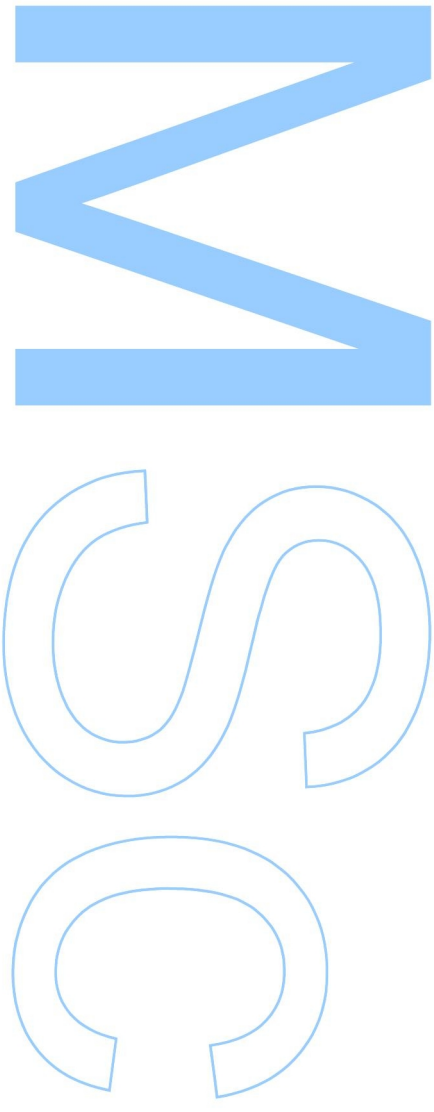
Rede ad hoc com priorização de tráfego

Fábio Rafael Araújo de Carvalho
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2016

Orientador
Pedro Brandão, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto

Coorientador
Ana Aguiar, Professora Auxiliar, Faculdade de Engenharia da Universidade do Porto



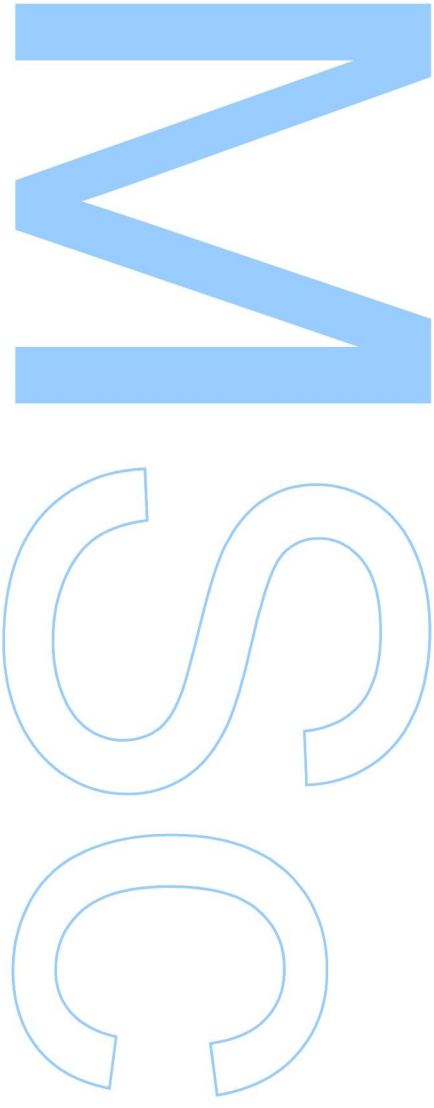
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /



Abstract

Provide quality of service requirements in Mobile Ad Hoc Network (MANET) is a non-trivial task because of all factors that characterize this type of networks. The constant movement of network nodes, are causing constants topological changes wich in turn causes instability on the network. The reduced processing capacity of network nodes, limits possible solutions for this type of networks. The solutions that are developed for this type of networks, have to take into account energy consumption, because the network nodes are battery powered. These solutions should be aware of energy consumption, because in certain types of applications the network nodes cannot be replaced during the network action time.

In this dissertation we present a solution, call algorithm PrioLoadBalancing (PLB), that provides two quality of service requirements in MANET. These two quality of service requirements are: delay and throughput. Both requirements are guaranteed until next hop. This solution uses the Optimized Link State Routing (OLSR)v2 as routing protocol to instantiate the MANET. In addition to guarantee the quality of service requirements presented above, our solution will also balance the traffic when the network topology allows.

We perform a set of tests to measure the impact of our solution on the network of Vital Responder project. Tests were performed in this project, because in this application network nodes do not take into account the possibility of traffic require quality of service requirements. This factor is essential to good behavior of this system.

The results show that traffic prioritization until next hop works as well as the load balancing. The hardware resources used by algorithm, were a bit excessive.

Resumo

Garantir requisitos de qualidade de serviço em redes Mobile Ad Hoc Network (MANET) não é uma tarefa fácil por causa de todos os fatores que caracterizam este tipo de redes. A movimentação constante dos nós da rede, provocam mudanças topológicas constantes que por sua vez provoca instabilidade na rede. O poder de processamento reduzido por parte dos nós da rede, limita possíveis soluções para este tipo de redes. As soluções que são desenvolvidas para este tipo de redes, têm de ter em consideração os consumos energéticos, visto que os nós da rede são alimentados por bateria. Deve-se tentar que esta dure o máximo tempo possível, porque em certo tipo de aplicações não é possível trocar os nós durante o período de ação da rede.

Nesta dissertação apresentamos uma solução, que designamos por algoritmo PrioLoadBalancing (PLB), que garante dois requisitos de qualidade de serviço nas redes MANET. Esses dois requisitos de qualidade de serviço são: atraso e largura de banda. Ambos os requisitos são garantidos até ao próximo salto. Esta solução usa o Optimized Link State Routing (OLSR)v2 como protocolo de encaminhamento para instanciar a rede MANET. Para além de garantir os requisitos de qualidade de serviço apresentados anteriormente, a nossa solução também vai balancear o tráfego quando a topologia de rede assim o proporcionar.

Realizamos um conjunto de testes para medir o impacto da nossa solução na rede MANET do projeto Vital Responder. Foram realizados testes neste projeto, porque nesta aplicação os nós da rede não têm em consideração a possibilidade do tráfego necessitar de requisitos de qualidade de serviço. Esse fator é muito importante para o bom funcionamento deste sistema.

Os resultados obtidos mostram que a priorização do tráfego até ao próximo salto está a funcionar, assim como o seu balanceamento. Os recursos de hardware gastos pelo algoritmo foram um pouco excessivos.

Agradecimentos

Quero agradecer aos meus pais que estiveram sempre presentes e foram os principais motivadores neste longo percurso. Fizeram tudo para facilitar este percurso, com o seu apoio, força e amor. Ao resto da minha família e amigos pelo apoio e motivação que me deram nas fases mais complicadas.

A todos os amigos que fiz durante o Mestrado, onde tive o privilégio de partilhar bons momentos que espero que se repitam por muito anos.

Aos meus orientadores Prof. Pedro Brandão e Prof. Ana Aguiar por me terem facultado a possibilidade de trabalhar com eles. Sem eles não era possível alcançar os objetivos propostos. Muito obrigado por todo o apoio, paciência e motivação durante este ano.

À minha namorada pelo apoio, força e paciência. Durante este percurso a disponibilidade nunca foi a desejada, mesmo assim nunca deixaste de apoiar e dar-me força para ultrapassar os momentos mais complicadas. Pela pessoa que és, um muito obrigado.

Dedico aos meus pais, namorada e amigos.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

AODV	Adhoc On-Demand Distance Vector Routing
ART	Active Route Timeout
BATMAN	Better Approach To Mobile Adhoc Networking
CBQ	Class Based Queueing
CPU	Central Processing Unit
CTP	Collection Tree Protocol
DSCP	Differentiated Service Code Point
DSDV	Destination-Sequenced Distance-Vector Routing Protocol
DSR	Dynamic Source Routing
DisToNet	Disconnection Tolerant Network
ECG	Eletrocardiograma
EQ	Echo Quality
ETX	Expected Transmission Count
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
GPS	Global Positioning System
HTB	Hierarchical Token Bucket
IETF	Internet Engineering Task Force
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	Internet Protocol
JSON	JavaScript Object Notation
LARTC	Linux Advanced Routing & Traffic Control
M-AODV	Modified - Ad hoc On-demand Distance Vector
MAC	Media Access Control
MANET	Mobile Ad Hoc Network
MPR	MultiPoint Relay
NHDP	Neighborhood Discovery Protocol
NTP	Network Time Protocol
OGM	Originator Menssages
OLSR	Optimized Link State Routing
OONF	OLSR.org Network Framework
OSI	Open Systems Interconnection
PLB	PrioLoadBalancing
QOLSR	Quality of Service for Ad hoc Optimized Link State Routing Protocol
QS-AODV	Quality of Service - Ad hoc On-demand Distance Vector
QoS	Quality of Service
RAM	Random Access Memory

RERR Route Error

RQ Receive Quality

RREP Route Reply

RREQ Route Request

RRepAck Route Reply Acknowledgment

SSH Secure Shell

SSID Service Set Identifier

TCP Transmission Control Protocol

TLV Type-Length-Value

TORA Temporally Ordened Routing Algorithm

TQ Transmit Quality

ToS Type of Service

UDP User Datagram Protocol

USB Universal Serial Bus

VoIP Voice over Internet Protocol

WBest Wireless Bandwidth EStimation Tool

XML eXtensible Markup Language

ZRP Zone Routing Protocol

Capítulo 1

Introdução

Nos últimos anos, houve um crescimento significativo do número de aplicações que usam comunicações sem fios. Atualmente, os utilizadores beneficiam de dispositivos pequenos (por exemplo, *smartphones*), que estão equipados com interfaces rádio com alta capacidade de largura de banda e com baterias com boa autonomia. A investigação e o desenvolvimento na área das redes *ad hoc* têm tido uma atenção especial no decorrer destes anos. Foram propostos vários protocolos e mecanismos para melhorar e estender o número de aplicações que possam ser implementadas sobre esta tecnologia.

O tema desta dissertação está relacionado com as redes *ad hoc*, mais precisamente com uma subcategoria que são as Mobile Ad Hoc Network (MANET) [19]. Neste tipo de redes sem fios os nós podem movimentar-se livremente. Como se trata de uma subcategoria das redes *ad hoc* não requer qualquer infraestrutura ou ponto central que coordene a rede. São facilmente instaláveis em qualquer tipo de ambientes e por isso tem uma aplicabilidade muito vasta. Algumas características deste tipo de redes tem sido alvo de investigação ao longo destes anos. A camada de rede e os protocolos a ela associados tem sido um ponto forte da investigação nesta área. Os requisitos de qualidade de serviço, diferenciação de tráfego e segurança também têm tido uma atenção especial em termos de desenvolvimento e investigação.

Em certo tipo de aplicações há uma vantagem clara em optar-se por uma implementação sobre uma rede MANET. Por vezes os ambientes onde estas aplicações vão trabalhar, não permitem que haja a possibilidade de montar uma infraestrutura ou recorrer a outras tecnologias de comunicação. Quando essas aplicações estão relacionadas com temas críticos, como por exemplo, a monitorização de pessoas, os requisitos apresentados anteriormente tem uma importância extra pelo facto de que uma pequena falha da rede pode ser fatal.

1.1 Enquadramento e motivação

Este trabalho, como foi referido anteriormente, está relacionado com uma subcategoria das redes *ad hoc* que são as redes MANET [19]. Neste tipo de redes os dispositivos podem movimentar-se

livremente, como consequência disso a topologia de rede está constantemente a alterar. Por estes motivos, alguns mecanismos que são usados nas redes infraestruturadas e *ethernet* para garantir requisitos de qualidade de serviço não podem ser aplicados nas redes MANET.

Este tipo de redes tem uma aplicabilidade muito vasta, em algumas dessas aplicações há necessidade de garantir certos requisitos de qualidade de serviço. Essa necessidade advém do facto do tráfego da rede ter características diferentes, ao contrário de outras aplicações onde o tráfego não difere em termos dos requisitos exigidos. Em aplicações críticas, como é o caso da monitorização de pacientes, catástrofe, as garantias de certos parâmetros de qualidade de serviço são essenciais para o seu bom funcionamento. Por exemplo, numa aplicação de monitorização de pacientes onde se está a medir várias variáveis (temperatura, oxigénio e etc) com prioridades e características diferentes, a rede tem de tratar de forma diferente os pacotes de dados. Os dados com maior prioridade têm de ser encaminhados de forma diferente dos dados com menor prioridade. As prioridades podem ser baseadas em diferentes características e/ou requisitos dos dados, por exemplo, fiabilidade, atraso, ordenação, *jitter*, largura de banda entre outros.

O que pretendemos com a realização deste trabalho é desenvolver uma solução onde os fatores apresentados anteriormente são tidos em conta pelos nós da rede. Essa solução vai ser testada na rede de uma aplicação de monitorização de bombeiros em situações de combate aos incêndios. Mas futuramente poderá ser usada em outro tipo de aplicações que necessitem de garantir certos requisitos de qualidade de serviço.

1.2 Objetivo do trabalho

O objetivo principal deste trabalho é desenvolver uma solução que possa ser incorporada em aplicações, que necessitem de garantir certos parâmetros de qualidade de serviço no transporte dos dados na rede MANET.

Os objetivos deste trabalho são:

- Levantamento das possibilidades de garantir certos parâmetros de qualidade de serviço neste tipo de redes.
- Comparação em termos de qualidade de serviço dos principais protocolos de encaminhamento existentes.
- Análise de mecanismos que possam ser incorporadas na rede, de modo a diferenciar o tráfego que circula na mesma.
- *Design* e desenvolvimento de um algoritmo que garante certos parâmetros de qualidade de serviço, por exemplo, atraso e largura de banda.
- Implementação da rede.

- Preparar a solução desenvolvida para futuramente incorporar o trabalho já desenvolvido, designado por Disconnection Tolerant Network (DisToNet) [30].
- Testes em ambientes reais.

1.3 Estrutura do documento

Esta dissertação está organizada da seguinte maneira:

- Capítulo 1 (Introdução): Apresentação da motivação para a realização deste trabalho assim como os objetivos que se pretende cumprir.
- Capítulo 2 (Estado da arte): Introdução às redes *ad hoc*, descrição de vários protocolo de encaminhamento para redes MANET relevantes para este trabalho e por último descrição dos problemas de Quality of Service (QoS) em redes *ad hoc*.
- Capítulo 3 (Arquitetura): Descrição do cenário alvo da nossa solução, dos problemas existente nesse cenário e a forma como tencionamos resolver esses problemas.
- Capítulo 4 (Avaliação dos protocolos de encaminhamento): Descrição dos elementos da rede usada para os testes. Realização de testes para comparar dois protocolos de encaminhamento. Descrição de algumas funcionalidades extra desses dois protocolos de encaminhamento.
- Capítulo 5 (Priorização com Hierarchical Token Bucket (HTB): Descrição mais detalhada da solução apresentada no capítulo 3.
- Capítulo 6 (Desenvolvimento): Apresentação do algoritmo desenvolvido no decorrer deste trabalho. Descrição do funcionamento global do algoritmo e dos módulos associadas ao mesmo.
- Capítulo 7 (Testes e Resultados): Realização de testes para verificar se as funcionalidades do algoritmo propostas estão a funcionar corretamente assim com o seu desempenho em termos de tempo e recursos de *hardware*.
- Capítulo 8 (Conclusão): Conclusões tiradas após a realização deste trabalho. Apresentação de alguns fatores que podem ser considerados no futuro de forma a melhorar o desempenho do algoritmo.

Capítulo 2

Estado da Arte

2.1 Redes Ad Hoc

Esta secção tem como objetivo descrever de forma mais detalhada as redes *ad hoc*. Primeiro faz-se uma abordagem sobre as principais características das redes *ad hoc*. Depois apresentamos os principais cenários de aplicação deste tipo de redes e por último algumas características do encaminhamento neste tipo de redes.

2.1.1 Características de uma rede *ad hoc*

Há dois tipos de redes sem fios, infraestruturadas e *ad hoc* (sem infraestrutura). Nas redes sem fios infraestruturadas todos os dispositivos comunicam através e dentro da cobertura de um ponto de acesso. Não podem comunicar diretamente entre si, em vez disso, todo o tráfego tem que passar pelo ponto de acesso que depois trata de encaminhá-lo para o destino. Os dispositivos podem mover-se dentro da cobertura de um único ponto de acesso (*Basic Service Set*) ou através de múltiplos pontos de acesso (*Extended Basic Service Set*).

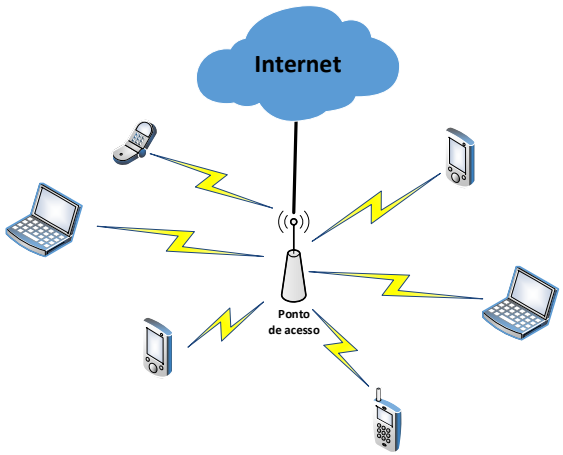


Figura 2.1: Rede infraestruturada

O segundo tipo, são as redes *ad hoc* (sem infraestrutura) onde os dispositivos comunicam diretamente entre si e sem necessidade de um ponto de acesso. Os dispositivos movem-se livremente podendo conectar-se dinamicamente a outros dispositivos. Além disso, alguns dispositivos da rede podem ser designados como encaminhadores para descobrir e manter rotas para outras redes.

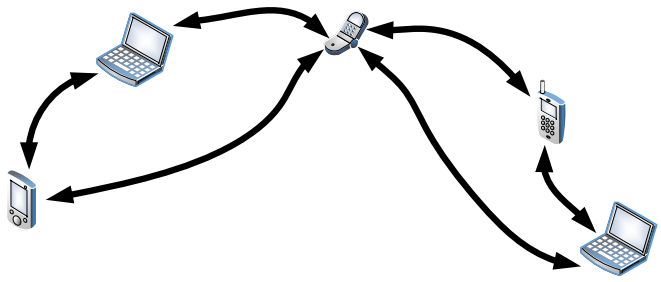


Figura 2.2: Rede Ad Hoc

As principais características de uma rede *ad hoc* são:

- Não necessita de um ponto de acesso para que haja comunicação entre dispositivos.
- Não necessita de uma infraestrutura de rede pré-instalada.
- Os dispositivos podem movimentar-se livremente.
- A topologia de rede está constantemente a mudar devido às movimentações dos dispositivos.
- Está constantemente a atualizar as suas rotas devido às mudanças de topologia constantes.

Para que este tipo de redes funcionem corretamente, cada nó da rede tem de identificar-se perante os outros nós, normalmente com um endereço Internet Protocol (IP). Este endereçamento faz com que os nós da rede sejam capazes de alcançar outros nós e executar funções de encaminhamento. Para além disto, a rede deve estar preparada para a perda e/ou entrada de novos nós. Estes mecanismos são alcançáveis recorrendo ao uso de protocolos de encaminhamento, que podem fornecer outras funcionalidades mas que tem como principal objetivo descobrir e estabelecer rotas entre os nós da rede. O protocolo de encaminhamento determina qual a melhor rota recorrendo ao uso de métricas, essas métricas variam de protocolo para protocolo. Deve-se escolher o protocolo de encaminhamento de acordo com as características da rede.

Nas redes *ethernet* as rotas estabelecidas pelo protocolo de encaminhamento no processo de inicialização, raramente sofrem alterações porque a topologia deste tipo de redes raramente muda. Nas redes *ad hoc* já não funciona assim, com a constante movimentação dos nós os protocolos de encaminhamento tem de estar constantemente a atualizar e recalcular as tabelas de encaminhamento. As características desejáveis para um protocolo de encaminhamento são: escolha do melhor caminho, simplicidade, robustez, estabilidade, rápida convergência, flexibilidade e requisitos de qualidade de serviço.

2.1.2 Cenários de aplicação de redes *ad hoc*

Com o crescimento significativo da popularidade das redes sem fio, cresce também o desenvolvimento e investigação dentro da mesma, assim como o número de aplicações que usam este tipo de tecnologia. A possibilidade de comunicação direta ou indirecta entre os nós da rede, serem auto-configuráveis e auto-organizáveis, faz com que os ambientes em que as redes Mobile Ad Hoc Network (MANET) podem ser aplicadas seja muito vasto. Na tabela 2.1 temos alguns exemplos de cenários de aplicação para este tipo de redes.

2.1.3 Encaminhamento em redes *ad hoc*

Como as redes *ad hoc* são redes muito dinâmicas, como já vimos anteriormente, os protocolos de encaminhamento utilizados nas redes *ethernet*, não são apropriados para realizar a função de encaminhamento neste tipo de redes. Os equipamentos utilizados neste tipo de redes, normalmente, não tem o mesmo poder de processamento dos equipamentos utilizados nas redes *ethernet*. Além disso, os equipamentos usados nas redes *ad hoc* podem estar limitados por questões de autonomia de bateria. Por isso, os protocolos de encaminhamento usados neste tipo de redes tem de ter em conta estes fatores.

O encaminhamento neste tipo de redes é atualmente uma das principais áreas de investigação. Este processo de encaminhamento pode ser direto, também conhecido como *single-hop*, ou, pode ser indireto, também conhecido como *multi-hop*. No encaminhamento direto, o destino dos dados é um nó vizinho, no indireto é necessário usar nós intermédios para alcançar o destino. Como se pode observar na figura 2.3, o nó **A** quer enviar dados para o nó **D** que está fora do seu alcance. Para que os dados cheguem até ao destino têm de passar por nós intermédios (**B** e **C**), o que designamos por encaminhamento *multi-hop*. Caso o nó **A** quisesse enviar dados para o nó **B** já era designado por encaminhamento *single-hop* ou direto, porque ambos os nós estão ao alcance um do outro.

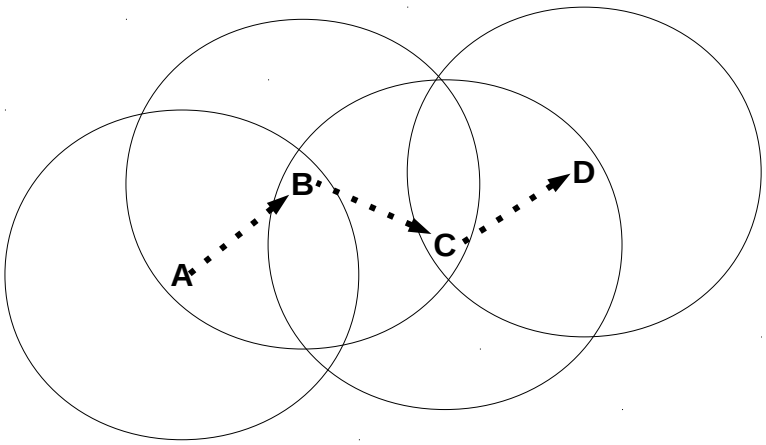


Figura 2.3: Encaminhamento multi-hop

Os protocolos de encaminhamento que tratam do mecanismo de encaminhamento, estão

Tabela 2.1: Cenários de aplicação das redes Ad Hoc

Cenário	Descrição
Aplicação Militar	As MANET tiveram origem dentro de ambientes militares de forma a possibilitar a comunicação entre soldados no campo de batalha, já que em tais ambientes não é possível a existência de uma infraestrutura fixa para a comunicação.
Redes de sensores	O uso das MANET possibilita aos sensores transmitir os dados recolhidos entre eles ou para determinado ponto da rede.
Assistência a desastres	Já que as MANET não necessitam de uma infraestrutura pré-existente, podem oferecer um método extremamente flexível em operações de resgate ou qualquer outro cenário que requeira um estabelecimento de comunicação que seja rapidamente organizado, possibilitando a troca de informações com sobreviventes ou com quem presta auxílio aos sobreviventes. Bem como auxiliando em situações de emergência, como policiamento e combate aos incêndios.
Conexões com a Internet	As vantagens encontradas em aumentar o alcance das redes sem fio infraestruturadas com a finalidade de se obter conexões à Internet em sítios onde não há conectividade vêm popularizando o uso de redes *ad-hoc* em redes domésticas, visto que podem ser estendidas conforme necessário através de nós encaminhadores (para garagens, para o carro estacionado na rua, para a casa do vizinho e etc).

divididos em três categoria; proactivos, reativos e híbridos. A diferença entre estes três tipos será explicada na secção 2.2.1. Algumas características importantes dos protocolos de encaminhamento neste tipo de redes são:

- **Escolha da rota:** Deve escolher sempre o melhor caminho disponível para encaminhar o tráfego.
- **Simplicidade do algoritmo:** Deve gastar o mínimo de recursos possíveis fornecendo os serviços necessários à rede.
- **Robustez:** Deve obter sempre os melhores resultados sem causar interrupções longas no funcionamento da rede.
- **Escalabilidade:** A eficiência do algoritmo deve ser a mesma independentemente do número de nós da rede.
- **Convergência:** Após uma falha, por exemplo a saída de um nó da rede, o algoritmo deve reagir o mais rápido possível e encontrar uma nova rota para compensar aquela perda.
- **Adaptabilidade:** Reagir de forma eficaz às mudanças constantes de topologia.
- **Independência de tecnologia:** Deve funcionar de igual forma, independentemente da tecnologia utilizada nas camadas abaixo.
- **Qualidade de serviço:** Em algumas aplicações é fundamental a aceitação de parâmetros de qualidade de serviço, de modo a colmatar a perda de pacotes, atrasos, baixa taxa de transferência e erros de transmissão.

2.1.4 Sumário

Nesta secção fez-se uma apresentação das principais características de uma rede *ad hoc* e o que as distingue de outro tipo de redes. Apresentamos ainda os seus principais cenários de aplicação, como funciona o encaminhamento neste tipo de redes e algumas características desse encaminhamento.

2.2 Protocolos de encaminhamento nas redes *ad hoc*

Para que numa rede *ad hoc* seja possível ter encaminhamento múltiplo salto, necessitamos que todos os nós da rede estejam a executar o mesmo protocolo de encaminhamento. Desta forma, um nó da rede sabe como chegar até todos os outros nós da rede, diretamente ou indiretamente. Por exemplo, um nó que esteja fora do alcance do nó que está a colecionar os dados, também designado como *sink*, não conseguirá chegar até ele se não estiver a executar um protocolo de encaminhamento. Este protocolo de encaminhamento vai calcular qual o melhor percurso para chegar até ao *sink*, e colocar essa rota na tabela de encaminhamento do respetivo nó.

Esta secção tem como objetivo fazer uma apresentação do funcionamento dos principais protocolos de encaminhamento para redes *ad hoc*. Primeiro descrevemos as diferentes classificações dos protocolo de encaminhamento e nas subsecções seguintes descrevemos o funcionamento dos

protocolos de encaminhamento Optimized Link State Routing (OLSR), Better Approach To Mobile Adhoc Networking (BATMAN), Adhoc On-Demand Distance Vector Routing (AODV) e Collection Tree Protocol (CTP), finalizando a secção com uma tabela comparativa dos protocolos de encaminhamento apresentados.

2.2.1 Classificação dos protocolos

Ao longo destes anos foram propostos alguns protocolos de encaminhamento para redes *ad hoc* que estão divididos nas seguintes categorias: *Table Driven* (também designados como Proactivos), *On-Demand* (também designados como Reativos) e Híbridos [26, 41, 44].

- **Protocolos de encaminhamento *Table-driven* (designados também por Proactivos):** Neste tipo de protocolos de encaminhamento cada nó mantém informações de encaminhamento para os outros nós, estas informações de encaminhamento são enviadas por cada nó periodicamente. Este tipo de protocolos tenta manter informações de encaminhamento consistentes e atualizadas em cada nó. Os protocolos deste tipo periodicamente fazem o designado como *flooding*, ou seja, inundam a rede com pacotes de controlo. Os principais protocolos de encaminhamento desta categoria são o OLSR [9], BATMAN, CTP [20] e o Destination-Sequenced Distance-Vector Routing Protocol (DSDV) [43].
- **Protocolos de encaminhamento *On-Demand* (designados também por Reativos):** Este tipo de protocolos de encaminhamento apenas estabelecem as rotas quando necessário, por isso são também conhecidos como *source-initiated routing protocols*. Quando um nó quer transmitir informação para outro nó da rede é iniciado um processo de descoberta de rota. Após essa rota ser estabelecida será mantida até não ser mais desejada. A principal vantagem deste tipo de protocolos é que impõe uma sobrecarga menor na rede, mas por outro lado a latência para a descoberta da rota é maior do que nos protocolos *Table-driven*. AODV [42] e Dynamic Source Routing (DSR) são alguns protocolos de encaminhamento deste tipo.
- **Híbridos:** Este tipo de protocolos de encaminhamento combinam as vantagens dos outros dois apresentados anteriormente. Inicialmente as rotas são estabelecidas de forma proativa, em seguida, são mantidas de forma reativa. O Zone Routing Protocol (ZRP) [22] e Temporally Ordened Routing Algorithm (TORA) são alguns protocolos de encaminhamento deste tipo.

