dados, mais especificamente *hashtables*. O OLSR, por seu lado, mantém oito repositórios de informação e necessita de processamento adicional para a funcionalidade de MPR. Faz ainda uso de um algoritmo *shortest path* para o cálculo dos caminhos.

Relativamente aos tipos de mensagens, ambos os protocolos apresentam dois tipos principais de mensagens, sendo no entanto as mensagens do OLSR mais complexas: este, com as mensagens HELLO e TC; o CHOPIN com as mensagens Base Station Beacon e Node Beacon.

6.3 Throughput

Uma preocupação frequente no desenvolvimento de um protocolo para redes móveis ad-hoc é a gestão da largura de banda. A comunicação *wireless* dá-se num meio volátil e muito susceptível a interferências e perdas. Desta forma é

importante ter em conta a quantidade de tráfego que as aplicações terão disponível para a realização das suas funções.

O teste de *throughput* coloca a rede sobre carga e verifica a média de transferência alcançada. Até três nós é efectuada dez vezes a transferência de um ficheiro de 50MB. Para quatro, e até seis nós, é feita a transferência de um ficheiro de 6MB.

A transferência é iniciada no primeiro nó, com destino ao nó mais afastado. A topologia definida é a mesma utilizada no teste anterior. De notar que para seis nós a informação tem que atravessar cinco nós intermédios até alcançar o seu destino.

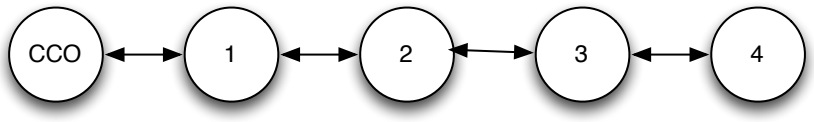


Figura 6.5 – Cenário de Teste 2

Resultados

	2	**3**	**4**	**5**	**6**
CHOPIN	1.38156	0.666709	0.566396	0.0512012	0.0520215
OLSR	0.937461	0.645273	0.557158	0.111758	0.0471094

Tabela 6 - Throughput para 2 - 6 nós

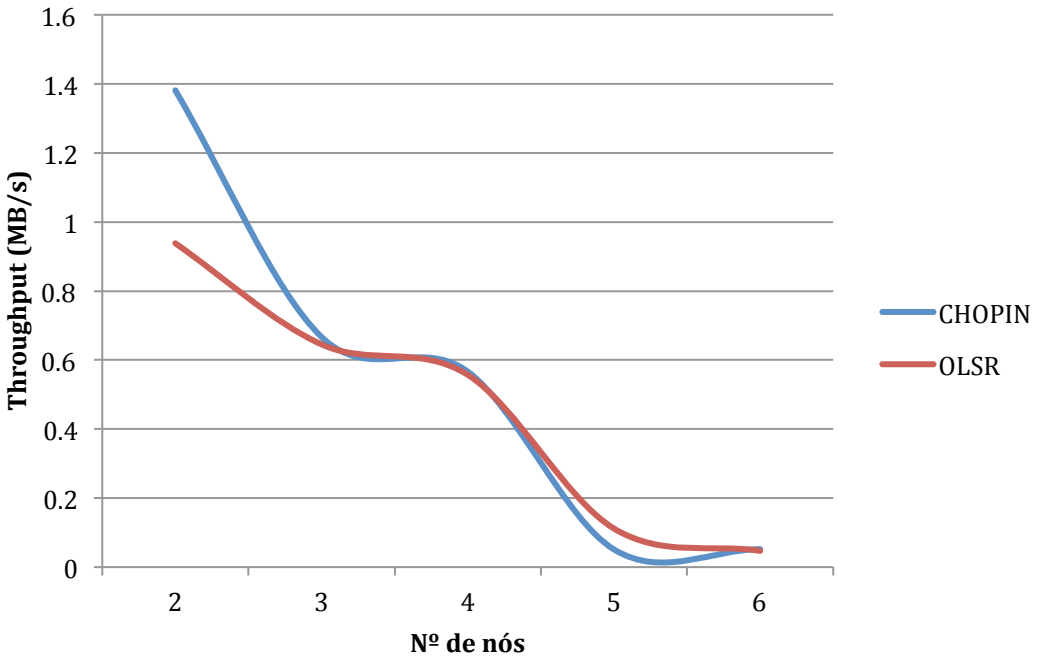


Gráfico 2 – Throughput para 2 - 6 nós

Verifica-se, de forma geral, que o comportamento dos dois protocolos se aproxima em termos de aproveitamento da capacidade da rede. À medida que o número de nós aumenta o débito diminui devido ao funcionamento *multi-hop* e à atenuação de sinal resultante .

6.4 Convergência com reencaminhamento

No âmbito de uma rede móvel ad-hoc, a mobilidade tem uma importância fundamental. Torna-se necessário ter em consideração, que para além das características do meio, uma rede móvel *ad-hoc* irá apresentar falhas de comunicação devido a mobilidade.

A topologia deste teste aproxima-se de uma possível configuração num cenário do projecto CHOPIN. Existe uma estrutura em árvore, com raiz no CCO, de onde se ramificam dois grupos de nós. Para simular a ocorrência de mobilidade é criada alternância de comunicação entre os dois ramos. Considera-se que o nó 5 assinalado na Figura 6.6, faz parte do ramo inferior, e que devido a

mobilidade a sua capacidade de comunicação com o CCO alterna entre o caminho disponibilizado pelo nó 4 e o caminho disponibilizado pelo nó 2.

É inicializada uma transferência do CCO para o último nó. Esta transferência, de um ficheiro de 700MB, coloca a rede sobre carga durante tempo considerável. Para este efeito recorreu-se ao programa SCP, que faz uso do SSH para transporte de dados por TCP. Durante este tempo, com intervalos entre 20 a 40 segundos faz-se alternância entre regras *Iptables* que concretizam as restrições ou permissões de comunicação entre os nós envolventes.

O objectivo do teste é verificar o tempo que a transferência demora a ser retomada na presença da necessidade de reencaminhamento.

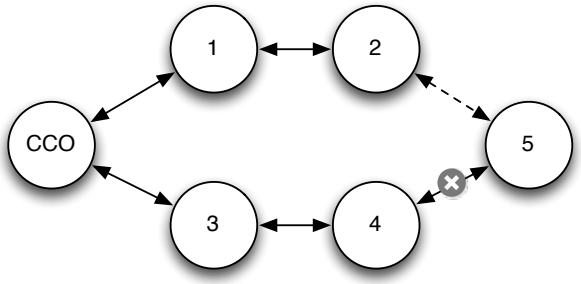


Figura 6.6 – Cenário de Teste 3

Resultados

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
CHOPIN	27	16	13	30	28	33	16	13	13	8
OLSR	69	28	27	27	44	30	93	27	49	35

