UNIVERSIDADE FEDERAL DO PARÁ
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

RODRIGO DA SILVA SARAIVA

PROPOSTA DE IMPLANTAÇÃO DA TOPOLOGIA DE REDES EM MALHA NA CIDADE DE CASTANHAL.

CASTANHAL-PA

Novembro / 2017

RODRIGO DA SILVA SARAIVA

PROPOSTA DE IMPLANTAÇÃO DA TOPOLOGIA DE REDES EM MALHA NA CIDADE DE CASTANHAL.

Trabalho de Conclusão de Curso apresentado no curso de Sistemas de Informação da Universidade Federal do Pará, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Orientador: Dr. José Jailton Henrique Ferreira Júnior

CASTANHAL-PA

Novembro / 2017

RODRIGO DA SILVA SARAIVA

PROPOSTA DE IMPLANTAÇÃO DA TOPOLOGIA DE REDES EM MALHA NA CIDADE DE CASTANHAL.

Trabalho de Conclusão de Curso apresentado no curso de Sistemas de Informação da Universidade Federal do Pará, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Jailton Henrique Ferreira Júnior
Universidade Federal do Pará
Orientador

Prof. Dr.
Universidade Federal do Pará

Prof. Dr.
Universidade Federal do Pará

Dedico este trabalho à minha família, em especial meus pais Pedro Arinaldo e Ruth Clea pelos conselhos, por toda ajuda e sacrifício todos estes anos, durante esta jornada. E à minha namorada e futura esposa Laryze Santos por estar ao meu lado em todos os momentos, fáceis ou não, felizes ou não, apesar de tudo, sempre esteve comigo. Também dedico este trabalho à memória de minha Avó, Ana Maria Ferreira, que há quase 3 anos não está mais entre nós, porém, sempre sonhou em ver minha mãe e eu formados.

Agradecimentos

Em primeiro lugar, gostaria de agradecer à Deus, por ter me possibilitado chegar até aqui, me proporcionando a saúde e o discernimento necessário, por todas as oportunidades, pessoas e desafios que colocou em meu caminho. Gostaria de agradecer também à meus pais, Pedro Arinaldo Rodrigues Saraiva e Ruth Clea da Silva Saraiva por todo o esforço, todo sacrifício e apesar de tudo, sempre lutarem para que eu tivesse o melhor deles. À Laryze Santos Silva, minha namorada e futura esposa, ofereço um agradecimento especial, pois sempre esteve ao meu lado em todos os momentos, por suportar todas as dificuldades junto comigo e estar sempre presente quando precisei dela.

Ao meu orientador, mestre e amigo Prof. Dr. José Jailton Júnior (JJJ), pelo voto de confiança depositado, pela oportunidade, pela enorme paciência, pela disposição em todas as reuniões e conversas. Sua orientação foi fundamental pra que eu conseguisse definir diversos aspectos deste trabalho. Muito obrigado por todos os conselhos e todo o conhecimento valioso repassado.

Aos meus colegas de classe, não poderia nunca deixar de citá-los aqui, meus companheiros de luta, os quais passei fome, sono, cansaço junto, agradeço pela conclusão desse trabalho, pois sem vocês, nunca teria chegado aqui: Auriane Marques, Karla Lima, Igor Falcão e um agradecimento mais que especial a minha equipe de trabalho, o "GofS": Ranilson Pereira (Delegado), Henrique Júnior Jaques (Steve), Iuri Ferreira (Negão), Bruno Gaignoux (Gaynois), João Paulo Cruz (Palermo) e Mayron Lúcio (Sub13).

Agradeço também, aos meus companheiros do Lades, que durante o curso, formei um vínculo não só de trabalho, mas de amizade e companheirismo, em especial, aos meus amigos: Hygor Jardim e Arnaldo Souza, por sempre disponibilizarem um pouco de seu tempo para me ajudar com o simulador e com as outras técnicas para a construção desse trabalho.

E por fim, mas de forma alguma menos importante, agradeço a Faculdade de Computação FACOMP/UFPA, aos discentes e professores que tive a honra de conhecer, por esta grande oportunidade de graduação.

Resumo

O uso da rede sem fio está disponível hoje em diversos equipamentos, como *notebooks*, *tablets*, celulares, e agora com o conceito de Internet das Coisas, até mesmo eletrodomésticos como geladeiras, ar condicionado, microondas, lâmpadas e TVs podem ser gerenciadas e acessadas através de tecnologias como *Wi-Fi* e *Bluetooth*. Um atrativo para o uso dessas redes reside em seu baixo em seu baixo custo de implantação e na possibilidade de obter uma conexão de dados em qualquer ponto dentro da área de cobertura. Atualmente, as redes locais sem fio costumam se organizar como redes infraestruturadas, em que uma estação base denominada ponto de acesso *(AP - Access Point)* encabeça um grupo de dispositivos, intermediando suas transmissões. No entanto, outras possibilidades de organização existem, tais como redes *Ad-Hoc* e redes *Mesh*. Uma rede *Mesh*, como qualquer rede *Ad-Hoc*, não necessita de um ponto de acesso para a comunicação entre os dispositivos, pois por definição, estes equipamentos podem se comunicar diretamente.

Nas redes *Mesh*, os dispositivos utilizam o padrão próprio da arquitetura, *IEEE 802.11s*, formando um cenário que lembra uma malha, pela organização dos nós, que realizam suas transmissões entre si através de múltiplos saltos. Outra grande vantagem deste tipo de rede, além de utilizar um protocolo roteamento exclusivo do padrão 802.11s, é que se algum dispositivo for removido da rede ou simplesmente desligar, a rede automaticamente muda sua organização, afim de cobrir a área que aquele nó abrangia antes, ou seja, cobrir aquele buraco na malha. Essa tolerância a falhas da arquitetura, veremos nos capítulos seguintes.

Neste trabalho realizou-se a simulação de uma rede *Mesh* na cidade de Castanhal, fazendo uso do Simulador de Redes *NS-3* e todas as bibliotecas suportadas pela arquitetura. Com essa rede se realizaram experimentos para medir a vazão, compatibilidade e comparação com outros protocolos de rede, inclusão e remoção de nós do cenário e alteração na alocação dos nós.

Palavras-chave: IEEE 802.11s; WiFi; Redes Ad Hoc; Mesh; Rede em Malha; Simulação; NS-3; Protocolo de Roteamento; HWMP; Redes Móveis Sem Fio; Access Point; Vazão.

Abstract

The use of the wireless network is available today in various devices like notebooks, tablets, cell phones, and now with the concept of Internet of Things, even home appliances how refrigerators, air conditioners, microwaves, lamps and TVs can be managed and accessed through technologies such as Wi-Fi (IEEE 802.11) and Bluetooth. One advantage to using these networks lies in their low cost of deployment and the ability to get a data connection at any point within the coverage area. Currently, wireless local area networks usually organize themselves as infrastructure networks, where a base station called the Access Point (AP) leads a group of devices, mediating their transmissions. However, other organizational possibilities exist, as the Ad Hoc networks and Mesh networks. A Mesh network, like any Ad Hoc network, does not need an access point for communication between devices, because by definition, these devices can communicate directly.

In this networks, IEEE 802.11s, forming a scenario that reminds a mesh, by the organization of the nodes, that carry out their transmissions to each other through multiple hops. Another large differential of the network kind, in addition to using a single frequency pattern and a unique routing protocol, if any device is removed from the network or simply disconnected, the network automatically changes its organization In order to cover the area that the node covered before. That is, cover that hole in the mesh. This mechanism to avoid failures of the architecture, we will see later.

In this work the simulation was performed of a Mesh network in the city of Castanhal, making use of the NS-3 network simulator and all the libraries supported by the architecture. With this network, experiments were made to measure the flow, compatibility and comparison with other network protocols, inclusion and removal of nodes from the scenario and change in the allocation of nodes.

Key words: IEEE 802.11, WiFi, Ad Hoc Networks, Mesh Network, Simulation, NS-3, Router Protocols, HWMP, Mobile Wireless Networks, Access Point, Network Throughput.

Conteúdo

Lista de Abreviaturas

AP	Access Point
IEEE	Institute Of Electrical and Electronic Engineers
LAN	Local Area Network
WAN	Wide Area Network
WWAN	Wireless Wide Area Network
PHY	Camada Fisica do Modelo OSI
MAC	Camada de Controle de Acesso ao Meio
BSS	Basic Service Set
STA	Stations
ESS	Extended Service Set
EM	Estações Moveis
BSA	Basic Set Area
MAP	Mesh Access Point
MPP	Mesh Portal Point
MBSS	Mesh Basic Service Set
HWMP	Hybrid Wireless Mesh Protocol
AODV	Ad-Hoc On-Demand Distance Vector
PREP	Path Reply
PREQ	Path Request
RANN	Root Announcement
MP	Mesh Points
ALM	Airtime Link Metric
NS-3	Network Simulator 3
OLSR	Optimized Link State Routing
NetAnim	Network Animator
SUMO	Simulator Of Urban Mobility

MTU Maximum Transmit Unit

Lista de Figuras

CAPÍTULO 1

Introdução

Este capítulo irá expor brevemente sobre Redes *Ad-Hoc*, mais especificamente, Redes em Malha sem fio *(Mesh)*, apresentando as principais vantagens e desafios enfrentados por esse tipo de rede, sintetizando suas principais aplicações. Também serão apresentados superficialmente os próximos capítulos.

1.1 Visão geral

As redes locais sem-fio podem se organizar de forma infraestruturada, *Ad-Hoc* e também em *Mesh*. Nas redes infraestruturadas, uma estação base denomina-se Ponto de Acesso ou *Access Point* (AP) que gerencia um determinado grupo de dispositivos, intermediando suas transmissões. Para que os dispositivos possam se comunicar pela rede sem fio, devem primeiro se associar a um *AP*. Nas redes *Ad-Hoc* não existe esta estação central, e assim, qualquer dispositivo na rede pode se comunicar com qualquer um em seu alcance. Nas redes *Mesh*, que podem ser entendidas como um caso especial das redes *Ad-Hoc*, os dispositivos são capazes de determinar suas rotas de transmissão dentro da rede. Com isso, um nó que queira realizar uma transmissão, pode usar outros nós como intermediários para encaminharem quadros para destinos que estejam fora de seu alcance (GOLDSMITH, 2015).

Uma rede sem fio é como qualquer tipo de rede móvel, cujos nós tem algum nível de mobilidade dentro do cenário, podendo existir, alguns nós fixos, sem mobilidade alguma. Uma das formas de classificação desse tipo de rede é quanto às características de mobilidade e de comunicação de seus nós.

Um grande diferencial dessa arquitetura é a sua escalabilidade, pois como o núcleo da rede é gerenciado pelos nós (roteadores) da borda, quando um novo dispositivo é

inserido no cenário, ele já tem conexão com todos os demais, bem como todos os outros podem chegar até o novo nó inserido. A rede em malha já é uma realidade nos EUA e em toda a Europa, no Brasil, mais especificamente no norte do país, está ainda sendo utilizada em caráter acadêmico, e através deste trabalho, almeja-se o emprego desta tecnologia para ampliar o desempenho na transmissão de dados na região.

Nas redes *Mesh*, as estações se comunicam diretamente, sem a necessidade de intermediação de um ponto de acesso. Caso a estação destino esteja fora do alcance do sinal do nó que deseja realizar a transmissão, os quadros a aquela estação podem ser encaminhados por estações intermediárias. Com isto, as estações possuem função de roteadores, o que as habilita a encaminhar pacotes de forma a manter a conectividade a todos os dispositivos da rede. Para isto, é necessário que estações sejam alocadas no cenário de forma estratégica, para cobrir a maior área possível, funcionando como uma infraestrutura estática com a finalidade de prover conectividade a dispositivos de usuários. No caso de estações que estejam a apenas um salto de distância, isso possibilita que se obtenha um melhor aproveitamento do canal sem fio, por reduzirem as quantidades de transmissões necessárias.

As redes *Mesh*, possuem uma habilidade de auto-organização, de forma que novos caminhos entre as estações possam ser formados caso ocorra uma mudança na topologia da rede (Ex: queda ou adição de uma nova estação, quebra de um *link*, sobrecarga de uma rota). Para o acesso a rede cabeada, é suficiente que ao menos uma estação da rede *Mesh* esteja a ela diretamente conectado, o que possibilita que compartilhe seu acesso com as demais estações dispostas na rede. Devido a esses atrativos, existe um esforço para que redes (IEEE 802.11 possam funcionar também como *Mesh* (JUNG, 2011).

A estrutura da rede *Mesh* no escopo de redes sem-fio *IEEE* 802.11 é objeto do padrão *IEEE* 802.11s, cuja elaboração iniciou em 2004 pelo grupo de estudos (Task Group - TGs). A proposta final foi aprovada em julho de 2011 e publicada no final de Novembro de 2011, sendo parte do padrão IEEE 802.11 - 2012 (BAUER, 2012).

Para chegarmos a um trabalho sobre Redes em malha, é necessário ter uma abordagem relativamente importante sobre redes sem fio, mais em especial, em Redes *Ad-Hoc*, que é onde o tipo malha está inserido. Abordagem esta que será feita nos capítulos seguintes.

1.2 Motivação

Nesse contexto, a transmissão de dados que é predominantemente utilizada entre empresas, pessoas e instituições é a cabeada, seja pelo comum cabo de rede ou fazendo uso do enlace óptico, por conta da disponibilidade das provedoras, porém, as redes sem fio, vem ganhando cada vez mais espaço no mercado de trabalho, com barateamento nos custos de instalação, equipamentos e instalação, maior alcance, e menos complexidade na configuração dos dispositivos, bem como praticidade na transferência dos dados e a mobilidade.

O grande avanço das redes sem fio, fez com que surgisse as redes *Mesh*, e sua característica é de grande usabilidade principalmente em regiões de acesso complicado, como também em regiões onde as provedoras de serviço cabeado não alcançam. Nessa arquitetura de rede, em que um nó é tido como um roteador móvel, alargando o raio da rede de forma que a organização de cada nó, pode inserir e se comunicar com outros nós, utilizando um protocolo de roteamento de múltiplos saltos para que a transmissão seja realizada (OLIVEIRA, 2012).

Segundo esse padrão, estações podem se comunicar diretamente e a conectividade em toda a rede deve ser provida por meio da descoberta de caminhos efetuada diretamente na camada de enlace. O padrão 802.11s faz uso de protocolos de descoberta de caminhos dinâmico, que na ocorrência de uma mudança na estrutura atual da rede, como queda ou acréscimo de uma estação, a rede se adapte de forma automática, sem a quebra de conectividade. Além disso, o padrão possibilita que uma ou mais estações possam prover acesso à rede cabeada ao resto da rede *Mesh*.

O surgimento das redes em malha sem fio trouxe a possibilidade da popularização da Internet, criando as chamadas "cidades digitais", onde o acesso é proporcionado através de pontos de acesso distribuídos estrategicamente em campus universitários, prédios, comerciais, hospitais, etc. As vantagens acerca dessa tecnologia beneficiam diversos segmentos da sociedade, criando uma rede de comunicação fácil implantação e baixo custo, podendo se expandir facilmente.