2.2.2 Protocolo OLSR

Versão 1

O OLSR é considerado um dos principais protocolos de encaminhamento para as redes MANET. É um protocolo do tipo *Table-driven*, desenvolvido pelo Institut National de Recherche en

Informatique et en Automatique (INRIA) e proposto pela primeira vez na conferência *IEEE International Multitopic Conference* em 2001. Em 2003 foi desenvolvido dentro no grupo MANET Internet Engineering Task Force (IETF), sendo proposto e definido através da RFC3626 [9].

Este protocolo faz uso do tradicional algoritmo de estado de ligação (*link state*) mas de uma forma otimizada, o conceito chave para esta otimização é o uso de MultiPoint Relay (MPR). MPR são nós selecionados para encaminhar as mensagens durante o processo de *flooding*, apenas os nós que são selecionados como MPR encaminham as mensagens, os outros nós limitam-se a processar as mensagens. Esta técnica reduz de forma significativa o tráfego que circula na rede comparado com a técnica pura de *flooding*, onde todos os nós da rede encaminham as mensagens. Além da redução do tráfego que circula na rede, esta técnica também reduz o número de mensagens repetidas que cada nó recebe.

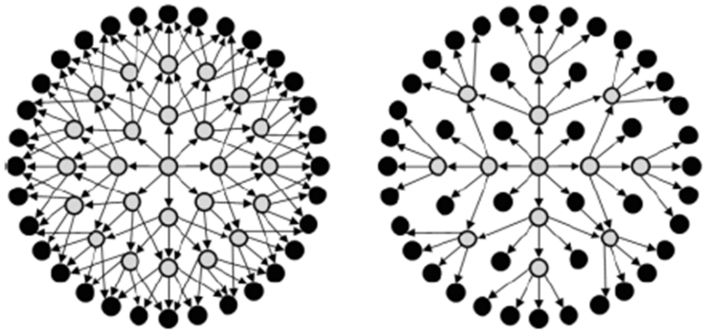


Figura 2.4: Comparação do protocolo OLSR sem e com nós MPR

Cada nó da rede seleciona um conjunto de MPR, este conjunto de MPR são nós da rede que estão a *1-hop* de distância do nó ao qual corresponde aquele conjunto de MPR. Este conjunto é selecionado de modo a que sejam abrangidos todos os nós a *2-hops* de distância do nó ao qual pertence o conjunto de MPR. Cada nó a *2-hops* é alcançável pelo menos por um dos nós do conjunto MPR. Este conjunto deve ser recalculado quando forem detetadas alterações de topologia a 1 ou 2 *hops* de distância.

Como foi referido anteriormente, os MPR são responsáveis por encaminhar as *topology control messages (TC)*, que descrevem as ligações em torno de um nó, *multiple interface declaration messages (MID)* onde os vários endereços de um nó são declarados e as *host and network association messages (HNA)* que anunciam as *gateways* da rede.

Em termos de descoberta de vizinhos, periodicamente cada nó envia mensagens `HELLO`, a sua estrutura pode ser visualizada na figura 2.5. Estas mensagens descrevem todas as ligações que estão diretamente ligadas a um nó. Consequentemente os nós que recebem estas mensagens ficam a conhecer os vizinhos a *2-hops* e o próximo salto para esses nós. Estas mensagens são enviadas com o *Time To Live* definido a 1 para que não sejam retransmitidas.

Todas as mensagens geradas pelos nós OLSR são encapsuladas num pacote User Datagram Protocol (UDP). Na figura 2.6 temos o formato de um pacote OLSR sem os cabeçalhos IP e UDP.

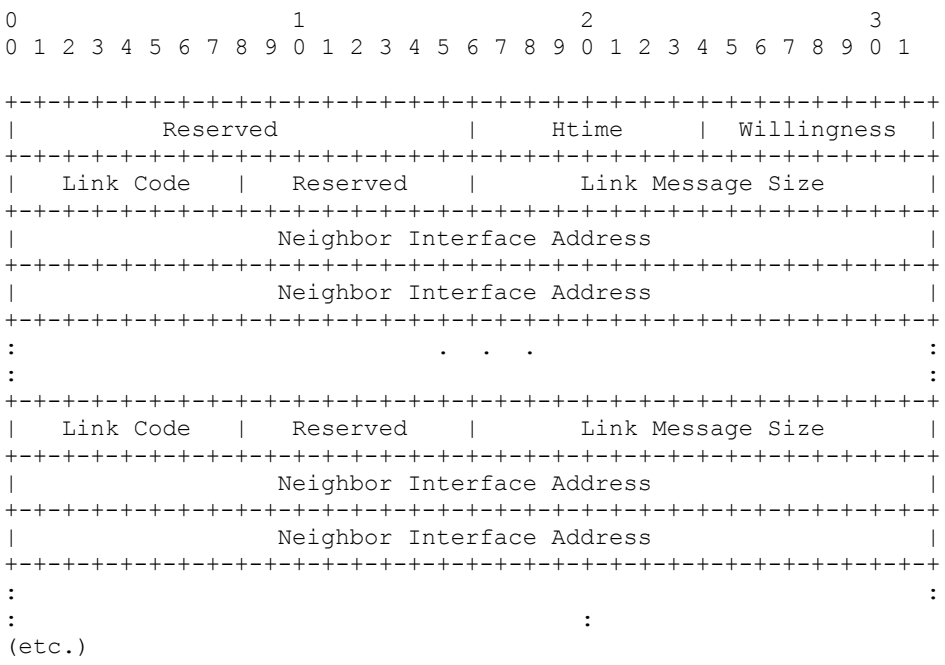


Figura 2.5: Formato da mensagem HELLO, retirado de [9]

Versão 2

O OLSRv2 [12] mantém os mesmos mecanismos básicos da versão anterior, contudo oferece alguns melhoramentos. Por exemplo, na sua arquitetura flexível e modular permitindo que sejam incorporadas extensões de segurança ou simples *add-ons* ao protocolo base sem ser necessário mexer no núcleo. Como mecanismo para a descoberta dos vizinhos esta nova versão usa o protocolo Neighborhood Discovery Protocol (NHDP) [11], que já tem incorporado os seguintes mecanismos:

- Teste da bidirecionalidade das ligações entre nós da rede.
- Deteção dos vizinhos a dois saltos de distância.
- Possibilidade de ter várias interfaces, com um ou mais endereços IP.

Cada nó envia mensagens HELLO periodicamente com a lista de todos os nós dos quais recentemente recebeu uma mensagem HELLO assim como o estado da ligação. O estado HEARD significa que o nó ainda não verificou a bidirecionalidade da ligação e o estado SYM significa que a ligação foi verificada e é bidirecional. Na figura 2.7 podemos ver como funciona o processo da verificação da bidirecionalidade das ligações e a descoberta dos nós a *2-hops*.

NHDP permite que cada nó aplique um mecanismo de qualidade da ligação que para além da troca de mensagens pode restringir quando uma ligação é considerada utilizável ou não. Por exemplo, um nó pode optar por não caracterizar a ligação como HEARD ou SYM se não recebeu

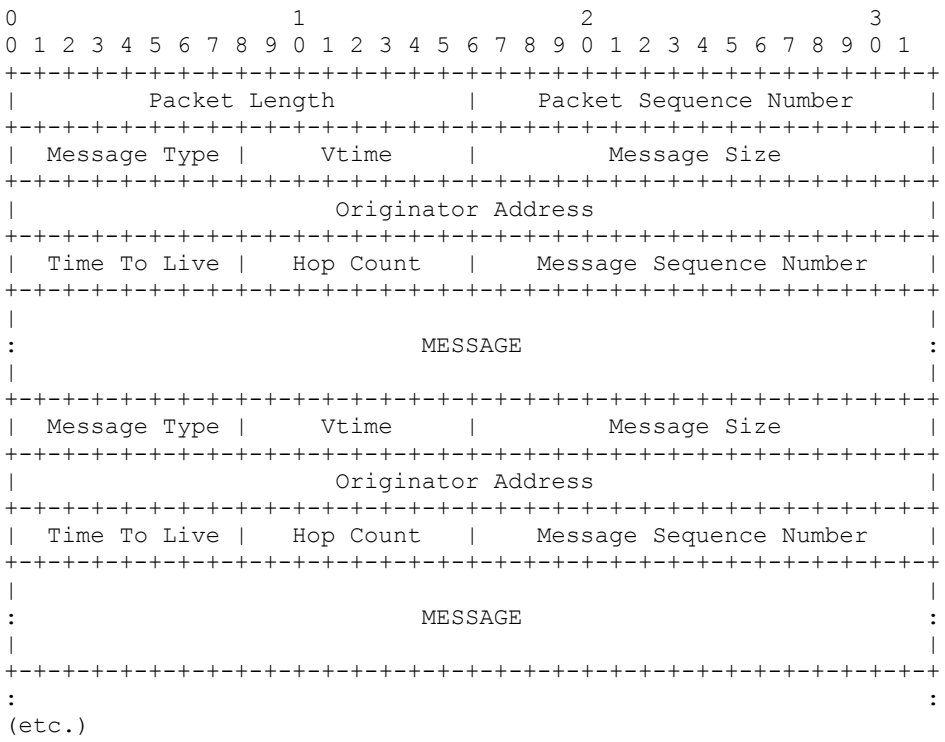


Figura 2.6: Formato do pacote OLSR, retirado de [9]

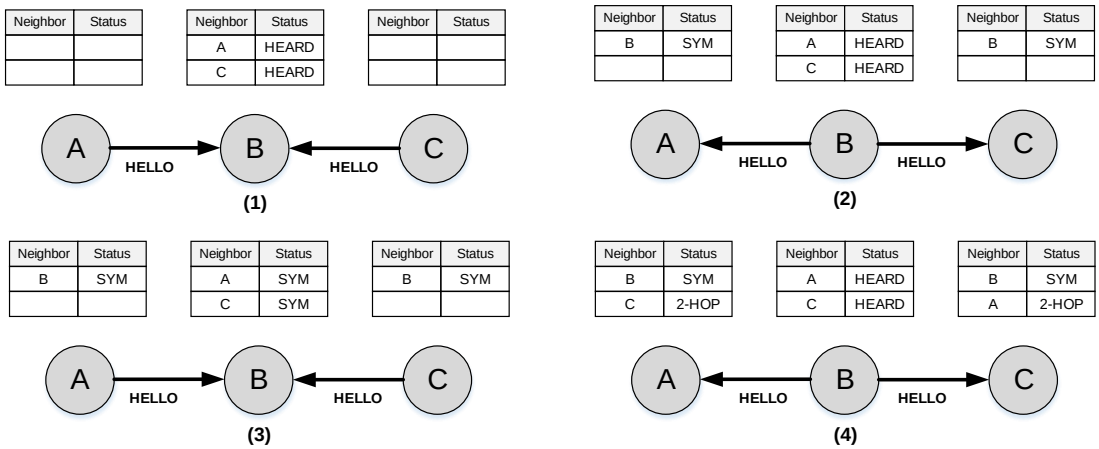


Figura 2.7: Descoberta dos vizinhos a 1 e 2 *hops*

um determinado número de mensagens HELLO ou se a relação sinal-ruído não tenha atingido determinado limiar.

Além do protocolo de descoberta dos vizinhos, o OLSRv2 utiliza um formato específico para as mensagens do protocolo, descrito em [10], permitindo desta forma algumas funcionalidades extra. Por exemplo, a capacidade de associar qualquer número de atributos arbitrários a qualquer uma das mensagens de controlo por meio da inclusão de Type-Length-Value (TLV). Na estrutura TLV ao analisarmos uma mensagem se não conhecermos aquele tipo, podemos saltar e indicá-lo como um TLV não conhecido. A vantagem é que podem ser desenvolvidas e incorporadas facilmente

extensões ao protocolo base nas mensagens de controlo existentes. A estrutura TLV é usada no OLSRv2, por exemplo, para indicar a seleção do MPR nas mensagens `HELLO` ou para indicar o intervalo de emissão de mensagens e a duração para o qual o conteúdo de mensagem é valido. O OLSRv2 e o NHDP são concebidos para incorporar facilmente extensões ao protocolo base. Para além do formato de mensagens descrito anteriormente, após a geração das mensagens de controlo base (`TC` e `HELLO`), as mensagens de saída podem ser entregues à extensão para um processamento adicional. Essa extensão pode inserir endereços, blocos de endereços e mensagens TLV. Além do processamento por parte da extensão das mensagens de saída, esse processamento também pode ser feito nas mensagens de entrada. Por exemplo, uma extensão para identificar a receção de mensagens mal formadas e proibir o processamento das mesmas pelo OLSRv2.

Outra melhoria incorporada no OLSRv2 foi a adição de métricas de ligação que são amplamente utilizadas pelos protocolos de encaminhamento do tipo estado da ligação. Algumas extensões propostas para a versão 1 tentavam colmatar este problema. Ao utilizar métricas de ligação temos vantagens claras, na figura 2.8 podemos ver um desses casos. Ao usarmos métricas de ligação vamos enviar o tráfego por um caminho melhor do que se tivéssemos a usar o mínimo número de saltos como acontece na versão 1.

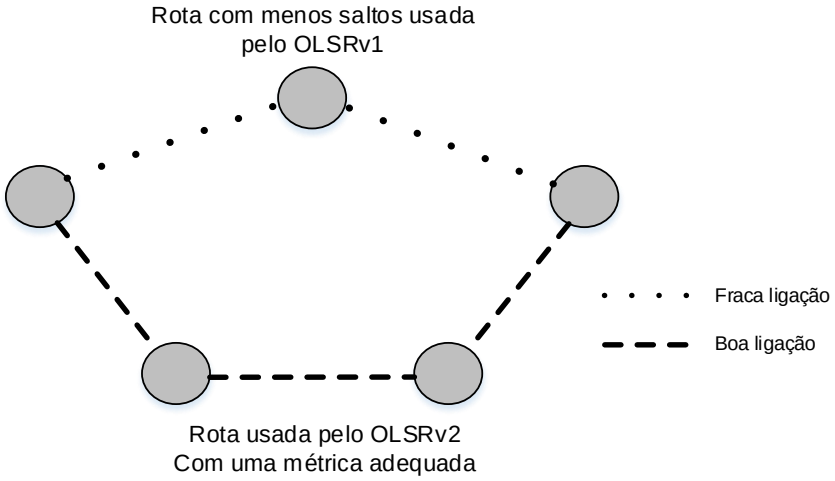


Figura 2.8: Encaminhamento QoS usando métricas de ligação

Como conclusão a versão 2 do OLSR mantém os mesmos mecanismos básicos de um protocolo de encaminhamento estado da ligação clássico: descoberta da vizinhança, anúncio do estado das ligações e *flooding* de mensagens pela rede. Além disso, devido ao *design* modular e ao formato flexível das mensagens, extensões OLSRv2 podem ser facilmente concebidas sem quebrar a compatibilidade da especificação do núcleo. Para finalizar a nível de implementações desenvolvidas deste protocolo de encaminhamento temos, a *framework* OLSR.org Network Framework (OONF) [40] que contém uma coleção de bibliotecas para executar todas as funções descritas anteriormente e uma implementação em `JAVA`, designada por `JOLSRv2` [23].

Qualidade de serviço

Na versão 1 do OLSR não há qualquer preocupação ao nível de requisitos de qualidade de serviço. Foram propostas algumas extensões para tentar melhorar esta versão em certos aspetos, tais como, qualidade de serviço, eficiência energética entre outros. Por exemplo, [31] ao nível de eficiência energética, [48] em termos da diminuição do tamanho de algumas mensagens e diminuição da sobrecarga na rede e [5] na agregação de mensagens de controlo.

Em termos de qualidade de serviço existe uma versão alterada do OLSR base que é o Quality of Service for Ad hoc Optimized Link State Routing Protocol (QOLSR) [2, 24, 36]. No QOLSR são adicionados alguns campos às mensagens padrão do OLSR para trazer alguma qualidade de serviço às redes MANET baseadas em OLSR. O QOLSR fornece alguns níveis de qualidade de serviço, onde cada nível equivale a um determinado requisito. A versão do QOLSR que está disponível[1], ainda não inclui estes mecanismos de qualidade de serviço. Outras extensões propostas tentam melhorar a escolha das rotas com base na energia local dos nós [4] ou alterar o mecanismo de escolha do MPR para garantir determinada largura de banda ao longo do caminho [18].

Na versão 2 do OLSR a introdução de métricas de ligação invés do tradicional mínimo número de saltos usado pela versão 1, permite melhores decisões de encaminhamento melhorando a qualidade do serviço prestado pela rede.

2.2.3 Protocolo BATMAN

O BATMAN é outro protocolo de encaminhamento que pertence ao grupo dos pró-ativos e foi desenvolvido com a intenção de substituir o OLSR. Os seus criadores dizem que a exigência do algoritmo estado da ligação para recalcular toda a topologia de rede torna-se um desafio quando os nós são limitados ao nível de processamento e memória. Visto isto, neste protocolo de encaminhamento, cada nó apenas tem conhecimento e apenas mantém informações sobre o seu melhor vizinho em direção a cada nó da rede. Desta forma, não há necessidade de manter um conhecimento global de mudanças locais. Apenas se acontecer alguma coisa a um dos melhores vizinhos é que a rede é avisada.

A evolução do algoritmo utilizado por este protocolo de encaminhamento está dividido por gerações. Na geração I o algoritmo não fazia verificação da bidirecionalidade das ligações, na geração II está verificação já foi considerada, a geração III e restantes representam a implementação do algoritmo. Nesta dissertação apenas serão apresentadas as gerações III e IV, que são as mais relevantes para se perceber o seu funcionamento e os mecanismos que utiliza.

[1] http://qolsr.lri.fr/

Geração III

Todos os nós da rede enviam periodicamente mensagens Originator Menssages (OGM), estas mensagens são encapsulados num pacote UDP que depois é enviado para todos os nós da rede. No total as mensagens OGM têm 52 Bytes (cabeçalho IP + cabeçalho UDP + campos OGM). Na figura 2.9 podemos ver o formato das mensagens OGM.

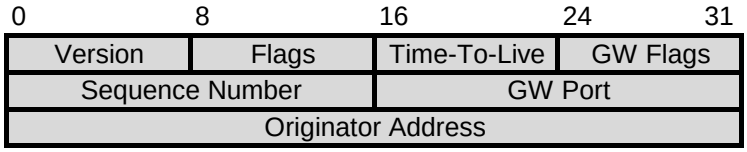


Figura 2.9: OGM na geração III

Os campos da mensagem OGM representam o seguinte:

- ***unidirectional flag:*** Indica se a ligação em direção ao originador da mensagem é bidirecional.
- ***is-direct-link flag:*** Indica se o OGM foi transmitido por um nó que é um vizinho do originador da mensagem.
- ***Time-To-Live:*** Define o limite da mensagem.
- ***sequence number:*** É incrementado pelo originador sempre que envia um OGM novo.
- ***originator address:*** Endereço do originador da mensagem.

As mensagens OGM são utilizadas para descobrir os vizinhos e definir qual o melhor vizinho em direção ao originador da mensagem OGM. Para determinar qual o melhor vizinho, cada nó guarda os números de sequência numa *sliding windows*, esses números são memorizados para depois determinar a qualidade da ligação. O campo *unidirectional flag* é utilizado para determinar se uma ligação entre dois nós é ou não é bidirecional. Ao fim de determinado tempo, chamado *purge timeout*, se um nó não receber nenhum OGM de um nó ao qual elegeu como melhor vizinho, a rota desse nó será eliminada da tabela de encaminhamento.

Geração IV

Esta nova geração é muito idêntica à geração anterior, com a exceção de alguns fatores que são tidos em conta durante o processo de escolha da melhor rota. O formato das mensagens OGM desta geração é idêntica ao da geração anterior à exceção da introdução de dois novos campos, o Transmit Quality (TQ) e *Previous Sender Address*. O TQ serve para denotar a qualidade do caminho em direção ao originador, nesta geração a eleição do melhor vizinho não se baseia apenas na qualidade local do nó vizinho mas sim na qualidade de todo o caminho. Nesta geração já não

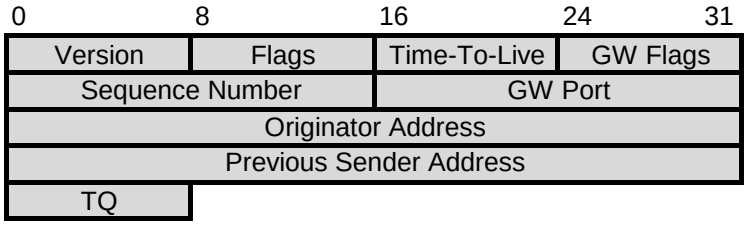


Figura 2.10: OGM na geração IV

é necessário identificar o endereço do nó que retransmitiu o pacote no cabeçalho IP, porque esse endereço já vem identificado na mensagem OGM, no campo *Previous Sender Address*.

O protocolo mede a qualidade da ligação entre dois vizinhos através de duas variáveis, o Receive Quality (RQ) e o Echo Quality (EQ). Com estas duas medições, obtém-se um outro valor que representa a probabilidade de sucesso de transmissão de um pacote, também chamado de TQ.

Figura 2.11: *Receive Quality*

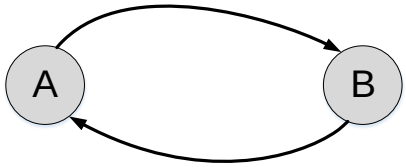


Figura 2.12: *Echo Quality*

Primeiro, o nó **A** obtém o RQ (figura 2.11), para isso calcula este valor dividindo o *charging level* (número de pacotes bem recebidos de determinado nó, neste caso **B**) pelo tamanho da *sliding window local*. Em segundo lugar, calcula o EQ (figura 2.12) que é a probabilidade de receção de um OGM originado por **A** que depois será retransmitido por **B**.

O valor do TQ que depois será passado para o campo TQ da mensagem OGM, é calculado da seguinte forma:

$$TQ = \frac{EQ}{RQ}$$

Sempre que um OGM é criado o campo TQ é inicializado com o valor de 255. Após a receção de um OGM, o nó que recebeu o pacote multiplica esse valor pelo valor do TQ em direção ao nó do qual recebe o pacote. Como podemos ver na figura 2.14.

$$TQ_{NOVO} = TQ_{OGM} \times TQ_{GLOBAL}$$

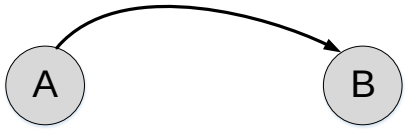


Figura 2.13: *Transmit Quality*

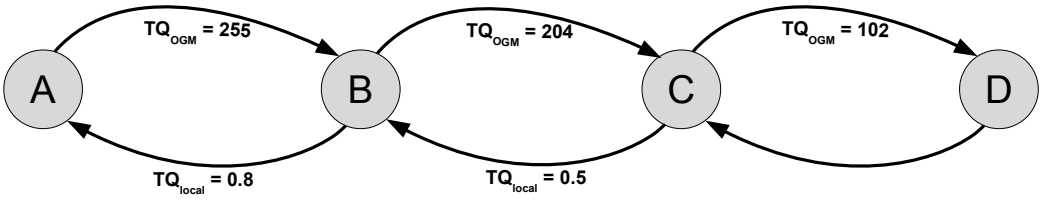


Figura 2.14: Versão IV propagação do TQ

Além dos cálculos que vimos anteriormente, também são aplicadas duas penalidades no valor do campo TQ. Uma dessas penalidades é aplicada caso a ligação não seja bidirecional (*asymetric penalty*) e a outra é aplicada por cada salto (*hop penalty*). Desta forma, a escolha do melhor vizinho em direção ao originador é feita através do valor obtido no TQ global. Assim, sabemos qual o melhor vizinho para chegar ao originador que oferece um melhor caminho global.

Esta nova geração também possibilita um uso eficiente do canal de transmissão, ou seja, é possível agregar várias mensagens OGM num único pacote UDP. Um nó espera determinado tempo e durante esse tempo vai agregando mensagens OGM. Essas mensagens são enviadas quando esse tempo é atingido ou quando é atingido o tamanho máximo do pacote. Em áreas de grande densidade esta técnica pode reduzir o número de colisões. Além disso, a sobrecarga de mensagens OGM é reduzida visto que vários OGM são encapsulados em um único pacote. Por outro lado a perda de um pacote UDP leva à perda de várias mensagens OGM, enquanto no outro caso apenas é perdida uma mensagem.

Qualidade de serviço

Nas subsecções anteriores foram apresentadas duas gerações do algoritmo utilizado pelo protocolo de encaminhamento BATMAN. Atualmente, este protocolo de encaminhamento tem duas versões, batmand [33] e batman-adv [32]. A primeira versão atua no nível 3 do modelo Open Systems Interconnection (OSI) e a segunda atua no nível 2. Em termos de qualidade de serviço, na geração IV do algoritmo com a introdução do campo TQ nas mensagens OGM, consegue-se ter uma perceção melhor da qualidade das ligações. Desta forma, a escolha do melhor vizinho é feita de acordo com a qualidade do caminho global e não da qualidade local.

2.2.4 Protocolo AODV

Versão 1

O AODV é um protocolo de encaminhamento do grupo dos reativos baseado no princípio vetor de distância, onde o caminho entre dois nós da rede é calculado quando necessário (se necessário). Quando um nó quer enviar dados para outro nó tem de procurar o caminho (fase de descoberta), usa o caminho durante a transmissão e mantém esse caminho durante a utilização (fase de manutenção). O processo de descoberta e manutenção dos caminhos é baseado na troca de mensagens de controlo, Route Request (RREQ), Route Reply (RREP), Route Error (RERR), Route Reply Acknowledgment (RRepAck) e `HELLO`. RREQ é utilizado pelo nó que quer transmitir informação para estabelecer o caminho até ao destino. O RREP é usado por um nó intermédio ou pelo nó de destino para responder ao pedido de estabelecimento de caminho sendo enviado em modo *unicast*. As mensagens `HELLO` são usadas para manter os caminhos estabelecidos. Para evitar a formação de *loops* usa o princípio do número de sequência que também é usado para manter as tabelas de encaminhamento atualizadas. Quando determinado caminho é perdido, por problemas com determinado nó, é ativado um mecanismo de reparação local que assume a reconstrução do caminho a partir deste ponto. Se este mecanismo não conseguir resolver o problema, o nó que ativou aquele caminho tenta encontrar outro caminho para conseguir chegar até ao nó de destino pretendido.

Descoberta de rota

Como foi referido anteriormente, quando um nó quer transmitir informação e não tem nenhuma rota válida na sua tabela de encaminhamento para o destino, ativa o processo de descoberta de rota. Primeiro, cria um pacote RREQ e transmite-o para os seus vizinhos, quando outro nó da rede recebe o pacote, caso não tenha nenhuma entrada na sua tabela de encaminhamento para aquele destino retransmite novamente o pacote. Caso tenha na sua tabela de encaminhamento uma entrada para aquele destino ou é o destino, incrementa o valor do contador de saltos e cria uma entrada na tabela de encaminhamento para o nó do qual recebeu o RREQ e para o nó fonte. Na figura 2.15 podemos ver como o RREQ é retransmitido até alcançar o destino (nó **D**). Após criar essa entrada envia um RREP para o nó fonte (nó **S**) em modo *unicast*, como se pode ver na figura 2.16. Quando o nó fonte recebe o RREP, cria uma entrada na tabela de encaminhamento para ambos os destinos, nó de destino e para o nó do qual recebeu o RREP (nó **A**). Se o nó fonte receber múltiplos RREP de diferentes caminhos, seleciona a rota com o melhor número de sequência.