Tabela 7 – Tempo de convergência com reencaminhamento, em segundos

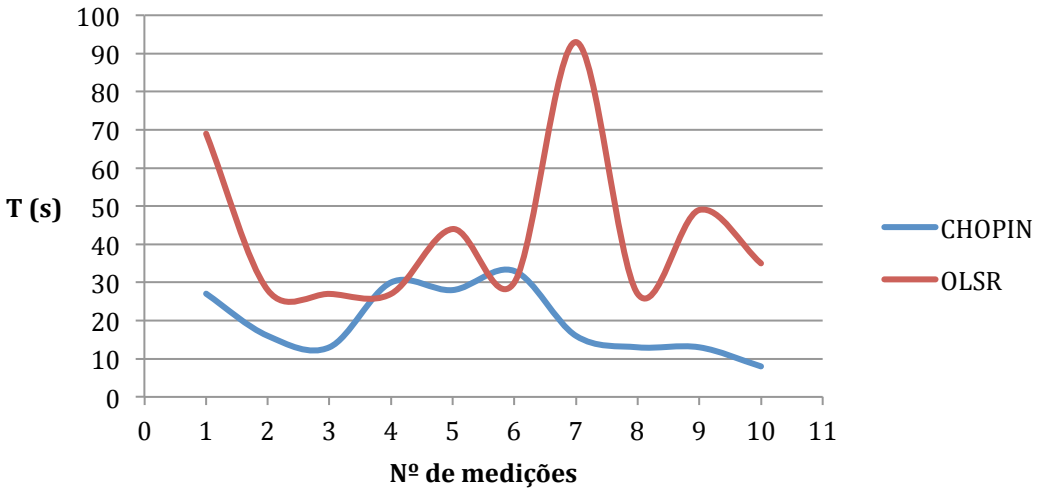


Gráfico 3 – Tempo de convergência com reencaminhamento, em segundos

É possível observar que o OLSR demonstra uma convergência mais lenta que o CHOPIN. Mais uma vez isto deve-se à maior complexidade do protocolo. Na presença de uma alteração topológica o funcionamento do OLSR implica recalcular toda a topologia, uma vez que esta é difundida para todos os nós. Devido à mobilidade pode haver necessidade de proceder a nova eleição de *MPRs*, influenciando o tempo de convergência.

O protocolo CHOPIN, ao manter apenas a informação básica necessária para a manutenção da rede, mais rapidamente se consegue adaptar a alterações topológicas.

Durante a mesma experiência foi medido o número de datagramas UDP transmitidos e recebidos. Esta medição corresponde ao número de mensagens de controlo trocadas pelos protocolos

	1		2		3	
	IN	**OUT**	**IN**	**OUT**	**IN**	**OUT**
CHOPIN	4.195	1.76	2.95	2.76	2.03	3.27
OLSR	2.19	1.73	1.08	0.56	0.84	2.23

Tabela 8 - Média de datagramas UDP enviados/recebidos em 3 nós

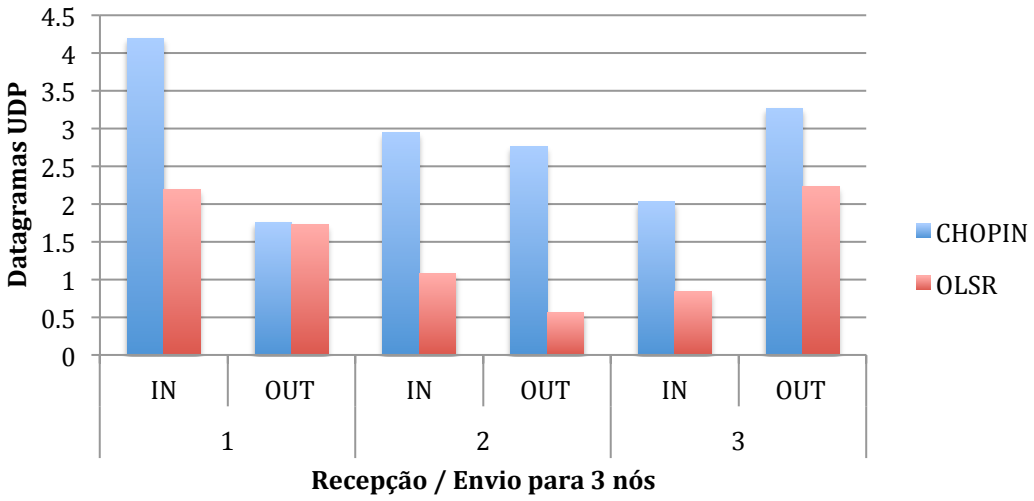


Gráfico 4 - Média de Datagramas UDP enviados/recebidos para 3 nós

O OLSR apresenta cerca de 50% menos mensagens de controlo. Este resultado é devido à utilização do mecanismo *MPRs* para optimização de difusão de mensagens, enquanto que o protocolo CHOPIN nesta altura não possui qualquer mecanismo de optimização, recorrendo à difusão tradicional.

6.5 Ferramentas de teste

Foram utilizados 6-7 computadores portáteis, com sistema operativo Ubuntu, com interfaces WiFi 802.11, formando uma rede ad-hoc com endereços IP estáticos. Sendo o Linux um sistema operativo aberto e por isso extremamente amigável no que diz respeito ao desenvolvimento de ferramentas, não é surpresa que existam bastantes aplicações para monitorizar diversos parâmetros do sistema.

O próprio sistema de ficheiros virtuais *proc* possui já uma grande quantidade de informação, sendo uma forma simples de aceder a estatísticas do sistema, através do espaço do utilizador.

A principal ferramenta utilizada na recolha de dados para os testes foi a aplicação de monitorização *SysStat*[8]. Trata-se de uma colecção de ferramentas de monitorização para Linux, que recolhe estatísticas do sistema através da consulta de informação do sistema de ficheiros *proc* do Linux.

Foi utilizada a ferramenta *sar*, disponível no *SysStat*, que através da opção *–n*, permite filtrar variada informação de rede.

```
sar -n DEV,EDEV,UDP,TCP,ETCP 1 -o utils/stats
```

Figura 6.7 – Comando *sar* utilizado nos testes

Este comando recolhe informação relativa à rede em geral, bem como erros associados. Recolhe também informação específica dos protocolos de transporte UDP e TCP.

Outras ferramentas utilizadas foram o *ping* para envio de pedidos ICMP ECHO e *SCP* para transmissão de ficheiros entre nós. O *SCP* funciona sobre TCP e foi utilizado para simular a troca de informação na rede, que ao nível aplicacional será feita através da plataforma ROS, que também usa maioritariamente tráfego TCP.

[8] http://sebastien.godard.pagesperso-orange.fr/

7 Conclusão

O presente trabalho propõe o protocolo CHOPIN como solução para manutenção de uma MANET em cenários de emergência. De acordo com as necessidades do projecto CHOPIN, fez-se o levantamento de requisitos e concluiu-se que as soluções existentes apresentavam abordagens genéricas, tendo-se optado pela exploração das necessidades específicas do projecto. O protocolo resultante apresenta-se conceptualmente simples, tirando alguma inspiração do OLSR.

A abordagem à implementação foi no sentido da facilidade de prototipagem e desenvolvimento. A escolha da linguagem Common Lisp não foi ocasional. Dadas as suas características permitiu um tipo de desenvolvimento propício à exploração e experimentação interactivos. O resultado foi um sistema de fácil compreensão e com grande capacidade de extensão.

Com os testes realizados pretendeu-se obter uma ideia do desempenho do protocolo, quando comparado com uma solução existente, mais estabelecida. Os resultados afiguram uma perspectiva interessante para futuros desenvolvimentos. O protocolo CHOPIN tem ainda muito espaço para melhoria, apresentando, no entanto, resultados positivos e interessantes quando comparado com o OLSR.