1.3 Objetivos

O objetivo geral deste trabalho é analisar e simular o emprego das redes *Mesh* na cidade de Castanhal, visando uma possível distribuição gratuita de internet em pontos específicos da cidade, bem como, em locais onde seja geograficamente e/ou financeiramente inviável de se ter uma internet cabeada, tendo como finalidade levar conexão a internet a um número ainda maior de pessoas. Este projeto teve como objetivo fazer um estudo da viabilidade e do impacto de se empregar em nível de rede e de aplicação da arquitetura de rede em malha *Mesh*, utilizando um simulador computacional de redes e a partir deste estudo, verificar qual o melhor cenário, protocolo, quantidade de nós irá se adaptar melhor ao ambiente simulado. A avaliação de desempenho, por meio de simulações de redes sem fio *Ad-Hoc*, teve como meta analisar os ganhos relativos para as transmissões nos cenários construídos.

1.3.1 *Objetivos Específicos*

Quanto aos objetivos específicos, englobam os seguintes tópicos:

- Comparar o desempenho dessa rede *Mesh* alocando manualmente seus pontos em lugares da cidade de Castanhal;

- Estimar a vazão e fatores que possam influenciar no desempenho dessas redes, tanto no caso de estações alocadas manualmente ou aleatórias.;
- Comparar o desempenho dessa rede comparando seu protocolo padrão de roteamento *(HWMP)* com outros protocolos de roteamento suportados pelo simulador, e teve como meta analisar os ganhos relativos ao usar cada protocolo;

Esta transmissão, será realizada no simulador, de uma forma simultânea entre os nós, fazendo uso de alguns dos vários protocolos de roteamento disponíveis no simulador, para que cada nó se comunique com os demais, de uma forma direta ou não, ou seja, utilizando de *multi-hops* ou múltiplos saltos para que a mensagem chegue ao destino.

1.4 Organização do texto

O restante do trabalho está dividido em 5 capítulos, ordenados de acordo com as descrições abaixo:

- Capítulo 2: Apresenta um estudo sobre os aspectos importantes relacionados às Redes Sem Fio com um aprofundamento em Redes *Ad-Hoc* que são onde as Redes em malha estão inseridas, como as aplicações, os desafios na transmissão de conteúdo e os principais modelos de transmissão e descoberta de rotas existentes.
- Capítulo 3: Apresenta alguns trabalhos relacionados ao controle de topologia das Redes em Malha. Ele descreve as características e o funcionamento de cada trabalho e as principais vantagens e desvantagens das soluções apresentadas em cada um.
- Capítulo 4: Detalha a proposta do trabalho, a construção do cenário no Simulador, A Arquitetura Utilizada, Protocolos e a Metodologia Aplicada nas Simulações.
- Capítulo 5: Apresenta as ferramentas utilizadas no decorrer do trabalho, bem como o que se obteve com cada uma. Exibe também a avaliação no desempenho de cada protocolo de rede utilizado, bem como, o quanto esse desempenho se altera quando demais fatores na simulação são alterados.
- Capítulo 6: Apresenta a conclusão do trabalho, resumindo as principais contribuições e resultados. Fazendo, também, direcionamentos para trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

Neste capítulo, será apresentada uma visão geral sobre as Redes *Mesh* em Malha, quanto às características da rede e os benefícios fornecidos pelo seu uso. Além disso serão apresentadas algumas aplicações que se beneficiam desse tipo de rede. É necessário também, fazer uma abordagem sobre o Padrão *IEEE* 802.11, bem como, sua relevância e aplicabilidade, analisar os agentes ativos em uma Rede sem fio, para poder entender métricas como custo, desempenho, interferência, atraso e perda dos pacotes.

2.1 Introdução as Redes Sem Fio *Ad-Hoc*

O processo evolutivo das tecnologias de telecomunicações juntamente com a informática, atribuído a crescente necessidade de manter pessoas conectadas a internet, tem motivado cada vez mais o desenvolvimento das redes sem fio. O crescimento das redes sem fio em conjunto com a informática tem o objetivo de atender necessidades, tais como: serviços celulares, redes sem fio, transmissões de dados via satélite, provedores via rádio, bem como a proposta principal deste trabalho, a distribuição gratuita de internet.

Da mesma forma que as redes cabeadas, as redes sem fio podem ser de dois tipos: LAN *(Local Area Network)* e WAN *(Wide Area Network)*. As redes sem fio do tipo WAN ou WWAN *(Wireless Wide Area Network)* baseiam-se principalmente nas redes de telefonia celular, a princípio utilizadas para comunicação de voz, porém, nos dia de hoje é bastante possível a transferência de dados nesse meio. Com a fabricação de redes de forma diversificada, de acordo com cada fabricante, em 1991 a *IEEE (The Institute of Electrical and Electronics Engineers, Inc)* é solicitada a elaboração de um padrão para redes sem fio locais e metropolitanas, configurando-se o grupo 802.11. Devido a atrasos, apenas em 1997 foi publicada a especificação que padronizava a conectividade sem fio entre equipamentos em uma área local e que permitia a utilização de equipamentos de

diferentes fabricantes.

As Redes *Ad-Hoc* também são constituídas por estações que utilizam comunicação sem-fio. A principal características dessas redes e a ausência de infra-estrutura, como pontos de acesso ou estações base, existentes em outras redes locais sem-fio. A comunicação entre estações que estão fora do alcance de transmissão deve ser feita pela camada de rede, o que implica o encaminhamento de mensagens por múltiplos saltos através da colaboração de estações intermediárias. Numa Rede *Ad-Hoc* não há topologia predeterminada, nem controle centralizado(JUNG, 2011).

2.2 O Padrão IEEE 802.11

O padrão *IEEE* 802.11 define a padronização relativa às camadas Físicas (PHY) e a de Controle de Acesso ao Meio (MAC) para redes sem fio. Uma rede baseada nesse padrão é composta pelos seguintes componentes:

- BSS - *(Basic Service Set)*: Corresponde a uma rede de comunicação Wireless;
- STA - *(Stations)*: Estações de trabalho que comunicam-se entre si dentro da BSS;
- AP - *(Access Point)*: Nó responsável por coordenar a comunicação entre as STA dentro da BSS;
- ESS - *(Extended Service Set)*: São células BSS próximas que se interceptam e que os AP estão ligados a uma mesma rede. Com isso, um STA pode se deslocar de um BSS para outro, mantendo a conexão com a rede;

As redes caracterizadas no padrão 802.11, podem operar de dois modos diferentes: *Infrastructure mode* (Redes de Infra-Estrutura) e *Ad-Hoc mode*.

2.2.1 *Redes Infraestruturadas*

Tem como característica possuir dois tipos de elementos: As Estações Móveis (EM) e os Pontos de Acessos (AP). Os *APs* são responsáveis pela conexão das *EM* com a rede fixa, cada ponto tem o controle de uma determinada Área de Cobertura BSA *(Basic Set Area)*. A transferência de dados nessas redes, acontece sempre entre uma *STA* e um *AP*. A transferência de dados nunca ocorre diretamente entre duas estações. O *AP* também pode agir como uma ponte para outra rede (cabeada ou sem fio)(REZENDE, 2014) . Redes com infra-estrutura perdem um pouco da flexibilidade que as Redes sem Fio podem oferecer, por exemplo, elas ficam inutilizadas no caso de um terremoto que provoque a destruição de toda a infraestrutura da rede ou somente dos seus *APs*.

2.2.2 *Redes Ad-Hoc*

As redes *Ad-Hoc* serão utilizadas como base dos cenários simulados no nosso trabalho, portanto, será dado um foco maior sobre suas tecnologias e vantagens, bem como em seu suporte aos protocolos que serão utilizados nas simulações.

As redes *Ad-Hoc* tem como característica não possuir nenhuma infraestrutura para apoiar a comunicação. Os diversos equipamentos móveis ficam localizados em uma área de cobertura dos demais nós e estabelecem comunicação ponto a ponto entre os mesmos. Nenhum *AP* é necessário para controlar o acesso ao meio. Uma estação A só pode se comunicar com B se B estiver dentro do raio de ação de A ou se existir uma ou mais estações entre A e B que possam encaminhar a mensagem, dependendo do protocolo de roteamento. Entenda-se por raio de ação a área de cobertura de uma estação, ou seja, todos os pontos geográficos onde o sinal desta estação chegue com um mínimo de clareza (REZENDE, 2014).

Numa rede *Ad-Hoc*, a complexidade e a taxa de processamento de cada estação é elevada porque toda estação deve ser munida com mecanismos de acesso ao meio, bem como de comunicação com as outras estações, mecanismos para controlar problemas com caminhos congestionados ou links rompidos, além de mecanismos para prover uma certa qualidade de serviço. Quando os nós se movem, a mudança resultante na topologia da rede deve ser informada a todos os outros nós, de tal forma que as informações sobre a topologia possam ser atualizadas frequentemente. Estas redes são tolerantes a falhas e a retirada de estações (Nós), pois sua organização e controle não dependem apenas de alguns nós determinados, todos os nós têm a função de fazer o roteamento e a retransmissão dos pacotes trafegados. Da mesma maneira, novos nós podem ser adicionados, facilmente, a essas redes (GOLDSMITH, 2015).

As funções de núcleo de rede, em geral, utilizam os protocolos de roteamento tradicionais já conhecidos, que fazem uso de técnicas de roteamento denominadas vetor de distância *(distance-vector ou hop-by-hop)* ou estado dos enlaces *(link-state)*, utilizados na operação das redes cabeadas. Ao contrário deste tipo de solução, a meta das redes móveis sem fio *Ad-Hoc* é estender o conceito de mobilidade para permitir que domínios móveis sem fio, totalmente autônomos e compostos por um conjunto de nós (que podem ser roteadores ou estações) formem, eles mesmos, uma infraestrutura de roteamento de rede em malha, do tipo *Ad-Hoc*.

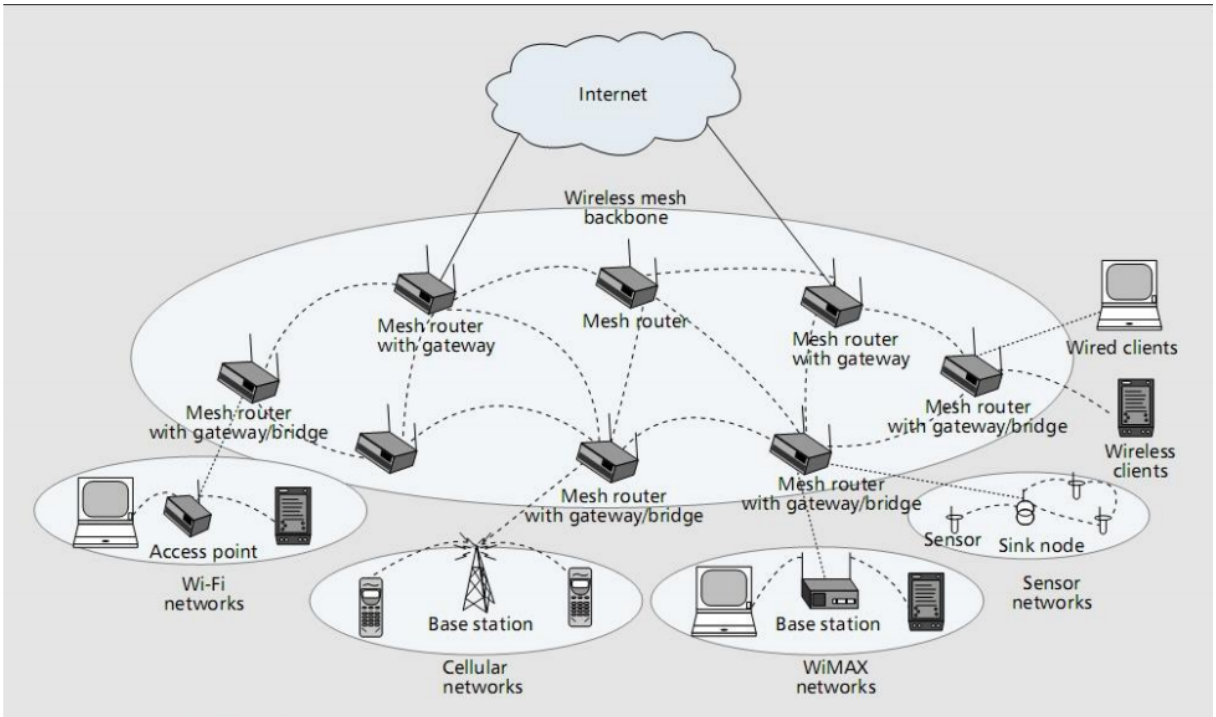
2.3 O Padrão IEEE 802.11s: Redes Sem Fio Mesh

Conforme (TEIXEIRA, 2011), as redes *Mesh* foram financiadas na década de 90 pelos militares americanos na *Defense Advanced Research Projects Agency* (DARPA), com o objetivo de comunicação fim a fim sem a necessidade de uma estação central. Foi usada pelos militares durante a guerra do Iraque, para a comunicação entre as diversas equipes, onde cada elemento, soldados, tanques e helicópteros, eram uma estação na rede.

Para os militares, isto evitaria a fragilidade de um ataque a uma estação central, se existente. Mas as redes sem fio possuem desvantagens, como a dificuldade em atender requisitos de qualidade de serviço. Diversos fatores contribuem para isso, dentre eles a maior sensibilidade a interferências causadas por outros equipamentos, devido ao uso de rádio na transmissão de dados, elevando assim a taxa de erro; taxa de transmissão muito baixa comparada a rede cabeada; problemas de segurança, pois com o uso de ondas de rádio, os dados podem ser interceptados mais facilmente. Dependendo do tipo de uso destas redes, os benefícios proporcionados, largamente compensam seus aspectos negativos, o que pode ser verificado pela disseminação de seu uso. (BAUER, 2012).

No escopo do padrão *IEEE* 802.11, uma rede local sem fio *Mesh* foi definida da seguinte maneira: Um *BSS* da rede *Mesh* (MBSS) é uma *WLAN IEEE* 802.11 composta por nós autônomos. Dentro do *MBSS*, todos os nós estabelecem enlaces sem fio com nós vizinhos para a troca de mensagens. Além disso, usando a capacidade dessa rede de se comunicar utilizando múltiplos saltos, pode haver comunicação entre nós que não possuem enlace direto entre si. Do ponto de vista de entrega de dados, a rede funciona como um único domínio de broadcast. Assim, todas os nós em um *MBSS* se comunicam diretamente em nível de enlace, mesmo se não estiverem dentro de alcance direto. A capacidade de múltiplos saltos tem o efeito de aumentar o alcance dos nós, e assim ampliar a conectividade da rede local sem fio.

Figura 1: Representação do Cenário com uma Rede em Malha, da Organização e Comunicação dos Nós.