Manutenção da rota

A manutenção da rota é feita através do uso de mensagens `HELLO` que verificam se o nó ainda se encontra dentro do alcance de conectividade. Se por alguma razão o nó sair fora do alcance de conectividade é iniciado de imediato um mecanismo de recuperação. Caso o mecanismo de recuperação não consiga restabelecer a rota, o nó remove a entrada da sua tabela de encaminhamento e envia uma mensagem RERR. Se o nó que originou aquela rota ainda

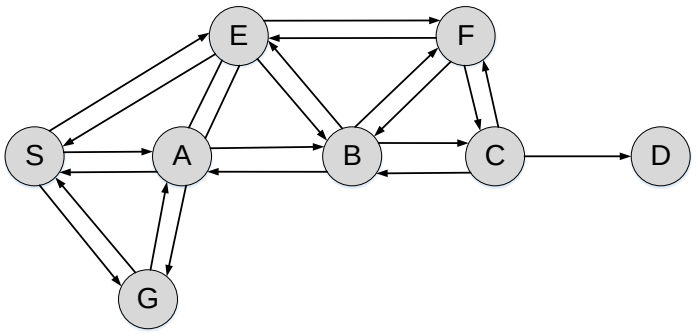


Figura 2.15: AODV *route request*

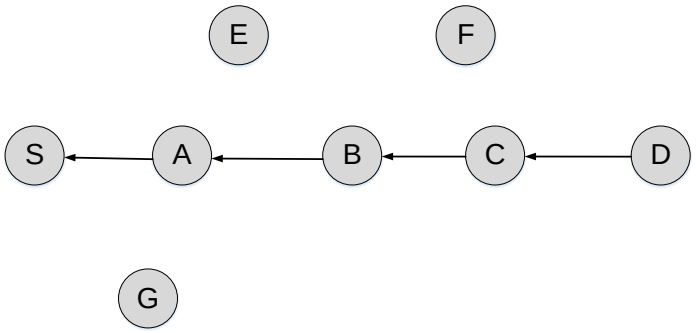


Figura 2.16: AODV *route reply*

necessitar dela, inicia novamente o mecanismo de descoberta de rota, caso contrário apenas elimina a entrada da sua tabela de encaminhamento.

Versão 2

AODVv2 como o nome indica é o sucessor do AODV que foi apresentado anteriormente. Também pertence ao grupo dos protocolos de encaminhamento reativos e partilha grande parte das suas características com a versão 1. A acumulação de informação de encaminhamento de todos os nós do caminho é uma das diferenças da versão 2 em relação à versão 1. O processo de descoberta de rota é igual ao da versão 1 com a exceção da característica de acumulação de informação. Enquanto o pacote RREQ é retransmitido cada nó intermédio adiciona o seu endereço, como se pode ver na figura 2.17. O destino responde com um pacote RREP, como na versão anterior, e o mesmo processo de acumulação de informação feito no RREQ é repetido no RREP. Desta forma, garantimos que o caminho é construído e que todos os nós intermédios conhecem um caminho para todos os nós ao longo do caminho.

Outra característica interessante é a sua eficiência energética, permitindo que um nó que esteja com pouca energia tenha a opção de não participar no processo de descoberta de rota. Caso opte por não participar não irá retransmitir as mensagens RREQ. O processo de manutenção mantém-se igual ao da versão 1.

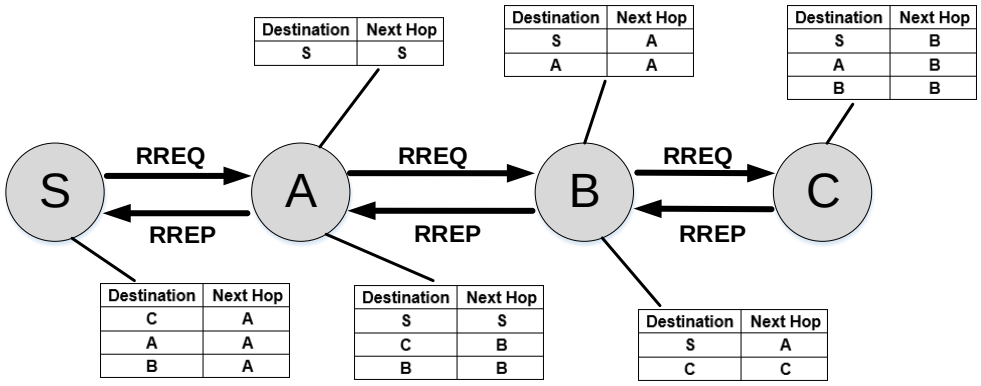


Figura 2.17: Exemplo da acumulação de informação de encaminahemnto

Qualidade de serviço

O AODVv2 funciona de forma muito idêntica ao AODV, mas no processo de descoberta de rota no AODVv2 o nó que origina o RREQ, ou seja, o nó que necessita de uma rota, vai obter informações sobre todos os nós intermédios. No AODV apenas sabe o destino e o próximo salto. Por obter informações de todos os nós que fazem parte da nova rota, o nó que origina o RREQ sabe como chegar até ao destino e a todos os nós dessa rota. Desta forma, segundo o estudo feito em [47], a sobrecarga de pacotes RREQ diminui.

Em termos de qualidade de serviço e focando apenas na versão 1, foram propostas algumas alterações para melhorar estas características. Em [1] alteram o parâmetro Active Route Timeout (ART) para melhorar a taxa de entrega de pacotes em determinado tipo de redes. Além desta análise, foram propostas algumas extensões para o protocolo base para que este tenha em conta certos parâmetros de qualidade de serviço, por exemplo [50] e [46]. Em [50] designada por Quality of Service - Ad hoc On-demand Distance Vector (QS-AODV), alteram os mecanismos base de descoberta de rotas e manutenção do AODV por forma a garantir que são estabelecidos caminhos com determinada largura de banda. Em [46] designada por Modified - Ad hoc On-demand Distance Vector (M-AODV), estabelecem vários caminhos entre a origem e o destino, onde um desses caminhos é o principal e os outros secundários. Caso algum nó do caminho principal falhe o nó de origem passa a enviar os dados pelo caminho secundário, desta forma a sobrecarga de mensagens para restabelecimento do caminho diminuiu.

2.2.5 Protocolo CTP

O CTP é um protocolo de encaminhamento do grupo dos proativos e o seu principal objetivo é descobrir as melhores rotas para determinados nós da redes. Esses nós são responsáveis por recolher e armazenar os dados que são enviados pelos outros nós. A implementação deste protocolo descrita em [15, 20], apresenta dois mecanismos fundamentais para lidar com as inconsistências das redes sem fios assim como as mudanças rápidas de topologia. O primeiro mecanismo é o *datapath validation*, que usa os pacotes de dados para validar a topologia e detetar

loops. O segundo mecanismo é o *adaptative beaconing*, que é uma extensão do algoritmo *Trickle* [27], originalmente desenvolvido para atualizações de *firmware*, mas que neste protocolo tem como função adaptar o tráfego de controlo consoante o estado da rede. Se a rede estiver estável aumenta o intervalo de tempo de envio dos pacotes de dados, caso contrário alterar esse intervalo de tempo para o valor mínimo de modo a estabilizar rapidamente a rede.

A especificação do CTP, descrita em [15], descreve o formato dos pacotes de dados e de controlo, mas algumas questões são deixadas em aberto para serem tomadas na implementação. Nesta subsecção vamos descrever o funcionamento da uma implementação do CTP designado por, CTP Noe [20]. Nesta implementação são usados os mecanismos apresentados anteriormente assim como outros mecanismos que são responsáveis por lidar com o plano de dados, de modo a melhorar o desempenho e fiabilidade do protocolo de encaminhamento.

Datapath validation* e *adaptive beaconing

Rápidas alterações da qualidade das ligações pode fazer com que os nós da rede contenham informações erradas sobre as rotas, posteriormente esta informação pode provocar *loops* e perda de pacotes. Os dois mecanismos apresentados anteriormente (*datapath validation* e *adaptative beaconing*) fazem com que este protocolo seja mais robusto contra estes problemas.

- ***Datapath validation*:** Todos os nós da rede mantêm uma estimativa de custo da rota para o *sink*, usando Expected Transmission Count (ETX) [14] como métrica. O custo de cada nó é o custo do seu próximo salto mais o custo da ligação com o próximo salto. O custo está sempre a diminuir, se um nó recebe um pacote com um custo associado ao nó que retransmitiu o pacote maior que o seu próprio custo, é sinal de que pode haver um *loop* ou informações topológicas desatualizadas. Caso isto se verifique, o nó deve desencadear o processo para atualizar a informação e reparar a rede.
- ***Adaptive beaconing*:** Protocolos de coleção normalmente enviam *beacons* (mensagens de controlo de topologia) num intervalo fixo de tempo. Se o intervalo de tempo for pequeno, a rede atualiza mais rapidamente mas por outro lado tem uma maior sobrecarga a nível de largura de banda e do consumo de energético. Se o intervalo for maior temos uma rede que reage mais lentamente às mudanças topológicas mas que tem uma sobrecarga menor. O mecanismo *adaptive beaconing* vem solucionar este problema, tendo uma rápida recuperação às mudanças topológicas com uma baixa sobrecarga. Ele baseia-se no algoritmo *Trickle* [27], que é um mecanismo para distribuir versões de código pelos nós usando um *timer* aleatório. Se determinado nó ouve os outros nós anunciar uma versão igual à sua, aumenta o seu *timer* até um valor máximo. Caso ouça uma versão diferente define o *timer* para o seu valor mínimo, para aprender rapidamente a nova versão.

Além dos mecanismos descritos anteriormente, ao nível do plano de dados esta implementação tem quatro mecanismos para tratar deste tipo de tráfego. Com estes mecanismos tenta ser mais eficiente, robusto e confiável. Esses mecanismos são: *per-client queuing*, *hybrid send queue*,

transmit timer e *transmit cache*. Na figura 2.18 podemos observar como estes mecanismos interagem entre si.

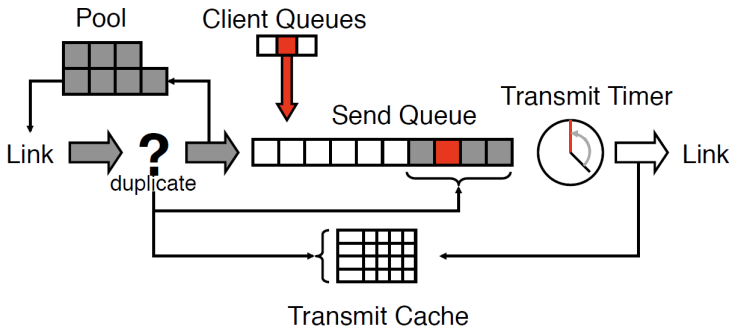


Figura 2.18: Encaminhamento no CTP Noe, retirado de [20]

O CTP Noe usa uma política de retransmissão agressiva, por defeito vai retransmitir um pacote até 32 vezes. Esta política deriva do facto que todos os pacotes terem o mesmo destino (nó que agregação de dados), e, portanto, o mesmo próximo salto. Além disto, o CTP Noe combina um atraso de retransmissão com o reparo topológico proactivo para aumentar a probabilidade de entrega dos pacotes. Em aplicações onde a receção do pacote mais recente é mais importante do que a receção de todos os pacotes este valor de retransmissão pode ser ajustado sem afetar o algoritmo de encaminhamento.

Qualidade de serviço

Em termos de qualidade de serviço, através dos mecanismos apresentados na subsecção anterior, o CTP Noe garante a fiabilidade na entrega dos pacotes. Segundo os testes realizados em [20], este protocolo de encaminhamento garante a entrega de mais de 90% dos pacotes. Outros requisitos de qualidade de serviço, como por exemplo, atraso mínimo de entrega, reserva de largura de banda, entre outros, não são garantidos por este protocolo de encaminhamento.

2.2.6 Sumário

Nesta secção, descrevemos os protocolos de encaminhamento mais usados nas redes *ad hoc* mostrando as suas principais características, mecanismos e funcionamento. Além disto, mostrou-se alguns requisitos de qualidade de serviço que são fornecidos pelos protocolos de encaminhamento ou por extensões propostas que tem como base estes protocolos de encaminhamento.

Para finalizar apresentamos uma tabela 2.2 comparativa dos diferentes protocolos de encaminhamento apresentados. Esta tabela tem em conta diferentes características/requisitos que serão relevantes para a solução que pretendemos desenvolver.

Tabela 2.2: Comparação dos protocolos de encaminhamento

Característica / Requisito	Protocolos de encaminhamento			
	OLSR	BATMAN	AODV	CTP
Otimização de tráfego para um único ponto	✗	✗	✗	✔
Diferenciação de tráfego	✗	✗	✗	✗
Fiabilidade	✗	✗	✗	✔
Métricas de ligação	ETX e ETT	*throughput*	Número de *hops*	ETX

2.3 Qualidade de serviço (QoS) em redes *ad hoc*

Esta secção tem como objetivo apresentar de forma mais detalhada as dificuldades de se garantir certos parâmetros de qualidade de serviço nas redes *ad hoc*. Na primeira subsecção descrevemos os problemas e desafios existentes nas redes *ad hoc*. Na subsecção seguinte faz-se uma abordagem das características do encaminhamento QoS. Estas duas subsecções são baseadas nas ideias apresentadas por Chakrabarti and Mishra em [7]. Nas últimas duas subsecções descrevemos duas soluções que podem ser usadas para diferenciar tráfego e/ou garantir certos parâmetros de qualidade de serviço na solução que pretendemos desenvolver.

2.3.1 Problemas e desafios

Uma rede *ad hoc* começa quando pelo menos dois nós transmitem mensagens que indicam a sua presença assim como o seu endereço IP, podendo também incluir a sua localização Global Positioning System (GPS). Se o nó **A** é capaz de estabelecer uma ligação direta com o nó **B**, através da troca de mensagens de controlo, então cada um deles terá de atualizar a sua tabela de encaminhamento para que passe a ter mais uma entrada. Quando um terceiro nó **C** junta-se à rede através do envia de mensagens de controlo, dois cenários são possíveis. No primeiro, poderá estabelecer uma ligação direta com os outros dois nós, **A** e **B**, caso esteja ao alcance dos dois. No segundo caso, apenas vai estabelecer uma ligação direta com um deles que depois será intermediário para que haja comunicação entre os três.

A mobilidade dos nós pode fazer com que as relações de acessibilidade mudem durante o tempo sendo necessário atualizar as tabelas de encaminhamento. Quantos mais nós se juntam à rede ou saem da rede, as atualizações de topologia tornam-se mais numerosas, complexas e, geralmente, mais frequentes, diminuindo assim os recursos disponíveis da rede. Encontrar

uma rota entre um par origem-destino pode tornar-se impossível se as mudanças de topologia ocorrerem com muita frequência. "Muito frequentemente" significa que a topologia de rede altera antes da última atualização de alteração de topologia propagar-se por todos os nós da rede. Num pior caso, antes da conclusão do cálculo de todas as rotas da rede após a última alteração. Dada uma janela de tempo específico, que chamamos "o comportamento de", uma rede *ad hoc* é combinatorialmente estável se e somente se a topologia alterar suficientemente devagar para permitir a propagação bem sucedida de todas as atualizações de topologia. Portanto, a estabilidade combinatória é uma consideração importante para garantir parâmetros de qualidade de serviço numa rede *ad hoc*. Estabilidade combinatória ocorre quando a distribuição geográfica dos nós móveis da rede não se altera durante um intervalo de tempo de interesse. Por exemplo, em uma sala de aula a comunicação entre computadores portáteis como nós da rede *ad hoc*. As rotas entre nós da rede, em tais casos, mudará pouco ou nada. Há outros casos, por exemplo, em operações de resgate, onde as atualizações de rotas ocorrem durante o intervalo de interesse, mas não com frequência suficiente para violar os limites da estabilidade combinatória.

Em [7], definem uma rede *ad hoc QoS-robust* em relação a um conjunto específico de garantias de qualidade de serviço somente, se tais garantias forem mantidas independentemente das atualizações de topologia que possam ocorrer. Mais estreitamente, também definem uma rede *ad hoc QoS-preserving*, se mantiver garantias de qualidade de serviço durante o intervalo entre o fim de uma atualização de topologia com sucesso até a ocorrência do próximo evento de mudança topológica. A rede *ad hoc QoS-robust* é, por definição, *QoS-preserving* mas o inverso é obviamente falsa.

A noção de qualidade de serviço, mencionada anteriormente, são garantias dadas pela rede para satisfazer um conjunto pré-determinado de restrições de desempenho do serviço. Esse conjunto de restrições pode ser, por exemplo, limite do atraso de encaminhamento, largura de banda disponível, probabilidade de pacotes perdidos entre outros. A primeira tarefa essencial é encontrar uma rota entre a origem e o destino, que tenha os recursos necessários disponíveis para atender às restrições de qualidade de serviço para o serviço desejado. Por exemplo, considerando a figura 2.19 onde os números próximos das ligações representam as suas respetivas larguras de banda (em Mbits/s). Minimizar o atraso, usar melhor os recursos da rede e minimizar o número de saltos intermédios são os principais objetivos para determinar as rotas adequadas. No entanto, suponha que o nó **A** quer estabelecer uma ligação com o nó **E** onde tenha garantido uma largura de banda de 3 Mbits/s. O encaminhamento QoS estabelecerá a rota **A-B-C-E** invés da rota **A-D-E**, embora esta última tenha menos saltos. Hoje em dia a diversidade de serviços é grande assim como os requisitos que os mesmos necessitam, por exemplo, voz, *streamming* e transferência de documentos, tem objetivos significativamente diferentes para atraso, largura de banda e perda de pacotes. Determinar a capacidade de QoS das ligações candidatas não é simples para tais cenários, e o processamento dos percursos é uma tarefa que não pode demorar "demasiado tempo". Uma vez que a rota foi selecionada para um determinado fluxo, os recursos necessários para esse fluxo devem ser reservados. Esses recursos não estão disponíveis para outros fluxos apenas ficam disponíveis quando o fluxo principal não necessitar mais deles. A minimização de atualizações de encaminhamento é um dos principais objetivos da engenharia de

redes. Estas atualizações consumem largura de banda e recursos físicos dos nós. Este objetivo é extremamente difícil de alcançar em redes *ad hoc* por causa de mudanças de estado involuntário da rede, como a entrada e saída de nós. Encaminhamento QoS está dependente da precisão do estado atual da rede, que se baseia em dois tipos: informação do estado local e informação do estado global. A informação do estado local é mantida por cada nó e inclui atraso nas filas e a capacidade residual da Central Processing Unit (CPU), atraso de propagação, largura de banda e as métricas de custo para cada ligação de saída. As informações de estado local de cada nó da rede representa a informação do estado global, que é construída pela troca de informações do estado local entre os nós da rede. Este processo de atualização da informação do estado global é também conhecido como atualizações de topologia. Como estas atualizações não são instantâneas, a informação do estado global pode ser apenas uma aproximação do estado real da rede. Em rede *ad hoc* com a mobilidade dos nós as informações do estado global podem nunca ser precisas.

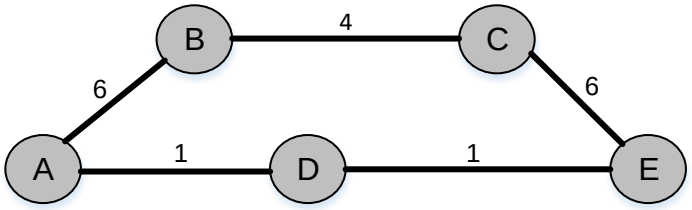


Figura 2.19: Rede ad hoc exemplo

O tráfego numa rede QoS é tratado de forma diferente. Os pacotes de controlo devem receber uma prioridade maior do que os pacotes de dados em uma rede QoS. A política de QoS pode permitir que haja diferentes prioridades para diferentes fluxos de pacotes de dados. Tratamento de pacotes de dados com múltiplas prioridades também apresenta dificuldades neste tipo de redes. Quando um pacote de dados chega à rede com uma certa prioridade a rede primeiro precisa de autenticar o pedido através da troca de pacotes de controlo. Em seguida, a rede deve encontrar uma rota com o QoS solicitado por aquela prioridade contra todos os outros fluxos de menor prioridade. Em situações de tráfego intenso, garantias de QoS para tráfego de menor prioridade pode ser extremamente difícil de alcançar ou até mesmo impossível. O desenvolvimento de políticas de encaminhamento QoS, algoritmos e protocolos para lidar com os pacotes de dados com múltiplas prioridades é uma área em constante evolução, daí a motivação para a realização deste trabalho.

2.3.2 Encaminhamento QoS em redes *ad hoc*

Os conceitos básicos apresentados na subsecção anterior constituem a base para o encaminhamento QoS nas redes *ad hoc*. Cada nó transmite *beacons* periodicamente a identificá-lo e desta forma também passa a conhecer os seus vizinhos. O mecanismo de *beaconing* é fundamental para as redes *ad hoc*, pois de outra forma o nó não saberia quem são os seus vizinhos sendo este conhecimento indispensável para o encaminhamento.

Em [8], são considerados vários mecanismos para a deteção de rotas perdidas e recuperação das mesmas. A probabilidade de não se satisfazer determinado requisitos de QoS é mais reduzida, recorrendo ao uso de rotas alternativas. As rotas perdidas são detetadas através do protocolo de *beaconing* para a deteção dos vizinhos. Na figura 2.20, se o nó **B** deteta que o nó **C** já não é seu vizinho, porque a ligação entre eles já não existe, pode tentar encontrar um caminho alternativo para chegar até **C**, neste caso seria **B-E-C** caso o requisito de QoS seja satisfeito. Caso contrário, o nó **B** notifica o nó que originou aquele caminho para que ele procure um caminho alternativo que satisfaça o requisito de QoS, como se pode observar na figura 2.21. Dependendo da política de rede, o nó **B** pode enviar a notificação para a origem sem tentar reparar o caminho perdido.

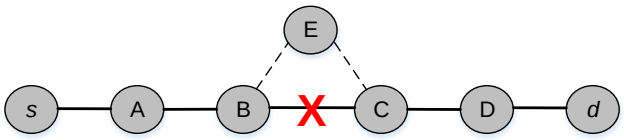


Figura 2.20: Reparação de rota

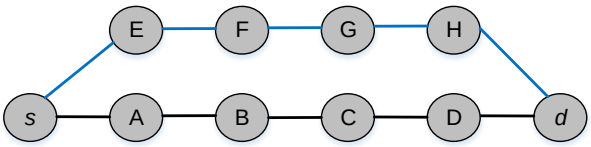


Figura 2.21: Rota alternativa

O mecanismo de estabelecimento de múltiplos caminhos redundantes, também são considerados em [8]. Estes mecanismos são usados para diminuir a probabilidade de insatisfação de requisitos de QoS assim como a falha de caminhos. Como se pode observar na figura 2.22, no mais alto nível de redundância são estabelecidas várias rotas alternativas com a mesma garantia de QoS e são usadas simultaneamente. As rotas alternativas devem ser, preferencialmente, separadas, embora isso por vezes não seja possível, os pacotes duplicados são descartados no destino. No nível mais abaixo, as rotas e os recursos associados são reservados mas não são utilizados a menos que a rota principal falhe, ou a primeira escolha para a rota alternativa falhe até a principal não estar disponível, e por aí em diante. No nível mais baixo de redundância apenas é identificada a rota mas nenhum recurso é reservado, quando a rota principal falhar os caminhos alternativos são verificados para determinar se os recursos necessários ainda estão disponíveis.

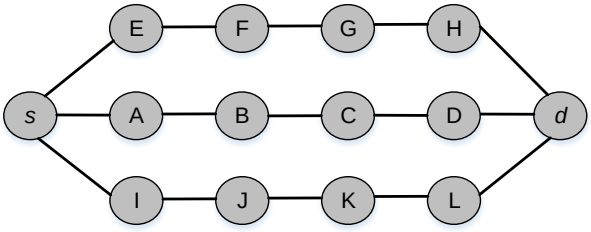


Figura 2.22: Rotas redundantes

Mudanças rápidas na topologia, como já referido, degradam as garantias de qualidade de serviço. Dado α, que se refere ao intervalo de tempo entre dois eventos de alteração de topologia consecutivos, e β, que se refere ao tempo de cálculo e propagação das atualizações topológicas resultantes da última alteração. Recordando que uma rede *ad hoc* é combinatorialmente estável somente se $\beta < \alpha$. Se a rota deixar de existir durante a atualização topológica correspondente, a garantia de QoS torna-se sem sentido, assim como manter limites de *jitter*, torna-se impraticável mesmo em uma rede *ad hoc* combinatorialmente estável se α está muito "perto" de β, segundo o que está descrito em [7].

2.3.3 Click

O `Click` [38] é um software que permite criar um router flexível e configurável. Um router baseado em `Click` é configurado a partir de módulos de processamento de pacotes chamados elementos. Cada elemento implementa uma função de um router normal, por exemplo, classificação de pacotes, filas e interface com dispositivos de rede. A configuração completa de um *router* `Click` é feita ligando os elementos em um grafo onde os pacotes fluem ao longo desse grafo passando pelos diversos elementos. Uma conexão entre dois elementos representa um possível caminho por onde os pacotes poderão passar. Segundo [35], um computador que esteja a executar `Linux`, um *router* `Click` pode encaminhar pacotes de 64-byte a uma taxa de 73000 pacotes por segundo, sendo apenas 10% mais lento do que o `Linux` sozinho.

Como foi referido anteriormente, um *router* `Click` é configurado com base em elementos que são interligados num grafo. Há vários elementos desenvolvidos que disponibilizam muitas funções para serem incorporados num *router* `Click`. Se não houver a funcionalidade que pretendemos, pode-se criar novos elementos que desempenhem funções mais específicas ou compor os elementos já existentes de novas maneiras para desempenhar outras funções. Cada elemento é um objeto `C++` e as conexões são apontadores para elementos, as principais propriedades de um elemento são:

- Classe do elemento
- Portas de entrada e saída
- *Strings* de configuração

A figura 2.23 representa as propriedades do elemento `Tee`. `Tee` é a classe do elemento, tem uma porta de entrada e duas portas de saída e a *string* de configuração é colocada dentro dos parêntesis, neste exemplo é `2`.

O `Click` fornece dois tipos de conexões para interligar dois elementos, *push* e *pull*. Na conexão *push* o elemento a montante entrega um pacote para o elemento a jusante, na conexão *pull* o elemento a jusante pede ao elemento a montante para retornar um pacote.

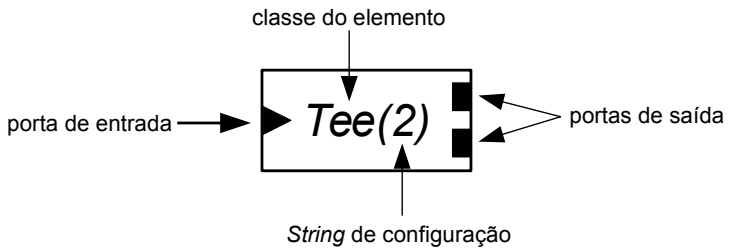


Figura 2.23: Propiedades de um elemento Click, adaptado de [35]

```
src :: FromDevice(eth1);
ctr :: Counter;
sink :: Discard;
src -> ctr;
ctr -> sink;
```

Bloco de Código 2.1: Exemplo da linguagem `Click`

A configuração do *router* é escrita em uma linguagem textual simples com duas construções importantes: declarações e conexões. A declaração diz qual o elemento que deve ser criado e as conexões definem como esses elementos estão ligados. Na figura 2.24 podemos ver um exemplo de uma configuração de um *router* `Click` que conta os pacotes de entrada e em seguida descarta-os. No bloco de código 2.1 temos a linguagem utilizada para criar essa configuração.

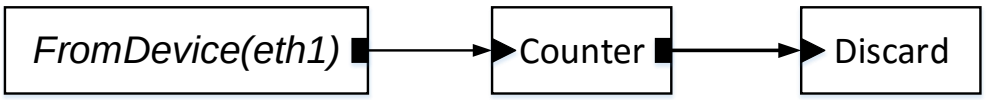


Figura 2.24: Configuração exemplo de um router em Click, adaptado de [35]

Para concluir e analisando configurações mais avançadas e mais relevantes para o trabalho que pretendemos desenvolver, em [35], apresentam um exemplo de como se pode diferenciar tráfego utilizando o `Click`. Os *routers* que se encontram à entrada e saída da uma rede onde há diferenciação de tráfego além de encaminhar os pacotes têm de os classificar e colocar uma etiqueta de acordo com o tipo de tráfego. Os *routers* do núcleo da rede encaminham com base na etiqueta colocada à entrada. A classificação, colocação da etiqueta e o encaminhamento com base nessa classificação são componentes que no `Click` correspondem a elementos, o que dá ao administrador total controlo sobre como estão organizados. Na figura 2.25 podemos ver um exemplo de uma configuração em `Click` com diferenciação de tráfego, assim como o seu encaminhamento diferenciado.

Nesta configuração o tráfego é separado em quatro tipos, consoante o seu IP Differentiated Service Code Point (DSCP). Os primeiros três tipo estão limitados a uma determinada taxa de transmissão e o último representa o encaminhamento normal.