Um dos pontos de possível melhoria é a optimização do processo de difusão de mensagens. É visível nos resultados que o protocolo OLSR utiliza cerca de menos 50% de mensagens de controlo para manutenção da rede. Como referido, deve-se à sua funcionalidade de *MPRs*. Será interessante explorar a redução de mensagens de controlo do protocolo CHOPIN através das capacidades de agregação do RFC5444 e a adição de um método de difusão selectivo.

Está ainda prevista a realização de experiências de patrulhamento cooperativo recorrendo aos robôs *Pioneer* 3DX[9], para inclusão num mostrador para o projecto CHOPIN.

.

[9] http://www.mobilerobots.com/researchrobots/pioneerp3dx.aspx

A. Organização do código

```
c/ - comunicação com kernel por sockets NETLINK para alteração da tabela de encaminhamento
utils/ - script de setup com diversas funcionalidades
.config – parâmetros de configuração do protocolo
library.lisp – definições FFI (Foreign Function Interface) para interacção com código em c/
daemon-class.lisp – definição das mensagens utilizadas
serialization.lisp – rotinas para serialização de mensagens
daemon.lisp – lógica do protocolo
udp-server.lisp – configuração de sockets e threads.
```

B. Protocolo CHOPIN

```
;;; -*- Mode: Lisp -*-
;;; daemon-class.lisp
;;;
(in-package :chopin-routing)

(defstruct config interface host-address broadcast-address port hop-limit refresh-interval
dup-hold-time neighb-hold-time timer-repeat-interval)

(defstruct rt-entry destination next-hop hop-count seq-num)

(defclass duplicate-tuple ()
  ((orig-addr :initarg :orig-addr :accessor orig-addr)
   (msg-type :initarg :msg-type :accessor msg-type)
   (seq-num :initarg :seq-num :accessor seq-num)
   (exp-time :initarg :exp-time :accessor exp-time)))

(defmethod print-object ((object duplicate-tuple) stream)
  (print-unreadable-object (object stream :type t)
    (with-slots (msg-type orig-addr seq-num exp-time) object
      (format stream "~A ~A ~A ~A" msg-type (usocket:hbo-to-dotted-quad orig-addr) seq-num
exp-time))))

(defclass link-tuple ()
  ((local-addr :initarg :local-addr :accessor local-addr)
   (neighbor-addr :initarg :neighbor-addr :accessor neighbor-addr)
   (l-time :initarg :l-time :accessor l-time)))

(defmethod print-object ((object link-tuple) stream)
  (print-unreadable-object (object stream :type t)
    (with-slots (local-addr neighbor-addr l-time) object
      (format stream "~A ~A ~A" local-addr neighbor-addr l-time))))

(defclass packet ()
  ((pkt-header :initarg :pkt-header :reader pkt-header)
   (message :initarg :message :reader message)))

(defmethod print-object ((object packet) stream)
  (print-unreadable-object (object stream :type t)
    (with-slots (message) object
      (format stream "SEQ: ~A ORIG: ~A BLOCK: ~A" (msg-seq-num (msg-header message))
(usocket:hbo-to-dotted-quad (msg-orig-addr (msg-header message))) (tlv (tlv-block
message))))))
```

```
(defclass pkt-header ()
  ((version :initarg :version
            :accessor version
            :type '(unsigned-byte 4))
   (pkt-flags :initarg :pkt-flags
              :accessor pkt-flags
              :type '(unsigned-byte 4))
   (pkt-seq-num :initarg :pkt-seq-num
                :accessor pkt-seq-num
                :type '(unsigned-byte 16)))
  (:default-initargs
   :version 0
   :pkt-flags #b1000
   :pkt-seq-num *pkt-seq-num*))

(defclass message ()
  ((msg-header :initarg :msg-header :reader msg-header)
   (tlv-block :initarg :tlv-block :reader tlv-block)))

(defclass msg-header ()
  ((msg-type :initarg :msg-type
             :accessor msg-type
             :type '(unsigned-byte 8))
   (msg-flags :initarg :msg-flags
              :accessor msg-flags
              :type '(unsigned-byte 4))
   (msg-addr-length :initarg :msg-addr-length
                    :accessor msg-addr-length
                    :type '(unsigned-byte 4))
   (msg-size :initarg :msg-size
             :accessor msg-size
             :type '(unsigned-byte 16))
   (msg-orig-addr :initarg :msg-orig-addr
                  :accessor msg-orig-addr
                  :type '(unsigned-byte 32))
   (msg-hop-limit :initarg :msg-hop-limit
                  :accessor msg-hop-limit
                  :type '(unsigned-byte 8))
   (msg-hop-count :initarg :msg-hop-count
                  :accessor msg-hop-count
                  :type '(unsigned-byte 8))
   (msg-seq-num :initarg :msg-seq-num
                :accessor msg-seq-num
                :type '(unsigned-byte 16)))
  (:default-initargs
   :msg-flags #b1111
   :msg-addr-length #b0011
   :msg-size 0
   :msg-orig-addr (usocket:host-byte-order (config-host-address *config*))
   :msg-hop-limit (config-hop-limit *config*)
   :msg-hop-count 0
```

```
   :msg-seq-num *msg-seq-num*))

(defclass tlv-block ()
  ((tlvs-length :initarg :tlvs-length :accessor tlvs-length
                    :type '(unsigned-byte 16))
   (tlv :initarg :tlv :accessor tlv))
  (:default-initargs
   :tlvs-length 0
   :tlv nil))

(defmethod path-destination ((tlv-block tlv-block))
  "Return last element of `tlv-block'."
  (with-slots (tlv) tlv-block
    (value (first (last tlv)))))

(defmethod next-hop ((tlv-block tlv-block))
  "Return next hop on `tlv-block'."
  (with-slots (tlv) tlv-block
    (value (first tlv))))

(defclass tlv ()
  ((tlv-type :initarg :tlv-type
               :accessor tlv-type
               :type '(unsigned-byte 8))
   (tlv-flags :initarg :tlv-flags
                :accessor tlv-flags
                :type '(unsigned-byte 8))
   (length :initarg :length
             :accessor vlength
             :type '(unsigned-byte 8))
   (value :initarg :value
            :accessor value
            :type '(unsigned-byte 32)))
  (:default-initargs
   :tlv-flags #b00010000
   :length 4)) ; for 32-bit address

(defmethod print-object ((object tlv) stream)
  (print-unreadable-object (object stream :type t)
    (with-slots (tlv-type tlv-flags length value) object
      (format stream "~A ~A ~A ~A" tlv-type tlv-flags length (usocket:hbo-to-dotted-quad
value)))))
```

```
;;; -*- Mode: Lisp -*-
;;; serialization.lisp
;;;
(in-package :chopin-routing)

;; As per RFC 5444 there are some 4 bit fields. Since they occur in pairs, we encode
;; them in a unsigned-byte 8.

(defun merge-4bit-fields (a b)
  (logior (dpb a (byte 4 4) 0)
          (dpb b (byte 4 0) 0)))

(defun extract-4bit-fields (v)
  (values (ldb (byte 4 4) v)
          (ldb (byte 4 0) v)))

(defun version+pkt-flags (pkt-header)
  (merge-4bit-fields (version pkt-header) (pkt-flags pkt-header)))

(defun (setf version+pkt-flags) (value pkt-header)
  (multiple-value-bind (version flags) (extract-4bit-fields value)