Fonte: (AKYILDIZ I. F.; WANG, 2005)

As redes *Mesh* como redes *Ad-Hoc*, possuem a característica de não necessitar

de um ponto de acesso para a comunicação entre os dispositivos, pois estes podem se comunicar diretamente. Cada dispositivo funciona como um roteador dentro da rede sem fio, encaminhando as informações vindas de seus vizinhos de forma cooperativa, ou seja, comunicação entre os dispositivos através de um ou mais saltos.

As redes *Mesh* são uma variação do gênero de redes *Ad-Hoc*, porém diferem no fato das redes *Ad-Hoc* serem redes que tem uma topologia altamente dinâmica, não infra estruturada, e suas estações podem apresentar mobilidade. Por causa disso, precisam de protocolos de roteamento que permitam que isso ocorra com o mínimo de problemas, ao contrário das redes *Mesh* que possuem um backbone formado por roteadores sem fio que mantém a rede com rotas mais estáveis, possuindo uma hierarquia relativamente estática, podendo ter um ou mais pontos de acesso com a rede externa. Outro ponto é o protocolo de roteamento, pois enquanto na rede *Ad-Hoc* os usuários devem entrar em um acordo para utilizarem o mesmo protocolo, na rede *Mesh* isso faz parte da própria tecnologia e assim se mostra transparente para o usuário. Nas redes *Ad-Hoc*, o roteamento ocorre na camada de rede "3", mas nas redes *Mesh* isso é realizado na camada de enlace "2"(FERNANDES, 2008).

As redes *Mesh* caracterizam-se também pela capacidade de seus dispositivos se auto organizarem em caso de mudança da topologia. No caso de perda de uma estação, a rede se reorganiza, formando novas rotas. Devido a sua capacidade de auto-organização, a rede incorpora uma nova estação *Mesh* de forma automática. A auto-organização na rede *Mesh* ajuda na escolha de melhores caminhos entre os dispositivos, de forma que as mensagens sejam transmitidas para uma estação vizinha mais apropriada para atingir o destino. Com isso, estas redes apresentam desafios como o desenvolvimento de protocolos de roteamento capazes de lidar com mudanças na topologia da rede e também escolher rotas que proporcionem transmissões com menos erros e melhor aproveitamento do canal. Na rede sem fio *Mesh*, somente um ponto precisa estar conectado fisicamente a uma rede que forneça acesso a Internet. Essa estação compartilha sua conexão à Internet, sem o uso de cabos, com todas as outras estações ao seu redor, os quais também compartilham com as estações mais próximas a elas, formando uma espécie de malha.

Devido a isso, novas estações são incorporadas de forma automática, devido ao uso de protocolos de roteamento dinâmico na camada de enlace*"2"*. Isso ocorre sem a necessidade de interferência do usuário, oferecendo a ele comodidade e facilidade para acesso. Essa característica deste tipo de rede pode ser útil para diversas aplicações. Entre elas citam-se redes domésticas, institucionais e metropolitanas (FERNANDES, 2008).

2.3.1 *Elementos e Arquitetura das Redes Mesh no Padrão 802.11*

A arquitetura da norma *IEEE* 802.11 consiste em vários componentes que interagem de forma a que seja possível a formação de uma rede local sem-fios com suporte de mobilidade e auto-configuração de estações de uma forma transparente para as camadas superiores (PEREIRA, 2009). A rede *IEEE* é composta por estações autônomas e interdependentes que funcionam como roteadores, que podem se comunicar de forma

direta ou a partir de múltiplos saltos, encaminhando quadros até o destino ou utilizando as estações vizinhas para alcançar o mesmo. Além disso, podem disponibilizar acesso a outras estações, não participantes da rede *Mesh*, e até mesmo acesso a Internet. Para isto, o padrão *IEEE* 802.11 definiu os seguintes elementos para a rede *Mesh*:

- STA - Cliente ou Estação: É uma estação que requer serviços, mas não repassa dados, nem participa da descoberta de caminhos feita pelos protocolos de roteamento;
- MP - *Mesh Point*: É uma estação que participa da formação e operação da rede *Mesh*, repassando dados e participando da descoberta de rotas;
- MAP - *Mesh Access Point*: É um MP agregado a um nó que está provendo acesso a internet a clientes STA em uma determinada área.
- MPP - *Mesh Portal Point*: É um MP com uma funcionalidade especial de atuar como um *gateway* entre a rede Mesh e a rede externa (internet, por exemplo);
- MBSS - *Mesh Basic Service Set*: É um grupo de estações que formam a Rede *Mesh*. Podendo conter MP, MAP e MPP. Um MBSS pode conter mais de um MPP.

As estações das redes em Malha criam enlaces com seus vizinhos e se comunicam através de caminhos que são descobertos dinamicamente, usando por padrão o protocolo de roteamento HWMP *(Hybrid Wireless Mesh Protocol)*. As transmissões nesse tipo de rede, ocorrem da existência de nós interconectados que devem sem comunicar e enviar dados de um lado a outro da rede, mas superar falhas causadas por nós que simplesmente podem deixar de funcionar em um determinado momento. Mais do que isso, essas redes devem ter a capacidade de criar e/ou decidir as melhores rotas sozinhas e dinamicamente, se adaptando ao funcionamento ou não dos nós ao seu redor(FERNANDES, 2008).

Quando um evento como esse acontece, a rede deve procurar outro caminho para que a informação chegue ao destino, procurando outros nós disponíveis que possam, sem passar pelo ponto danificado, levar a informação ao destino traçado na rota. O fato de redes *Mesh* tolerarem tais falhas, as torna suficientemente confiáveis, isso sera melhor explicado nos capítulos a seguir.

2.3.2 *Protocolos de Roteamento*

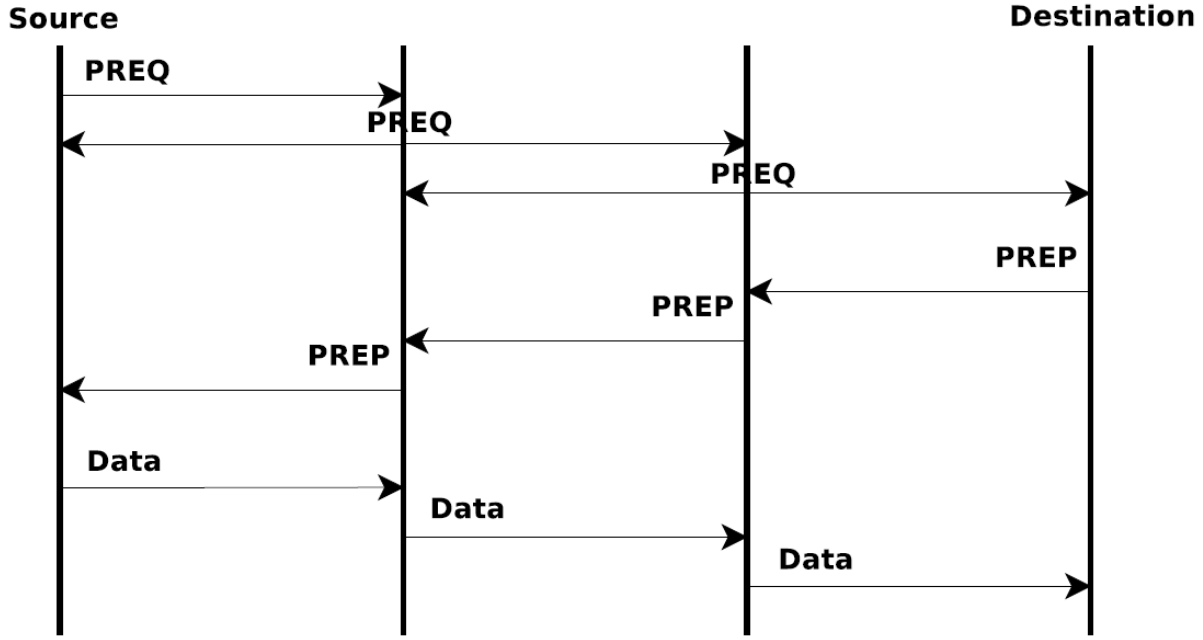
Por ser dinâmica e com alta mobilidade, as redes em malha sem fio, em questão de protocolos de roteamento, estão em constantes pesquisas para o desenvolvimento de protocolos que atenda essa topologia. As soluções hoje utilizadas são os protocolos de roteamento: *Unicast e Multicast.* O protocolo de roteamento *Unicast* é a transmissão mais comum, ponto a ponto, ou seja, um pacote do nó de origem terá somente um destino. Já no *Multicast*, o pacote do nó de origem é transmitido a um grupo de nós destino. (MEDEIROS, 2012).

O *IEEE* 802.11s propõe como base para o uso de roteamento em Redes *Mesh*, o protocolo *HWMP (Hybrid Mesh Wireless Protocol)*. O protocolo *HWMP* foi baseado no protocolo AODV *(Ad Hoc On-Demand Distance Vector)*, adaptado para camada de enlace. O protocolo *HWMP* fornece dois modos de operação: *Roteamento sob-demanda* e *Roteamento proativo*.

- Modo Sob-demanda: A descoberta das rotas nesse modo, é realizada somente quando necessário. Como exemplo, se uma estação precisar transmitir um quadro a um destino, porém não conhece uma rota adequada até esse destino, inicia-se a descoberta de caminho sob demanda. Esse modo possibilita que os MPs se comuniquem usando caminhos fim a fim, além de não congestionar a rede, visto que a comunicação para descoberta de rotas só é iniciada quando a transmissão é requerida.(ANDREEV, 2010)

  No modo sob demanda, a descoberta do caminho começa quando uma estação da malha tem dados para transmitir para um destino desconhecido. Ele transmite um pedido de caminho *(Path Request - PREQ)* informando o destino para os demais nós. Cada estação que recebe esta PREQ, cria uma rota para a origem, com métricas de atualizações e mensagens de encaminhamento. Se a estação tiver uma informação de roteamento válida para o destino ou a estação é o próprio destino, é gerado um quadro de gerenciamento de resposta de caminho (*Path Reply* - PREP).

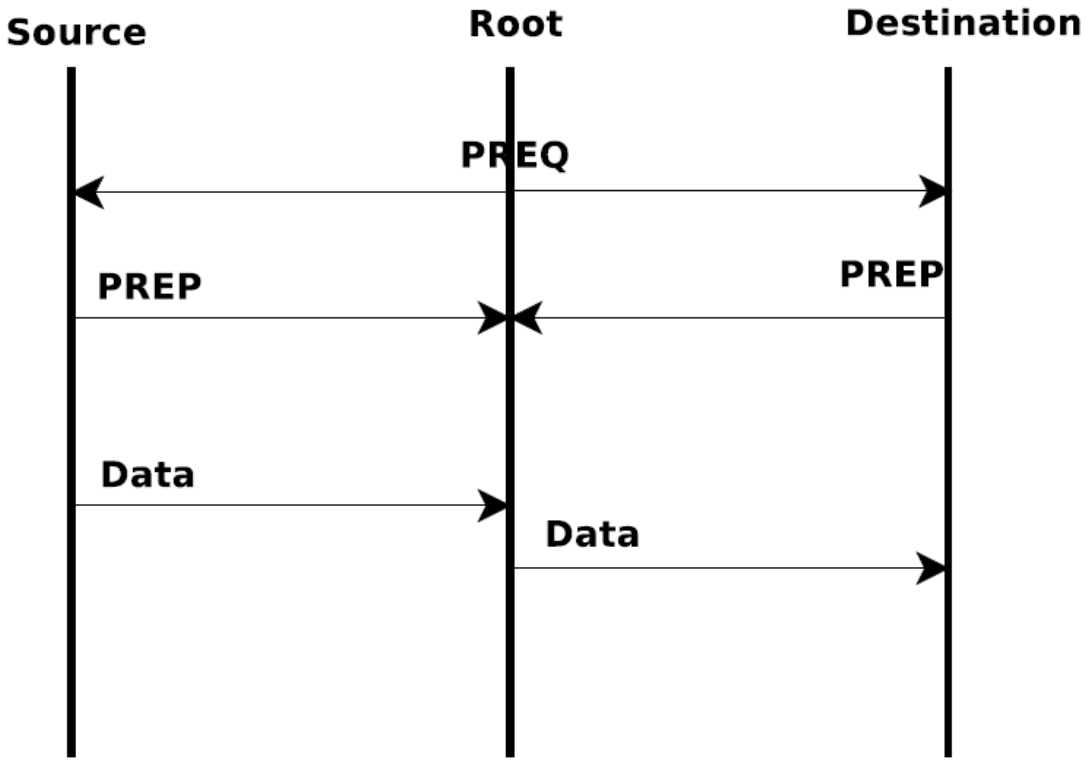
Figura 2: Representação do esquema de descoberta de rotas do modo sob demanda. *(On-Demand Mode)*



Fonte: (ANDREEV, 2010)

- Modo proativo: Nesse modo, a funcionalidade de construção proativa de uma árvore de caminhos é adicionado ao modo sob demanda. Essa árvore tem como base um MP, em que se usam os mecanismos PREQ *(Path Request)* ativo ou RANN *(Root Announcement)* para descobrir caminhos para todas as demais estações da rede. O mecanismo PREQ ativo cria os caminhos a partir de cada MP para a estação origem da transmissão, podendo ser bidirecional. O mecanismo RANN cria caminhos bidirecionais entre a estação origem e cada MP da rede.

Figura 3: Representação do esquema de descoberta de rotas do modo Pró Ativo. *(Proactive Mode)*



Fonte: (ANDREEV, 2010)

O HWMP conta com a flexibilidade da descoberta de rotas do modo sob-demanda, além de um roteamento eficiente do modo proativo, para os seus portais *Mesh*. Segundo (ALBUQUERQUE, 2011), durante o processo de descoberta de um caminho, cada estação irá contribuir com seus cálculos de métrica acrescentando ou atualizando dados nos quadros de gerenciamento dedicados à troca de informações de encaminhamento. Independente do modo de operação (proativo ou reativo) as funções do HWMP são implementadas pelos seguintes agentes de gerenciamento:

- *Path Request* (PREQ): Requisição de caminho - estes quadros são enviados em broadcast por um MP que deseja encontrar um caminho para outro MP;

- *Path Reply* (PREP): Resposta de Caminho - estes quadros são enviados pelo MP de destino, em resposta ao recebimento de uma requisição de caminho PREQ;
- *Path Error* (PERR): Erro no caminho - estes quadros são utilizados para notificação de que um caminho não está mais disponível;
- *Root Announcement* (RANN): Anúncio da estação origem - estes quadros são utilizados pela estação que se anuncia como estação de origem daquela transmissão. Como visto antes, existem duas formas de comportamento do HWMP, sendo o quadro RANN utilizado em um deles;

2.4 Conclusão do Capítulo

Este capítulo teve como objetivo apresentar as principais tecnologias e conceitos que serão utilizados neste trabalho, apresentando as principais características das Redes *Ad-Hoc*, bem como das Redes em Malha, seus desafios, benefícios e aplicações dos elementos que agem nessas redes. Além disso, foram descritas as métricas que podem ser obtidas com a utilização dessa arquitetura.