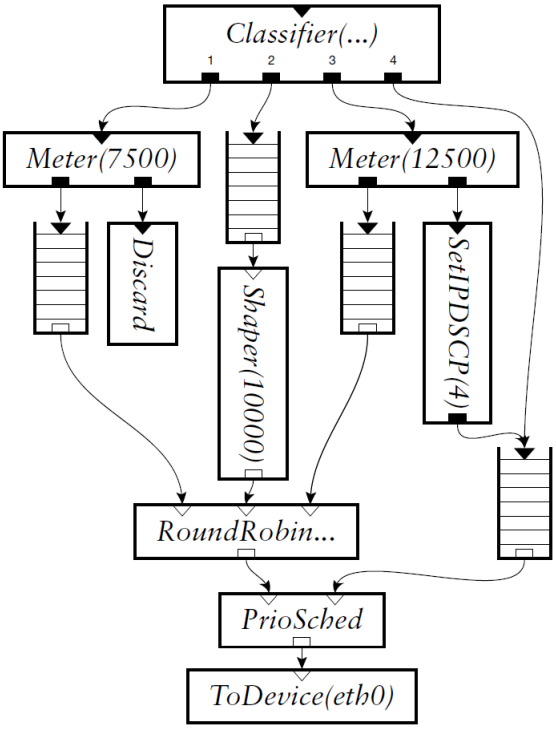


Figura 2.25: Configuração `Click` para diferenciação de tráfego, retirado de [35]

2.3.4 Controlo de tráfego em Linux

A partir da versão 2.2 do *kernel* `Linux` o seu subsistema de rede foi completamente redesenhado e passou a incluir o `iproute2`. O `iproute2` consiste num pacote de *software* que fornece várias ferramentas de rede que suportam encaminhamento avançado, túneis e controlo de tráfego. Esta última ferramenta é a mais relevante do pacote de *software* `iproute2` para o trabalho que pretendemos desenvolver. Esta ferramenta permite classificar, priorizar, partilhar e limitar o tráfego de entrada e saída. Além disto, já tem vários algoritmos de escalonamento que permitem modificar a forma como o tráfego é tratado. Para chegarem a esta fase de encaminhamento diferenciado, têm de passar pelas seguintes fases:

- Os pacotes são aceites ou descartados consoante a política implementada.
- Caso sejam aceites, são filtrados para ser atribuída uma classe.
- Os pacotes são enviados para as filas associadas à classe que foi atribuída.
- Cada fila tem um algoritmo de escalonamento associado e os pacotes são encaminhados consoante esse algoritmo.

Ferramenta de controlo de tráfego (`tc`)

Para termos acesso as funções descritas anteriormente, o `iproute2` disponibiliza um comando, designado por `tc`, que permite definir as políticas de QoS, filtros e algoritmos de escalonamento

associados. Os elementos que são definidos pelo `tc` são os seguintes:

- ***Queueing Disciplines* (`qdisc`)**: Algoritmos que controlam a entrada e saída dos pacotes.
- **Classes (`class`)**: São usadas para diferenciar os pacotes.
- **Filtros (`filter`)**: São usados para classificar/atribuir as classes aos pacotes.
- **Políticas (`policers`)**: São usadas para evitar que o tráfego ultrapasse determinado limite.

Existe dois tipos de `qdisc`, as *classless* e *classful*. O primeiro tipo de `qdisc` apenas permitem controlar uma interface no seu todo sem subdivisões. O segundo tipo de `qdisc` são muito usadas em situações mais complexas ou situações onde queremos ter mais flexibilidade no controlo do tráfego.

Classless `qdisc`

Temos várias `qdisc` já incorporadas no *kernel* `Linux` do tipo *classless*:

- ***FIFO* (`pfifo` e `bfifo`)**: Uma `qdisc` simples que implementa o algoritmo *First In First Out*. O seu tamanho pode ser limitado através do número de pacotes (`pfifo`) ou por bytes (`bfifo`).
- **`pfifo_fast`**: A *qdisc* por defeito do *kernel* `Linux`. Muito idêntica à `pfifo` mas com priorização de tráfego com base no Type of Service (ToS).
- ***Token Bucket Filter* (`tbf`)**: Uma `qdisc` simples que é perfeita para definir uma determinada taxa de transmissão numa interface ou classe.
- ***Stochastic Fair Queuing* (`sfq`)**: Uma `qdisc` que tenta distribuir de forma justa o tempo de transmissão pelos diversos fluxos.
- ***Random Early Detection (RED)*** e ***Generic Random Early Detection (GRED)***: São `qdisc` adequadas para os *routers* de *backbone* com taxas de dados superiores aos 100 Mbits/s.
- ***DiffServ Mark* (`ds_mark`)**: com base no campo DSCP dos pacotes implementa uma metodologia de diferenciação de serviços.

Classful `qdisc`

Este tipo de `qdisc` são usadas para modelar diferentes tipos de tráfego e a principal `qdisc` usada é a Hierarchical Token Bucket (HTB). Este tipo de `qdisc` são baseadas em uma hierarquia, cada interface tem uma *root* `qdisc` que comunica com o *kernel*. Essa *root* `qdisc` tem classes associadas e `qdisc` associadas a essas classes. Na figura 2.26 temos uma hierarquia exemplo,

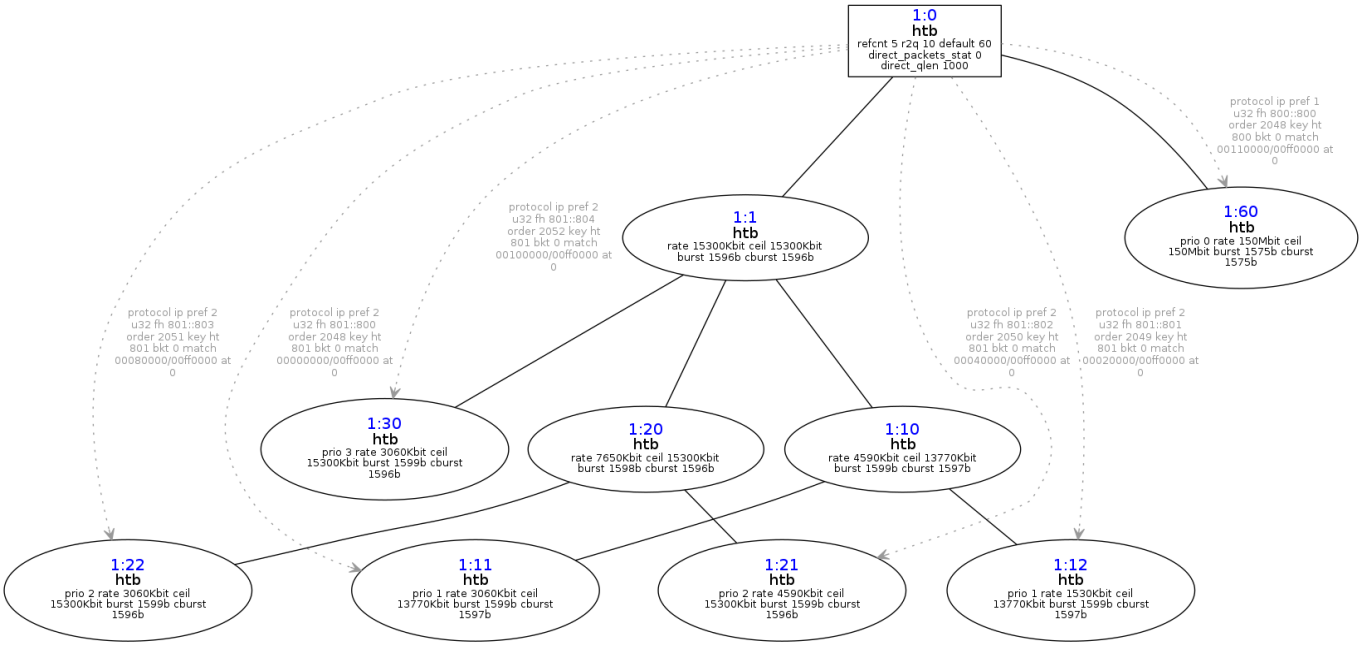


Figura 2.26: Estrutura exemplo do HTB

com 6 classes que podem ser associadas ao tráfego. Essas classes são: `1:11`, `1:12`, `1:21`, `1:22`, `1:30` e `1:60`.

Basicamente, cada classe abaixo da *root* `qdisc` pode ser responsável por determinado tipo de tráfego. Por exemplo, se quisermos tratar de forma diferenciada o tráfego Voice over Internet Protocol (VoIP) do tráfego File Transfer Protocol (FTP) podemos criar duas classes diferentes. Cada classe está associada a um tipo de tráfego e pode ter diferentes `qdisc`. Como foi referido a principal `qdisc` é a HTB, mas não é a única, por omissão no *kernel* `Linux` temos ainda a `PRIO`.

- **HTB**: Coordena a ordem em que as classes são servidas com base num esquema hierárquico. A HTB é a substituta da Class Based Queueing (CBQ) que está obsoleto mas que ainda está disponível no *kernel* `Linux`.
- **`PRIO`**: Coordena a ordem como as classes são servidas segundo um esquema de prioridades.

2.3.5 Sumário

Nesta secção foram apresentados os problemas existentes nas redes *ad hoc* para se garantir certos requisitos de qualidade de serviço. Para além dos problemas, apresentamos também alguns mecanismos usados para colmatar esses problemas. Por fim, apresentamos duas ferramentas que podem ser usadas para garantir requisitos de qualidade de serviço em redes *ethernet* e *wireless*. Estas duas ferramentas podem ser usadas na solução que pretendemos implementar.

Capítulo 3

Arquitetura

Este capítulo tem como objetivo descrever os problemas endereçados no contexto dos cenários de aplicações que temos como alvo. Além disso, vamos dar alguns exemplos de aplicações e explicar porque que o tráfego pode necessitar de requisitos de qualidade de serviço diferentes. Para finalizar o capítulo vamos descrever o algoritmo de escalonamento que vamos usar para garantir os requisitos de qualidade de serviço e explicar conceitos que pretendemos implementar.

3.1 Aplicação e problemas

O cenário alvo da solução que pretendemos desenvolver no decorrer deste projeto, consiste num conjunto de nós que formam uma rede Mobile Ad Hoc Network (MANET). Nessa rede o tráfego que está a ser gerado pelos nós tem requisitos específicos e o destino desse tráfego é o mesmo para todos os nós da rede. Esse ponto comum é designado por nó coordenador ou *sink*, e a sua função é guardar e tratar todo o tráfego da rede. Tráfego com requisitos específicos consiste, por exemplo, em determinado tráfego necessitar de requisitos mínimos de largura de banda e a outro necessitar de fiabilidade na entrega dos dados. Visto isto, os nós da rede têm de tratar de forma diferente cada tipo de tráfego.

O cenário que vamos usar para testar a nossa solução é a rede do projeto Vital Responder [45]. O objetivo deste projeto de investigação é desenvolver um sistema de monitorização de bombeiros para situações críticas, tais como, combate a incêndios, resgates, catástrofe entre outras. Cada bombeiro está equipado com uma t-shirt, desenvolvida pela empresa Biodevices[1], que é capaz de medir informações relevantes sobre o indivíduo (Eletrocardiograma (ECG), temperatura corporal, movimento através de acelerómetro e giroscópio). Os dados obtidos pela t-shirt assim como os dados que são obtidos por outros sensores são enviados para o nó coordenador. Após processar os dados recebidos o nó coordenador pode enviar alertas caso algo não esteja dentro da normalidade ou apenas representar os dados graficamente. O sistema de alertas refere-se a mensagens ou pedidos de controlo enviados pelo capitão da equipa que está com o coordenador.

[1] http://www.vitaljacket.com

A rede MANET do projeto Vital Responder é composta por 6 elementos. Desses 6 elementos, 5 são responsáveis por medir os valores dos sensores e enviar esses dados para o coordenador, ou reencaminhar os dados provenientes de outros nós. Estes 5 elementos representam os 5 bombeiros que estão no terreno. O sexto elemento é o coordenador que é responsável por receber os dados de todos os nós, guardá-los e tratá-los. Os 5 elementos que representam os bombeiros que estão no terreno, têm de ter em consideração a possibilidade do tráfego da rede necessitar de requisitos de qualidade de serviço diferentes. Na versão corrente do projeto, os nós da rede MANET estão a enviar/encaminhar o tráfego sem ter em consideração os seus requisitos.

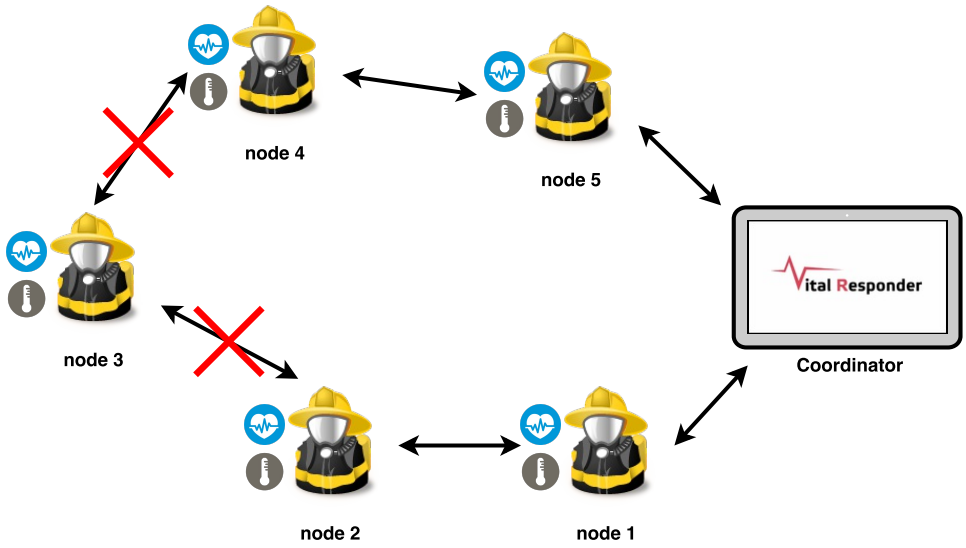


Figura 3.1: Rede MANET do Vital Responder

3.2 Tráfego com requisitos diferentes

Quando uma rede MANET é estabelecida não tem em consideração a possibilidade de existir tráfego com características diferentes e assim sendo irá encaminhá-lo de forma homogénea. Este método de encaminhamento não é prejudicial caso a aplicação que está a correr sobre a rede MANET não necessite dessa diferenciação para o seu bom funcionamento. Por exemplo, uma aplicação de *chat* não necessita desta diferenciação, porque apenas tem um tipo de tráfego que são mensagens de texto. Em outro tipo de aplicações esta diferenciação é exigida e caso se trate de uma aplicação crítica, como no caso do projeto Vital Responder, essa diferenciação ainda é mais importante.

Na rede MANET do Vital Responder os pacotes de dados que transportam informação sobre o ECG tem maior prioridade do que os pacotes de dados que transportam informação sobre a temperatura. Isto é apenas um exemplo do Vital Responder, mas que serve de exemplo para outro tipo de aplicações. Por exemplo, numa aplicação de vídeo conferência com *chat*, os pacotes de dados que transportam informação sobre o vídeo têm requisitos diferentes dos pacotes de dados que transportam as mensagens de texto do *chat*. O tráfego necessita de requisitos

de qualidade de serviço diferentes pelo facto de ter características diferentes ou até mesmo pela sua importância para o bom funcionamento da aplicação. Voltando ao exemplo do Vital Responder, os pacotes de dados do ECG necessitam de ser encaminhados o mais rápido possível, porque contêm dados sensíveis que caso ocorra uma situação de perigo devem ser recebido pelo coordenador atempadamente. Os pacotes de dados de temperatura transportam parâmetros que demoram algum tempo a variar e então não necessitam de um requisito mínimo de atraso, podem simplesmente ser enviados usando a abordagem de *best-effort*. Apesar de não haver exemplos no Vital Responder de tráfego com requisitos de largura de banda, noutro tipo de aplicação pode haver essa necessidade, que é o caso da aplicação de vídeo conferência.

3.3 Disconnection Tolerant Network (DisToNet)

Este trabalho foi desenvolvido por Lima et al. em [30] e tem como objetivo colmatar a perda de pacotes em redes MANET com desconexões esporádicas entre os nós da rede. A solução proposta para resolver este problema, consiste no armazenamento local dos pacotes quando a conexão para o destino não está disponível. Nesta implementação existem dois *buffers* designados por *Small Buffer* (`SB`) e *Large Buffer* (`LB`). O primeiro é responsável por armazenar a cópia de todos os pacotes que passam pela *stack* (os que são originados localmente e os que vão ser encaminhados). O segundo *buffer* armazena os pacotes durante o período de desconexão. O `SB` é muito mais pequeno do que o `LB`, porque apenas precisa de guardar a cópia dos pacotes durante o período entre a desconexão e a deteção da desconexão por parte do protocolo de encaminhamento. Quando um nó recebe um pacote verifica se a ligação para o destino desse pacote está disponível. Se essa ligação estiver disponível o pacote é copiado para o `SB` e depois é enviado normalmente pela *stack*. Se essa ligação não estiver disponível o pacote é copiado para o `LB` e quando a ligação estiver novamente disponível é enviado. Os dados do `SB` são colocados no `LB` quando a desconexão é detetada. Servem deste modo para salvaguardar perdas que possam ter ocorrido antes da deteção da desconexão que já podia existir. Na figura 3.1 podemos ver que o `node3` não tem nenhuma ligação disponível para chegar até ao nó coordenador. Todos os pacotes que o `node3` tente enviar para o nó coordenador vão ser armazenados localmente pela DisToNet, até que uma das ligações fique novamente disponível. Na figura 3.2 temos o diagrama de fluxo da solução que foi implementada na DisToNet. No desenvolvimento do trabalho da DisToNet foram descobertos alguns problemas de implementação e *design* que invalidam o seu funcionamento. No entanto o conceito subjacente foi tido em conta neste trabalho.

3.4 Problemas dos protocolos de encaminhamento

Os principais problemas dos protocolos de encaminhamento nas redes MANET incluem: sobrecarga do tráfego de controlo e o tempo de propagação da informação de uma alteração. Ambos os problemas devem ser considerados quando se escolhe um protocolo de encaminhamento para uma determinada aplicação.

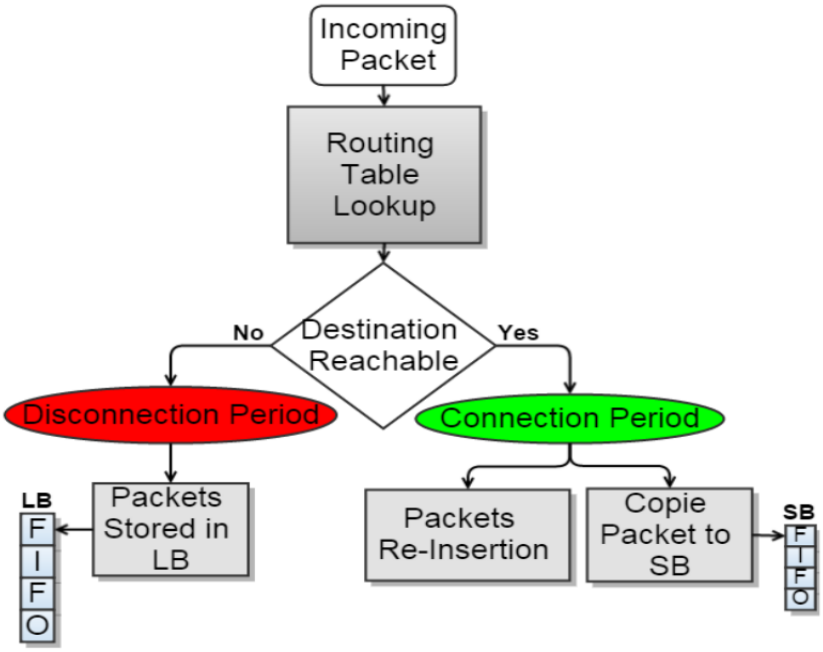


Figura 3.2: Diagrama de fluxo da DisToNet, retirado de [30]

O tempo de propagação da informação de uma alteração, também designado por tempo de convergência, é o fator mais importante para o bom funcionamento da solução que pretendemos desenvolver. A nossa solução está dependente da informação local que cada nó contém sobre a rede. É através dessa informação que vamos escolher um dos dois estados apresentados nas secções 3.5 e 3.6. Após o protocolo de encaminhamento detetar uma alteração, demora algum tempo até que todos os nós atualizem a sua informação local. Esse tempo está relacionado com as mensagens de controlo demorarem até alcançarem os nós da rede para serem tratadas.

3.5 Priorizar o tráfego

Para conseguirmos priorizar o tráfego, acomodando o seu requisito de atraso, na nossa solução optamos por usar o algoritmo de escalonamento Hierarchical Token Bucket (HTB) que após a análise dos artigos [3, 25], concluímos que se trata de um algoritmo preciso e eficiente. Ao usarmos este algoritmo o termo "qualidade de serviço" na nossa rede refere-se à disponibilização de classes de tráfego com taxas de transmissão e prioridades diferentes. O tráfego com maior prioridade tem uma taxa de transmissão maior do que o tráfego com menor prioridade. Cada uma dessas classes tem uma taxa de transmissão mínima, máxima e uma prioridade. A taxa de transmissão mínima será garantida independentemente da ocupação das restantes classes. Se a taxa de ocupação das restantes classes não for elevada cada classe poderá aumentar a sua taxa de transmissão até ao valor máximo. Por exemplo, no caso de termos 3 classes onde uma dessas classes tem taxa de ocupação nula, os seus recursos vão ser partilhados pelas outras duas classes. Esses recursos vão ser partilhados tendo em conta a prioridade das outras duas classes. A classe com maior prioridade tem maior prioridade sobre os recursos.

O número de classes criadas e a forma como dividimos a largura de banda disponível por essas classes foi baseada numa solução proposta no projeto **ADHOCSYS** [6]. Este projeto pode ser aplicado em vários cenário, mas o principal objetivo é fornecer o acesso à Internet em zonas rurais e de montanha, onde não há a possibilidade de ter esse acesso via cabo. Cada aldeia e cidade tem uma *gateway* que liga o *backbone* da rede **ADHOCSYS** aos utilizadores, fornecendo desta forma o acesso à Internet. O tráfego da rede **ADHOCSYS** foi dividido em várias classes, como se pode ver na tabela 3.1, de forma a garantir um mecanismo eficiente de diferenciação de tráfego.

Tabela 3.1: Classes da rede ADHOCSYS [6]

Classe	Aplicação
I	Requisitos de latência e pequena largura de banda (VoIP, chat)
II	Alta largura de banda (transferência de ficheiros)
III	Interativo, *best-effort* (*Web browsing*, e-mail)
IV	*Routing* e informações de bateria
V	Chamadas de emergência
VI	Alta largura de banda e requisitos de latência (*streaming video*)
VII	Aplicações P2P
VIII	Tráfego não classificado

Na figura 3.3 podemos ver a estrutura do HTB usada na rede **ADHOCSYS** e a divisão dos diversos serviços pelas três classes criadas. As características de cada classe criada estão representadas na tabela 3.2.

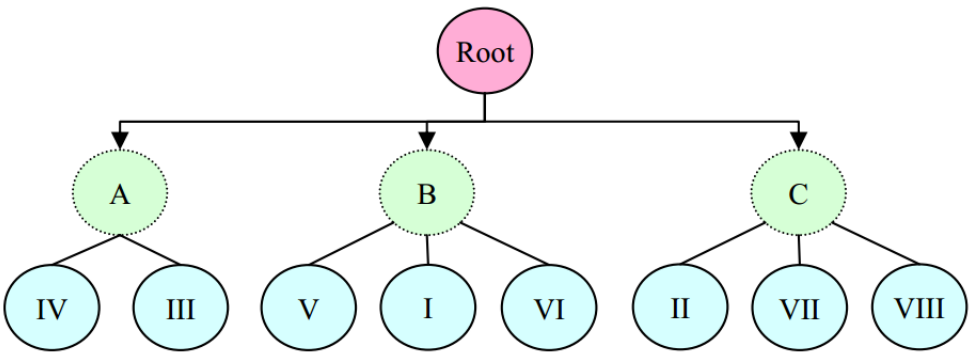


Figura 3.3: Estrutura HTB da rede ADHOCSYS, retirado de [6]

A largura de banda mínima da classe **A** é superior às das restantes classes, porque pretende servir o tráfego com requisitos de largura de banda. A classe **B** tem um valor de prioridade inferior (maior prioridade sobre as outras duas classes) devido às restrições de atraso. A classe **A** pretende servir o tráfego essencial para os utilizadores e para a rede **ADHOCSYS**, a classe **B** o tráfego com restrições de atraso e a classe **C** o tráfego não caracterizado.

As características das classes criadas na nossa solução são apresentadas com mais detalhe

Tabela 3.2: Características da classes da rede ADHOCSYS [6]

Parâmetros do HTB			
Classe	Taxa mínima	Taxa máxima	Prio
A	70%	100%	2
B	20%	90%	1
C	10%	90%	3

no capítulo 5. Na nossa solução pretendemos garantir dois requisitos de qualidade de serviço, atraso e largura de banda. Este dois requisitos são garantidos apenas até ao próximo salto e não até ao nó coordenador. Na figura 3.4 podemos ver a estrutura de classes usada na nossa solução, onde temos três classes para servir três tipos de tráfego diferentes. Como no projeto Vital Responder está incorporada a DisToNet, que foi apresentada na secção 3.3, foram criadas classes específicas para servir o tráfego proveniente da DisToNet. Por exemplo, o tráfego que circula na rede com requisitos de qualidade de serviço da classe **A** vai para a classe **A1**, o tráfego que sai da DisToNet com o mesmo requisito vai para a classe **A2**, porque são domínios diferentes e cada domínio tem as suas classes. Apenas a classe **C** é partilhada tanto pelo tráfego que circula na rede como pelo tráfego proveniente da DisToNet.

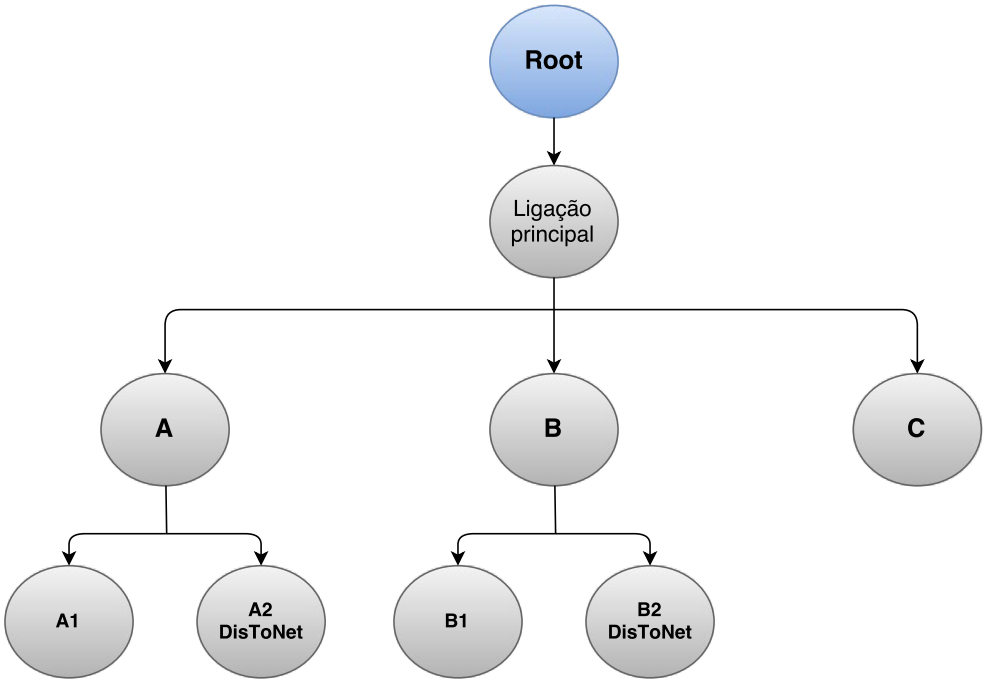


Figura 3.4: Estrutura HTB primeiro caso

3.6 Balancear o tráfego

Para além de garantir os dois requisitos de qualidade de serviço apresentados anteriormente, queremos também balancear o tráfego quando a rede assim o proporcionar. Se houver apenas uma ligação para chegar até ao nó coordenador, vamos priorizar de forma a garantir os dois requisitos de qualidade de serviço apresentados na secção anterior. Caso existam duas ligações para chegar até ao nó coordenador vamos priorizar e balancear o tráfego. Por exemplo, na figura 3.1 caso o `node 3` tivesse as ligações para o `node 2` e `node 4` disponíveis, podia balancear o tráfego. O conceito de balanceamento de tráfego que pretendemos implementar não deve ser comparado com a tradicional definição usada nas redes Internet Protocol (IP) convencionais com duas ou mais ligações à Internet. Nesse tipo de redes a segunda ligação apenas é usada quando a primeira está sobrecarregada, não está disponível ou quando se pretende distribuir de forma uniforme o tráfego entre as ligações. Neste projeto o conceito de balanceamento de tráfego é referente à priorização de tráfego, ou seja, o tráfego com maior prioridade usa a ligação principal e o restante tráfego usa a ligação secundária.