    (setf (version pkt-header) version
          (pkt-flags pkt-header) flags)))

(userial:make-accessor-serializer (:pkt-header ph-instance (make-instance 'pkt-header))
                                          :uint8 version+pkt-flags
                                          :uint16 pkt-seq-num)

(defun serialize-pkt-header (pkt-header)
  (userial:serialize :pkt-header pkt-header))

(defun unserialize-pkt-header (pkt-header)
  (userial:unserialize :pkt-header :ph-instance pkt-header))

(defun msg-flags+msg-addr-length (msg-header)
  (merge-4bit-fields (msg-flags msg-header) (msg-addr-length msg-header)))

(defun (setf msg-flags+msg-addr-length) (value msg-header)
  (multiple-value-bind (flags length) (extract-4bit-fields value)
    (setf (msg-flags msg-header) flags
          (msg-addr-length msg-header) length)))

(userial:make-accessor-serializer (:msg-header mh-instance (make-instance 'msg-header))
                                          :uint8 msg-type
                                          :uint8 msg-flags+msg-addr-length
                                          :uint16 msg-size
                                          :uint32 msg-orig-addr
                                          :uint8 msg-hop-limit
                                          :uint8 msg-hop-count
                                          :uint16 msg-seq-num)
```

```
(defun serialize-msg-header (msg-header)
  (userial:serialize :msg-header msg-header))

(defun unserialize-msg-header (msg-header)
  (userial:unserialize :msg-header :mh-instance msg-header))

(userial:make-accessor-serializer (:tlv-block tlv-block-instance (make-instance 'tlv-block))
                                          :uint16 tlvs-length)

(defun serialize-tlv-block (tlv-block)
  (with-accessors ((tlv tlv)) tlv-block
    (userial:serialize :tlv-block tlv-block)
    (dolist (entry tlv)
      (serialize-tlv entry)))
  (userial:get-buffer))

(defun unserialize-tlv-block (tlv-block)
  "A bit of a hack. Since we're using 32bit addresses, the tlv field is always 7 octets."
  (let* ((tlvb (userial:unserialize :tlv-block :tlv-block-instance tlv-block)))
    (setf (tlv tlvb)
            (loop repeat (/ (tlvs-length tlvb) 7)
                    collect (unserialize-tlv (make-instance 'tlv))))
    tlvb))

(userial:make-accessor-serializer (:tlv tlv-instance (make-instance 'tlv))
                                          :uint8 tlv-type
                                          :uint8 tlv-flags
                                          :uint8 vlength
                                          :uint32 value)

(defun serialize-tlv (tlv)
  (userial:serialize :tlv tlv))

(defun unserialize-tlv (tlv)
  (userial:unserialize :tlv :tlv-instance tlv))

(defun serialize-packet (packet)
  "Packet, Message and Tlv-Block are encapsulation. What we want is the bytes from
pkt-header, msg-header and tlv-block."
  (let ((buffer (userial:make-buffer)))
    (userial:with-buffer buffer
      (let ((pkt-header (pkt-header packet))
              (msg-header (msg-header (message packet)))
              (tlv-block (tlv-block (message packet))))
          (serialize-pkt-header pkt-header)
          (serialize-msg-header msg-header)
          (serialize-tlv-block tlv-block)
          buffer))))
```

```
(defun unserialize-packet (buffer)
  "pkt-header, msg-header, tlvs-length, tlvs"
  (userial:with-buffer buffer
    (userial:buffer-rewind)
    (let* ((pkt-header (unserialize-pkt-header (make-instance 'pkt-header)))
           (msg-header (unserialize-msg-header (make-instance 'msg-header)))
           (tlv-block (unserialize-tlv-block (make-instance 'tlv-block))))
      (values pkt-header msg-header tlv-block))))


;;; -*- Mode: Lisp -*-
;;; udp-server.lisp
;;;
(in-package :chopin-routing)

;; sockets
(defvar *writer-thread* nil)
(defvar *reader-thread* nil)
(defvar *semaphore* nil)
(defparameter *broadcast-socket* nil)

(defun start-server ()
  (load-config)
  (let ((socket (usocket:socket-connect nil nil :protocol :datagram
                                                :local-host usocket:*wildcard-host*
                                                :local-port (config-port *config*))))
    (setf (usocket:socket-option socket :broadcast) t)
    (setf *broadcast-socket* socket)
    (setf *semaphore* (sb-thread:make-semaphore))
    (setf *writer-thread*
          (bt:make-thread #'(lambda ()
                              (unwind-protect
                                   (writer socket)
                                (usocket:socket-close socket))) :name "WRITER Thread"))
    (setf *reader-thread*
          (bt:make-thread #'(lambda ()
                              (unwind-protect
                                   (reader socket)
                                (usocket:socket-close socket))) :name "READER Thread"))
    (start-timers)))

(defun reader (socket)
  (loop
   (multiple-value-bind (buf size host port)
       (usocket:socket-receive socket (make-array 128 :element-type '(unsigned-byte 8) :fill-
pointer t) nil)
     (unless (host-address-p (usocket:host-byte-order host))
       (retrieve-message buf size)))))
```

```
(defun writer (socket)
  (loop
   (let ((out (out-buffer-get)))
     (when out
       (usocket:socket-send socket out (length out) :host (config-broadcast-address *config*)
                                    :port (config-port *config*))))
   (sb-thread:wait-on-semaphore *semaphore*)))

(defun stop-server ()
  (setf *out-buffer* (sb-concurrency:make-queue))
  (stop-timers)
  (let ((threads `(,*writer-thread* ,*reader-thread*)))
    (mapcar #'(lambda (th)
                  (let ((cur (shiftf th nil)))
                    (when (and cur (not (eql th (bt:current-thread))))
                      (bt:destroy-thread cur)))) threads)))


;;; -*- Mode: Lisp -*-
;;; daemon.lisp
;;; Project CHOPIN http://chopin.isr.uc.pt

(in-package :chopin-routing)

(defparameter *messages-received* 0)

(defparameter *config* nil)
(defparameter *max-jitter* nil) ; (/ refresh-interval 4)

(defparameter *msg-seq-num* 0) ; wrap-around is 65535
(defparameter *pkt-seq-num* 0) ; same here

(defparameter *base-station-p* nil)

(defparameter *msg-types* '(:base-station-beacon 1 :node-beacon 2))
(defparameter *tlv-types* '(:relay 1 :path 2))

(defparameter *out-buffer* (sb-concurrency:make-queue))

(defparameter *duplicate-set* (make-hash-table :test 'equal))
(defparameter *link-set* (make-hash-table :test 'equal))
(defparameter *routing-table* (make-hash-table :test 'equal))

;; Object Factories
(defun make-pkt-header ()
  (make-instance 'pkt-header :pkt-seq-num (incf *pkt-seq-num*)))