A característica peculiar do HWMP, de se auto-organizar perante a uma alteração na sua organização, ajuda na escolha de melhores caminhos entre os dispositivos. Com isso, estas redes apresentam desafios como o desenvolvimento de protocolos de roteamento capazes de lidar com mudanças na topologia da rede e também pode escolher rotas que proporcionem transmissões com menos erros e melhor aproveitamento do canal. O protocolo de roteamento ou seleção de caminho é utilizado para descobrir ou manter os caminhos de múltiplos saltos da rede *Mesh*. É comum, o protocolo de roteamento utilizar métricas de seleção de caminhos para escolher um caminho melhor entre os outros opcionais (ANDREEV, 2010).

CAPÍTULO 3

Trabalhos Relacionados

Este capítulo tem como objetivo apresentar alguns trabalhos presentes na literatura, relacionados a Redes *Ad-Hoc*, Protocolos de Roteamento e as Redes *Mesh*. Descreve-se as principais características e o funcionamento de cada trabalho e compara suas vantagens e desvantagens no processo de comparação de qual melhor protocolo utilizar e onde melhor se aplica a Rede em Malha.

Bem como entendermos onde melhor aplicar esta topologia, o capítulo também tem como objetivo explanar as diversas técnicas de descoberta de rotas bem como atualização da tabela de roteamento dos *Mesh Points* (MP) e da auto organização da rede citada nos capítulos anteriores.

3.1 Redes Mesh: Topologia e Aplicação

3.1.1 *Aplicação da Arquitetura*

As redes em malha sem fio ou Redes *Mesh* já são uma realidade em diversas cidades no mundo, especialmente em lugares onde há obstáculos a redes cabeadas, como terrenos montanhosos. No Brasil temos o exemplo da cidade de Tiradentes no Estado de Minas Gerais, onde foram implantados equipamentos em pontos estratégicos, sendo um grande desafio, pois o local é montanhoso e as casas possuem paredes espessas dificultando a passagem do sinal *Wireless*.

No presente trabalho, serão abordadas algumas aplicações que já fazem uso dessa tecnologia. Tratando-se de aplicação de redes *Mesh*, podemos citar as seguintes:

- Lugares isolados: como mencionado anteriormente, a aplicação de redes *mesh* em

lugares de difícil acesso, como regiões montanhosas onde prover o acesso a Internet só é possível por transmissões via *wireless*, mostrou-se satisfatório até então. Nessas regiões são instalados diversos pontos de acesso, utilizando-se de equipamentos de baixa potência e longo alcance, geralmente alimentados por energia solar (MEDEIROS, 2012).

- Educação: a implantação das redes *mesh* em universidades, escolas, provê o acesso a Internet a alunos e professores, promovendo a comunicação rápida e a troca de informações. Não há necessidade de cabeamento, pois são instalados diversos pontos de acesso *wireless* que podem ser internos e externos (MEDEIROS, 2012).

- Saúde: como nem todas as construções são adaptadas para receber uma infraestrutura de rede, a rede *mesh* é ideal nesses casos. Em hospitais, principalmente em prédios mais antigos, a rede mesh pode enviar sinais a curtas distâncias, através de vidros espessos. Isso garante a conexão entre os consultórios e laboratórios, mantendo atualizadas as informações sobre exames e históricos médicos, podendo o profissional de saúde ter acesso a essas informações quando necessário (MEDEIROS, 2012).

- Locais temporários: eventos ao ar livre como shows, comícios políticos e também construções, podem usufruir da tecnologia das redes *mesh*, pelo fato de ser de fácil instalação e remoção (MEDEIROS, 2012).

3.2 Mesh Networking NS-3 Model

3.2.1 *Interoperabilidade*

O padrão 802.11s permite ao usuário implementar qualquer software de roteamento ou uma métrica de seleção de caminho para atender às necessidades da organização. Ao mesmo tempo, o padrão inclui um protocolo de roteamento padrão *HWMP* e uma métrica de seleção do caminho também padrão (ALM)*(Airtime Link Metric)* para assegurar a todos os dispositivos, uma interoperabilidade mínima, mesmo eles sendo de diferentes fornecedores. Apenas um protocolo de roteamento e apenas uma métrica de seleção de caminho deve ser usada por um *Mesh Point* por vez (FOUNDATION PLANETE GROUP, 2016).

Além dessas funções principais, padrão *IEEE* 802.11s possui uma série de recursos avançados disponíveis. Embora não sejam implementados no simulador (NS-3), este tópico irá expor quais maneiras de transmissão a arquitetura possui. A primeira e uma das coisas mais importantes é o *(Mesh Coordination Function)* que é um método de acesso opcional que permite a estação acesse o meio sem fio em horários previamente programados, de forma que a disputa normalmente não seria possível. Também a implementação do protocolo *Peering Management* não fornece suporte à segurança do *link* de malha (ou seja, transmissões de todos os quadros necessários para estabelecer uma ligação segura

entre um par do nó de malha não são implementadas). Esses protocolos são necessários para possibilitar uma comunicação entre nós de malha e uma estação externa a rede *Mesh*, através de portais e pontos de acesso da malha (ANDREEV, 2010).

3.3 Multihop MAC: Desvendando o Protocolo 802.11s

3.3.1 *Descoberta e Manutenção de Rotas*

Para a descoberta de rotas, os *MPs* utilizam dois pacotes, um de requisição de rota, o PREQ *(Path Request)* e outro de resposta à requisição da rota, o PREP *(Path Reply)*. Quando um nó fonte deseja transmitir dados para um destino, ele realiza uma inundação na rede com mensagens *PREQ* com informações que denunciam a intenção da fonte em estabelecer uma rota para determinado destino.

Quando um nó recebe um *PREQ*, se ele não for o destino requerido, ele checa na sua tabela de roteamento se há alguma rota válida para o destino. Se não houver, ele então repassa para os seus vizinhos, e este processo se repete até atingir o destino ou algum nó que conheça a rota até ele. A cada retransmissão, o campo contador de saltos é incrementado e os nós registram em uma entrada de suas tabelas de roteamento o endereço do vizinho de quem recebeu o primeiro *PREQ* e seu respectivo número de seqüência, além do seu tempo de expiração. Durante este procedimento, os nós intermediários aprendem o caminho reverso até a fonte e podem utilizar esta rota para transmitir dados. O aprendizado da rota reversa possibilita também que a resposta de requisição de rota possa ser transmitida em unicast diretamente para a fonte(FERNANDES, 2008).

Quando um nó detecta uma quebra de enlace, ou através da análise das informações da camada de enlace, ou pelo cessar do envio de mensagens *hello* por parte de algum vizinho, ele deverá avisar a todos os nós que se utilizam desta rota sobre a indisponibilidade da mesma. Como cada nó mantém em sua tabela de roteamento uma lista de todos os nós predecessores de cada rota, quando detectada a quebra de algum enlace, o nó detector sabe os nós que utilizam este enlace no encaminhamento de seus dados e são eles que devem ser notificados. Esta mensagem é geralmente enviada em *broadcast*, porém se apenas um nó for atingido pela quebra de algum enlace, o nó que detectou pode enviar o PERR *(Path Error)* em *unicast* para o nó prejudicado.

Assim que os nós receberem o *PERR*, a rota para o destino dito como inalcançável é apagada da tabela de roteamento e um novo processo de descobrimento de rota é iniciado.

3.4 Conclusões do capítulo

Com base na análise dos trabalhos relacionados, é possível concluir que pelo fato do protocolo *HWMP* ser recente, ainda há bastante implementação para agregar a ele. Mesmo assim, seu uso não é de forma alguma dispensável, vendo que existem inúmeras

situações em que este tipo de arquitetura é considerada viável e até mesmo auxiliando órgãos como hospitais e movimentos sociais. As vantagens dessa tecnologia fazem da rede *mesh* a solução mais adequada quando é necessária a implantação de uma rede que traga maior mobilidade, economia, menor manutenção e fácil expansão. Seus protocolos de roteamento mantêm a rede em constante atualização, fazendo com que as melhores rotas sejam descobertas, tornando assim uma rede flexível a possíveis mudanças (REZENDE, 2014).

Enfim, a tecnologia *mesh*, é a solução para expansão e provimento da Internet a população que vive em lugares isolados. Na cidade, é a forma mais rápida e econômica de expandir redes em empresas, universidades, escolas, hospitais, podendo transformar cidades em "cidades digitais", trazendo assim a informação onde e na hora em que necessitamos.

CAPÍTULO 4

Metodologia

4.1 Recursos Utilizados

Esta seção descreve os materiais e métodos utilizados para o desenvolvimento do trabalho e da proposta de implantar uma topologia *Mesh* na cidade de Castanhal. Esta arquitetura tem como finalidade a conexão a internet do máximo de dispositivos possíveis através de conexão sem fio barata e sem limitações geográficas. Nesse sentido, a viabilidade do projeto consiste em que a cidade de Castanhal tem muitos pontos onde não há conexão a internet, logo, com esta proposta, teríamos internet móvel como uma malha em diversos pontos da cidade.

4.2 Implantação

Para atingir os objetivos do projeto, utilizaremos as técnicas de simulação da área de abrangência da rede, bem como dos nós, da região em que efetuaremos a implantação da arquitetura. Isso seria feito através da ferramenta de simulação discreto de redes, NS-3 *(Network Simulator 3)*. As simulações foram realizadas utilizando a linguagem de programação C++, com bibliotecas e softwares de captura de tráfego e dos pacotes que seriam transmitidos na rede, bem como tornar possível a coleta dos dados gerados pela própria rede simulada, bem como velocidade do fluxo, perda, atraso, etc.

Foram utilizados nas simulações, o total de até 35 nós, organizados de forma manual e aleatória pelo cenário e que transmitem informações entre si. Cada nó se comunica com os outros 34, pelo protocolo de roteamento próprio da rede mesh *(HWMP)*, bem como, também foram utilizados outros dois protocolos de roteamento reconhecidos pelo simulador: AODV *(Ad-Hoc On-Demand Distance Vector)* e OLSR *(Optimized Link*

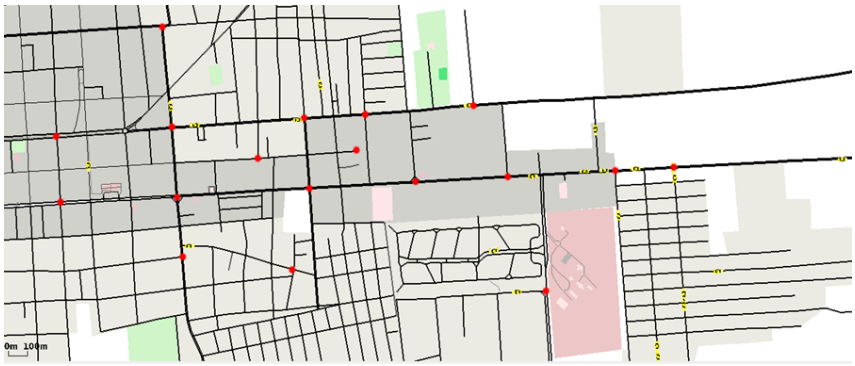
State Routing); Com o objetivo de ter uma comparação dos resultados e analisar a diferença de desempenho dos protocolos. Os pacotes têm um tamanho padrão de 1024 *bytes* e foi definida uma *range* de IPs para os nós de 10.0.0.1 até 10.0.0.35. Este simulador, é bastante utilizado pela comunidade acadêmica, para diversas finalidades no estudo de Redes de Computadores. Ele é bastante complexo desde a sua instalação porém abrange inúmeros conceitos da área, além de poder simular qualquer cenário que se possa imaginar de tráfego de dados e organização de dispositivos em uma rede, bem como protocolos, topologias, fluxo de dados, vazão, atraso, *jitter* e outras métricas.

Além do próprio simulador que foi utilizado, fez se também o uso de ferramentas tanto de análise do tráfego dos nós que eram criados no código fonte, bem como de ferramentas para plotagem dos gráficos gerados através das informações coletadas na simulação. Utilizamos o NetAnim (do inglês *Network Animator*), um animador de rede que ilustra graficamente como eram dispostos os nós no cenário, bem como as transmissões eram feitas e fluxo de cada uma. Outro *software* utilizado foi o Gnuplot, uma ferramenta de projeção de gráficos, que foi utilizada afim de comparar o desempenho das topologias utilizadas, protocolos, quantidade de nós em cada cenário e tempo de simulação. Também fizemos uso do SUMO (Simulador de Mobilidade Urbana), para criar as rotas da cidade de Castanhal em que os nós ficariam alocados e ter uma avaliação real do desempenho *(Vazão e Atraso)* desse tráfego entre os *MPs*.

4.2.1 *Construção do Cenário*

Para representar fielmente o desempenho do cenário na cidade de Castanhal, os nós foram organizados através da ferramenta *SUMO*, de acordo com um trecho da cidade. Estrategicamente, os nós ficaram posicionados de forma que um sempre esteja na área de cobertura de seus vizinhos, para que a transmissão seja a mais direta possível e não haja nenhum buraco na malha.

Figura 4: Representação da Rede em Malha no cenário simulado, através do *SUMO*.



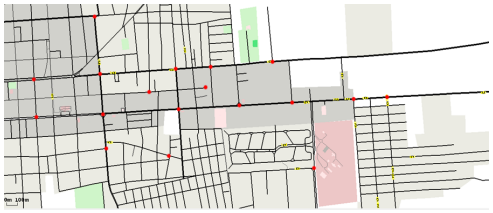
Fonte: Autor, 2017

De acordo com a Figura 4, foi desenhada no simulador o caminho feito para chegar na UFPA - Campus Castanhal, pela BR 316. Qualquer dispositivo com suporte

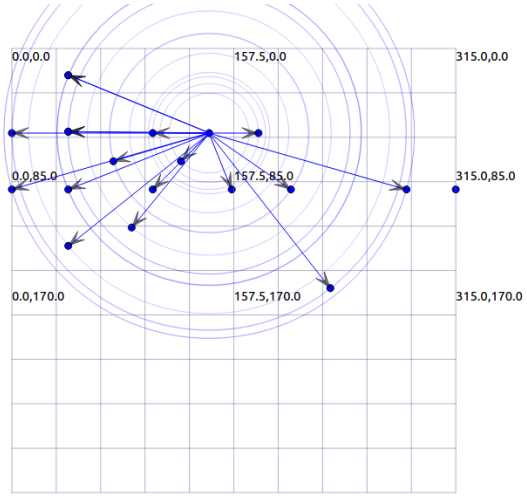
a *Wi-Fi*, ao entrar na malha, já terá acesso a internet, não precisando ficar parado no alcance de apenas 1 nó que esteja conectado a internet, mas sim em todo o decorrer da malha. A representação feita no SUMO, serviu como base para as simulações futuras, que serão mostradas ao longo deste tópico e no capítulo seguinte.

Figura 5: Representação da Alocação dos Nós.