Neste caso a estrutura de classes do HTB é diferente da que foi apresentada na secção anterior. Apesar do número de classes ser o mesmo as suas características e o nó de destino são diferentes. A figura 3.5 representa a estrutura usada neste caso, onde temos duas classes principais referentes às duas ligação válidas e as classes que pertencem a cada uma dessas classes. Nesta situação o tráfego que seja classificado com o requisito da classe **A** é enviado pelo vizinho principal e o tráfego que seja classificado com requisito da classe **B** e **C** é enviado pelo vizinho secundário.

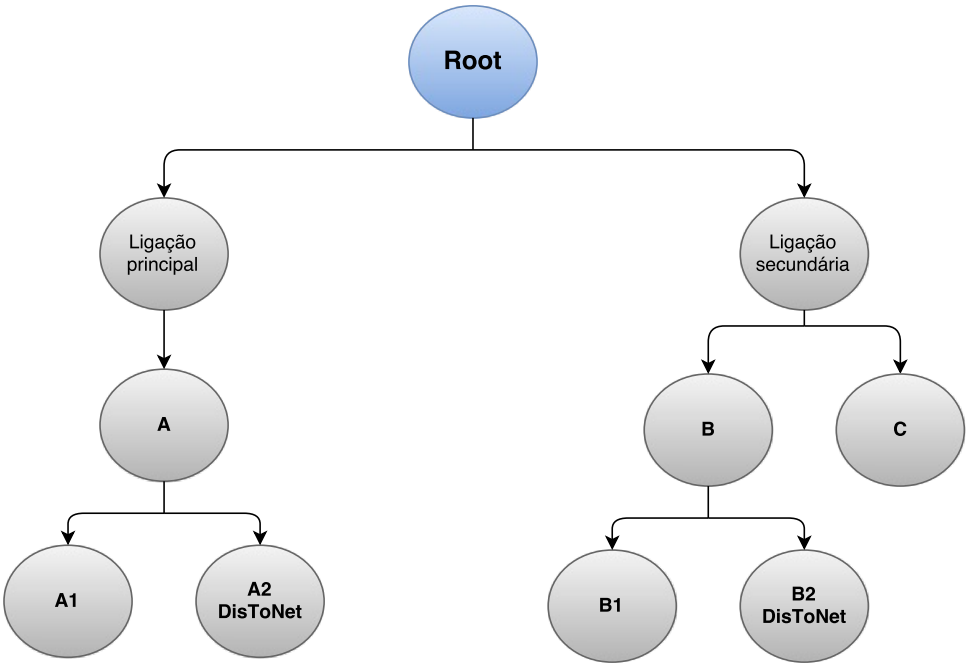


Figura 3.5: Estrutura HTB segundo caso

3.7 Sumário

Neste capítulo apresentamos o cenário alvo da nossa solução e o problema que pretendemos resolver nesse cenário. Introduzirmos o conceito de DisToNet para se perceber porque que vamos criar classes específicas para este domínio. Para finalizar, apresentamos a estrutura de classes do HTB da nossa solução assim como o conceito de balanceamento de tráfego.

Capítulo 4

Avaliação dos protocolos de encaminhamento

Este capítulo tem como objetivo medir o desempenho de dois protocolos de encaminhamento diferentes e descrever algumas das suas funcionalidades extra. Os dois protocolos de encaminhamento são o Optimized Link State Routing (OLSR)v2 e o Better Approach To Mobile Adhoc Networking (BATMAN). A versão do BATMAN que vamos testar neste capítulo é a que atua no nível 3 do modelo Open Systems Interconnection (OSI), também designada por BATMANd. Vamos medir o desempenho dos protocolos de encaminhamento em termos do tempo de convergência em quatro situações diferentes. Como vimos, na secção 3.4, o tempo de convergência é um fator importante para o desempenho da solução que pretendemos desenvolver.

Nas três primeiras secções vamos descrever a arquitetura da rede de testes, os elementos da rede Mobile Ad Hoc Network (MANET) usada para realizar os testes assim como todos os *scripts* criados para instanciar a rede. Esta rede para além de ser usada para testar os protocolos de encaminhamento, será também usada para testar o algoritmo desenvolvido no decorrer deste projeto.

4.1 Arquitetura da rede de testes

Para conseguirmos fazer a gestão dos nós da rede MANET, montámos uma infraestrutura que permite gerir os nós localmente e remotamente. Essa infraestrutura tem três redes distintas, a primeira é a rede do edifício onde estão os dispositivos (rede interna), a segunda é a rede MANET e a última é a rede de gestão.

A rede de gestão serve somente para dar acesso à **Internet** e configurar via Secure Shell (SSH) os nós. Cada nó da rede tem uma interface *ethernet* e *wireless*, a interface *ethernet* está ligada na rede de gestão e a interface *wireless* serve para ligar-se à rede MANET. A infraestrutura montada está representada na figura 4.1, o `ROUTER ISP` fornece o acesso à **Internet** na rede interna e na rede de gestão. Como não é possível ligar os nós diretamente à rede interna, por

causa de políticas de rede e também por não ser possível fazê-lo fisicamente, colocamos um *router* intermédio (`ROUTER PI`) que faz a comunicação entre a rede *wireless* interna e a rede de gestão. Todos os nós da rede MANET têm configurado como *default gateway* o `ROUTER PI`.

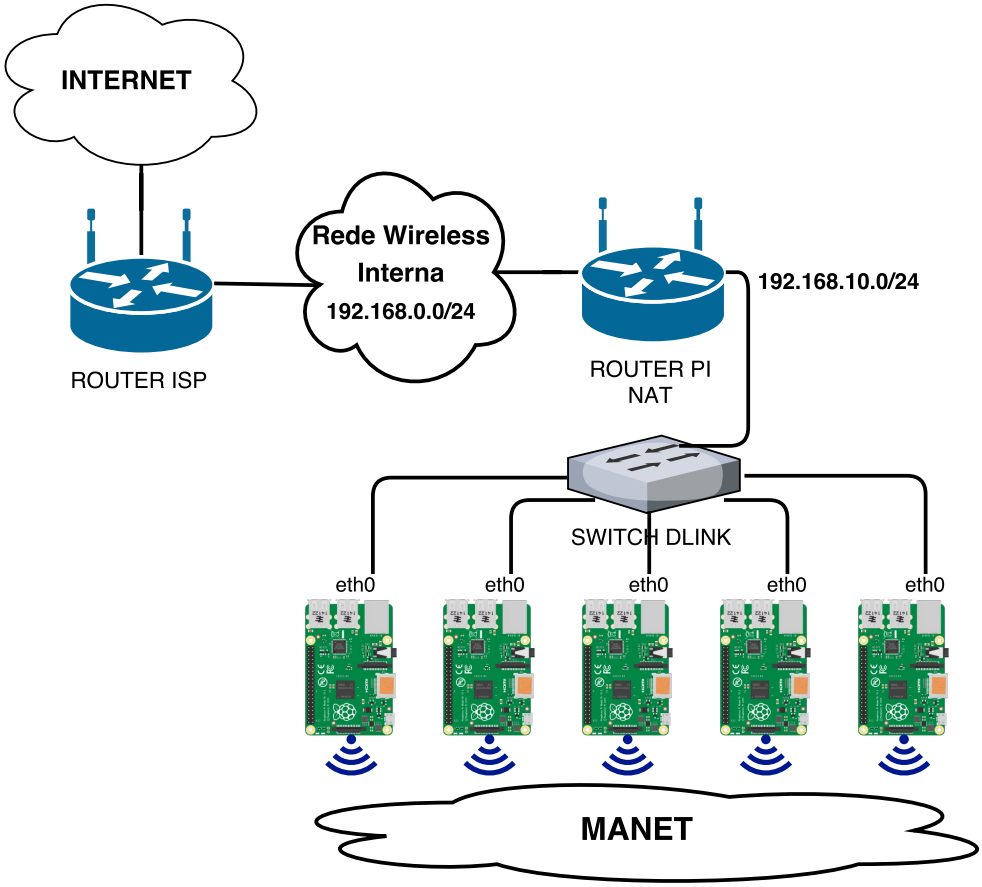


Figura 4.1: Arquitetura da rede de testes

Para conseguirmos configurar os nós via SSH quando estamos fora da rede interna, da rede de gestão e da rede MANET, redirecionamos a porta do serviço SSH de cada nó no `ROUTER ISP` para o `ROUTER PI` e no `ROUTER PI` para o respetivo nó. Cada nó tem o serviço SSH configurado na porta 220α, onde α corresponde ao `ID` do nó. Por exemplo, o `node1` tem o serviço SSH configurado na porta `2201`. Desta forma, quando estamos na sala onde estão os dispositivos, ligamos o nosso computador ao `SWITCH DLINK` e temos acesso ao terminal SSH via endereçamento interno. Quando estamos fora da sala sem acesso direto à rede de gestão, o terminal SSH de cada nó é acessível via endereçamento público.

4.2 Elementos da rede

4.2.1 Raspberry Pi

O Raspebrry Pi é um computador de tamanho reduzido que foi desenvolvido pela Raspberry Pi Foundation [17] com a intenção de promover o ensino de programação nas escola. Passou também a ser integrado no mundo da investigação devido ao seu preço, sistema operativo e propriedades do *hardware*. Devido a estas características facilmente se adapta para o desenvolvimento de protótipos, testes e em alguns casos produtos finais. Tem sido lançadas diferentes versões do Raspberry Pi: A, A+, B, B+, 2 e por último lançado em Fevereiro de 2016, Raspberry Pi 3. Na rede de testes são usados os modelos B e B+, que estão representados na figura 4.2.

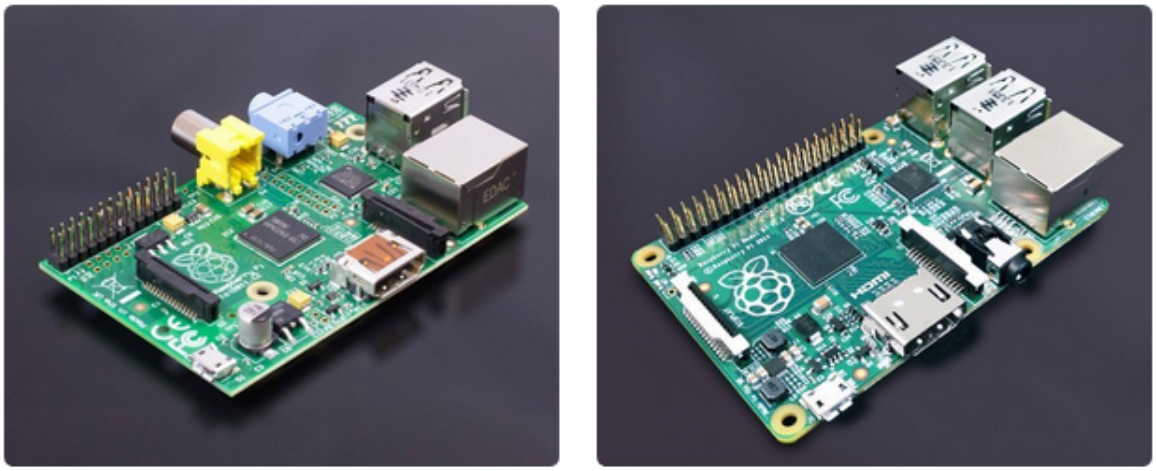


Figura 4.2: Rpi B vs Rpi B+

Em termos das propriedades do *hardware* o modelo B contém um processador ARM1176JZF-S de 700 MHz, 512 MB de memória Random Access Memory (RAM), duas portas Universal Serial Bus (USB) e uma interface ethernet 10/100. O modelo B+ mantém o mesmo processador e memória RAM, difere apenas no número de portas USB que passam a ser 4 em vez de 2, suporte para cartões de memória microSD, melhor eficiência enérgica e mais pinos General Purpose Input/Output (GPIO).

O sistema operativo escolhido foi o Raspbian[1] que é uma versão baseada em Debian para a plataforma Raspberry Pi, desenvolvido por uma pequena equipa externa à Raspebrry Pi Foundation. Mesmo não estando afiliada à Raspebrry Pi Foundation este sistema operativo é fornecido como o sistema operativo de suporte oficial para a plataforma Raspberry Pi.

[1] https://www.raspbian.org/

4.2.2 Placa de rede *wireless*

As placas de rede *wireless* usadas na rede MANET são o modelo TL-WN722N da marca TP-Link. Escolhemos este modelo porque num trabalho que foi realizado anteriormente [13], foram testados vários modelos de placas de rede *wireless* e este modelo foi o que obteve melhores resultados em termos da relação de alcance e consumo energético.

Figura 4.3: TP Link TL-WN722N

4.3 Instanciar a MANET

Na lista abaixo podemos ver todos os passos que são necessários para os nós conseguirem instanciar a rede MANET. Em cada passo mencionamos o *script* que foi criado para automatizar essa tarefa:

1. Configurar a interface de rede *wireless* em modo *ad hoc* com o mesmo Service Set Identifier (SSID) e canal dos restantes nós.
   - `create-adhoc-interface.sh <nodeID> <interface>`
2. Executar o *daemon* do protocolo de encaminhamento.
   - OLSRv2: `meshNetwork-OLSRv2.sh <nodeID> <interface>`
   - BATMANd: `meshNetwork-Batmand.sh <nodeID> <interface>`
3. Bloquear determinado nó através do endereço Media Access Control (MAC) da sua interface de rede *wireless* (apenas aplicável em cenários de teste).
   - `firewall-rule-lockMAC.sh <nodeX> <nodeY> ... <nodeZ>`

Os *scripts* principais são o `meshNetwork-OLSRv2.sh` e `meshNetwork-Batmand.sh` que recebem como argumentos o `ID` do nó e a interface de rede na qual vai ser configurada a rede

MANET. Ambos os *scripts* usam o *script* `create-adhoc-interface.sh` que é responsável por configurar a interface de rede *wireless* em modo *ad hoc* com o SSID `"meshNetwork"`.

Como as interfaces de rede *wireless* usadas na rede de teste têm um alcance de 150 metros, torna-se impossível simular topologias com mais de um salto dentro de uma sala. Por isso, criamos o *script* `firewall-rule-lockMAC.sh` que cria regras `IPTABLES` para descartar os pacotes com um determinado endereço MAC de origem. Como se pode observar na figura 4.4, as linhas tracejadas representam o alcance real de cada nó e as setas as ligações válidas após criarmos as seguintes regras:

Executar no `node1`

- Bloquear todos os pacotes com o endereço MAC de origem do `node3`
  - `firewall-rule-lockMAC.sh node3`

Executar no `node3`

- Bloquear todos os pacotes com o endereço MAC de origem do `node1`
  - `firewall-rule-lockMAC.sh node1`

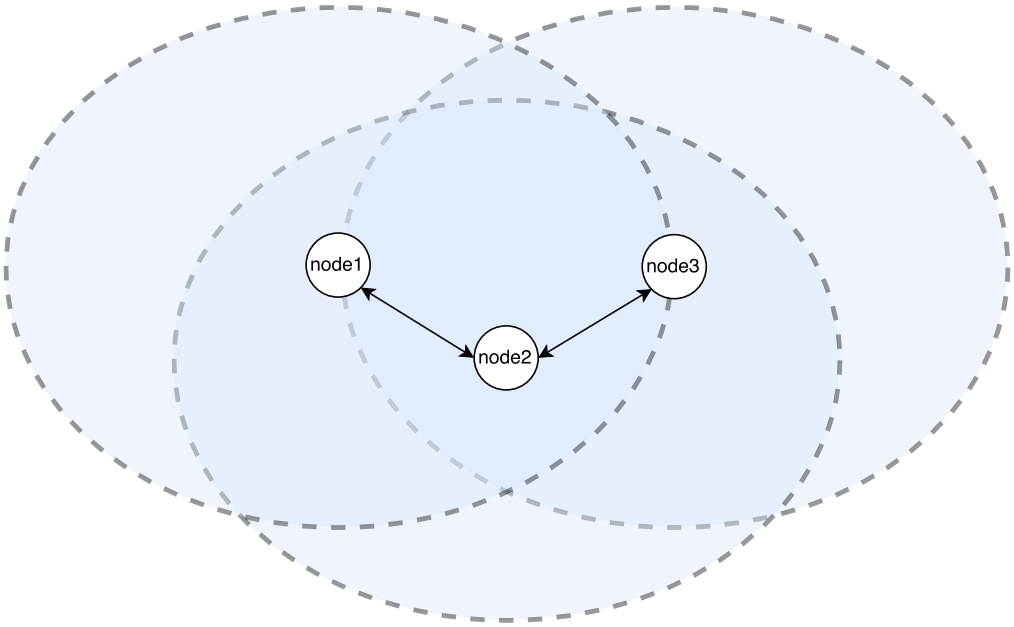


Figura 4.4: Topologia de rede real vs topologia de rede simulada

4.4 Testes e resultados

Nesta secção vamos descrever os resultados obtidos nos testes realizados para comparar os protocolos OLSRv2 e o BATMANd (*layer* 3), em termos do tempo de convergência em quatro

situações diferentes. Após a realização destes testes conseguimos perceber qual o melhor protocolo de encaminhamento para a solução que pretendemos implementar.

4.4.1 Inicialização da rede

O primeiro teste realizado tem como objetivo medir o tempo de convergência de cada protocolo de encaminhamento quando a rede é inicializada. Inicialmente os nós não tem nenhuma rede configurada e posteriormente passam a instanciar a rede MANET com o devido protocolo de encaminhamento. Os valores temporais são guardados em dois momentos, o primeiro momento é quando o *daemon* do protocolo de encaminhamento é inicializado e o segundo é quando a tabela de encaminhamento contém uma entrada para cada nó da rede.

$$Tempo\ de\ convergência = Valor\ do\ segundo\ momento - Valor\ do\ primeiro\ momento$$

Este teste foi realizado em três topologias diferentes, como se pode ver na figura 4.5, em cada topologia foram realizadas várias medições até se obter valores fiáveis. Estas medições foram realizadas no `node1`.

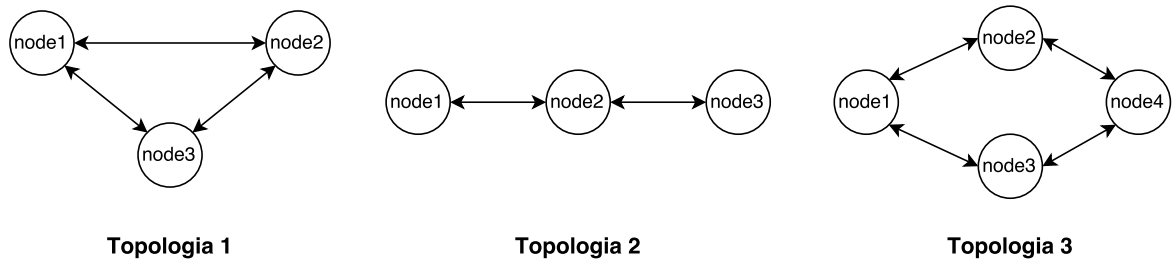


Figura 4.5: Diferentes topologias testadas

De modo a facilitar a realização deste teste, criamos um *script* usando a linguagem `Python` que está representado no algoritmo 1. Os resultados obtidos estão representados no gráfico 4.6.

4.4.2 Deteção da alteração de topologia

Neste segundo teste pretendemos verificar quanto tempo é que o protocolo de encaminhamento demora a atualizar as tabelas de encaminhamento dos nós da rede após uma alteração de topologia. A figura 4.7 representa a topologia que vamos testar, as setas tracejadas representam a movimentação dos nós e as setas com linha contínua representam as ligações válidas entre os nós. Como podemos ver na figura 4.7 o `node2` está a movimentar-se para fora da rede e o `node4` para dentro da rede. O que pretendemos verificar é o tempo que demora a atualizar a tabela de encaminhamento do `node1`. Inicialmente o `node1` tem como próximo salto para chegar até ao `node3` o `node2`, mas que depois da saída do `node2` da rede tem de atualizar essa entrada para ter como próximo salto o `node4`.

Algorithm 1 testConvergenceProtocol

```
1:  function detectConvergence()
2:      while numberRoutes ≠ (numberRoutesBeforeMANET + networkNodes) do
3:          numberRoutes ← getRoutes()
4:      end while
5:  end function
6:  function main()
7:      loop
8:          firstMomentTime ← datetime(now)
9:          startProtocol()
10:         detectConvergence()
11:         secondMomentTime ← datetime(now)
12:         convergenceTime ← secondMomentTime − firstMomentTime
13:     end loop
14: end function
```

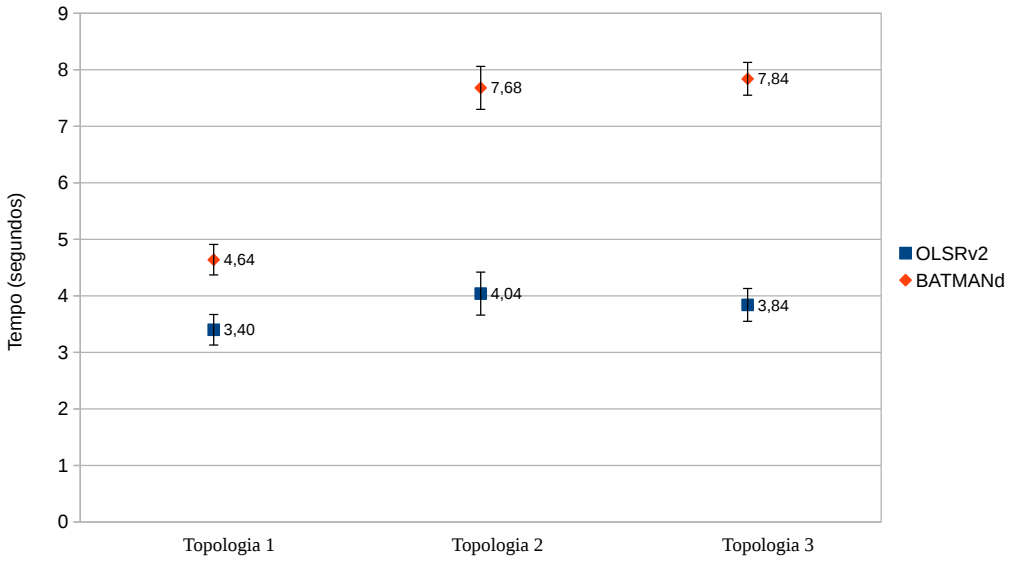


Figura 4.6: Tempo de convergência inicial dos protocolos de encaminhamento

No teste anterior apenas estávamos a registar variáveis temporais em um nó (`node1`). A sincronização do relógio desse nó com os relógios dos restantes nós não é relevante, porque não vamos comparar tempos registados em nós diferentes, apenas vamos comparar tempos registados no mesmo nó. No teste que vamos realizar nesta secção, vamos registar valores temporais em nós diferentes logo a sincronização dos relógios é essencial. Tanto o `node1`, `node2` e `node4` têm os seus relógios sincronizados usando Network Time Protocol (NTP) e estão configurados para usar os mesmos servidores NTP. A comunicação com os servidor NTP é feita através da rede de gestão e não da rede MANET. A diferença entre os relógios do `node1`, `node2` e `node4` nos testes realizados em cada protocolo de encaminhamento está representada na tabela 4.1.

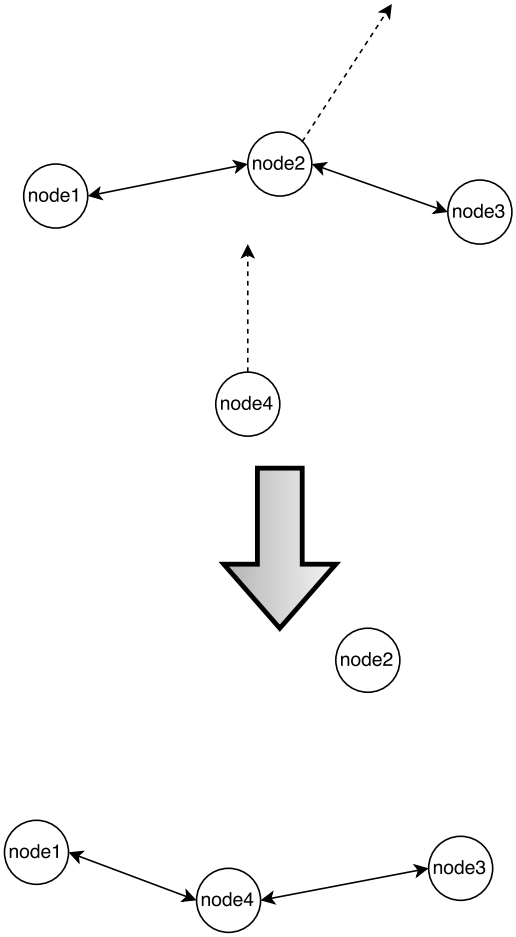


Figura 4.7: Alteração da topologia

De modo a facilitar a realização deste teste criamos um *script* usando a linguagem `Python`, que está representado no algoritmo 2. A variável `destinations` corresponde a um *array* com todos os destinos da tabela de encaminhamento. A variável `nextHops` também é um *array* e contém o próximo salto dos destino guardados na variável `destinations`. Ambos os *arrays* estão alinhados, ou seja, o próximo salto para o destino que está na posição 0 do *array* `destinations` encontra-se também na posição 0 do *array* `nextHops`. Este *script* vai ser executado no `node1` e vai registar o instante de tempo quando a tabela de encaminhamento do `node1` estiver atualizada. Como o `node2` sai da rede no mesmo instante em que o `node4` entra na rede, vamos guardar o valor temporal registado no `node4`, esse valor corresponde ao momento em que o `node4` entra na rede. O tempo de convergência é calculado da seguinte forma;

$$Tempo\ de\ convergência = Instante\ de\ tempo\ do\ node1 - Instante\ de\ tempo\ do\ node4$$

Os resultados obtidos estão representados na figura 4.8.

Tabela 4.1: Erro entre relógios em segundos

Protocolo em teste	node1 → node2	node1 → node4
OLSRv2	0,009770	0,001526
BATMANd	0,012457	0,000775

Algorithm 2 detectTopologyChange

```
1:  function MAIN()
2:      loop
3:          destinations, nextHops ← GETDESTINATIONSANDNEXTHOPS(routingTable)
4:          nextHopToNode3 ← nextHops.GETNEXTHOPTODESTINATION(ipNode3)
5:          if nextHopToNode3 = correctNextHop then
6:              updatedRoutingTableTime ← DATETIME(now)
7:              break
8:          end if
9:      end loop
10: end function
```

4.5 Funcionalidades

Nesta secção pretendemos descrever algumas funcionalidades extras dos dois protocolos de encaminhamento testados anteriormente, OLSRv2 e BATMANd (*layer* 3). A principal funcionalidade extra e a mais relevante para a solução que pretendemos desenvolver é a possibilidade de termos acesso a informações sobre a rede. Estas informações contém todas as ligações entre os nós da rede assim como as características dessas ligações e dos respetivos nós.

No OLSRv2 essa informação é disponibilizada através do *plugin* `telnet`[2] e `netjson`[3]. O *plugin* `telnet` permite executar comandos na *framework* OLSR.org Network Framework (OONF) que retornam informações sobre a rede. Essas informações podem ser, por exemplo, as características de nível 2 ou 3 dos vizinhos, `netjson graph`, `netjson route`, `netjson domain`, `netjson filter` entre outras. O `netjson` [37] é um padrão JavaScript Object Notation (JSON) para guardar várias informações da rede, uma dessas informações e a que é mais relevante para a nossa solução é o `netjson graph`. O `netjson graph` contém a topologia da rede assim como as características de cada nó e das ligações entre eles. Por exemplo, no bloco de código 4.1 temos o `netjson graph` da topologia representada na figura 4.9.