(defun make-msg-header (&key (msg-type :base-station-beacon))
  (make-instance 'msg-header :msg-type (getf *msg-types* msg-type) :msg-seq-num (incf *msg-
seq-num*)))
```

```
(defun make-tlv (value &key (tlv-type :relay))
  (make-instance 'tlv :tlv-type (getf *tlv-types* tlv-type) :value (usocket:host-byte-order
value)))

(defun make-tlv-block (tlvs)
  "Return a `tlv-block' composed of TLVS. Mid-way serialization to obtain TLVS-LENGTH."
  (let ((buff (userial:make-buffer)))
    (userial:with-buffer buff
      (dolist (entry tlvs)
          (serialize-tlv entry)))
    ;; tlvs-length is number of octets of tlvs
    (make-instance 'tlv-block :tlvs-length (length buff) :tlv tlvs)))

(defun make-message (&key msg-header tlv-block)
  (make-instance 'message :msg-header msg-header :tlv-block tlv-block))

(defun make-packet (&key (msg-header (make-msg-header)) tlv-block)
  (make-instance 'packet :pkt-header (make-pkt-header)
                      :message (make-message :msg-header msg-header :tlv-block tlv-block)))

;;; Message Building
(defun build-tlvs (tlv-values &key (tlv-type :relay))
  "Return a `list' of `tlv' instances, based on TLV-VALUES."
  (loop for value in tlv-values
          collect (make-tlv value :tlv-type tlv-type)))

(defun build-packet (msg-header tlv-block)
  (userial:with-buffer (userial:make-buffer)
    (serialize-msg-header msg-header)
    (serialize-tlv-block tlv-block)
    ;; msg-size is size of message including msg-header, that is msg-header+tlv-block
    (setf (msg-size msg-header) (userial:buffer-length))
    (make-packet :msg-header msg-header :tlv-block tlv-block)))

(defun generate-message (&key msg-header (msg-type :base-station-beacon) (tlv-type :relay)
                                tlv-block (tlv-values (list (config-host-address
*config*))))
  "Enqueue `packet' in *OUT-BUFFER*."
  (let* ((msg-header (or msg-header (make-msg-header :msg-type msg-type)))
           (tlvs (unless tlv-block (build-tlvs tlv-values :tlv-type tlv-type)))
           (tlvblock (or tlv-block (make-tlv-block tlvs))))
    (with-accessors ((orig-addr msg-orig-addr) (hop-count msg-hop-count) (hop-limit msg-hop-
limit)) msg-header
      (incf hop-count)
      (decf hop-limit)
    (sb-concurrency:enqueue (build-packet msg-header tlvblock) *out-buffer*)
    (sb-thread:signal-semaphore *semaphore*)))
```

```
(defun new-beacon (msg-type)
  "Enqueue a beacon in *OUT-BUFFER* given `msg-type'."
  (assert (valid-msg-type-p msg-type))
  (sb-concurrency:enqueue (build-packet
                           (make-msg-header :msg-type msg-type)
                           (make-tlv-block (build-tlvs (list (config-host-address
*config*))))) *out-buffer*)
  (sb-thread:signal-semaphore *semaphore*))

(defun message-hash (&rest rest)
  "Generate hash key based on passed arguments."
  (ironclad:byte-array-to-hex-string
   (ironclad:digest-sequence :sha1 (ironclad:ascii-string-to-byte-array (format nil "~{~a~}"
rest)))))

;;; Processing
(defun check-duplicate-set (msg-type orig-addr seq-num)
  "Return T if *DUPLICATE-SET* contains an entry for MSG-TYPE and ORIG-ADDR. Otherwise, return
NIL."
  (gethash (message-hash msg-type orig-addr seq-num) *duplicate-set*))

(defun link-set-params ()
  (values
   (config-host-address *config*)
   (config-refresh-interval *config*)
   (config-neighb-hold-time *config*)))

(defun update-link-set (msg-header tlv-block)
  "Add or update *LINK-SET* entry. For an existing entry update L-TIME. Otherwise, create new
`link-tuple'."
  (multiple-value-bind (local-addr ref-interval neighb-holding)
      (link-set-params)
    (let* ((l-time (dt:second+ (dt:now) (* neighb-holding ref-interval)))
           (ls-hash (message-hash (msg-orig-addr msg-header)))
           (current-link (gethash ls-hash *link-set*)))
      (if (and current-link (equal (neighbor-addr current-link) (next-hop tlv-block)))
          (setf (l-time current-link) l-time)
          (progn
            (setf (gethash ls-hash *link-set*) (make-instance 'link-tuple :local-addr
local-addr :neighbor-addr (usocket:hbo-to-dotted-quad (msg-orig-addr msg-header)) :l-time l-
time))
            (update-routing-table msg-header tlv-block))))))

(defun update-duplicate-set (msg-header)
  "Create a `duplicate-tuple' from MSG-HEADER to be added to *DUPLICATE-SET*."
  (with-slots (msg-type msg-orig-addr msg-seq-num) msg-header
    (setf (gethash (message-hash msg-type msg-orig-addr msg-seq-num) *duplicate-set*)
          (make-instance 'duplicate-tuple :orig-addr msg-orig-addr :msg-type msg-type :seq-
num msg-seq-num :exp-time (dt:second+ (dt:now) (config-dup-hold-time *config*))))))
```

```
(defun update-kernel-routing-table (destination gateway iface metric)
  "Call ADD-ROUTE foreign function to update OS routing table."
  (rcvlog (format nil "KERNEL: iface -> ~A dest -> ~A gw -> ~A metric -> ~A" iface
(usocket:hbo-to-dotted-quad destination) (usocket:hbo-to-dotted-quad gateway) metric))
  #-darwin
  (add-route (usocket:hbo-to-dotted-quad destination) (usocket:hbo-to-dotted-quad gateway)
iface metric))

(defun update-routing-table (msg-header tlv-block)
  "Create `rt-entry' and add to *ROUTING-TABLE*. DESTINATION is the last of the TLV values in
TLV-BLOCK."
  (with-slots (msg-orig-addr msg-seq-num msg-hop-count) msg-header
    (let ((destination (path-destination tlv-block)))
      (setf (gethash (message-hash destination) *routing-table*)
            (make-rt-entry :destination (usocket:hbo-to-dotted-quad destination)
                           :next-hop (tlv tlv-block) :hop-count (1+ msg-hop-count) :seq-num
msg-seq-num))
      (if (= msg-orig-addr (next-hop tlv-block))
          (update-kernel-routing-table destination 0 (config-interface *config*) (1+ msg-hop-
count))
          (update-kernel-routing-table destination (next-hop tlv-block) (config-interface
*config*) (1+ msg-hop-count))))))

(defun del-routing-table (destination)
  (let* ((rt-hash (message-hash (usocket:host-byte-order destination)))
         (rt-entry (gethash rt-hash *routing-table*)))
    (remhash rt-hash *routing-table*)
    #-darwin
    (del-route (rt-entry-destination rt-entry) "0" (config-interface *config*) (rt-entry-hop-
count rt-entry))))

(defun valid-tlv-block-p (tlv-block)
  "Return NIL if `tlv-block' contains invalid tlvs. An invalid tlv contains the current node
address or 0.0.0.0."
  (let ((tlvs (mapcar #'(lambda (x)
                                (value x)) (tlv tlv-block))))
    (and (notany #'zerop tlvs) (notany #'host-address-p tlvs))))

(defun process-message (pkt-header msg-header tlv-block)
  "Update *ROUTING-TABLE*, *DUPLICATE-SET* and *LINK-SET*. If MSG-TYPE is :BASE-STATION-BEACON