(a) Redes Mesh - SUMO



(b) Redes Mesh - NetAnim



Fonte: Autor, 2017

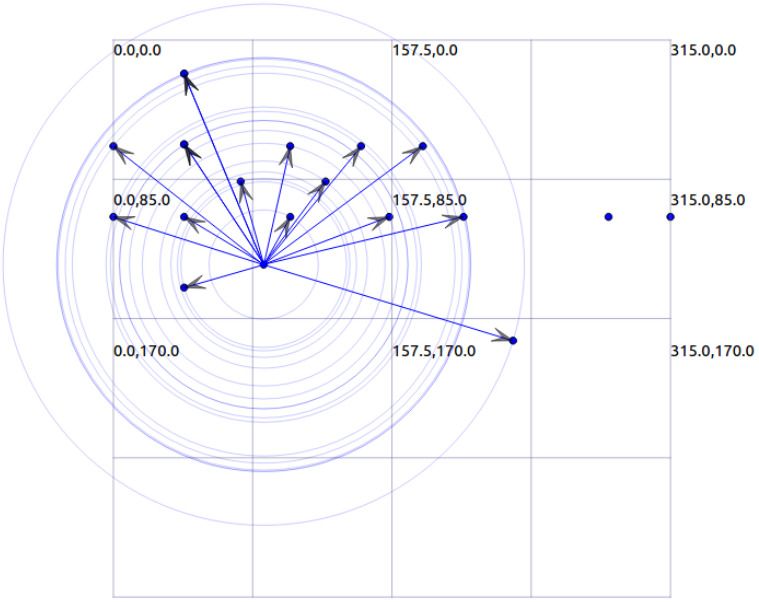
A alocação dos nós no arquivo das Redes *Mesh* no *NS-3* vem por padrão em forma de matriz (*Grid*). Foi necessário fazer a alteração dessa forma de alocação através da classe *PositionAllocator* para que fosse possível alocar os nós da forma desejada, ou seja, no modo estratégico como foi mostrado acima.

Será possível analisar, no *Capítulo 5*, que a alocação dos nós, bem como tamanho dos pacotes que são transmitidos no cenário, protocolo utilizado, largura de banda, entre outros fatores, influenciam fortemente para o desempenho da rede. Para que se tenha uma transmissão considerada aceitável no cenário simulado, é preciso levar em consideração todos estes fatores e aplicar tratativas para cada um deles.

Como foi abordado no *Capítulo 3*, que a descoberta de rotas é realizada enviando um pacote de reconhecimento *PREQ ou RANN* para todos os nós da rede e com a resposta deles é traçada uma rota de transmissão. É possível visualizar esse envio de requisição de caminho sendo realizado bem como o nó origem da requisição tendo conhecimento dos caminhos e as rotas sendo traçadas no simulador, na *Figura 06*.

Os avanços alcançados com os estudos direcionados a redes *Mesh*, fez com que fossem realizadas inúmeras simulações em diferentes cenários de possíveis regiões e organizações dos nós, bem como alteração na taxa de transmissão e tamanho dos pacotes utilizados (ANDREEV, 2010). Nesta etapa final da simulação, o código manipulado foi o código padrão de redes *Mesh (mesh.cc)* encontrado no próprio simulador. As demais ferramentas foram incorporadas de forma manual neste código, bem como as bibliote-

Figura 6: Representação da Descoberta de Rotas do Cenário Simulado.



Fonte: Autor, 2017

cas e demais parâmetros utilizados. Um código que também deu suporte neste trabalho e também pode ser encontrado nos diretórios do simulador foi o *("hwmp.cc")*, código este que é responsável por gerenciar o protocolo de roteamento próprio de redes *Mesh*, e fez se necessário alterá-lo para a implantação de outros protocolos, acréscimo de nós, reorganização e acréscimo de métricas entre os *MPs*.

Figura 7: Ilustração dos dados sobre os nós, utilizados para a construção da simulação, bem como seus parâmetros.

ID dos Nós	**IP dos Nós**	**Protocolo de Roteamento do Nó**	Tamanho dos Pacotes Transmitidos	Total De Nós
00	10.0.0.1	HWMP	1024 bytes	35
01	10.0.0.2	HWMP	1024 bytes	
02	10.0.0.3	HWMP	1024 bytes	
03	10.0.0.4	HWMP	1024 bytes	
04	10.0.0.5	HWMP	1024 bytes	
05	10.0.0.35	HWMP	1024 bytes	

Tempo de Simulação	**Tecnologia Utilizada**	**Tipo de Aplicação**
100 segundos	Nós Wireless Enlace sem Fio Topologia Mesh	UDP

Fonte: Autor, 2017

4.3 Conclusões do Capítulo

Este trabalho foi iniciado com um amplo levantamento bibliográfico das publicações que abordam a tecnologia de redes móveis sem fio *Ad-Hoc* e foram estudados, em cada publicação, os protocolos discutidos e o tipo de estudo realizado a respeito. A partir desse levantamento foi estabelecido um processo de classificação que permitisse discutir cada um dos principais protocolos e estabelecer o conjunto de atributos e métricas a serem consideradas na elaboração da síntese contida na tabela de síntese comparativa dos protocolos, organização dos nós e na eficiência em descobrir a melhor rota de transmissão.

Nesta proposta, foi discutido o compromisso que é assumido ao se implantar uma arquitetura sem fio. As Redes *Mesh* podem ser utilizadas para prover ganhos de acessibilidade de transmissão e/ou recepção ou para prover ganhos de mobilidade ao estabelecer uma comunicação entre dispositivos não importando seus sistemas operacionais, requerendo apenas que estejam sobre a malha da rede.

Esta seção descreveu a metologia para se implantar a topologia na Cidade de Castanhal. Assim também como suas características, operações, parâmetros utilizados nos elementos da arquitetura bem como a funcionalidade das transmissões.

CAPÍTULO 5

Resultados

Este capítulo é responsável por detalhar como foi realizada as simulações da arquitetura no *NS-3*, bem como descrever a avaliação de desemepenho do protocolo *HWMP* em relaçao a outros protocolos de rede disponíveis no simulador utilizado. Para isto, será apresentado os critérios utilizado nas simulações em relação a quantidade de nós, tamanho dos pacotes, controle de fluxo e estabilidade da transmissão.

A avaliação de desempenho da rede é feita em termos da Vazão (*Throughput*), do Atraso (*Delay*) dos pacotes e do índice de perda de pacotes (*Packet Lost*) durante o tempo proposto da simulação. Por vazão efetiva, entende-se o tráfego total transmitido com sucesso, e por atraso, entende-se o atraso total dos pacotes transmitidos com sucesso. O índice de perda de pacotes é medido através de um monitor incorporado ao simulador: (*FlowMonitor*), que coleta e organiza as métricas geradas com base nos parâmetros utilizados para os elementos da simulação. Entende-se por Índice de perda de pacotes, os pacotes que não conseguem chegar ao seu destino por excesso de saltos ou pela tomada de decisão do nó remetente daquele pacote, não ter sido a mais adequada e aquele pacote terminar se perdendo no caminho e ter que ser retransmitido.

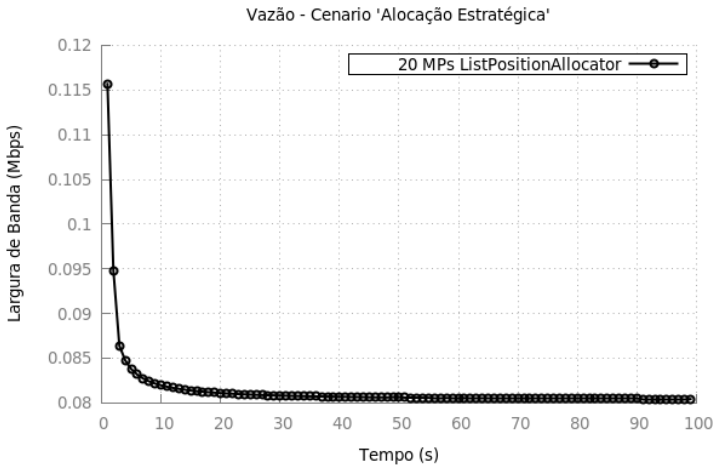
5.1 Comparativo: Cenário Simulado vs Cenário Aleatório

Dado o interesse em investigar o impacto da implantação das Redes *Mesh* no cenário simulado referente a vazão global da rede, focamos nossas simulações em redes *Ad-Hoc* sem fio estáticas e com transmissões simultâneas.

Na Figura 8, foram consideradas topologias de 20 nós, manualmente posicionados em um terreno plano de 1000 por 1000 metros, de forma que cada nó atua tanto como transmissor quanto como receptor, permitindo a existência de 20 pares fonte-destino si-

multâneos. A Avaliação dos Resultados tem o objetivo de provar que essa organização estratégica dos nós, faz com que a rede tenha um desempenho melhor do que se os nós fossem alocados de forma randômica pelo cenário. Tanto em questão de vazão como de atraso da transmissão dos nós, serão comparados os dois cenários e as métricas geradas por eles.

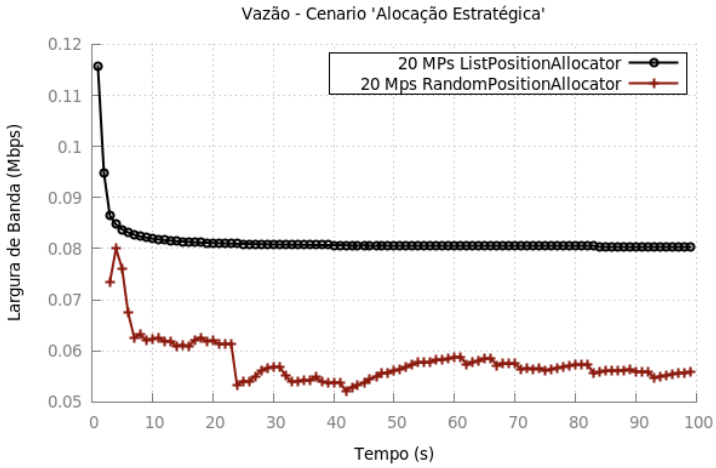
Figura 8: Taxa de Vazão dos nós organizados no Cenário Simulado.



Fonte: Autor, 2017

Na Figura 9, é mostrado a diferença da taxa de vazão projetada pelo cenário simulado acima e um cenário ilustrativo, onde os nós são gerados em uma mesma área de 1000 por 1000 metros, porém a organização dos nós é aleatória, ou seja, gerenciada pelo próprio simulador através da classe *RandomPositionAllocator*.

Figura 9: Comparativo da taxa de Vazão entre os Cenários.



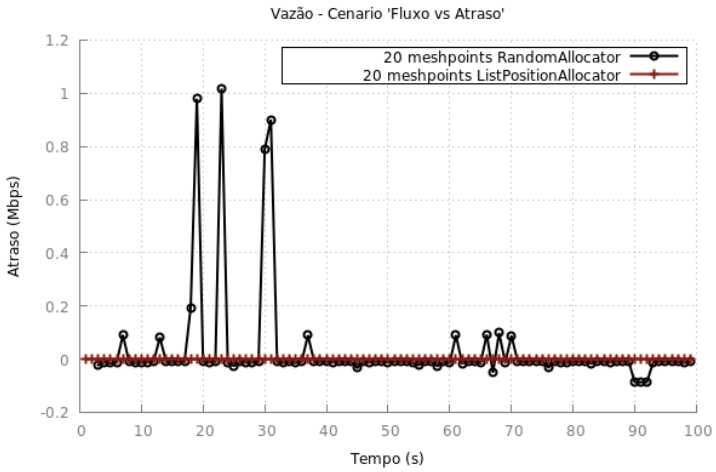
Fonte: Autor, 2017

A taxa de transmissão do cenário simulado na Figura 8, em relação ao cenário aleatório que é mostrado também na Figura 9, é maior e mais elevada, pois o simulador

não leva em consideração a distância de um nó para outro no momento da alocação dos mesmos, ou seja, ele não procura posicionar os nós de forma que a malha tenha uma maior área de cobertura, influenciando assim no desempenho de toda a rede, critério este que é aplicado no Cenário Simulado.

A métrica de Vazão é de vital importância em uma arquitetura de rede, e foi mostrado que tendo um cenário simulador com uma organização manual dos nós, o rendimento de transmissão é mais efetivo, porém, é necessário que o fluxo da transmissão seja estável, sem atrasos nem picos de perdas de pacotes. Por isso, é realizada também a avaliação de desempenho da métrica *Delay* nos dois cenários, fazendo um comparativo para fidelizar ainda mais a viabilidade da proposta.

Figura 10: Comparativo da taxa de Atraso entre os Cenários.



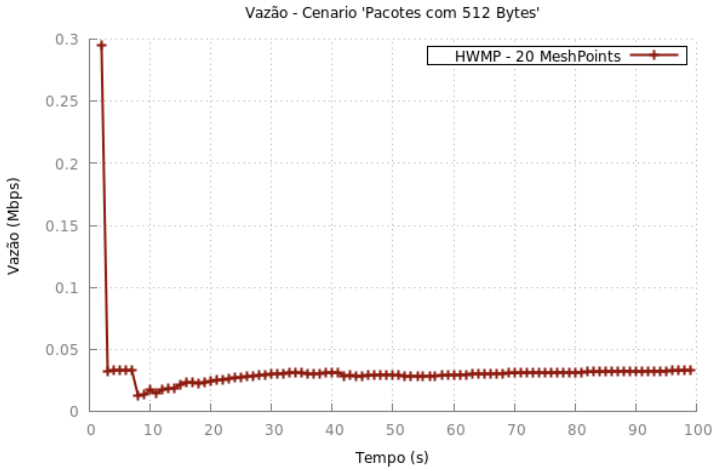
Fonte: Autor, 2017

Na Figura 10, é mostrado a taxa de Atraso dos dois cenários, para comparação. Nota-se que alocando randomicamente os MPs têm-se picos de atraso de transmissão. Isso acontece por que como suas posições não podem ser controladas para que tenham um fluxo de transmissão constante, o atraso e a perda de pacotes é bem alta, se comparado com o Cenário Simulado, onde tem-se a alocação estratégica dos nós e uma transmissão contínua, com a taxa de atraso bem próxima de zero.

5.2 Avaliação de Desempenho

Como fora estudado anteriormente, as redes *Mesh* são similares as outras formas de comunicação sem fio, porém o seu diferencial encontra-se na organização dos nós, onde cada nó pode ser um roteador, logo, um fato que implica diretamente no desempenho da rede, além da localização de cada nó no cenário, também é o tamanho dos pacotes que são transmitidos na rede. Em Figura 11, será projetado um gráfico com a vazão no cenário, porém com pacotes reduzidos a metade do seu tamanho normal, ou seja, 512 *Bytes*.

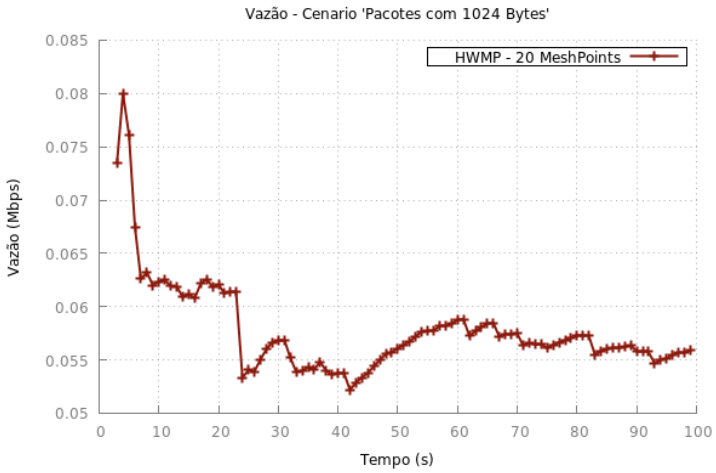
Figura 11: Taxa de Vazão do algoritmo *mesh.cc*, com pacotes de 512 Bytes.