O BATMAN também disponibiliza essa informação através de um `Vis Server` [34]. Cada nó BATMAN apenas conhece o seu melhor vizinho mas não sabe nada sobre o que está para além desse vizinho. Por exemplo, o nó **A** sabe que existe o nó **C** na rede e que para chegar até ele tem de usar o seu vizinho nó **B**, mas não sabe quantos *hops* existem entre ele e o nó **C**. O `Vis Server` guarda informações locais sobre os nós da rede para depois disponibilizar essa

[2]http://www.olsr.org/mediawiki/index.php/OONF_Telnet_Plugin
[3]http://www.olsr.org/mediawiki/index.php/NetJson_Info_Plugin

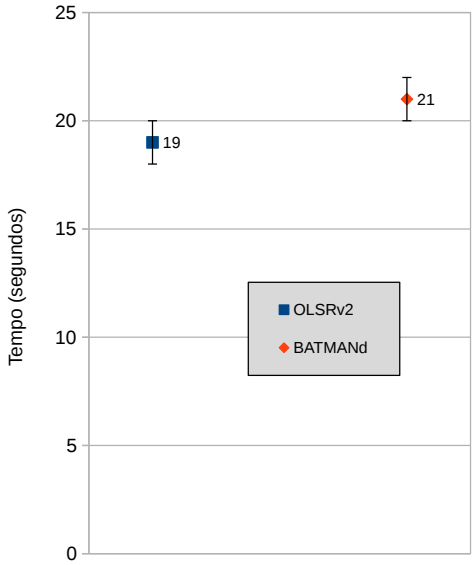


Figura 4.8: Tempo de convergência do teste de alteração de topologia

```
{
    "nodes": [
        { "id": "172.16.40.24", "label": "node A"},
        { "id": "172.16.40.60", "label": "node B"},
        { "id": "172.16.41.1", "label": "node C"}],
    "links": [
        {"source": "172.16.40.24","target": "172.16.40.60","cost": 1.000},
        {"source": "172.16.40.24","target": "172.16.41.1","cost": 1.000},
        {"source": "172.16.40.60","target": "172.16.41.1","cost": 1.000}]
}
```

Bloco de Código 4.1: Exemplo de um JSON do `netjson graph`

informação. Recorrendo a ferramentas específicas podemos fazer o tratamento dessa informação e representá-la graficamente.

O `Vis Server` pode disponibilizar a informação em vários formatos, nas versões mais recentes há a possibilidade de ser em JSON. Através do exemplo de um *output* do `Vis Server`, representado em 4.2, podemos ver que o nó `5.174.37.225` tem uma ligação com o nó `5.224.160.202` com uma qualidade de `2.13`. Além disso, ligação é bidirecional porque na linha seguinte temos a rota inversa, do nó `5.224.160.202` para o nó `5.174.37.225`. Nesse sentido a qualidade da ligação é de `1.28` e não de `2.13` como no sentido inverso. O valor que

```
"5.174.37.225" -> "5.224.160.202"[label="2.13"]
"5.224.160.202" -> "5.174.37.225"[label="1.28"]
```

Bloco de Código 4.2: Exemplo de um *output* do `Vis Server`

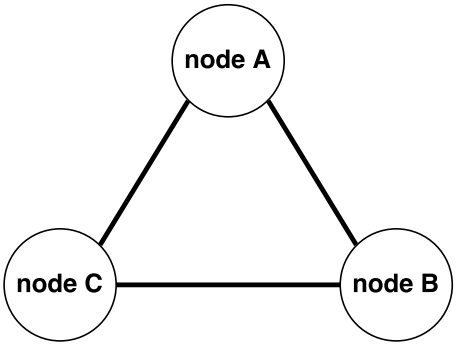


Figura 4.9: Topologia exemplo do `netjson graph`, adaptado de [37]

representa a qualidade da ligação diz respeito ao valor do campo Transmit Quality (TQ) dos pacotes Originator Menssages (OGM). O valor `1.00` corresponde a 100%, `2.00` a 50%, `3.00` a 33% e `4.00` corresponde a 25%. Como já foi referido, podemos usar a informação que é retornada pelo `Vis Server` e representá-la graficamente, usando por exemplo a ferramenta `mesh3d`[4].

4.6 Sumário

Neste capítulo foram realizados alguns teste para medir o desempenho de dois protocolos de encaminhamento. Após a realização desses testes concluímos que o OLSRv2 obteve melhores resultados em todos os testes. Em termos das funcionalidades extra, ambos os protocolos de encaminhamento fornecem as informações que necessitamos para a nossa solução. Por isso, este fator não altera a decisão tomada com base nos resultados obtidos nos testes realizados. Visto isto, optamos por escolher o OLSRv2 como protocolo de encaminhamento para a nossa solução.

[4]http://s3d.sourceforge.net/

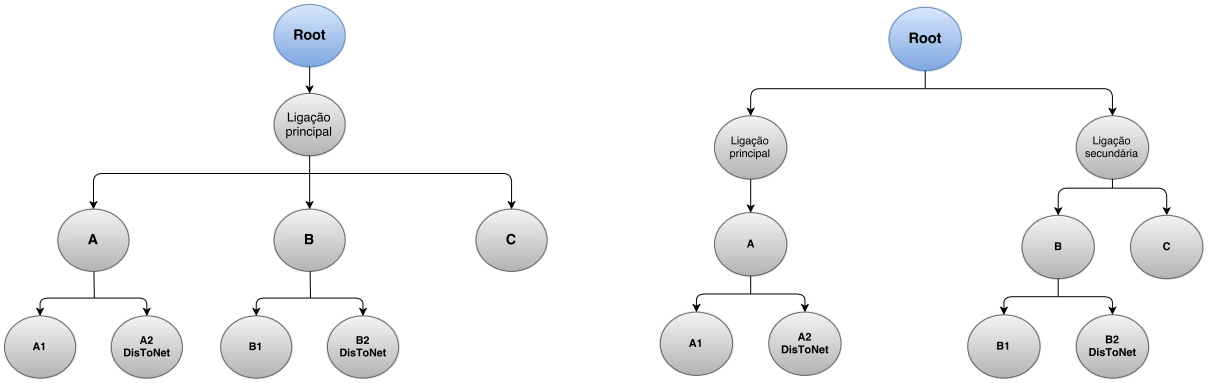
Capítulo 5

Priorização com HTB

Este capítulo tem como objetivo descrever com mais detalhe as classes criadas em cada uma das estruturas apresentadas nas secções 3.5 e 3.6. Vamos também descrever o método usado para configurar essas estruturas ou outras que podem ser adicionadas à nossa solução.

5.1 Estruturas do HTB

Como vimos nas secções 3.5 e 3.6 temos duas estruturas diferentes, representadas em 5.1a e 5.1b, e em cada uma delas temos classes direcionadas para o tráfego que circula na rede e para o tráfego proveniente da Disconnection Tolerant Network (DisToNet). Apesar das classes estarem em domínios diferentes os requisitos de qualidade de serviço são iguais. Tanto a classe **A1** como a **A2** tem o mesmo requisito de qualidade de serviço, mas em domínios diferentes. As taxas de transmissão dessas classes (**A1** e **A2**) são diferentes, apesar de terem o mesmo requisito de qualidade de serviço, porque estão a servir domínios diferentes.



(a) Estrutura da secção 3.5

(b) Estrutura da secção 3.6

Figura 5.1: Estruturas do HTB da nossa solução

O requisito que cada classe pretende servir é:

- **Classe A (atraso):** tráfego com restrições ao nível do atraso.
- **Classe B (largura de banda):** tráfego com necessidade de largura de banda.
- **Classe C (outros):** tráfego que não obedeça a nenhuma das restrições aplicadas nas classes anteriores.

As características das classes da estrutura 5.1a estão representadas na tabela 5.1. Os valores das taxas de transmissão estão representados com a percentagem de utilização da largura de banda disponível. Na coluna **Prio** temos a prioridade de cada classe, onde o valor inferior representa maior prioridade. Por último a coluna Type of Service (ToS) representa os valores que os pacotes devem conter no campo ToS do cabeçalho Internet Protocol (IP) para que sejam enviados para a respetiva classe.

Tabela 5.1: Parâmetros do HTB da estrutura 3.4

Classe	Taxa mínima	Taxa máxima	Prio	ToS	Objetivo
A	30%	90%	1	-	Atraso
A1 (Dados)	20%	90%	1	0x00	-
A2 (DisToNet)	10%	90%	1	0x02	-
B	50%	100%	2	-	Largura de banda
B1 (Dados)	30%	100%	2	0x04	-
B2 (DisToNet)	20%	100%	2	0x08	-
C	20%	100%	3	0x10	Outros

Na tabela 5.2 temos as características das classes da estrutura 5.1b. Como temos duas ligações podemos ser mais "generosos" na largura de banda que vamos atribuir a cada classe. Neste caso, a classe **A** tem disponível a capacidade total da ligação principal para o seu tráfego, a classe **B** e **C** tem de partilhar entre ambas a capacidade da ligação secundária. Como podemos ver na tabela 5.2, a taxa mínima e máxima da classe **A** está definida com 100%. Visto isto, neste caso a classe **A** não serve apenas o tráfego com requisito de atraso, mas sim de atraso e largura de banda. Isto também é verdade para a classe **B**.

5.2 Configuração do ficheiro XML

A solução que pretendemos desenvolver vai adaptar-se a qualquer estrutura do HTB que queiramos testar. Para que essa adaptação seja feita vamos uniformizar as *tags* de um ficheiro eXtensible Markup Language (XML) que depois será carregado. Deste modo, conseguimos definir estruturas do HTB diferentes consoante os requisitos da aplicação que está a correr sobre a rede Mobile Ad Hoc Network (MANET). Cada estrutura necessita de dois ficheiros XML, um para as características de cada classe e o outro com as características dos filtros, que serão criadas para encaminhar o tráfego para a respetiva classe. O ficheiro XML das classes precisa de ter as

Tabela 5.2: Parâmetros do HTB da estrutura 3.5

Classe	Taxa mínima	Taxa máxima	Prio	ToS	Objetivo
A	100%	100%	1	-	Atraso e largura de banda
A1 (Dados)	80%	100%	1	0x00	-
A2 (DisToNet)	20%	100%	1	0x02	-
B	90%	90%	1	-	Atraso e largura de banda
B1 (Dados)	60%	90%	1	0x04	-
B2 (DisToNet)	30%	90%	1	0x08	-
C	10%	100%	2	0x10	Outros

características (taxa de transmissão mínima, máxima e prioridade) da classe principal e das classes secundárias.

No ficheiro XML das classes vamos ter as seguintes *tags*:

- **<class> :** Representa uma classe e tem o atributo `classid`.
- **<subclass> :** Representa uma subclasse e tem o atributo `classid`.
- **<parent> :** Nome da chave ou o valor do `ID` da classe que está acima na hierarquia.
- **<rate> :** Percentagem da taxa de transmissão mínima.
- **<ceil> :** Percentagem da taxa de transmissão máxima.
- **<prio> :** Valor da prioridade da classe.

No ficheiro XML dos filtros vamos ter as seguintes *tags*:

- **<filter> :** Representa um filtro e tem o atributo *title*.
- **<rootID> :** Nome da chave que contém o `root ID`.
- **<prio> :** Valor da prioridade do filtro.
- **<tos> :** Valor do ToS a filtrar.
- **<flowID> :** `ID` da classe para onde será enviado o tráfego.

Desta forma, ao utilizarmos outro ficheiro XML que respeite as *tags* apresentadas anteriormente, a nossa solução vai priorizar com base nessa nova estrutura. Deste modo, a nossa solução é flexível em termos dos requisitos de qualidade de serviço. Por exemplo, se a aplicação que está a correr sobre a rede MANET necessitar de outras classes, apenas precisa de alterar o conteúdo do ficheiro XML com essa informação. O bloco de código 5.1 representa o ficheiro XML das classes do vizinho principal numa situação de balanceamento de tráfego, as características dessas classes estão representadas na tabela 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
    <!-- Classe A -->
    <class classid="1:10">
        <parent>mainClass_id</parent>
        <rate>1</rate>
        <ceil>1</ceil>
        <prio>1</prio>
        <!-- Classe A1 -->
        <subclass classid="1:11">
            <parent>1:10</parent>
            <rate>0.8</rate>
            <ceil>1</ceil>
            <prio>1</prio>
        </subclass>
        <!-- Classe A2 (DisToNet) -->
        <subclass classid="1:12">
            <parent>1:10</parent>
            <rate>0.2</rate>
            <ceil>1</ceil>
            <prio>1</prio>
        </subclass>
    </class>
</classes>
```

Bloco de Código 5.1: Exemplo do XML das classes do vizinho principal da estrutura 5.1b

5.3 Sumário

Neste capítulo apresentamos os valores definidos nos parâmetros das classes do HTB, para se perceber como se garantem certos requisitos de qualidade de serviço manipulando esses parâmetros. Por último, apresentamos uma forma de tornar a nossa solução flexível a qualquer estrutura do HTB. Com as *tags* do XML uniformizadas para a nossa solução, conseguimos testar qualquer estrutura que cumpra essas regras. Desta forma, a nossa solução pode ser usada em outras aplicações, que necessitam de classes diferentes das que foram propostas.

Capítulo 6

Desenvolvimento

A solução desenvolvida no decorrer deste projeto foi o algoritmo PrioLoadBalancing (PLB). Para o desenvolvimento deste algoritmo optou-se pela linguagem de programação `Python` [16], porque é a linguagem de programação onde temos mais experiência e tem uma vasta comunidade para suporte. Na figura 6.1 estão representados todos os componentes do algoritmo PLB. Os componentes a azul representam os módulos do algoritmo PLB, cada um deles executa uma função específica. Essas funções podem ser, por exemplo, comunicar com a *framework* OLSR.org Network Framework (OONF), medir a capacidade das ligações, criar/remover as classes do Hierarchical Token Bucket (HTB), criar/remover regras Internet Protocol (IP) entre outras. As funções e características de cada um desses módulos serão apresentadas nas secções seguintes.

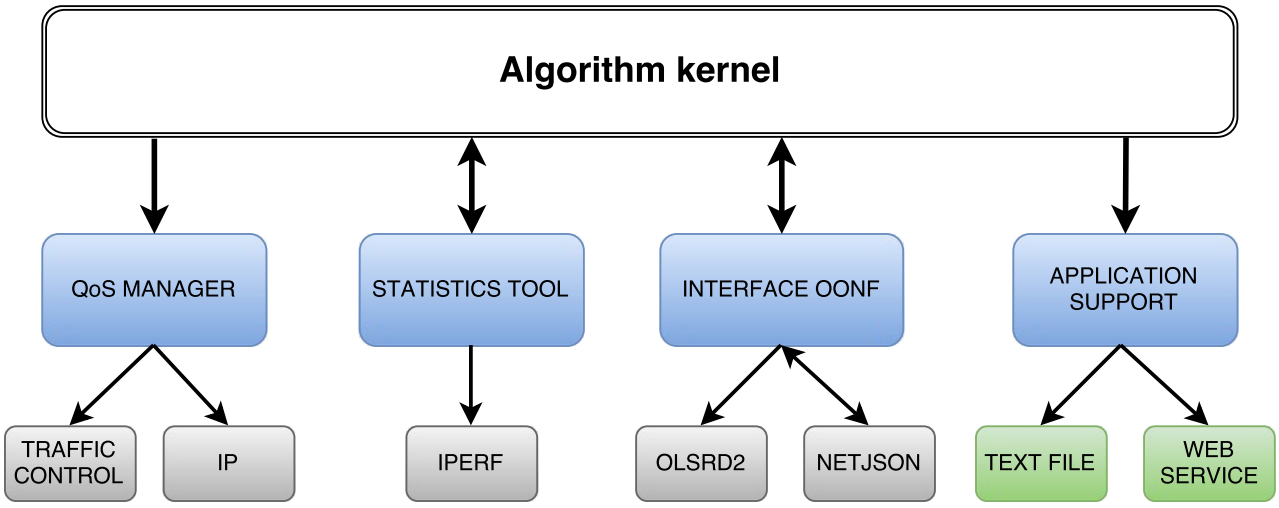


Figura 6.1: Componentes do algoritmo PLB

6.1 Descrição do algoritmo

O algoritmo PLB tem quatro estados e cada um deles usa os módulos de maneiras diferentes. Esses estados podem ser designados por: "zero vizinhos", "um vizinho", "dois vizinhos" ou "os

mesmos vizinhos". A figura 6.2 representa o fluxo do algoritmo PLB sobre os vários estados.

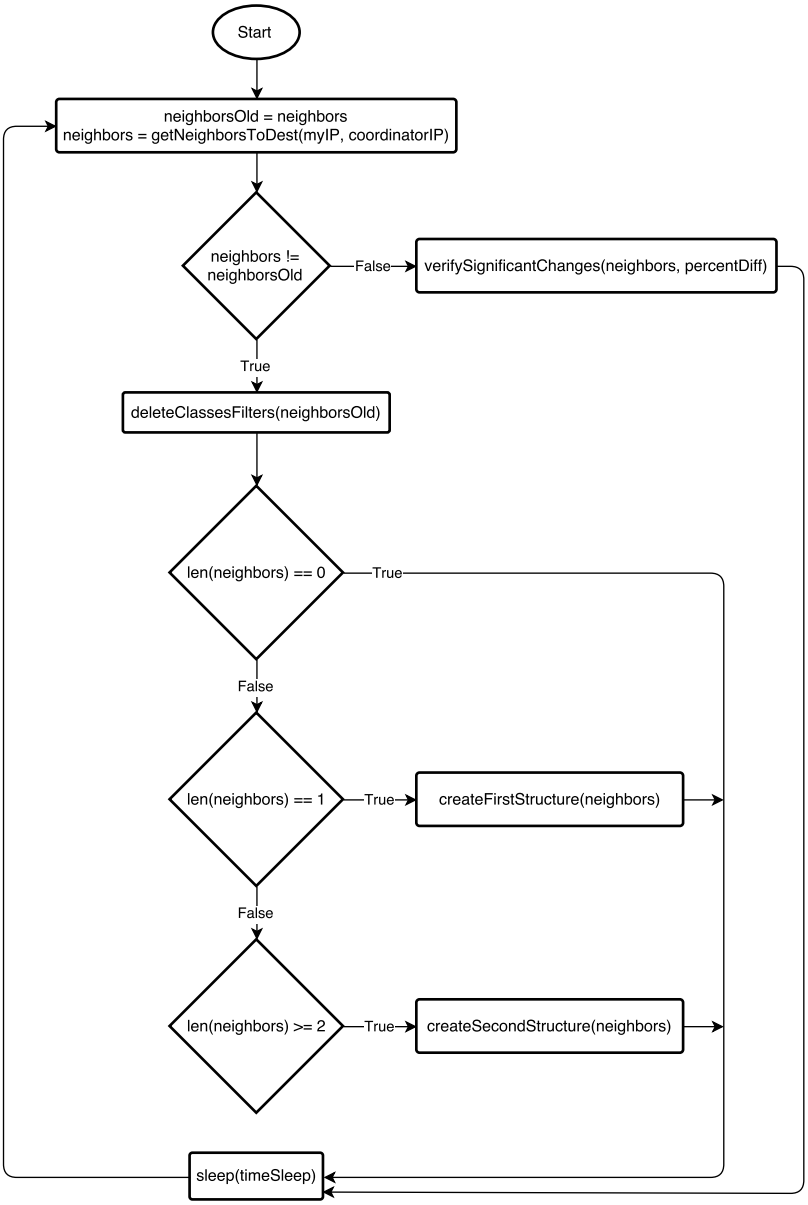


Figura 6.2: Diagrama de fluxo do algoritmo

Na figura 6.1 os componentes a azul representam os módulos do algoritmo PLB que foram desenvolvidos no decorrer deste projeto. Alguns desses módulos usam ferramentas ou tecnologias que não foram desenvolvidas neste projeto, representadas a cor cinza na figura 6.1. Por último, os componentes de cor verde, representam os ficheiros ou serviços criados pelo algoritmo PLB para dar suporte ao nível da aplicação.

Criámos variáveis globais para facilitar a configuração de certos parâmetros do algoritmo PLB, essas variáveis são as seguintes:

- `interface`: *interface* de rede na qual está configurada a rede Mobile Ad Hoc Network (MANET).

- `tableLoadBalancing`: nome da segunda tabela de encaminhamento usada numa situação de balanceamento de tráfego.
- `percentDiff`: diferença percentual máxima quando se verifica alterações na capacidade da ligação.
- `timeSleep`: tempo de espera entre cada iteração do algoritmo.

Estado zero vizinhos

No estado "zero vizinhos" o algoritmo apenas se limita a analisar se já tem vizinhos válidos. E entende-se como vizinho válido um vizinho com uma rota para o nó coordenador. O tempo entre cada verificação é definido na variável `timeSleep`, que por omissão são 2 segundos.

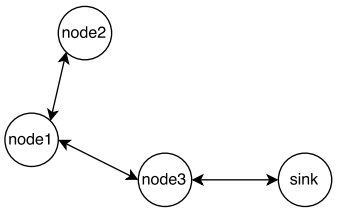


Figura 6.3: Vizinho válido e vizinho não válido

Como podemos ver na figura 6.3, o `node1` tem como vizinho válido apenas o `node3` porque é o único vizinho com uma rota para o coordenador (*sink*). O `node2` é vizinho do `node1` mas não pode ser considerado como um vizinho válido, porque não tem uma rota para o coordenador (*sink*).

Estado um vizinho

No estado "um vizinho" o algoritmo primeiro necessita de saber qual é a capacidade da ligação entre o nó onde está a ser executado e o seu vizinho válido. Para medir essa capacidade usa o módulo desenvolvido `statistics tool` que por sua vez usa a ferramenta `IPERF`. Quando o nó inicializa o algoritmo são inicializados alguns serviços de suporte ao seu funcionamento. Um desses serviços é o servidor `IPERF` para que cada nó consiga receber pedidos dos clientes `IPERF` dos outros nós da rede. Para uniformizar a porta onde o servidor `IPERF` está à escuta vamos usar o endereço IP de cada nó, a porta do servidor `IPERF` será a 100α onde o α corresponde ao valor do último octeto do endereço IP. A necessidade de saber a capacidade da ligação deve-se ao facto de ser necessário este valor para depois subdividi-lo pelas classes do HTB. Essas classes vão ser criadas pelo módulo `qos manager`. As suas características estão definidas nos ficheiros eXtensible Markup Language (XML), que no estado "um vizinho" é o ficheiro `oneNeighbor_classes.xml`. A configuração deste ficheiro XML foi apresentada na secção 5.2. Depois das classes estarem criadas é necessário criar os filtros, usando novamente o módulo `qos manager`. Os filtros têm como objetivo encaminhar os pacotes para a respetiva classe. Esse

encaminhamento será feito com base no Type of Service (ToS) do cabeçalho IP. As características destes filtros estão definidas no ficheiro XML `oneNeighbor_filters.xml`.

Os ficheiros XML são modificados (se necessário) pelo utilizador e têm de respeitar as *tags* apresentadas na secção 5.2. Nenhum módulo é responsável por fazer as alterações nos ficheiros, visto que é o utilizador que decide quais as classes que pretende criar para a sua aplicação. Os ficheiros XML por omissão estão configurados para criar as estruturas apresentadas no capítulo 5.

Estado dois vizinhos

No estado "dois vizinhos" como também acontece no estado "um vizinho" temos de medir a capacidade da ligação (neste caso a capacidade de duas ligações) para depois criar as classes e os respetivos filtros. Neste estado os ficheiros XML que definem as características das classes e dos respetivos filtros são:

- `loadBalancing_mainNeighbor_classes.xml`
- `loadBalancing_mainNeighbor_filters.xml`
- `loadBalancing_secundaryNeighbor_classes.xml`
- `loadBalancing_secundaryNeighbor_filters.xml`

Para além disto, o algoritmo tem de realizar mais algumas configurações que tornam este estado um pouco mais complexo do que os anteriores. Como o próprio nome indica temos mais do que um vizinho, mais propriamente dois vizinhos, ou seja, se temos duas possibilidades para chegar até ao nó coordenador vamos enviar parte do tráfego pelo vizinho principal (vizinho escolhido pelo protocolo de encaminhamento) e o restante tráfego pelo vizinho secundário. Para que a troca do endereço Media Access Control (MAC) de destino de cada pacote seja feita de acordo com o vizinho para o qual queremos enviar o tráfego, necessitamos de ter duas tabelas de encaminhamento. Uma é a tabela base do `Linux`, também designada como `main`, a outra é a tabela que definimos na variável `tableLoadBalancing`, que por omissão é `algorithmPLBTable`. Esta tabela tem apenas uma rota por omissão onde o próximo salto é o endereço IP do vizinho secundário. Depois de termos as tabelas configuradas é necessário criar regras IP e `IPTABLES`, usando o módulo `qos manager`. Caso estas regras não sejam criadas todo o tráfego vai usar a tabela `main`, e queremos que o tráfego que vai ser enviado para o vizinho secundário use a tabela `algorithmPLBTable` e não a tabela `main`. As regras `IPTABLES` servem somente para o tráfego que é gerado localmente, porque as regras IP baseadas no valor do ToS não afetam estes pacotes. Para que os pacotes gerados localmente usem a respetiva tabela de encaminhamento temos que os marcar com um `FWMARK` baseado no valor do ToS usando regras `IPTABLES` e depois criar regras IP baseadas no `FWMARK`. Depois de termos as regras IP e `IPTABLES` criadas tanto o tráfego que é gerado localmente como o tráfego que

está a ser encaminhado, vai usar a tabela de encaminhamento correspondente ao seu valor de ToS. A figura 6.4, representa todos os passos realizados para que cada tipo de tráfego use as respetivas tabelas de encaminhamento.

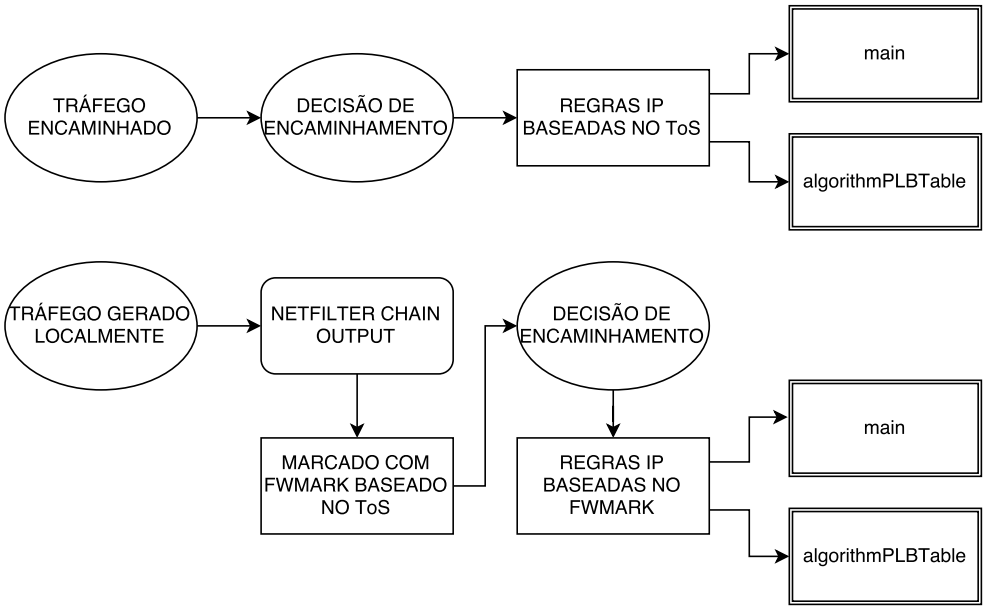


Figura 6.4: Processo de escolha da tabela de encaminhamento do tráfego gerado localmente e do tráfego encaminhado

Estado os mesmos vizinhos

No estado "os mesmos vizinhos" o algoritmo limita-se a analisar se há alterações significativas na capacidade das ligações. Considera-se uma alteração significativa se a diferença entre o valor da capacidade da ligação no instante t e $t-1$ for maior que o valor percentual definido na variável `percentDiff`. Caso hajam alterações significativas as classes do HTB são novamente criadas com o valor da capacidade da ligação atualizado.

6.2 QoS Manager

Este módulo é responsável por criar e remover as classes do HTB usando a configuração estipulada nos ficheiros XML referidos na secção anterior. As taxas de transmissão mínima e máxima de cada classe são calculadas através do valor percentual lido do ficheiro XML e do valor da capacidade da ligação lido através do módulo `statistics tool`. Por exemplo, se no ficheiro XML estiver o valor `0.2` (20%) para a taxa mínima e `0.5` (50%) para a taxa máxima da classe, e a capacidade da ligação for de 10 Mbits/s, os valores são calculados da seguinte forma:

- **Taxa mínima** $= 10 \times 0.2 = 2$ Mbits/s
- **Taxa máxima** $= 10 \times 0.5 = 5$ Mbits/s

Este módulo também é responsável por criar e remover as regras IP, IPTABLES e fazer a gestão da segunda tabela de encaminhamento usada numa situação de balanceamento de tráfego. A gestão da segunda tabela de encaminhamento, consiste na atualização do endereço IP do próximo salto da rota por omissão. Sempre que o algoritmo PLB está no estado "dois vizinhos", este módulo verifica qual é o endereço IP do vizinho secundário e atualiza essa informação na segunda tabela de encaminhamento.

Para que no nível da aplicação se tenha noção das características da classes disponíveis, criamos duas formas de aceder a essa informação. Essa informação pode ser acedida através da leitura de um ficheiro de texto, designado por `auxApplication.txt`, ou através de um `Web Service`. O módulo responsável pela atualização destes dois mecanismos, é o módulo `application support`. Sempre que são criadas novas classes do HTB o módulo `qos manager` é responsável por passar essa informação ao módulo `application support`, que depois atualiza o ficheiro `auxApplication.txt` e o `Web Service`.

6.3 Statistics tool

Este módulo como já foi referido anteriormente, trabalha com a ferramenta IPERF e tem como função medir a capacidade das ligações. Para fazer essas medições usa dois *scripts*, o `statistics-iperf-long.sh` e o `statistics-iperf-short.sh`. O primeiro faz uma medição da capacidade da ligação durante três segundos enquanto que o segundo faz uma medição de apenas um segundo. Criamos dois *scripts* para termos um balanceamento em termos de precisão e ocupação do canal. O primeiro ocupa mais tempo o canal, porque faz a mediação durante três segundos, mas o resultado final é mais preciso, enquanto que o segundo ocupa menos tempo o canal mas é menos preciso. Ambos os *scripts* o que fazem é iniciar um cliente IPERF que vai comunicar com o servidor IPERF do vizinho válido. O servidor IPERF é inicializado pelo algoritmo PLB, como foi explicado anteriormente. No estado "os mesmos vizinhos" este módulo também é usado para verificar se há alterações significativas na capacidade das ligações dos vizinhos atuais. Para fazer essa verificação, volta a fazer uma leitura da capacidade da ligação e verifica a diferença entre esse valor e o valor lido na iteração anterior.

Futuramente pretende-se substituir esta ferramenta por outra ferramenta que seja menos intrusiva, o IPERF inunda a rede com os seus pacotes para conseguir medir a capacidade das ligações. O que se pretende fazer é usar uma ferramenta que faça essa leitura em modo passivo, ou seja, essa leitura é feita sem inundar a rede com os seus pacotes. A capacidade das ligações é deduzida através do tráfego que circula na rede.

6.4 Interface OONF

Este módulo é responsável pela comunicação com o OONF. Esta *framework* permite executar o *daemon* do protocolo Optimized Link State Routing (OLSR)v2 e obter informações sobre a rede

através dos *plugins* que estão incorporados na *framework*[1]. Os *plugins* usados por este módulo são o `telnet`[2] e o `netjson`[3]. O *plugin* `telnet` permite executar comandos que retornam informações sobre a rede que o *daemon* OLSR interrogado conhece. O `netjson` [37] é um padrão JavaScript Object Notation (JSON) para guardar várias informações da rede, uma dessas informações, e a que é usada por este módulo, é o `netjson graph`. O `netjson graph` contém a topologia da rede assim como as características de cada nó e das ligações entre eles. Este módulo depois de obter o `netjson graph` usa essa informação para criar um grafo usando a biblioteca `networkx` [39]. A biblioteca `networkx` fornece vários algoritmos que podem ser usados para retirar informações do grafo. Na nossa implementação usamos o algoritmo *Simple Paths* que devolve todos os caminhos possíveis entre uma origem e destino sem nós repetidos.

6.5 Support Application

Este módulo é responsável por atualizar o ficheiro `auxApplication.txt` e o `Web Service` através da informação recebida pelo módulo `qos manager`. Tanto o ficheiro `auxApplication.txt` como o `Web Service` servem para que a camada de aplicação saiba quais foram as classes criadas e a suas características. O ficheiro `txt` permite um acesso à informação de uma forma mais rápida e direta caso a aplicação esteja a correr localmente. O `Web Service` permite o acesso a essa informação quando a aplicação não está a ser executada localmente.

Para facilitar o uso das classes do HTB ao nível da aplicação, criamos duas classes em `JAVA`, que designamos por `VRSocket` e `VRDatagramSocket`. Estas duas classes estabelecem um `socket` ou `datagramSocket` com o valor do ToS escolhido. No bloco de código 6.1 temos um exemplo do uso dessas duas classes. Optamos pela linguagem `JAVA` por atualmente ser a linguagem de programação mais usada.

Caso a aplicação seja desenvolvida numa outra linguagem de programação tem de usar o ficheiro `auxApplication.txt` ou o `Web Service` para obter as informações sobre as classes criadas. Se tivermos três classes criadas o ficheiro `auxApplication.txt` fica estruturado da seguinte forma:

<classeA>;<taxaMin>;<taxaMax>;<enviarPara>;<valorTOS>
<classeB>;<taxaMin>;<taxaMax>;<enviarPara>;<valorTOS>
<classeC>;<taxaMin>;<taxaMax>;<enviarPara>;<valorTOS>

Para obter as informações das classes através do `Web Service`, é necessário fazer um pedido `HTTP GET` ao endereço `http://ip_node/classes`. A resposta a esse pedido é um JSON com as características das classes criadas. Após obter as informações através do ficheiro

[1] http://www.olsr.org/mediawiki/index.php/OONF_Plugins
[2] http://www.olsr.org/mediawiki/index.php/OONF_Telnet_Plugin
[3] http://www.olsr.org/mediawiki/index.php/NetJson_Info_Plugin