broadcast. If MSG-TYPE is :NODE-BEACON unicast to next-hop to Base Station. "
  (with-accessors ((msg-type msg-type) (orig-addr msg-orig-addr) (seq-num msg-seq-num) (hop-
count msg-hop-count) (hop-limit msg-hop-limit)) msg-header
    (incf *messages-received*)
    (update-link-set msg-header tlv-block)
    (update-duplicate-set msg-header)
    (let ((new-tlv-block (make-tlv-block (adjoin (make-tlv (config-host-address *config*))
(tlv tlv-block)))))
      (unless *base-station-p* ; Base Station does not forward messages
          (cond
            ((= msg-type (getf *msg-types* :base-station-beacon))
```

```
            (generate-message :msg-header msg-header :msg-type msg-type :tlv-type :relay
:tlv-block new-tlv-block))
            ((= msg-type (getf *msg-types* :node-beacon))
             (generate-message :msg-header msg-header :msg-type msg-type :tlv-type :path
:tlv-block new-tlv-block))
            (t (rcvlog (format nil "!!!! THIS SHOULD NOT BE REACHED!!!!!")))))))

(defun retrieve-message (buffer size)
  "Unserialize BUFFER and into PKT-HEADER, MSG-HEADER and TLV-BLOCK. Parse MSG-HEADER
according to RFC 5444."
  (multiple-value-bind (pkt-header msg-header tlv-block)
      (unserialize-packet buffer)
    (when (= size (length buffer)) ;; read size must match unserialized length
      (with-accessors ((msg-type msg-type) (orig-addr msg-orig-addr) (seq-num msg-seq-num)
(hop-limit msg-hop-limit) (hop-count msg-hop-count)) msg-header
          (cond
            ((not (valid-tlv-block-p tlv-block)) nil) ; discard
            ((= hop-limit 0) nil) ; discard
            ((= hop-count 255) nil) ; discard
            ((host-address-p orig-addr) (rcvlog (format nil "TALKING TO SELF"))) ; discard
            ((check-duplicate-set msg-type orig-addr seq-num) (rcvlog (format nil
"DUPLICATE"))) ; discard
            ((not (member msg-type *msg-types*)) (rcvlog (format nil "UNRECOGNIZED TYPE")))
;discard
            (t (process-message pkt-header msg-header tlv-block)))))))

(defun out-buffer-get ()
  "Dequeue element from *OUT-BUFFER* and serialize it into a PACKET."
  (let ((packet (sb-concurrency:dequeue *out-buffer*)))
    (when packet
      (serialize-packet packet))))

;;; timer / event scheduling
(defun check-duplicate-holding ()
  "Remove *DUPLICATE-SET* entries with expired timestamp."
  (loop for key being the hash-keys in *duplicate-set* using (hash-value val)
          when (dt:time>= (dt:now) (slot-value val 'exp-time))
          do (remhash key *duplicate-set*)))

(defun check-link-set-validity ()
  "Remote *LINK-SET* entries with expired timestamp."
  (loop for key being the hash-keys in *link-set* using (hash-value link-tuple)
          when (dt:time>= (dt:now) (slot-value link-tuple 'l-time))
          do (progn
               (remhash key *link-set*)
               (del-routing-table (neighbor-addr link-tuple)))))
```

```
(defun start-timers ()
  "Setup and start timers."
  (sb-ext:schedule-timer (sb-ext:make-timer #'check-duplicate-holding :thread t) 10 :repeat-
interval (config-dup-hold-time *config*))
  (sb-ext:schedule-timer (sb-ext:make-timer #'check-link-set-validity :thread t) 10 :repeat-
interval (config-neighb-hold-time *config*))
  (sb-ext:schedule-timer
    (sb-ext:make-timer #'(lambda ()
                           (if *base-station-p*
                               (new-beacon :base-station-beacon)
                               (new-beacon :node-beacon)))
                       :thread t) 0 :repeat-interval (config-refresh-interval *config*))
  (sb-ext:schedule-timer (sb-ext:make-timer #'screen :thread t) 3 :repeat-interval (config-
refresh-interval *config*)))

(defun stop-timers ()
  (dolist (timer (sb-ext:list-all-timers))
    (sb-ext:unschedule-timer timer)))

;;--- TODO(jsmpereira@gmail.com): http://tools.ietf.org/html/rfc5148 Jitter Considerations in
MANETs
(defun jitter (time)
  "Add some noise to TIME."
  (dt:second+ time (- (random (float *max-jitter*)))))

;;; debug
(defun print-hash (hash)
  (loop for k being the hash-keys in hash using (hash-value v)
          collect (format nil "K:~A V:~A~%" k v)))

(defun screen ()
  (with-open-file (s (merge-pathnames "screen" (user-homedir-pathname)) :direction :output
:if-exists :supersede)
    (format s "----- Routing Table -----~%
~A~%
----- Duplicate Set ----~%
~A~%
------ Link Set -----~%
~A~%
------- OUT BUFFER -----~%
~A~%" (print-hash *routing-table*) (print-hash *duplicate-set*) (print-hash *link-set*) (sb-
concurrency:list-queue-contents *out-buffer*))))

(defun rcvlog (&rest rest)
  (with-open-file (s (merge-pathnames "received" (user-homedir-pathname)) :direction :output
                     :if-exists :append)
    (format s "~{~A ~}~%" rest)))
```

```
;;; util
(defmacro with-hash (hash &body body)
  `(loop for k being the hash-keys in ,hash using (hash-value v)
          do ,@body))

(defun load-config (&optional (path "quicklisp/local-projects/chopin-routing/.config"))
  (with-open-file (in (merge-pathnames path (user-homedir-pathname)) :direction :input)
    (let ((conf (read in)))
      (setf *config* (apply #'make-config conf))))
  (setf *max-jitter* (/ (config-refresh-interval *config*) 4)))

(defun valid-msg-type-p (msg-type)
  (getf *msg-types* msg-type))

(defun host-address-p (orig-addr)
  "Return T if ORIG-ADDR equals current node address. Otherwise return NIL."
  (string= (usocket:hbo-to-dotted-quad orig-addr) (config-host-address *config*)))

(defun kernel-table-cleanup ()
  "Loop through *ROUTING-TABLE* and cleanup routing entries from Kernel IP table."
  #-darwin
  (with-hash *routing-table*
    (del-route (rt-entry-destination v) "0" (config-interface *config*) (rt-entry-hop-count
v))))

;;; -*- Mode: Lisp -*-
;;; library.lisp
(in-package #:chopin-routing)

;; Call shared library libkernel_routes to manage Kernel Routing Table
(cffi:define-foreign-library libkernel-routes
  (:unix "~/quicklisp/local-projects/chopin-routing/c/libkernel_routes.so"))
(cffi:use-foreign-library libkernel-routes)

(defun modify-route (dst gw iface metric op)
  "Call to foreign function modify_route. _op_ = 1 add route, otherwise del route."
  (cffi:foreign-funcall "modify_route" :string dst :string gw :string iface :int metric :int
op))
(defun add-route (dst gw iface metric)
  (modify-route dst gw iface metric 1))
(defun del-route (dst gw iface metric)
  (modify-route dst gw iface metric 0))

;;; Exemplo de ficheiro de configuração .config
(:host-address "192.168.0.1"
 :broadcast-address "192.168.0.255"
 :port 269
 :hop-limit 255
 :refresh-interval 2
 :dup-hold-time 30
 :neighb-hold-time 6 ; 3 * refresh-interval)
```