Fonte: Autor, 2017

Os mesmos parâmetros utilizados na simulação ilustrada pela Figura 12, foram repetidos e simulados novamente, porém, com o tamanho dos pacotes normalizado, cada um com 1024 *Bytes*, para ter-se novamente uma mudança no desempenho e fazer a comparação.

Figura 12: Taxa de Vazão do algoritmo *mesh.cc*, com pacotes de 1024 Bytes.

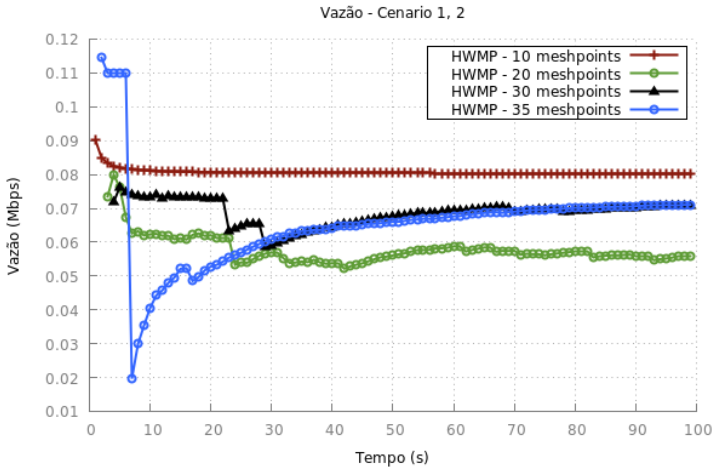


Fonte: Autor, 2017

O segundo gráfico, da Figura 12, teve o desempenho maior do que o cenário com nós de 512 *Bytes*, Figura 11, pois seus MPs tiveram um aproveitamento melhor da rede, por estarem no seu tamanho padrão, assim, puderam transmitir o dobro de dados em relação ao primeiro cenário. Isso é explicado pelo fato do protocolo HWMP ter o aproveitamento máximo da rede, não tendo a necessidade de diminuir o tamanho de seus pacotes para ter um melhor rendimento.

Outra métrica que também influencia diretamente no desempenho da rede, além do tamanho dos pacotes, é o número de nós que estão sendo distribuídos no cenário. Dependendo do protocolo de roteamento e da distância desses nós um do outro, o desempenho da rede pode cair ou aumentar. Observe na Figura 13, em que utilizaremos o próprio protocolo da rede *mesh* (HWMP), porém variamos a quantidade de nós para percebermos a diferença no fluxo de dados no cenário.

Figura 13: Taxas de Vazão comparando a quantidade de nós.



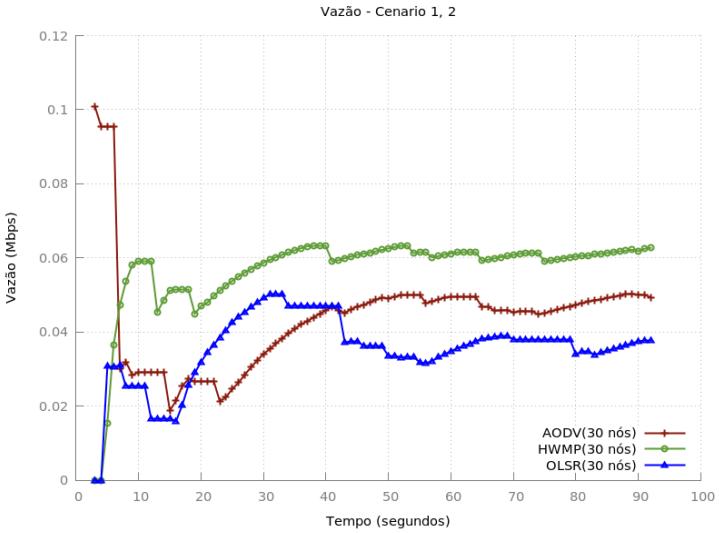
Fonte: Autor, 2017

Na Figura 13, é projetada uma comparação entre dois fatores que foram determinantes nos cenários que foram desenhados no simulador, para um desempenho aceitável da rede. O protocolo de roteamento utilizado e a quantidade de nós alocada no cenário. Na Figura 13, é mostrado que a com quantidades diferentes de nós a rede tem comportamentos diferentes em relação a vazão. Isso acontece por que cada nó é um concorrente com os demais em relação a largura de banda e vez de transmissão, logo quanto mais nós estão presentes e ativos no cenário, pode-se ter uma instabilidade maior no período inicial da simulação.

Ao decorrer da simulação, pode-se observar que o desempenho começa a se comportar de forma diferente, isto acontece porque fatores como a quantidade de nós, também interferem na eficiência da transmissão. Na Figura 14, explanamos também outro fator que também interfere bastante no desempenho de uma rede. Trata-se de protocolos de roteamento, que tem eficiência inconstante para cada situação. Se o protocolo tem tratativas de retansmissão de pacotes, se ele é pró-ativo quanto a descoberta das rotas de seus vizinhos ou se ele tem algum mecanismo de tolerância a falhas no cenário, influencia no desempenho da minha rede. O fato do protocolo HWMP ter o conjunto das vantagens dos outros dois protocolos que lhe foram comparados, explica o seu desempenho no gráfico.

Como foi mencionado nos tópicos e capítulos anteriores, o Atraso (*Delay*) é crucial para o desempenho de uma rede. Ter a taxa de atraso sob controle é vital para que se tenha uma transmissão adequada entre os nós, principalmente em arquiteturas sem fio,

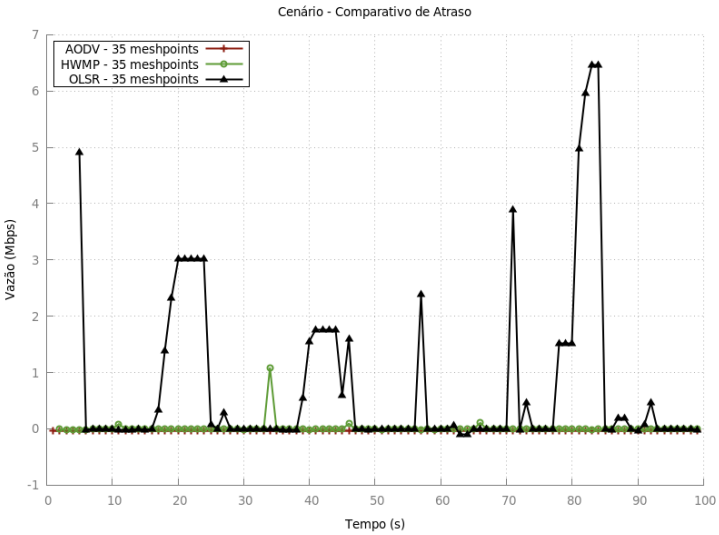
Figura 14: Taxa de Vazão do algoritmo *mesh.cc*: Comparativo dos protocolos de Roteamento.



Fonte: Autor, 2017

onde já se temos atraso e perda de pacotes naturalmente, por questões de interferências de outras redes, equipamentos, etc. É de suma importância que se tenha uma avaliação de desempenho voltada para a métrica de atraso exclusivamente em relação aos protocolos de roteamento, que é mostrada na Figura 15.

Figura 15: Taxa de Atraso do algoritmo *mesh.cc*: Comparativo dos protocolos de Roteamento.



Fonte: Autor, 2017

5.3 Conclusões do capítulo

Fazendo uma rápida análise nos gráficos, é notório a diferença no comportamento das transmissões, isso acontece pois apesar da quantidade dos nós ser a mesma nos cenários, os protocolos se comportam de forma diferente, tem critérios diferentes e por conta disso, fazem tomadas de decisões diferentes. A ideia de "melhor caminho"muda de protocolo para protocolo, um exemplo disso é que no *AODV* , sempre o melhor caminho vai ser aquele que mais fizer uma requisição, visto que ele funciona sob-demanda. Já no *OLSR*, o caminho que tiver o link disponível no momento da requisição vai ser o escolhido, não importando quanto saltos ele tenha que realizar até o destino, fazendo com que a sua escolha nem sempre seja a mais adequada. Como o *HWMP* combina o melhor dos dois protocolos e tenta suprir suas deficiências, isso explica o por que dele ter o desempenho mais aceitável na Figura 14.

Relacionado ao tamanho dos pacotes, o cenário onde os pacotes tem o tamanho maior *1024 bytes* tem o desempenho mais efetivo pois ele aproveita melhor a largura de banda disponível para enviar mais informações do que seu cenário concorrente. Ou seja, o protocolo *HWMP* utiliza a capacidade máxima de transmissão de cada pacote em sua malha, não tendo problemas com (MTU) (*Maximum Transmit Unit*) durante o caminho dos pacotes para ser preciso redimensioná-los depois.

CAPÍTULO 6

Conclusões

Levando em consideração que essas redes de longo alcance e ótimo custo-benefício são consideravelmente recentes, os estudos sobre elas são cada vez mais valorizados, bem como, é comum procurarmos ajuda em fóruns de dúvida sobre as tecnologias, por serem recentes, não há muito material sobre as mesmas, porém, a evolução do trabalho é constante. Após a análise das simulações, observou-se que graficamente os algoritmos se comportam de forma similar, apesar da mudança do tamanho máximo dos pacotes, porém, se formos olhar os números/métricas que são geradas pelo monitor, vemos que a diferença de comportamento entre eles é significativa, tanto em tempo de simulação quanto em desempenho.

Dispositivos de comunicação portáteis tais como PDAs, telefones celulares, iPods, entre outros encontram-se atualmente presentes e amplamente difundidos na sociedade mundial. O potencial de comunicação sem fio destes dispositivos será em um futuro próximo responsável por um grande volume de tráfego de dados nas redes metropolitanas (PEREIRA, 2009). Em termos de desempenho, ressalta-se ainda que informações do subsistema de rádio estão naturalmente disponíveis à camada 2. Tais informações são de extrema relevância para redes sem fio onde o canal de comunicação é sujeito a interferências complexas e variações freqüentes. Métricas de qualidade de enlace precisas são vitais para o pleno funcionamento de algoritmos de escolha de melhor caminho. Uma grande dificuldade de protocolos de roteamento de camada 3 é a estimativa acurada da qualidade dos enlaces. Por essas razões principais, iniciativas de padronização e implementação de redes sem fio com comunicação de múltiplos saltos no nível de enlace, como a proposta do *IEEE* 802.11s que foi apresentada neste trabalho, são de extremo valor e têm potencial para uso prático largamente difundido.

O padrão ainda é imaturo em alguns pontos, e precisa de melhorias. O uso de múltiplos saltos na rede *Mesh* apresentou uma queda no rendimento da rede. Verificou-

se que a comunicação com até dois nós intermediários é possível, mas a redução na vazão obtida e o aumento das perdas de quadros e de atrasos fim-a-fim foram significativos(AKYILDIZ I. F.; WANG, 2005). Outro ponto a ser verificado é o uso dos MPP, para comunicação com a rede cabeada e funcionando como gargalo da rede, o qual teve um melhor aproveitamento, quando as estaçõe estavam próximas no cenário (FOUNDATION PLANETE GROUP, 2016). Quando as estações estão distantes e não há um MP entre elas, ocorre perda na comunicacão e ambos cenários possuem mudança nos enlaces de comunicação constante. Estas mudanças de enlace, são devido a diminuição da taxa de bits da estação, fazendo o valor da métrica aumentar, e a estação assim procurar um novo enlace. Esses efeitos sugerem que a implantação de uma rede Mesh depende de uma cuidadosa e estratégica disposição dos dispositivos, para que os enlaces possuam qualidade aceitável.

Por fim, vislumbra-se que um estudo mais aprofundado sobre Redes *Mesh* e uma utilização da tecnologia no mundo real poderia atender aplicações com dispositivos que geram pouca ou média carga de tráfego, tais como automação residencial ou redes de sensores. Outro possível uso seria interligar pontos de acesso via canais sem-fio.

Referências

AKYILDIZ I. F., X. W.; WANG, W. Wireless mesh networks: a survey. *Computer Networks*, Akyldiz, v. 47, n. 4, p. 445—487, 2005. Citado 2 vezes nas páginas 8 e 31.

ALBUQUERQUE, R. M. G. D. *Routing Metrics and Protocols for Wireless Mesh Networks*. [S.l.]: IEEE Network, 2011. 6-12 p. Citado na página 12.

ANDREEV, P. B. K. *Mesh Networking NS-3 Model*. [S.l.]: IEEE, 2010. Citado 5 vezes nas páginas 11, 12, 13, 16 e 20.

BAUER, M. *Redes sem Fio Mesh: Padrão 802.11s*. [S.l.]: IFSC-Campus São José, 2012. Citado 2 vezes nas páginas 2 e 8.

FERNANDES, C. *Desvendando o Protocolo 802.11s*. [S.l.]: IME-USP, 2008. Citado 3 vezes nas páginas 9, 10 e 16.

FOUNDATION PLANETE GROUP, I. S. A. N. S. *NS-3 Documentation*. NS-3 Network Simulator, 2016. Disponível em: ⟨http://www.nsnam.org/.⟩ Citado 2 vezes nas páginas 15 e 31.

GOLDSMITH, A. *Wireless Communications*. [S.l.]: Cambridge University Press, 2015. Citado 2 vezes nas páginas 1 e 7.

JUNG, K.-W. L. J.-S. *Wireless Mesh Networks for Reliable Routing in the Smart Grid Infrastructure*. [S.l.]: IEEE, 2011. Citado 2 vezes nas páginas 2 e 6.

MEDEIROS, P. C. F. T. *Redes Mesh: Topologia e Aplicação*. [S.l.]: Revista iTEC, 2012. Citado 2 vezes nas páginas 10 e 15.

OLIVEIRA, P. A. *Redes em Malha sem Fio*. [S.l.]: IME-USP, 2012. Citado na página 3.

PEREIRA, N. J. *Redes Mesh Sem Fios*. [S.l.]: Faculdade de Engenharia do Porto, 2009. Citado 2 vezes nas páginas 9 e 30.

REZENDE, N. S. de. *Redes Móveis Sem Fio Ad-Hoc*. [S.l.]: Núcleo de Computação Eletrônica - UFRJ, 2014. Citado 3 vezes nas páginas 6, 7 e 17.

TEIXEIRA, E. R. D. *Wireless Mesh Networks*. [S.l.]: http://http://www.telecom.com.br/tutoriais/tutorialwmn, 2011. Citado na página 7.