```
public static void main(String[] args) {
    //Get all information about classes
    HTBProperties[] htbPropertiesList = HTBUtil.getHTBProperties();

    //Test VitalResponderSocket (JAVA Socket)
    //htbPropertiesList[0].getTos() -> Get ToS of class 0 (first class)
    VitalResponderSocket vrSocket = new VitalResponderSocket(port, server,
        htbPropertiesList[0].getTos());
    OutputStream outputStream = vrSocket.getOutputStream();
    outputStream.write(data);
    vrSocket.close();

    //Test VitalResponderDatagramSocket (JAVA DatagramSocket)
    //htbPropertiesList[0].getTos() -> Get ToS of class 0 (first class)
    VitalResponderDatagramSocket vrDatagramSocket = new
        VitalResponderDatagramSocket(port, server,
        htbPropertiesList[0].getTos());
    byte[] buf = new byte[256];
    buf = data.getBytes();
    DatagramPacket packet = new DatagramPacket(buf, buf.length,
        InetAddress.getByName(server), port);
    vrDatagramSocket.send(buf);
    vrDatagramSocket.close();
}
```

Bloco de Código 6.1: Exemplo do `VRSocket` e `VRDatagramSocket`

`auxApplication.txt` ou do `Web Service`, escolhe a classe que pretende usar e verifica qual é o ToS associado a essa classe e cria um canal de comunicação com o valor do respetivo ToS.

6.6 Sumário

Neste capítulo descrevemos a solução que implementámos no decorrer deste projeto. Começámos por fazer uma descrição dos vários estados presentes no algoritmo PLB e a forma como os módulos são usados em cada um desses estados. Ao desenvolvermos a nossa solução por módulos, possibilita que no futuro se possa substituir esses módulos sem necessidade de reescrever todo o algoritmo.

Capítulo 7

Testes e Resultados

Este capítulo tem como objetivo testar o algoritmo PrioLoadBalancing (PLB) que foi desenvolvido no decorrer deste projeto. Com estes testes pretendemos verificar se as funcionalidade implementadas estão a operar corretamente. Desta forma, conseguimos perceber se os objetivos planeados inicialmente foram cumpridos. A rede usada para realizar estes testes é a mesma que foi apresentada em 4.2.

7.1 Atualização do estado

Neste teste o que se pretende medir é o tempo que o algoritmo demora a atualizar o seu estado após o protocolo de encaminhamento ter atualizado a tabela de encaminhamento do respetivo nó. Para realizar este teste vamos passar de uma topologia de dois vizinhos para uma de um vizinho, como ilustrado na figura 7.1. O nó onde está a ser executado o algoritmo (`node4`) tem dois vizinhos válidos (`node1` e `node2`) para chegar até ao coordenador (`node3`). Após alguns segundos o `node4` passa a ter apenas um vizinho (`node1`).

Para sabermos o instante de tempo em que o protocolo atualizou a tabela de encaminhamento vamos executar um *script* desenvolvido em `Python`, que vai estar constantemente a comparar o *output* do comando `ip route` no instante t e $t-1$. Este *script* vai ser executado no `node4`, e quando detetar alguma diferença guarda o instante de tempo. Como vimos na secção 4.4.2 o tempo de atualização do protocolo é de aproximadamente 19 segundos, logo o algoritmo PLB irá demorar no mínimo 19 segundos para atualizar o seu estado. O que pretendemos saber com este teste é quanto tempo a mais para além dos 19 segundos o algoritmo demora a atualizar o seu estado. Quando o algoritmo PLB trocar de estado vamos guardar esse instante de tempo, para depois compararmos com o valor obtido no protocolo de encaminhamento.

Depois de sabermos os dois instantes temporais conseguimos saber quanto tempo a mais é que foi necessário para o algoritmo atualizar o seu estado, representado por α, calculando:

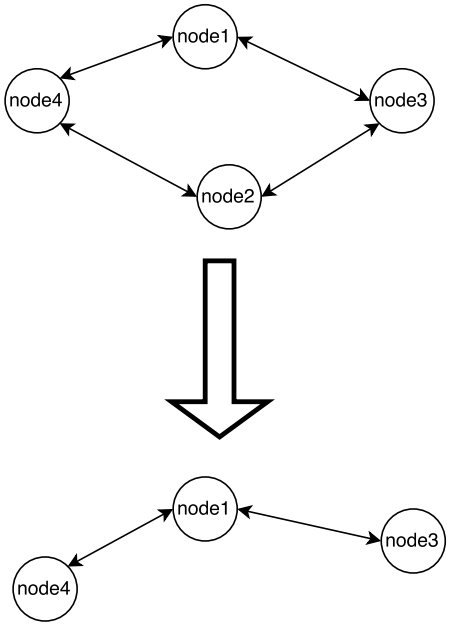


Figura 7.1: Teste troca de estados

$$\alpha = Tempo\ de\ mudança\ do\ algoritmo - Tempo\ de\ mudança\ do\ protocolo$$

Como foi referido anteriormente o tempo de convergência do protocolo quando o nó sai da rede é de aproximadamente 19 segundos que para certo tipo de aplicações pode ser muito tempo. A *framework* OLSR.org Network Framework (OONF) permite alterar este valor através da variável `hello_validity` do *plugin* Neighborhood Discovery Protocol (NHDP)[1]. O valor por omissão desta variável é de 20 segundos e define o tempo em que uma mensagem `HELLO` é válida para os vizinhos. O tempo de convergência está associado às mensagens `HELLO`, porque estas mensagens são usadas pelos nós da rede para informarem os seus vizinhos que ainda estão ativos. Nos testes que vamos realizar o valor da variável `hello_validity` fica com o valor por omissão.

Como se pode ver no gráfico 7.2, o algoritmo PLB deteta antes do tempo de convergência do protocolo de encaminhamento que o nó que saiu da rede não é válido. Podemos verificar isso no ponto do gráfico, designado por `iperf`. Como o último pacote `HELLO` foi enviado à menos de 20 segundos o protocolo de encaminhamento ainda pensa que o nó que saiu da rede é válido. Mas quando o `node4` tentar medir a capacidade da ligação, através do módulo `statistics tool`, esta medição vai falhar. Esta falha deve-se ao fato do servidor `IPERF` do nó que saiu de rede já não estar acessível. O ponto `iperf`, no gráfico 7.2, representa o momento da primeira falha na tentativa de medição da capacidade da ligação.

Para além disso, olhando novamente para o gráfico 7.2, vemos que o algoritmo PLB demora 11 segundos para atualizar o seu estado após o protocolo de encaminhamento convergir. Isto acontece porque o cliente `IPERF` tenta comunicar várias vezes com o servidor mesmo não estando

[1] http://www.olsr.org/mediawiki/index.php/NHDP_Plugin

acessível. Antes do algoritmo PLB recolher informações atualizadas sobre a rede, fica parado nestas tentativas de comunicação por parte do cliente `IPERF`.

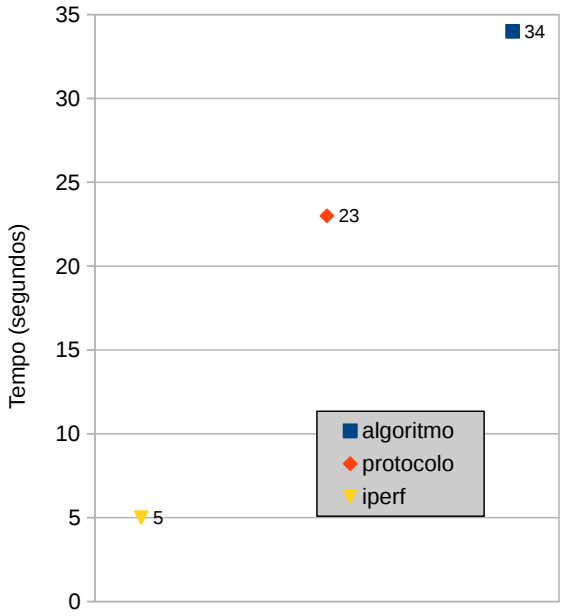


Figura 7.2: Tempo de atualização do estado do algoritmo sem `smartMode` após falha do `node2`

Introdução do modo `smartMode`

Devido ao que foi apresentado anteriormente, decidimos criar um modo de funcionamento do algoritmo PLB que designamos por `smartMode`. Este modo apenas é ativado quando: o módulo `statistics tool` falhar duas medições consecutivas e o algoritmo PLB estiver no estado "dois vizinhos". Quando estes dois fatores se verificam o algoritmo ativa o modo `smartMode`. O que este modo faz é eliminar o nó com o qual não conseguiu estabelecer a conexão `IPERF` da lista de vizinhos válidos sem esperar pela convergência do protocolo de encaminhamento. Na iteração seguinte, mesmo que o protocolo de encaminhamento diga que aquele nó é um vizinho válido o algoritmo não o considera como tal. Apenas o considera como vizinho válido se após 3 iterações do algoritmo o protocolo de encaminhamento ainda disser que é um vizinho válido. Este modo apenas é ativado no estado "dois vizinhos" porque caso o algoritmo esteja errado não perdemos a comunicação com o nó coordenador. Neste estado ao eliminarmos o nó que supostamente não é válido, temos outro vizinho válido o que não acontece no estado "um vizinho". Caso o algoritmo esteja errado no estado "dois vizinhos" apenas vamos perder a possibilidade de balancear o tráfego, mas não deixamos de enviar os dados. Enquanto que no estado "um vizinho" deixamos de enviar os dados, porque eliminámos o único vizinho válido para chegar até ao nó coordenador. Na imagem 7.3 podemos ver que no pior dos casos, que é quando o nó eliminado pelo algoritmo volta para a rede no instante seguinte, o algoritmo fica durante 6 segundos no estado "um vizinho" quando podia estar no estado "dois vizinhos". Neste caso consideramos que o tempo entre cada iteração do algoritmo é o valor definido por omissão, ou seja, 2 segundos.

Para além disso, colocamos o cliente `IPERF` com limite de tempo de execução, de modo a evitar que o algoritmo PLB fique parado nas sucessivas tentativas de comunicação com o servidor. Se o cliente `IPERF` não conseguir comunicar com o servidor dentro do limite de tempo definido, informa o algoritmo que aquele nó não está acessível. Caso a comunicação seja feita com sucesso dentro do limite de tempo, informa o algoritmo qual é a capacidade da ligação com o respetivo vizinho.

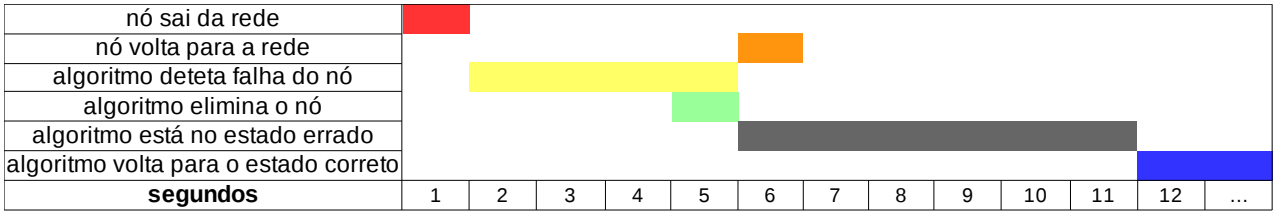


Figura 7.3: Diagrama temporal do `smartMode`

O gráfico 7.4 representa o tempo de atualização do algoritmo PLB com o modo `smartMode`.

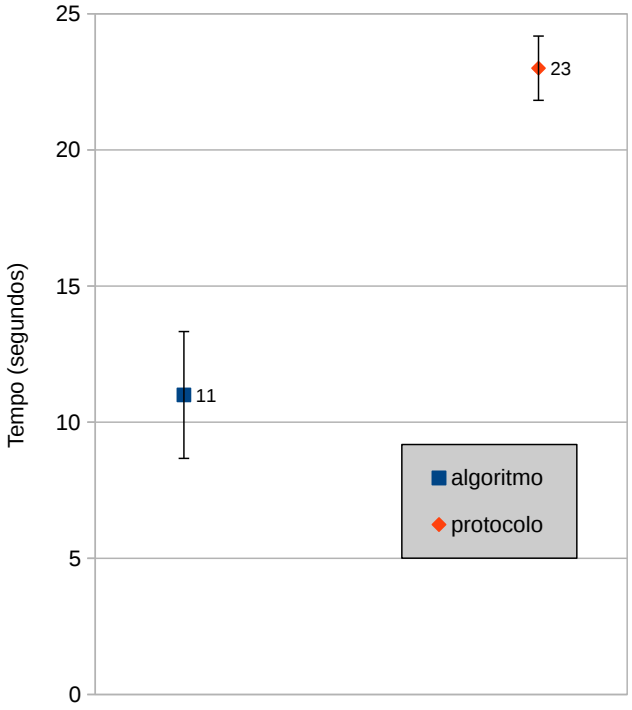


Figura 7.4: Tempo de atualização do estado do algoritmo com `smartMode` após falha do `node2`

7.2 Priorização de tráfego

Neste teste o que pretendemos verificar é se os diferentes fluxos de tráfego estão a ser enviados para as respetivas classes. Para realizar este teste vamos recorrer à ferramenta `IPERF` e através da *flag* `-tos (-S)` vamos alterar o valor do Type of Service (ToS) de cada fluxo de tráfego. A figura 7.5 representa a topologia que será usada neste teste.

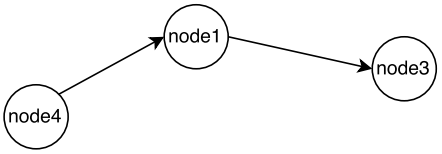


Figura 7.5: Topologia teste um vizinho

Para gerar vários fluxos de tráfego em simultâneo com valores de ToS diferentes criamos um *script* utilizando a linguagem `Python`. Neste *script* vamos usar a biblioteca `threading` para lançar uma *thread* por cada argumento passado para o *script*, esses argumentos correspondem ao valor do campo do ToS. Cada *thread* corresponde a um cliente `IPERF` que está a gerar tráfego do tipo User Datagram Protocol (UDP) com uma taxa de transmissão de 150 Mbits/s.

Este teste foi realizado no `node4` e os resultados obtidos estão representados no gráfico 7.6. A capacidade da ligação entre os dois nós (`node4` e `node1`) é de aproximadamente 15 Mbits/s, este valor foi obtido através do módulo `statistics tool`. As características de cada classe estão representadas na tabela 7.1.

Tabela 7.1: Características da classes

Classe	Taxa mínima	Taxa máxima	Prio	ToS
A1	20%	90%	1	0x00
B1	30%	100%	2	0x04
C	20%	100%	3	0x10

Os três tipos de tráfego foram gerados em simultâneo e a taxa mínima garantida para cada um deles é:

- Classe **A1** (`0x00`): $15 \times 0.2 = 3$ Mbits/s
- Classe **B1** (`0x04`): $15 \times 0.3 = 4.5$ Mbits/s
- Classe **C** (`0x10`): $15 \times 0.2 = 3$ Mbits/s

No total a taxa mínima garantida para as três classes é de 10.5 Mbits/s, ficando 4.5 Mbits/s livres para serem partilhados entre as classes. Essa largura de banda é partilhada consoante a prioridade de cada classe, como foi explicado no capítulo 5. Devido a isto, os valores das taxas de transmissão de cada classes são superiores ao valor mínimo garantido.

7.3 Balanceamento de tráfego

Neste teste o que pretendemos verificar é se o balanceamento de tráfego está a ser feito corretamente. Para verificarmos isso vamos usar a ferramenta `traceroute` recorrendo a

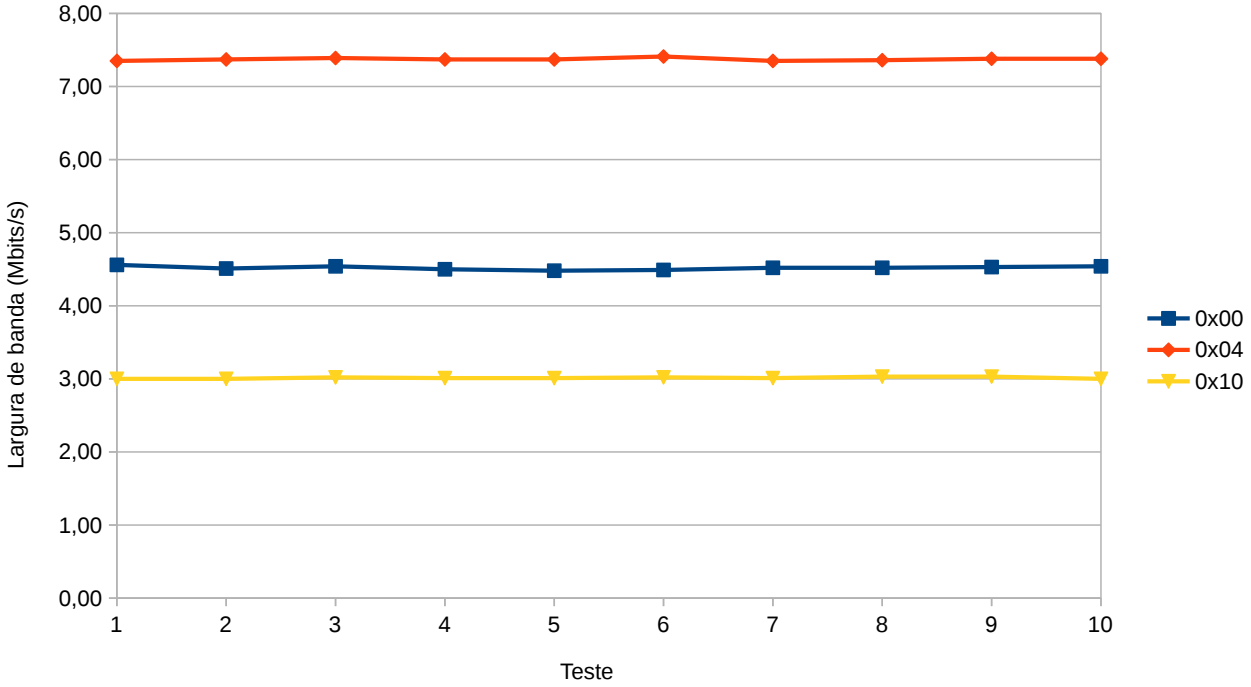


Figura 7.6: Valores de largura de banda

flag `tos (-t)` para alterar o valor do ToS dos pacotes. A figura 7.7 representa a topologia usada neste teste, o `node4` tem como vizinho principal o `node1` e como vizinho secundário o `node2`. Os endereços Internet Protocol (IP) de cada nó da rede estão representados na tabela 7.2. Os resultados obtidos através da ferramenta `traceroute` podem ser visualizados em 7.1. O tráfego da classe **A** deve ser enviado pelo vizinho principal e o restante tráfego pelo vizinho secundário. Na nossa estrutura do Hierarchical Token Bucket (HTB) o tráfego da classe **A** tem o valor `0x00` e `0x02` no campo ToS.

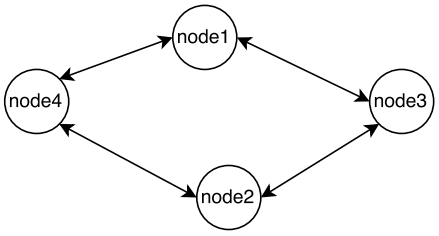


Figura 7.7: Topologia teste dois vizinhos

Após analisarmos os resultados obtidos, vemos que o balanceamento de tráfego está a funcionar corretamente. Os valores da *flag* `tos (-t)` correspondem ao valor que pretendemos definir no campo ToS em decimal. Para a classe **A** os valores em decimal são o `0` e `2`. Olhando para os resultados obtidos o tráfego com o valor de ToS `0` e `2` chega até ao nó coordenador (3.3.3.3) através do vizinho principal (1.1.1.1). O restante tráfego chega até ao nó coordenador (3.3.3.3) através do vizinho secundário (2.2.2.2).

Tabela 7.2: Endereços IP de cada nó da rede

Nó	Endereço IP
node1	1.1.1.1
node2	2.2.2.2
node3	3.3.3.3
node4	4.4.4.4