C. Biblioteca kernel_routes

```
#include <errno.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <linux/netlink.h>
#include <linux/rtnetlink.h>
#include <net/if.h>
#include <arpa/inet.h>
#include <unistd.h>

#define BUFFER_SIZE 8192

extern int modify_route(char* destination, char* gateway, char* interface, int metric, int
set)
{
  struct {
    struct nlmsghdr nl;
    struct rtmsg rt;
    char buf[BUFFER_SIZE];
  } req; // RTNETLINK request

    int sock = -1;
    struct sockaddr_nl addr;
    struct msghdr msg;
    struct sockaddr_nl pa;
    struct iovec iov;

    unsigned int if_idx = if_nametoindex(interface);

    int rtn;
    int rtl;
    struct rtattr *rtap;

    /* Setup socket */
    bzero (&addr, sizeof(addr));

    if ((sock = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE)) < 0)
        perror("socket");

    addr.nl_family = AF_NETLINK;
    addr.nl_groups = RTMGRP_IPV4_ROUTE;
```

```
if (bind(sock,(struct sockaddr *)&addr,sizeof(addr)) < 0)
    perror("bind");

// build request
bzero(&req, sizeof(req)); // init request buffer
rtl = sizeof(struct rtmsg);

// Add attributes
// First: destination IP`
rtap = (struct rtattr *) req.buf;
rtap->rta_type = RTA_DST;
rtap->rta_len = sizeof(struct rtattr) + 4;
inet_pton(AF_INET, destination, ((char *) rtap) + sizeof(struct rtattr));
rtl += rtap->rta_len;

// Second: interface index
rtap = (struct rtattr *) (((char *) rtap) + rtap->rta_len);
rtap->rta_type = RTA_OIF;
rtap->rta_len = sizeof(struct rtattr) + 4;
memcpy(((char *) rtap) + sizeof(struct rtattr), &if_idx, 4);
rtl += rtap->rta_len;

// Third: set metric
rtap = (struct rtattr *) (((char *) rtap) + rtap->rta_len);
rtap->rta_type = RTA_PRIORITY;
rtap->rta_len = sizeof(struct rtattr) + 4;
memcpy(((char *) rtap) + sizeof(struct rtattr), &metric, 4);
rtl += rtap->rta_len;

// If present, add Gateway
if (gateway) {
  rtap = (struct rtattr *) (((char *) rtap) + rtap->rta_len);
  rtap->rta_type = RTA_GATEWAY;
  rtap->rta_len = sizeof(struct rtattr) + 4;
  inet_pton(AF_INET, gateway, ((char *) rtap) + sizeof(struct rtattr));
  rtl += rtap->rta_len;
}

req.nl.nlmsg_len = NLMSG_LENGTH(rtl);
req.nl.nlmsg_flags = NLM_F_REQUEST;

if (set)
  req.nl.nlmsg_flags |= NLM_F_CREATE | NLM_F_REPLACE;

req.nl.nlmsg_type = set ? RTM_NEWROUTE : RTM_DELROUTE;

req.rt.rtm_family = AF_INET;
req.rt.rtm_table = RT_TABLE_MAIN;
req.rt.rtm_protocol = RTPROT_STATIC;
req.rt.rtm_scope = RT_SCOPE_UNIVERSE;
```

```
    req.rt.rtm_type = RTN_UNICAST;
    req.rt.rtm_dst_len = 32;

    // send request to kernel
    bzero(&pa, sizeof(pa));
    pa.nl_family = AF_NETLINK;

    bzero(&msg, sizeof(msg));
    msg.msg_name = (void *) &pa;
    msg.msg_namelen = sizeof(pa);

    iov.iov_base = (void *) &req.nl;
    iov.iov_len = req.nl.nlmsg_len;
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;

    rtn = sendmsg(sock, &msg, 0);

    if((rtn = sendmsg(sock, &msg, 0)) < 0){
      fprintf(stderr, "ERROR: %s\n", strerror(errno));
      exit(1);
    }

    /* Close socket */
    close(sock);

    return 0;
}


int main(int argc, char **argv) {

  if (argc < 5) {
    printf("Usage: modify_route <dst> <gw> <if> <metric> <op>\n");
    exit(0);
  }

  modify_route(argv[1], argv[2], argv[3], atoi(argv[4]), atoi(argv[5]));

  return 0;
}
```