ANEXO A – Script Base de Redes Mesh

```
1
2  #include "ns3/core-module.h"
3  #include "ns3/internet-module.h"
4  #include "ns3/network-module.h"
5  #include "ns3/applications-module.h"
6  #include "ns3/wifi-module.h"
7  #include "ns3/mesh-module.h"
8  #include "ns3/mobility-module.h"
9  #include "ns3/mesh-helper.h"
10
11 #include <iostream>
12 #include <sstream>
13 #include <fstream>
14
15 using namespace ns3;
16
17 NS_LOG_COMPONENT_DEFINE ("TestMeshScript");
18
19
20
21 class MeshTest
22 {
23 public:
24
25   MeshTest ();
26
27
28   void Configure (int argc, char ** argv);
29
30
31   int Run ();
32 private:
33   int        m_xSize;
34   int        m_ySize;
35   double     m_step;
36   double     m_randomStart;
```

```
37    double      m_totalTime;
38    double      m_packetInterval;
39    uint16_t    m_packetSize;
40    uint32_t    m_nIfaces;
41    bool        m_chan;
42    bool        m_pcap;
43    std::string m_stack;
44    std::string m_root;
45
46    NodeContainer nodes;
47
48    NetDeviceContainer meshDevices;
49
50    Ipv4InterfaceContainer interfaces;
51
52    MeshHelper mesh;
53
54  private:
55
56    void CreateNodes ();
57
58    void InstallInternetStack ();
59
60    void InstallApplication ();
61
62    void Report ();
63  };
64  MeshTest::MeshTest () :
65
66    m_xSize (3),
67    m_ySize (3),
68    m_step (100.0),
69    m_randomStart (0.1),
70    m_totalTime (100.0),
71    m_packetInterval (0.1),
72    m_packetSize (1024),
73    m_nIfaces (1),
74    m_chan (true),
75    m_pcap (false),
76    m_stack ("ns3::Dot11sStack"),
77    m_root ("ff:ff:ff:ff:ff:ff")
78  {
79  }
80
81  void
82  MeshTest::Configure (int argc, char *argv[])
```

```
83
84 {
85   CommandLine cmd;
86   cmd.AddValue ("x-size", "Number of nodes in a row grid",
       m_xSize);
87   cmd.AddValue ("y-size", "Number of rows in a grid", m_ySize);
88   cmd.AddValue ("step",   "Size of edge in our grid (meters)",
       m_step);
89   cmd.AddValue ("start",  "Maximum random start delay for beacon
       jitter (sec)", m_randomStart);
90   cmd.AddValue ("time",  "Simulation time (sec)", m_totalTime);
91   cmd.AddValue ("packet-interval",  "Interval between packets in
       UDP ping (sec)", m_packetInterval);
92   cmd.AddValue ("packet-size",  "Size of packets in UDP ping (
       bytes)", m_packetSize);
93   cmd.AddValue ("interfaces", "Number of radio interfaces used by
        each mesh point", m_nIfaces);
94   cmd.AddValue ("channels",   "Use different frequency channels
       for different interfaces", m_chan);
95   cmd.AddValue ("pcap",   "Enable PCAP traces on interfaces",
       m_pcap);
96   cmd.AddValue ("stack",  "Type of protocol stack. ns3::
       Dot11sStack by default", m_stack);
97   cmd.AddValue ("root", "Mac address of root mesh point in HWMP",
        m_root);
98
99   cmd.Parse (argc, argv);
100  NS_LOG_DEBUG ("Grid:" << m_xSize << "*" << m_ySize);
101  NS_LOG_DEBUG ("Simulation time: " << m_totalTime << " s");
102
103 }
104
105 void
106 MeshTest::CreateNodes ()
107 {
108
109   nodes.Create (m_ySize*m_xSize);
110
111   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
112   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::
       Default ();
113   wifiPhy.SetChannel (wifiChannel.Create ());
114
115   mesh = MeshHelper::Default ();
116   if (!Mac48Address (m_root.c_str ()).IsBroadcast ())
117     {
```

```
118       mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue
              (Mac48Address (m_root.c_str ())));
119     }
120   else
121     {
122
123       mesh.SetStackInstaller (m_stack);
124     }
125   if (m_chan)
126     {
127       mesh.SetSpreadInterfaceChannels (MeshHelper::
              SPREAD_CHANNELS);
128     }
129   else
130     {
131       mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
132     }
133   mesh.SetMacType ("RandomStart", TimeValue (Seconds (
         m_randomStart)));
134
135   mesh.SetNumberOfInterfaces (m_nIfaces);
136
137   meshDevices = mesh.Install (wifiPhy, nodes);
138
139   MobilityHelper mobility;
140   mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
141                                  "MinX", DoubleValue (0.0),
142                                  "MinY", DoubleValue (0.0),
143                                  "DeltaX", DoubleValue (m_step),
144                                  "DeltaY", DoubleValue (m_step),
145                                  "GridWidth", UintegerValue (
                                        m_xSize),
146                                  "LayoutType", StringValue ("
                                        RowFirst"));
147   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel"
         );
148   mobility.Install (nodes);
149   if (m_pcap)
150     wifiPhy.EnablePcapAll (std::string ("mp-"));
151 }
152 void
153 MeshTest::InstallInternetStack ()
154 {
155   InternetStackHelper internetStack;
156   internetStack.Install (nodes);
157   Ipv4AddressHelper address;
```

```
158   address.SetBase ("10.1.1.0", "255.255.255.0");
159   interfaces = address.Assign (meshDevices);
160 }
161 void
162 MeshTest::InstallApplication ()
163 {
164   UdpEchoServerHelper echoServer (9);
165   ApplicationContainer serverApps = echoServer.Install (nodes.Get
          (0));
166   serverApps.Start (Seconds (0.0));
167   serverApps.Stop (Seconds (m_totalTime));
168   UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);
169   echoClient.SetAttribute ("MaxPackets", UintegerValue ((uint32_t
        )(m_totalTime*(1/m_packetInterval))));
170   echoClient.SetAttribute ("Interval", TimeValue (Seconds (
        m_packetInterval)));
171   echoClient.SetAttribute ("PacketSize", UintegerValue (
        m_packetSize));
172   ApplicationContainer clientApps = echoClient.Install (nodes.Get
          (m_xSize*m_ySize-1));
173   clientApps.Start (Seconds (0.0));
174   clientApps.Stop (Seconds (m_totalTime));
175 }
176 int
177 MeshTest::Run ()
178 {
179   CreateNodes ();
180   InstallInternetStack ();
181   InstallApplication ();
182   Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report,
        this);
183   Simulator::Stop (Seconds (m_totalTime));
184   Simulator::Run ();
185   Simulator::Destroy ();
186   return 0;
187 }
188 void
189 MeshTest::Report ()
190 {
191   unsigned n (0);
192   for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i
        != meshDevices.End (); ++i, ++n)
193     {
194       std::ostringstream os;
195       os << "mp-report-" << n << ".xml";
196       std::cerr << "Printing mesh point device #" << n << "
```

```
          diagnostics to " << os.str () << "\n";
197       std::ofstream of;
198       of.open (os.str ().c_str ());
199       if (!of.is_open ())
200         {
201           std::cerr << "Error: Can't open file " << os.str () <<
                "\n";
202           return;
203         }
204       mesh.Report (*i, of);
205       of.close ();
206     }
207 }
208 int
209 main (int argc, char *argv[])
210 {
211   MeshTest t;
212   t.Configure (argc, argv);
213   return t.Run ();
214 }
215
216 }
```

ANEXO B – Script Principal de Implantação do Cenário Aleatório

```
1
2
3  #include "ns3/core-module.h"
4  #include "ns3/internet-module.h"
5  #include "ns3/network-module.h"
6  #include "ns3/applications-module.h"
7  #include "ns3/wifi-module.h"
8  #include "ns3/mesh-module.h"
9  #include "ns3/mobility-module.h"
10 #include "ns3/mesh-helper.h"
11 #include "ns3/netanim-module.h"
12 #include "ns3/gnuplot.h"
13 #include "ns3/flow-monitor-module.h"
14 #include <ns3/flow-monitor-helper.h>
15 #include "ns3/olsr-module.h"
16 #include "ns3/aodv-module.h"
17 #include <iostream>
18 #include <sstream>
19 #include <fstream>
20
21 using namespace ns3;
22
23 void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<
      FlowMonitor> flowMon,Gnuplot2dDataset DataSet);
24 void DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor>
      flowMon, Gnuplot2dDataset Dataset3);
25
26 NS_LOG_COMPONENT_DEFINE ("TestMeshScript");
27
28 class MeshTest
29 {
30 public:
31
32   MeshTest ();
```

```
33
34    void Configure (int argc, char ** argv);
35
36    int Run ();
37  private:
38    int        m_xSize;
39    int        m_ySize;
40    double     m_step;
41    double     m_randomStart;
42    double     m_totalTime;
43    double     m_packetInterval;
44    uint16_t   m_packetSize;
45    uint32_t   m_nIfaces;
46    bool       m_chan;
47    bool       m_pcap;
48    std::string m_stack;
49    std::string m_root;
50
51    NodeContainer nodes;
52
53    NetDeviceContainer meshDevices;
54
55    Ipv4InterfaceContainer interfaces;
56
57    MeshHelper mesh;
58  private:
59
60    void CreateNodes ();
61
62    void InstallInternetStack ();
63
64    void InstallApplication ();
65
66  void Report ();
67
68  };
69
70  MeshTest::MeshTest () :
71    m_xSize (3),
72    m_ySize (2),
73    m_step (100.0),
74    m_randomStart (0.1),
75    m_totalTime (100.0),
76    m_packetInterval (0.1),
77    m_packetSize (1024),
78    m_nIfaces (1),
```

```
79    m_chan (true),
80    m_pcap (false),
81    m_stack ("ns3::Dot11sStack"),
82    m_root ("ff:ff:ff:ff:ff:ff")
83  {
84  }
85  void
86  MeshTest::Configure (int argc, char *argv[])
87  {
88    CommandLine cmd;
89
90    cmd.AddValue ("x-size", "Number of nodes in a row grid. [6]",
        m_xSize);
91    cmd.AddValue ("y-size", "Number of rows in a grid. [1]",
        m_ySize);
92    cmd.AddValue ("step",   "Size of edge in our grid, meters.
        [1000 m]", m_step);
93
94
95    cmd.AddValue ("start",  "Maximum random start delay, seconds.
        [0.1 s]", m_randomStart);
96    cmd.AddValue ("time",  "Simulation time, seconds [100 s]",
        m_totalTime);
97    cmd.AddValue ("packet-interval",  "Interval between packets in
        UDP ping, seconds [0.001 s]", m_packetInterval);
98    cmd.AddValue ("packet-size",  "Size of packets in UDP ping",
        m_packetSize);
99  }
100 void
101 MeshTest::CreateNodes ()
102 {
103
104 nodes.Create (35);
105
106
107   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
108   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::
        Default ();
109   wifiPhy.SetChannel (wifiChannel.Create ());
110
111   mesh = MeshHelper::Default ();
112
113   if (!Mac48Address (m_root.c_str ()).IsBroadcast ())
114     {
115       mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue
          (Mac48Address (m_root.c_str ())));
```

```
116     }
117   else
118     {
119
120       mesh.SetStackInstaller (m_stack);
121     }
122   if (m_chan)
123     {
124       mesh.SetSpreadInterfaceChannels (MeshHelper::
             SPREAD_CHANNELS);
125     }
126   else
127     {
128       mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
129     }
130   mesh.SetMacType ("RandomStart", TimeValue (Seconds (
        m_randomStart)));
131
132   mesh.SetNumberOfInterfaces (m_nIfaces);
133
134   meshDevices = mesh.Install (wifiPhy, nodes);
135
136
137
138 MobilityHelper mobility;
139   mobility.SetPositionAllocator ("ns3::RandomBoxPositionAllocator
        ",
140                                  "X", StringValue("ns3::
                                     UniformRandomVariable[Min=0|
                                     Max=500]"),
141                                  "Y", StringValue("ns3::
                                     UniformRandomVariable[Min=0|
                                     Max=500]"),
142                                 "Z", StringValue("ns3::
                                    UniformRandomVariable[Min=0|Max
                                    =0]"));
143
144
145   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel"
        );
146   mobility.Install (nodes);
147
148
149 if (m_pcap)
150     wifiPhy.EnablePcapAll (std::string ("mp-"));
151 }
```

```
152 void
153 MeshTest::InstallInternetStack ()
154 {
155   InternetStackHelper internetStack;
156
157   internetStack.Install (nodes);
158   Ipv4AddressHelper address;
159   address.SetBase ("192.168.0.0", "255.255.255.0");
160   interfaces = address.Assign (meshDevices);
161 }
162 void
163 MeshTest::InstallApplication ()
164 {
165   UdpEchoServerHelper echoServer (9);
166   ApplicationContainer serverApps = echoServer.Install (nodes.Get
        (0));
167   serverApps.Start (Seconds (0.0));
168   serverApps.Stop (Seconds (m_totalTime));
169   UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);
170   echoClient.SetAttribute ("MaxPackets", UintegerValue ((uint32_t
        )(m_totalTime*(1/m_packetInterval))));
171   echoClient.SetAttribute ("Interval", TimeValue (Seconds (
        m_packetInterval)));
172   echoClient.SetAttribute ("PacketSize", UintegerValue (
        m_packetSize));
173   ApplicationContainer clientApps = echoClient.Install (nodes.Get
        (1));
174   clientApps.Start (Seconds (0.0));
175   clientApps.Stop (Seconds (m_totalTime));
176 }
177
178
179
180 int
181 MeshTest::Run ()
182 {
183   CreateNodes ();
184   InstallInternetStack ();
185   InstallApplication ();
186
187
188
189     std::string fileNameWithNoExtension = "FlowVSThroughput_";
190     std::string graphicsFileName        = fileNameWithNoExtension
          + ".png";
191     std::string plotFileName            = fileNameWithNoExtension
```

```
        + ".plt";
192     std::string plotTitle                = "Flow vs Throughput";
193     std::string dataTitle                = "Throughput";
194
195     Gnuplot gnuplot (graphicsFileName);
196     gnuplot.SetTitle (plotTitle);
197     gnuplot.SetTerminal ("png");
198     gnuplot.SetLegend ("Flow", "Throughput");
199
200
201    Gnuplot2dDataset dataset;
202    dataset.SetTitle (dataTitle);
203    dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);
204
205
206   FlowMonitorHelper fmHelper;
207   Ptr<FlowMonitor> allMon = fmHelper.InstallAll();
208
209   ThroughputMonitor(&fmHelper, allMon, dataset);
210
211
212
213   std::string fileNameWithNoExtension3 = "FlowVSDelay_";
214   std::string graphicsFileName3      = fileNameWithNoExtension3 +
        ".png";
215   std::string plotFileName3        = fileNameWithNoExtension3 + "
       .plt";
216   std::string plotTitle3       = "Flow vs Delay";
217   std::string dataTitle3       = "Delay";
218
219       Gnuplot gnuplot3 (graphicsFileName3);
220       gnuplot3.SetTitle(plotTitle3);
221
222       gnuplot3.SetTerminal("png");
223
224       gnuplot3.SetLegend("Flow", "Delay");
225
226       Gnuplot2dDataset dataset3;
227       dataset3.SetTitle(dataTitle3);
228       dataset3.SetStyle(Gnuplot2dDataset::LINES_POINTS);
229
230       DelayMonitor(&fmHelper, allMon, dataset3);
231
232
233
234   Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report,
```

```
      this);
235   Simulator::Stop (Seconds (m_totalTime));
236
237 uint64_t maximo = 10000000000000000;
238         AnimationInterface anim ("meshteste.xml");
239         anim.SetMaxPktsPerTraceFile(maximo);
240
241
242   Simulator::Run ();
243
244   gnuplot.AddDataset (dataset);
245   std::ofstream plotFile (plotFileName.c_str());
246   gnuplot.GenerateOutput (plotFile);
247   plotFile.close ();
248
249
250   gnuplot3.AddDataset(dataset3);;
251   std::ofstream plotFile3 (plotFileName3.c_str());
252   gnuplot3.GenerateOutput(plotFile3);
253   plotFile3.close();
254
255
256
257   Simulator::Destroy ();
258   return 0;
259 }
260
261 void
262 MeshTest::Report ()
263 {
264   unsigned n (0);
265   for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i
        != meshDevices.End (); ++i, ++n)
266     {
267       std::ostringstream os;
268       os << "mp-report-" << n << ".xml";
269       std::cerr << "Printing mesh point device #" << n << "
           diagnostics to " << os.str () << "\n";
270       std::ofstream of;
271       of.open (os.str ().c_str ());
272       if (!of.is_open ())
273         {
274           std::cerr << "Error: Can't open file " << os.str () <<
              "\n";
275           return;
276         }
```

```
277       mesh.Report (*i, of);
278       of.close ();
279     }
280 }
281 int
282 main (int argc, char *argv[])
283 {
284   MeshTest t;
285   t.Configure (argc, argv);
286   return t.Run ();
287 }
288
289 void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<
    FlowMonitor> flowMon,Gnuplot2dDataset DataSet)
290         {
291         double localThrou=0;
292                 std::map<FlowId, FlowMonitor::FlowStats>
                    flowStats = flowMon->GetFlowStats();
293                 Ptr<Ipv4FlowClassifier> classing = DynamicCast<
                    Ipv4FlowClassifier> (fmhelper->GetClassifier())
                    ;
294                 for (std::map<FlowId, FlowMonitor::FlowStats>::
                    const_iterator stats = flowStats.begin ();
                    stats != flowStats.end (); ++stats)
295                 {
296                       if(stats->first == 2){
297                         Ipv4FlowClassifier::FiveTuple fiveTuple =
                             classing->FindFlow (stats->first);
298                         std::cout<<"Fluxo                    :
                             " << stats->first <<" ; "<< fiveTuple.
                            sourceAddress <<" -----> "<<fiveTuple.
                            destinationAddress<<std::endl;
299                         std::cout<<"Transmitidos = " << stats->
                            second.txPackets<<std::endl;
300                         std::cout<<"Recebidos = " << stats->
                            second.rxPackets<<std::endl;
301       std::cout<<"Duração:   " <<  (stats->second.
             timeLastRxPacket.GetSeconds()-stats->second.
             timeFirstTxPacket.GetSeconds())<<std::endl;
302                         std::cout<<"Ultimo pacote recebido      :
                             "<< stats->second.timeLastRxPacket.
                            GetSeconds()<<" Seconds"<<std::endl;
303                         std::cout<<"Vazão: " << stats->second.
                            rxBytes * 8.0 / (stats->second.
                            timeLastRxPacket.GetSeconds()-stats->
                            second.timeFirstTxPacket.GetSeconds())
```

```
                              /1024/1024  << " Mbps"<<std::endl;
304             localThrou=(stats->second.rxBytes * 8.0 / (stats->
                   second.timeLastRxPacket.GetSeconds()-stats->second.
                   timeFirstTxPacket.GetSeconds())/1024/1024);
305
306             DataSet.Add((double)Simulator::Now().GetSeconds(),(
                   double) localThrou);
307                       std::cout<<"
                             ------------------------------------------------
                             "<<std::endl;
308             }
309 }
310                       Simulator::Schedule(Seconds(1),&
                             ThroughputMonitor, fmhelper, flowMon,
                             DataSet);
311
312       {
313         flowMon->SerializeToXmlFile ("ThroughputMonitor.xml",
               true, true);
314       }
315
316         }
317
318   void  DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor
         > flowMon, Gnuplot2dDataset Dataset3)
319   {
320     double localDelay=0;
321
322     std::map<FlowId, FlowMonitor::FlowStats> flowStats3 = flowMon
         ->GetFlowStats();
323     Ptr<Ipv4FlowClassifier> classing3 = DynamicCast<
         Ipv4FlowClassifier> (fmHelper->GetClassifier());
324     for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
         stats3 = flowStats3.begin(); stats3 != flowStats3.end(); ++
         stats3)
325     {
326                   if(stats3->first == 2){
327             Ipv4FlowClassifier::FiveTuple fiveTuple3 = classing3
                   ->FindFlow (stats3->first);
328           std::cout<<"Flow ID : "<< stats3->first <<";"<<
                 fiveTuple3.sourceAddress <<"------>" <<fiveTuple3.
                 destinationAddress<<std::endl;
329             localDelay = stats3->second.timeLastTxPacket.
                   GetSeconds()-stats3->second.timeLastRxPacket.
                   GetSeconds();
330             std::cout<<"Atraso: "<< localDelay <<std::endl;
```

```
331             Dataset3.Add((double)Simulator::Now().GetSeconds(), (
                    double) localDelay);
332             std::cout<<"
                    ----------------------------------------------------------------
                    "<<std::endl;
333           }
334       }
335       Simulator::Schedule(Seconds(1),&DelayMonitor, fmHelper,
              flowMon, Dataset3);
336       {
337          flowMon->SerializeToXmlFile("DelayMonitor.xml", true, true
              );
338       }
339    }
340
341 }
```