```
fabio@fabio-pc:/$ traceroute 3.3.3.3 -t 0
traceroute to 3.3.3.3 (3.3.3.3), 30 hops max, 60 byte packets
1  node1 (1.1.1.1)  1037.879 ms  1038.064 ms  1041.856 ms
2  coordinator (3.3.3.3)  1052.615 ms  1059.035 ms  1062.419 ms

fabio@fabio-pc:/$ traceroute 3.3.3.3 -t 2
traceroute to 3.3.3.3 (3.3.3.3), 30 hops max, 60 byte packets
1  node1 (1.1.1.1)  3.444 ms  5.205 ms  6.739 ms
2  coordinator (3.3.3.3)  15.162 ms  25.965 ms *

fabio@fabio-pc:/$ traceroute 3.3.3.3 -t 4
traceroute to 3.3.3.3 (3.3.3.3), 30 hops max, 60 byte packets
1  node2 (2.2.2.2)  50.372 ms  52.352 ms  52.383 ms
2  coordinator (3.3.3.3)  61.866 ms  62.378 ms  68.109 ms

fabio@fabio-pc:/$ traceroute 3.3.3.3 -t 8
traceroute to 3.3.3.3 (3.3.3.3), 30 hops max, 60 byte packets
1  node2 (2.2.2.2)  716.702 ms  717.183 ms  717.644 ms
2  coordinator (3.3.3.3)  725.252 ms  728.562 ms  733.391 ms

fabio@fabio-pc:/$ traceroute 3.3.3.3 -t 16
traceroute to 3.3.3.3 (3.3.3.3), 30 hops max, 60 byte packets
1  node2 (2.2.2.2)  1047.580 ms  1048.274 ms  1048.700 ms
2  coordinator (3.3.3.3)  1054.161 ms  1059.402 ms  1060.998 ms
```

Bloco de Código 7.1: Resultados obtidos no balanceamento de tráfego

7.4 Recursos de *hardware* usados

Neste teste pretendemos saber qual a quantidade de Central Processing Unit (CPU) e memória que é usada pelo algoritmo PLB. Vamos monitorizar estes dados recorrendo à ferramenta `sar` do pacote `sysstat`[2]. Estes recursos têm impacto direto no consumo da bateria dos dispositivos, e visto que são dispositivos móveis o gasto excessivo de bateria pode prejudicar o bom funcionamento da rede. Os nós têm de aguentar o máximo tempo possível sem recarregar a bateria, caso contrário terão de estar constantemente a ser substituídos, o que causa alguma instabilidade na rede ou até mesmo o término da mesma.

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

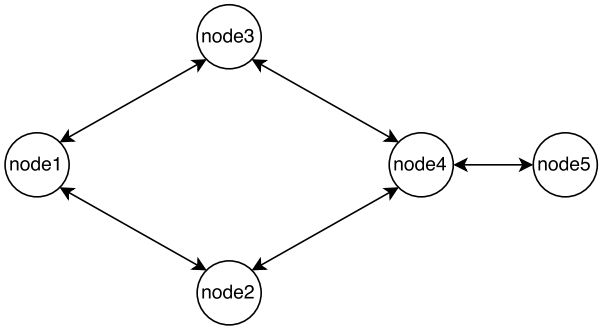


Figura 7.8: Topologia usada no teste de recursos

Para a realização deste teste vamos usar a topologia representada na figura 7.8 e vamos monitorizar os recursos de *hardware* no `node1`. Vamos simular alguma instabilidade na rede, de forma a obrigar o `node1` a estar constantemente a criar e remover as classes do HTB, filtros, regras `IP` e `IPTABLES`. Para gerar esse instabilidade, o `node2` a cada 3 minutos sai da rede e fica 2 minutos fora da rede, após esses 2 minutos volta a ligar-se à rede e fica conectado durante 3 minutos. Como se pode ver na figura 7.9.

node2 conectado à rede													
node2 desconectado da rede													
minutos	1	2	3	4	5	6	7	8	9	10	11	12	...

Figura 7.9: Instabilidade do `node2` na rede

O gráfico 7.10 corresponde aos recursos de CPU usados pelo sistema operativo sem nenhum processo relacionado com o algoritmo PLB ou com a rede Mobile Ad Hoc Network (MANET). O gráfico 7.11 corresponde aos recursos de CPU usados quando o protocolo de encaminhamento e todos os *plugins* da *framework* OONF estão a ser executados. Para finalizar o gráfico 7.12 corresponde aos recursos de CPU usados pelo algoritmo PLB e por todos os processos lançados pelo mesmo. Esses processos correspondem ao servidor e cliente `IPERF`, `tc (traffic control)`, `iproute2`, `IPTABLES`, *framework* OONF e `Web Service`. Por fim, o gráfico 7.13 correspondem ao recursos de memória usados nos três caso apresentados acima.

Em cada teste foram guardados os valores de utilização do CPU e memória a cada 1 segundo durante 15 minutos. No gráfico 7.13 os valores apresentados das três situações diferentes foram guardados em momentos temporais diferentes, mas foram representados no mesmo gráfico para facilitar a análise.

Após analisarmos os gráficos podemos concluir que em termos de utilização da memória, o aumento derivado da utilização do algoritmo PLB não é excessivo. Sendo apenas 10% superior em relação à utilização da memória, quando temos o sistema operativo a funcionar sem processos relacionados com o algoritmo PLB e a rede MANET inicializados, identificado no gráfico 7.13 por `Sistema Operativo`. Em termos de utilização de CPU o aumento é um pouco excessivo, sendo aproximadamente 80% superior em relação ao primeiro caso (sistema operativo sem processos extra).

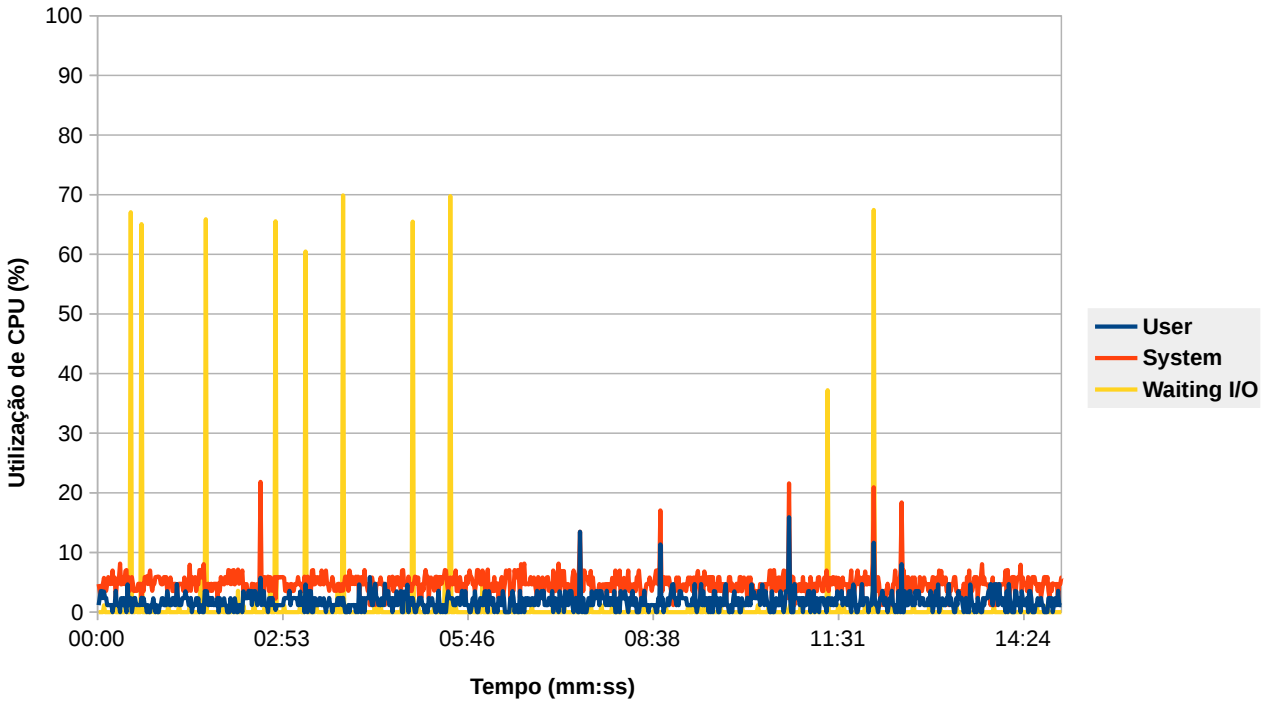


Figura 7.10: Utilização do CPU sem processos extra

7.5 Sumário

Neste capítulo realizamos alguns teste para perceber se todas as funcionalidades do algoritmo PLB estão a funcionar corretamente. Concluímos que a priorização de tráfego até ao próximo salto e o balanceamento de tráfego estão a funcionar corretamente. Além disso, conseguimos perceber que havia uma possibilidade de otimização do algoritmo introduzindo o modo `smartMode`. Detetamos ainda um problema no cliente `IPERF`, devido às suas sucessivas tentativas de comunicação com o servidor, que foi resolvido limitando temporalmente esta tarefa. Por fim, verificamos que algumas componentes do algoritmo PLB podem ser melhoradas no futuro. Esses melhoramentos serão apresentados na secção 8.2.

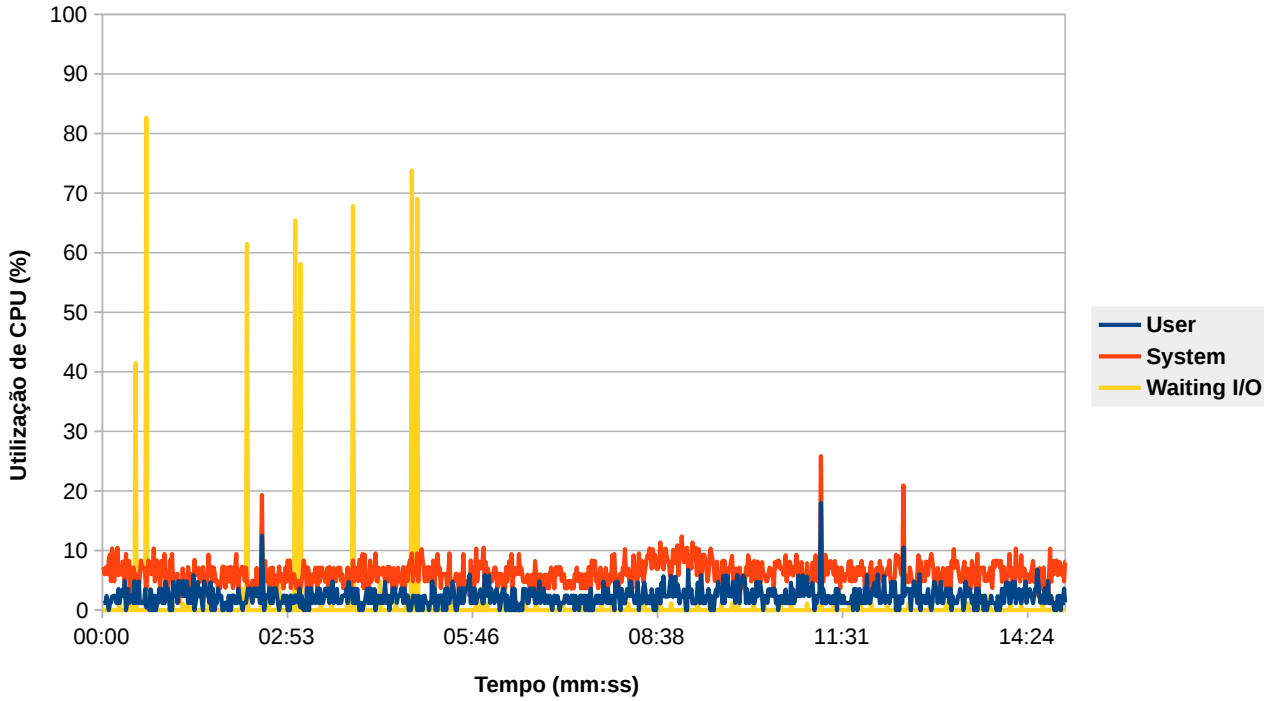


Figura 7.11: Utilização do CPU com a *framework* OONF

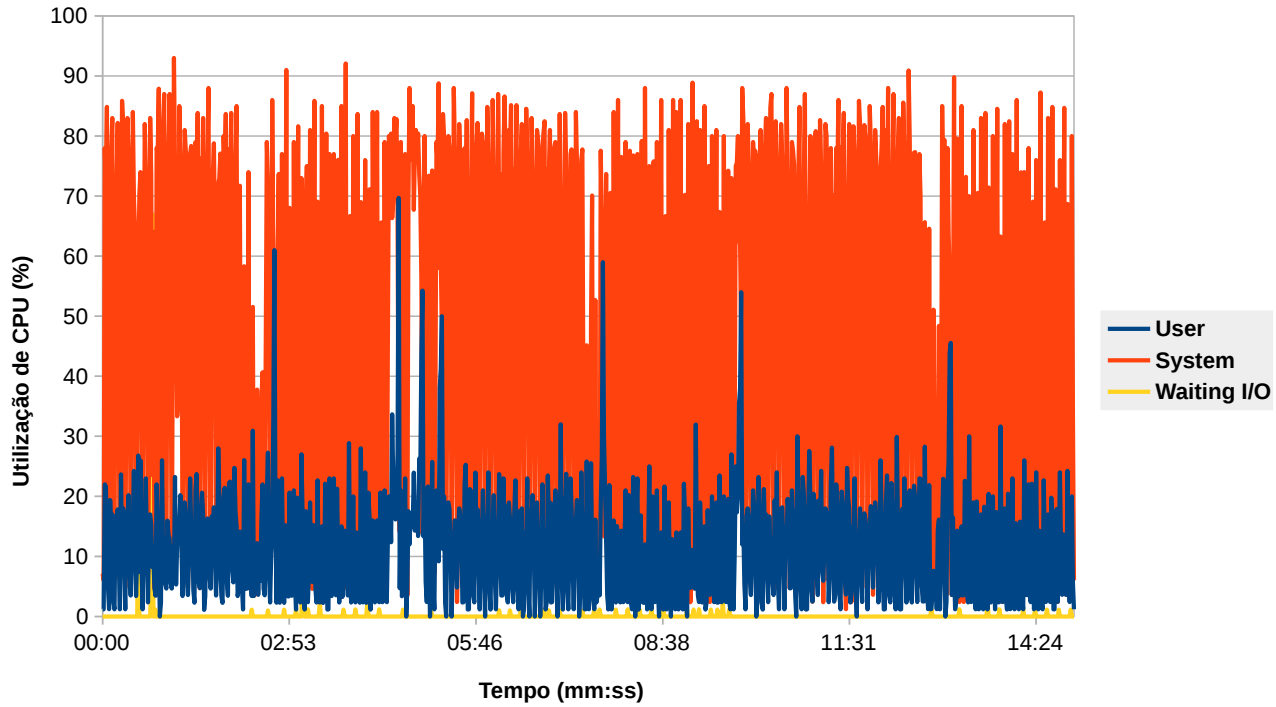


Figura 7.12: Utilização do CPU com algoritmo PLB

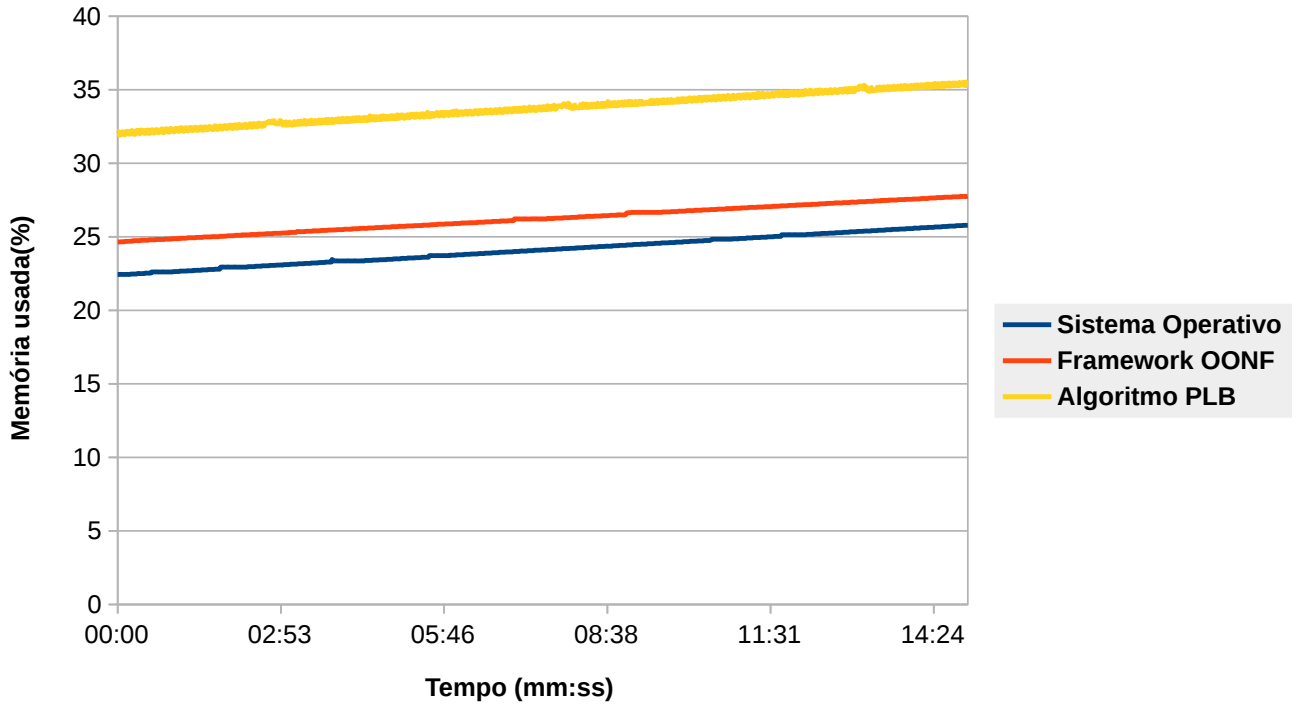


Figura 7.13: Utilização da memória

Capítulo 8

Conclusões

Neste capítulo vamos apresentar as conclusões desta dissertação assim como algumas ideias para melhorar o desempenho do nosso algoritmo.

8.1 Conclusão

Nesta dissertação o nosso objetivo foi fornecer requisitos de qualidade de serviço para o tráfego de uma rede Mobile Ad Hoc Network (MANET). Usámos a rede do projeto Vital Responder para testar a nossa solução, visto que os nós da mesma não tem em consideração a possibilidade do tráfego ser diferente, e para além disso trata-se de uma aplicação onde esta necessidade é fundamental. Este tipo de redes são muito instáveis, o que dificulta a tarefa de fornecer qualidade de serviço para as aplicações que estão a funcionar sobre as mesmas. A movimentação dos nós faz com que a topologia esteja constantemente a mudar. A qualidade das ligações também é afetada por estas movimentações, passando facilmente de ligações de boa qualidade para ligações de má qualidade e vice-versa. Para além da movimentação, normalmente estes nós têm menos poder de processamento do que os nós das redes convencionais. Tarefas que rapidamente são executadas pelos nós das redes convencionais, neste tipo de nós demoram mais tempo. Normalmente também são alimentados por baterias, o que também não ajuda na rapidez do processamento, e faz com que soluções para este tipo de redes tenham de ter em consideração os gastos energéticos. A conjugação de todos os fatores apresentados anteriormente, faz com que o nosso objetivo seja difícil de alcançar e daí advém a motivação para a realização deste trabalho.

Após a realização desta dissertação, concluímos que é possível melhorar o desempenho de dois requisitos de qualidade de serviço neste tipo de redes. Desenvolvemos uma solução, designada por algoritmo PrioLoadBalancing (PLB), que fornece classes de tráfego com taxas de transmissão diferentes para melhorar o desempenho em termos de requisitos de atraso e largura de banda em redes MANET. O melhoramento do desempenho destes dois requisitos de qualidade de serviço é feito até ao próximo salto. A nossa solução usa o algoritmo de escalonamento Hierarchical Token Bucket (HTB) para fornecer as classes com taxas de transmissão diferentes.

No projeto Vital Responder para além dos dois requisitos que são melhorados pela nossa solução, pode ser necessário garantir mais um. A fiabilidade na entrega dos dados é um requisito importante para o bom funcionamento da aplicação, mas não é possível garantir este requisito através do algoritmo de escalonamento usado na nossa solução. A necessidade deste requisito pode ser minimizada através da Disconnection Tolerant Network (DisToNet), ou através do uso de protocolos da camada de transporte apropriados para este tipo de redes.

Nos testes realizados, concluímos que as funcionalidades de priorização e balanceamento de tráfego estão a funcionar devidamente. Como se pode observar pelos resultados obtidos do teste realizado em 7.2, há uma diferenciação no tratamento dos três tipos de tráfego. Visto isto, podemos afirmar que os nós da rede já não estão a tratar de igual forma todo o tráfego da rede. Em termos do balanceamento de tráfego, através do teste realizado em 7.3 conseguimos comprovar que quando a topologia permite, o nó usa ambos os vizinhos para enviar os dados para o nó coordenador.

8.2 Trabalho Futuro

8.2.1 Módulo `statistics tool`

Este módulo usa a ferramenta `IPERF` para medir a capacidade das ligações entre os nós da rede. Futuramente pretende-se substituir esta ferramenta por outra ferramenta que seja menos intrusiva, o `IPERF` inunda a rede com os seus pacotes para conseguir medir a capacidade das ligações. O que se pretende fazer é usar uma ferramenta que faça essa leitura em modo passivo, ou seja, essa leitura é feita sem inundar a rede com os seus pacotes. A capacidade das ligações é deduzida através do tráfego que circula na rede.

Durante a realização deste trabalho testamos uma outra ferramenta, designada por Wireless Bandwidth EStimation Tool (WBest), que segundo o artigo [28] foi desenvolvida para ser rápida, não intrusiva e precisa. Os resultados obtidos nos testes realizados não foram os esperados. O WBest [29] baseia-se em dois estados, primeiro estado usa a técnica de *packet pair* para estimar a capacidade efetiva do caminho onde o último *hop* é *wireless*. No nosso caso todos os *hops* da rede são *wireless*. O segundo estado usa a técnica de *packet train* com taxa igual à capacidade efetiva para estimar o *throughput* alcançável. Para além de termos testado esta ferramenta, fizemos também um estudo sobre o comportamento em redes *wireless* de algumas ferramentas que são usadas em redes *ethernet*. Esse estudo baseou-se na análise de alguns artigos, tais como [21, 49], que testaram estas ferramentas em redes *wireless* mas os resultados obtidos não foram satisfatórios, devido ao diferente funcionamento das duas redes.

8.2.2 Recursos de *hardware*

Com o teste realizado em 7.4 observamos que há um uso excessivo do Central Processing Unit (CPU) quando o algoritmo PLB está em funcionamento. Visto que a maioria dos nós das redes MANET são alimentados através de bateria, esse uso excessivo do CPU faz com que a duração da bateria seja muito curta. Deste modo, os nós da rede MANET têm de ser constantemente substituídos causando instabilidade na rede ou até mesmo ao término da mesma. O término da rede pode acontecer porque pode não ser possível substituir os nós durante o período de ação da mesma.

De modo a melhorar este fator, primeiramente deve-se fazer uma análise do tempo que cada nó com bateria consegue-se manter ativo na rede estando a executar o algoritmo PLB. Caso esse tempo seja suficiente para o período ação da rede MANET, não é obrigatório passar para a segunda fase mas não implica que a mesma não seja feita. Essa segunda consiste numa análise para verificar qual a(s) componente(s) do algoritmo PLB que está a consumir mais recursos. Após essa análise deve-se tentar otimizar essa componente ao máximo sem alterar o objetivo principal do algoritmo PLB.

Bibliografia

[1] Wadhah Al-Mandhari, Koichi Gyoda, and Nobuo Nakajima. Ad-hoc On Demand Distance Vector (AODV) Performance Enhancement with Active Route Time-Out parameter. *WSEAS Transactions on Communications*, 7(9):912–921, 2008.

[2] H Badis and K Al Agha. Quality of service for ad hoc optimized link state routing protocol. *IETF DRAFT*, 2007.

[3] Doru Gabriel Balan and Dan Alin Potorac. Linux HTB queuing discipline implementations. In *Networked Digital Technologies, 2009. NDT'09. First International Conference on*, pages 122–126. IEEE, 2009.

[4] Suman Banik, Bibhash Roy, Parthi Dey, Nabendu Chaki, and Sugata Sanyal. QoS Routing using OLSR with Optimization for Flooding. *arXiv preprint arXiv:1108.4138*, 2011.

[5] Michal Beno. The OLSR Optimization for Large Semi-structured Wireless Ad-hoc Networks. Master's thesis, Masaryk University, 2011.

[6] Paolo Bucciol, Luca Leschiutta, Nikos Fragoulis, Frank Y Li, Giampietro Zicca, and Lorenzo Vandoni. Providing reliability and Qos in multi-hop wireless networks: the ADHOCSYS approach. In *Proceedings of the 3rd international conference on Mobile multimedia communications*, page 25. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[7] Satyabrata Chakrabarti and Amitabh Mishra. QoS issues in ad hoc wireless networks. *Communications Magazine, IEEE*, 39(2):142–148, 2001.

[8] Shigang Chen. Routing support for providing guaranteed end-to-end quality-of-service. 1999.

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

[10] T Clausen, C Dearlove, J Dean, and Adjih C. Generalized Mobile Ad Hoc Network (MANET) Packet Message Format. RFC 5444, IETF, February 2009.

[11] T Clausen, C Dearlove, and J Dean. Mobile ad hoc network (MANET) neighborhood discovery protocol (NHDP). RFC 6130, IETF, April 2011.

[12] T Clausen, C Dearlove, P Jacquet, and U Herberg. The optimized link state routing protocol version 2. RFC 7181, IETF, April 2014.

[13] Orangel José Azuaje Contreras. Performance Evaluation of an IEEE 802.11 Mobile AdHoc Network on the Raspberry Pi. Master's thesis, Faculdade de Engenharia da Universidade do Porto, October 2015.

[14] Douglas SJ De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434, 2005.

[15] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo. The collection tree protocol (CTP). *TinyOS TEP*, 123:2, 2006.

[16] Python Software Foundation. Python. Online, 2016. Acedido: 2016-09-30.

[17] Raspberry Pi Foundation. Raspberry Pi. Online, 2015. Acedido: 2016-09-30.

[18] Ying Ge, Thomas Kunz, and Louise Lamont. Quality of service routing in ad-hoc networks using OLSR. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 9–pp. IEEE, 2003.

[19] Silvia Giordano et al. Mobile ad hoc networks. *Handbook of wireless networks and mobile computing*, pages 325–346, 2002.

[20] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2009.

[21] Dhruv Gupta, Daniel Wu, Prasant Mohapatra, and Chen-Nee Chuah. Experimental Comparison of Bandwidth Estimation Tools for Wireless Mesh Networks. In *INFOCOM*, pages 2891–2895, 2009.

[22] Zygmunt J Haas, Marc R Pearlman, and Prince Samar. The zone routing protocol (ZRP) for ad hoc networks. Technical report, IETF Draft, July 2002.

[23] Ulrich Herberg. JOLSRv2 - An OLSRv2 implementation in Java. In *Proceedings of the 4th OLSR Interop workshop*, volume 280. Citeseer, 2008.

[24] Jiri Hosek, Pavel Vajsar, and Roman Figurny. OLSR-based QoS support in Mobile Ad-hoc Networks. *Advances in Data Networks, Communications, Computers and Materials*, 2012.

[25] Dorian Ivančić, Nikola Hadjina, and Danko Basch. Analysis of precision of the HTB packet scheduler. In *Applied Electromagnetics and Communications, 2005. ICECom 2005. 18th International Conference on*, pages 1–4. IEEE, 2005.

[26] Trapti Jain and Savita Shiwani. Analysis of OLSR, DSR, DYMO routing protocols in mobile Ad-Hoc Networks using OMNeT++ Simulation. *Global Journal of Computer Science and Technology*, 14(1), 2014.

[27] Philip Alexander Levis, Neil Patel, David Culler, and Scott Shenker. *Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks*. Computer Science Division, University of California, 2003.

[28] Mingzhe Li, Mark Claypool, and Robert Kinicki. WBest: A bandwidth estimation tool for IEEE 802.11 wireless networks. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 374–381. IEEE, 2008.

[29] Mingzhe Li, Mark Claypool, and Robert Kinicki. WBest: a Bandwidth Estimation Tool for IEEE 802.11 Wireless Networks. Online, 2016. Acedido: 2016-09-30.

[30] Emanuel Lima, Pedro Brandão, Orangel Azuaje, and Ana Aguiar. Demo: DisToNet: Disconnection Tolerant Mobile Ad-Hoc Networks. In *Proceedings of the 10th ACM MobiCom Workshop on Challenged Networks*, pages 63–64. ACM, 2015.

[31] Saoucene Mahfoudh and Pascale Minet. An energy efficient routing based on OLSR in wireless ad hoc and sensor networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1253–1259. IEEE, 2008.

[32] Open Mesh. B.A.T.M.A.N. Advanced Documentation Overview. Online, 2014. Acedido: 2016-09-30.

[33] Open Mesh. B.A.T.M.A.N. daemon Documentation Overview. Online, 2014. Acedido: 2016-09-30.

[34] Open Mesh. Visualize the mesh. Online, 2014. Acedido: 2016-09-30.

[35] Robert Morris, Eddie Kohler, John Jannotti, and M Frans Kaashoek. The Click modular router. In *ACM SIGOPS Operating Systems Review*, volume 33, pages 217–231. ACM, 1999.

[36] Anelise Munaretto, Hakim Badis, Khaldoun Al Agha, and Guy Pujolle. A link-state QoS routing protocol for ad hoc networks. In *MWCN*, pages 222–226, 2002.

[37] NetJSON. NetJSON data interchange format for networks. Online, 2016. Acedido: 2016-09-30.

[38] Mazu Networks, ICIR, UCLA, and Meraki. The Click Modular Router Project. Online, March 2014. Acedido: 2016-09-30.

[39] NetworkX developer team. NetworkX. Online, 2016. Acedido: 2016-09-30.

[40] OLSR.org. OLSR.org Network Framework. Online, October 2015. Acedido: 2016-09-30.

[41] Hrituparna Paul and Priyanka Sarkar. A study and comparison of OLSR, AODV and ZRP routing protocols in ad hoc networks. *International Journal of Research in Engineering and Technology*, 2(8), 2013.

[42] Charles Perkins, E Belding-Royer, and Samir Das. Ad hoc on demand distance vector (AODV) routing. RFC 3561, IETF, August 2003.

[43] Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM computer communication review*, volume 24 Issue 4, pages 234–244. ACM, 1994.

[44] Tiago Miguel da Cunha Pimenta. Simulação de protocolos de encaminhamento para redes AdHoc usando o NS-3. Master's thesis, Universidade do Minho, Dezembro 2013.

[45] Vital Responder. VitalResponder Project. Online, 2013. Acedido: 2016-09-30.

[46] Maamar Sedrati, Azeddine Bilami, and Mohamed Benmohamed. M-AODV: AODV variant to improve quality of service in MANETs. *arXiv preprint arXiv:1104.1186*, 2011.

[47] Jatinder Singh and Anuj Gupta. Performance Analysis of AODVv2 Protocol vs. AODV Protocol in MANET. *Internacional Journal of Emerging Technologies in Computacional and Applied Sciences (IJETCAS)*, 2013.

[48] Yong Xue, Hong Jiang, and Hui Hu. Optimization on OLSR protocol for lower routing overhead. In *Rough Sets and Knowledge Technology*, pages 723–730. Springer, 2008.

[49] Zhenyu Yang, Chandrakanth Chereddi, and Haiyun Luo. Bandwidth measurement in wireless mesh networks. *Course Project Report, URL: http://www. crhc. uiuc. edu/ cchered2/pubs. html (checked 2005-05-23)*, 2007.

[50] Yihai Zhang and T Aaron Gulliver. Quality of service for ad hoc on-demand distance vector routing. In *Wireless And Mobile Computing, Networking And Communications, 2005.(WiMob'2005), IEEE International Conference on*, volume 3, pages 192–196. IEEE, 2005.