D. Diff wifi_comm

Alterações ao pacote ROS *wifi_comm* de forma a remover a dependência para com o OLSR e permitir a recolha de informação de rede directamente do sistema operativo. Desta forma é possível fazer a publicação da vizinhança de um nó num tópico ROS, independentemente do protocolo utilizado.

```
--- /Users/josesantos/wifi_discovery_node_orig.cpp
+++ /Users/josesantos/wifi_discovery_node.cpp
@@ -58,101 +58,54 @@
     return str;
 }

-/*
- * EXAMPLE OF THE INPUT
-Table: Neighbors
-IP address      SYM     MPR     MPRS    Will.   2 Hop Neighbors
-192.168.10.4    NO      NO      NO      6       1
-192.168.10.8    YES     NO      NO      3       1
-192.168.10.7    YES     YES     NO      3       2
-*
-Table: Links
-Local IP        Remote IP       Hyst.   LQ      NLQ     Cost
-192.168.10.6    192.168.10.8    0.00    1.000   1.000   1.000
-192.168.10.6    192.168.10.4    0.00    1.000   0.000   INFINITE
-192.168.10.6    192.168.10.7    0.00    1.000   1.000   1.000
- *
- */
-
 void getNeighboursInfo(wifi_comm::WiFiNeighboursList * neighbours)
 {
-       FILE *ptr;
-       char buf[BUFFSIZE];
-       char link_cmd[] = "wget -q -O - localhost:8080/link";
-       std::vector<std::string> neighboursList;
-       std::string myIP;
-       std::vector<int> signalList;
+  FILE *ptr;
+  char buf[BUFFSIZE];
+  char cmd[] = "route -n";
+  std::vector<std::string> neighboursList;
+  std::vector<int> metricList;

-       // check if can be used
-       if((ptr = popen(link_cmd, "r")) != NULL)
-       {
-               //read the first line
-           fgets(buf, BUFFSIZE, ptr);
+  if ((ptr = popen(cmd, "r")) != NULL) {
+    fgets(buf, BUFFSIZE, ptr); // ignore title
+    fgets(buf, BUFFSIZE, ptr); // ignore header
+    fgets(buf, BUFFSIZE, ptr); // ignore network
+    fgets(buf, BUFFSIZE, ptr); // ignore network

-           if(strcmp(buf, LINK_HEADER) == 0)
-           {
-                       ROS_INFO("buffer:%s",buf);
-                       ROS_WARN("Received Header is not a valid Header (%s)", LINK_HEADER);
-                       exit(1);
```

```
-               }
-
-               // read next line and ignore - column description
-               fgets(buf, BUFFSIZE, ptr);
-
-               // parsing variables
-               char my_ip[16], other_ip[16];
-               float hyst, lq, nlq, cost;
-               std::string ip_str;
-
-               while(fgets(buf, BUFFSIZE, ptr) != NULL && strlen(buf) > 1)
-               {
-                       //ROS_INFO("rcv:%s", buf);
-                       sscanf (buf,"%s %s %f %f %f %f", my_ip, other_ip, &hyst, &lq, &nlq,
&cost);
-                       //ROS_INFO("Parsed - my:%s   other:%s  H:%2.3f  LQ:%2.3f  NLQ:%2.3f
cost:%2.3f", my_ip, other_ip, hyst, lq, nlq, cost);
-
-                       // my IP
-                       myIP = char2string(my_ip);
-                       // other IP
-                       ip_str = char2string(other_ip);
-
-                       // add to neigbour vector if link quality (LQ) is not zero
-                       // if LQ is zero the IP can be listed but is not connected
-                       if(lq > 0.0)
-                       {
-                               neighboursList.push_back(ip_str);
-
-                               int ilq = (int)(lq*100);
-                               // add to link quality vector
-                               signalList.push_back(ilq);
-                       }
-               }
-
-               neighbours->self_ip = myIP;
-               // clear neigbours buffer from MSG
-               neighbours->neighbours.clear();
+    while(fgets(buf, BUFFSIZE, ptr) != NULL && strlen(buf) > 1) {
+      // parsing variables
+      char destination[16], gw[16], mask[16], flags[6];
+      int metric;
+      std::string ip_str;
+
+      sscanf(buf,"%s %s %s %s %d", destination, gw, mask, flags, &metric);
+
+      ip_str = char2string(destination);

-               // create a vector<string>::iterator and set it to the beginning of the
vector
-               std::vector<std::string>::iterator it;
-
-               int index;
-               // Now, we iterate through the array until the iterator exceeds
-               for(it = neighboursList.begin(); it != neighboursList.end(); it++)
-               {
-                       //ROS_INFO("index:%d ", index);
-                       // list index
-                       index = int(it - neighboursList.begin());
-
-                       // one node
-                       wifi_comm::WiFiNeighbour * nod = new wifi_comm::WiFiNeighbour();
-
-                       nod->ip = (*it);
-                       nod->quality = signalList[index];
-                       neighbours->neighbours.push_back(*nod);
-               }
-       }
-       pclose(ptr);
+      neighboursList.push_back(ip_str);
+      metricList.push_back(metric);
+    }
+    neighbours->neighbours.clear();
+
```

```
+    std::vector<std::string>::iterator it;
+
+    int index;
+    // Now, we iterate through the array until the iterator exceeds
+    for(it = neighboursList.begin(); it != neighboursList.end(); it++)
+      {
+       // list index
+       index = int(it - neighboursList.begin());
+
+       // one node
+       wifi_comm::WiFiNeighbour * nod = new wifi_comm::WiFiNeighbour();
+
+       nod->ip = (*it);
+       nod->quality = metricList[index];
+       neighbours->neighbours.push_back(*nod);
+      }
+  }
+  pclose(ptr);
 }
-

 // ******************************************************************************
 // Main function for the multiRobotCom node
```

Referências

Abramson, N. THE ALOHA SYSTEM - Another alternative for computer communications.

Abramson, N., & Schwartz, M. (2009). The Alohanet - surfing for wireless data [History of Communications]. IEEE Communications Magazine , 47 (12), 21-25.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. Computer Networks , 38 (4), 393–422.

Barraquand, R., & Negre, A. (2012). The ROS Framework at a Glance. Obtido de http://barraq.github.io/fOSSa2012/slides.html

Binder, R., Abramson, N., Kuo, F., Okinaka, A., & Wax, D. (1975). ALOHA packet broadcasting: a retrospect. AFIPS '75 Proceedings of the May 19-22, 1975, national computer conference and exposition, (pp. 203-215).

Cabrita, G., & Sousa, P. (2011). wifi_comm - ROS Wiki. Obtido de ROS Wiki: http://www.ros.org/wiki/wifi_comm

Cavin, D., Sasson, Y., & Schiper, A. (2002). On the accuracy of MANET simulators. Proceedings of the second ACM international workshop on Principles of mobile computing (pp. 38-43). Toulouse: ACM.

Chakeres, I. D. (Março de 2009). IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols. Obtido de http://tools.ietf.org/html/rfc5498

Chlamtac, I., Conti , M., & Liu, J. J.-N. (2003). Mobile ad hoc networking: imperatives and challenges. Ad Hoc Networks , 1 (1), 13-64.

Clausen, T. H., & Jacquet, P. (October de 2003). Optimized Link State Routing Protocol (OLSR). Obtido de The Internet Engineering Task Force (IETF) - IETF Tools: http://tools.ietf.org/html/rfc3626

Clausen, T. H., Dearlove, C. M., Dean, J. W., & Adjih, C. (Fevereiro de 2009). Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format. Obtido de The Internet Engineering Task Force (IETF) - IETF Tools: http://tools.ietf.org/html/rfc5444

Clausen, T. H., Dearlove, C., Jacquet, P., & Herberg, U. (23 de Março de 2013). The Optimized Link State Routing Protocol version 2. Obtido de http://tools.ietf.org/html/draft-ietf-manet-olsrv2-19

Gassend, B. (2012). foreign_relay - ROS Wiki. Obtido de ROS Wiki: http://www.ros.org/wiki/foreign_relay

Hong , X., Xu, K., & Gerla, M. (2002). Scalable routing protocols for mobile ad hoc networks. IEEE Network , 16 (4), 11-21.

IEEE-SA. (2012). 802.11-2012 - IEEE Standard for Information technology -- Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

Jörg, D. O. (2003). Performance Comparison Of MANET Routing Protocols In Different Network Sizes.

Johnson, D. B., Maltz, D. A., & Hu, Y.-C. (Fevereiro de 2007). The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. Obtido de The Internet Engineering Task Force (IETF) - IETF Tools: http://tools.ietf.org/html/rfc4728

Kawadia, V., Zhang, Y., & Gupta, B. (2002). System Services for Implementing Ad-Hoc Routing Protocols. International Workshop on Ad Hoc Networking.

Perkins, C. E., & Bhagwat, P. (1994). Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers., (pp. 234-244).

Perkins, C. E., Belding-Royer, E. M., & Das, S. R. (Julho de 2003). Ad hoc On-Demand Distance Vector (AODV) Routing. Obtido de The Internet Engineering Task Force (IETF) - IETF Tools: http://tools.ietf.org/html/rfc3561

Perkins, C. E., Ratliff, S., & Dowdell, J. (25 de Fevereiro de 2013). Dynamic MANET On-demand (AODVv2) Routing. Obtido de The Internet Engineering Task Force (IETF) - IETF Tools: http://tools.ietf.org/html/draft-ietf-manet-aodvv2-00

R. BINDER, *. N. ALOHA packet broadcasting - A retrospect.

Royer, E. M., & Toh, C.-K. (1999). A review of current routing protocols for ad hoc mobile wireless networks. Personal Communications , 6 (2), 46-55.