ANEXO C – Script Principal de Implantação do Cenário Simulado (Castanhal)

```
1
2  #include "ns3/core-module.h"
3  #include "ns3/internet-module.h"
4  #include "ns3/network-module.h"
5  #include "ns3/applications-module.h"
6  #include "ns3/wifi-module.h"
7  #include "ns3/mesh-module.h"
8  #include "ns3/mobility-module.h"
9  #include "ns3/mesh-helper.h"
10 #include "ns3/netanim-module.h"
11 #include "ns3/gnuplot.h"
12 #include "ns3/flow-monitor-module.h"
13 #include <ns3/flow-monitor-helper.h>
14 #include "ns3/olsr-module.h"
15 #include "ns3/aodv-module.h"
16
17 #include <iostream>
18 #include <sstream>
19 #include <fstream>
20
21 using namespace ns3;
22
23 void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<
      FlowMonitor> flowMon,Gnuplot2dDataset DataSet);
24 void DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor>
      flowMon, Gnuplot2dDataset Dataset3);
25
26 NS_LOG_COMPONENT_DEFINE ("TestMeshScript");
27
28
29 class MeshTest
30 {
31 public:
32
```

```
33   MeshTest ();
34
35    void Configure (int argc, char ** argv);
36
37   int Run ();
38 private:
39   int        m_xSize;
40   int        m_ySize;
41   double     m_step;
42   double     m_randomStart;
43   double     m_totalTime;
44   double     m_packetInterval;
45   uint16_t   m_packetSize;
46   uint32_t   m_nIfaces;
47   bool       m_chan;
48   bool       m_pcap;
49   std::string m_stack;
50   std::string m_root;
51
52   NodeContainer nodes;
53
54   NetDeviceContainer meshDevices;
55
56   Ipv4InterfaceContainer interfaces;
57
58   MeshHelper mesh;
59 private:
60
61   void CreateNodes ();
62
63   void InstallInternetStack ();
64
65   void InstallApplication ();
66
67   void Report ();
68 };
69 MeshTest::MeshTest () :
70   m_xSize (3),
71   m_ySize (3),
72   m_step (500.0),
73   m_randomStart (0.1),
74   m_totalTime (100.0),
75   m_packetInterval (0.1),
76   m_packetSize (1024),
77   m_nIfaces (1),
78   m_chan (true),
```

```
79    m_pcap (false),
80    m_stack ("ns3::Dot11sStack"),
81    m_root ("ff:ff:ff:ff:ff:ff")
82  {
83  }
84  void
85  MeshTest::Configure (int argc, char *argv[])
86  {
87    CommandLine cmd;
88    cmd.AddValue ("x-size", "Number of nodes in a row grid",
        m_xSize);
89    cmd.AddValue ("y-size", "Number of rows in a grid", m_ySize);
90    cmd.AddValue ("step",   "Size of edge in our grid (meters)",
        m_step);
91    cmd.AddValue ("start",  "Maximum random start delay for beacon
        jitter (sec)", m_randomStart);
92    cmd.AddValue ("time",  "Simulation time (sec)", m_totalTime);
93    cmd.AddValue ("packet-interval",  "Interval between packets in
        UDP ping (sec)", m_packetInterval);
94    cmd.AddValue ("packet-size",  "Size of packets in UDP ping (
        bytes)", m_packetSize);
95    cmd.AddValue ("interfaces", "Number of radio interfaces used by
         each mesh point", m_nIfaces);
96    cmd.AddValue ("channels",   "Use different frequency channels
        for different interfaces", m_chan);
97    cmd.AddValue ("pcap",   "Enable PCAP traces on interfaces",
        m_pcap);
98    cmd.AddValue ("stack",  "Type of protocol stack. ns3::
        Dot11sStack by default", m_stack);
99    cmd.AddValue ("root", "Mac address of root mesh point in HWMP",
         m_root);
100
101   cmd.Parse (argc, argv);
102   NS_LOG_DEBUG ("Grid:" << m_xSize << "*" << m_ySize);
103   NS_LOG_DEBUG ("Simulation time: " << m_totalTime << " s");
104 }
105 void
106 MeshTest::CreateNodes ()
107 {
108
109   nodes.Create (20);
110
111   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
112   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::
        Default ();
113   wifiPhy.SetChannel (wifiChannel.Create ());
```

```
114
115
116   mesh = MeshHelper::Default ();
117   if (!Mac48Address (m_root.c_str ()).IsBroadcast ())
118     {
119       mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue
          (Mac48Address (m_root.c_str ())));
120     }
121   else
122     {
123
124       mesh.SetStackInstaller (m_stack);
125     }
126   if (m_chan)
127     {
128       mesh.SetSpreadInterfaceChannels (MeshHelper::
          SPREAD_CHANNELS);
129     }
130   else
131     {
132       mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
133     }
134   mesh.SetMacType ("RandomStart", TimeValue (Seconds (
       m_randomStart)));
135
136   mesh.SetNumberOfInterfaces (m_nIfaces);
137
138   meshDevices = mesh.Install (wifiPhy, nodes);
139
140
141   Ptr<ListPositionAllocator> nodePosition = CreateObject<
       ListPositionAllocator>();
142   nodePosition->Add(Vector(40,19,0));
143   nodePosition->Add(Vector(40,59,0.0));
144   nodePosition->Add(Vector(0,60,0.0));
145   nodePosition->Add(Vector(0,100,0.0));
146   nodePosition->Add(Vector(40,100,0));
147   nodePosition->Add(Vector(40,140,0));
148   nodePosition->Add(Vector(100,60,0.0));
149   nodePosition->Add(Vector(100,100,0.0));
150   nodePosition->Add(Vector(72,80,0.0));
151   nodePosition->Add(Vector(120,80,0.0));
152   nodePosition->Add(Vector(140,60,0.0));
153   nodePosition->Add(Vector(85,127,0.0));
154   nodePosition->Add(Vector(175,60,0.0));
155   nodePosition->Add(Vector(156,100,0.0));
```

```
156   nodePosition->Add(Vector(198,100,0.0));
157   nodePosition->Add(Vector(226,170,0.0));
158   nodePosition->Add(Vector(280,100,0.0));
159   nodePosition->Add(Vector(315,100,0.0));
160
161
162   MobilityHelper mobility;
163   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel"
        );
164   mobility.SetPositionAllocator (nodePosition);
165   mobility.Install (nodes);
166   if (m_pcap)
167
168     wifiPhy.EnablePcapAll (std::string ("mp-"));
169 }
170 void
171 MeshTest::InstallInternetStack ()
172 {
173   InternetStackHelper internetStack;
174   internetStack.Install (nodes);
175   Ipv4AddressHelper address;
176   address.SetBase ("10.1.1.0", "255.255.255.0");
177   interfaces = address.Assign (meshDevices);
178 }
179 void
180 MeshTest::InstallApplication ()
181 {
182   UdpEchoServerHelper echoServer (9);
183   ApplicationContainer serverApps = echoServer.Install (nodes.Get
        (0));
184   serverApps.Start (Seconds (0.0));
185   serverApps.Stop (Seconds (m_totalTime));
186   UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);
187   echoClient.SetAttribute ("MaxPackets", UintegerValue ((uint32_t
        )(m_totalTime*(1/m_packetInterval))));
188   echoClient.SetAttribute ("Interval", TimeValue (Seconds (
        m_packetInterval)));
189   echoClient.SetAttribute ("PacketSize", UintegerValue (
        m_packetSize));
190   ApplicationContainer clientApps = echoClient.Install (nodes.Get
        (m_xSize*m_ySize-1));
191   clientApps.Start (Seconds (0.0));
192   clientApps.Stop (Seconds (m_totalTime));
193 }
194 int
195 MeshTest::Run ()
```

```
196 {
197   CreateNodes ();
198   InstallInternetStack ();
199   InstallApplication ();
200
201
202     std::string fileNameWithNoExtension = "FlowVSThroughput_";
203     std::string graphicsFileName        = fileNameWithNoExtension
          + ".png";
204     std::string plotFileName            = fileNameWithNoExtension
          + ".plt";
205     std::string plotTitle               = "Flow vs Throughput";
206     std::string dataTitle               = "Throughput";
207
208     Gnuplot gnuplot (graphicsFileName);
209     gnuplot.SetTitle (plotTitle);
210     gnuplot.SetTerminal ("png");
211     gnuplot.SetLegend ("Flow", "Throughput");
212
213
214    Gnuplot2dDataset dataset;
215    dataset.SetTitle (dataTitle);
216    dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);
217
218
219   FlowMonitorHelper fmHelper;
220   Ptr<FlowMonitor> allMon = fmHelper.InstallAll();
221   ThroughputMonitor(&fmHelper, allMon, dataset);
222
223
224   std::string fileNameWithNoExtension3 = "FlowVSDelay_";
225   std::string graphicsFileName3      = fileNameWithNoExtension3 +
        ".png";
226   std::string plotFileName3        = fileNameWithNoExtension3 + "
       .plt";
227   std::string plotTitle3       = "Flow vs Delay";
228   std::string dataTitle3       = "Delay";
229
230       Gnuplot gnuplot3 (graphicsFileName3);
231       gnuplot3.SetTitle(plotTitle3);
232
233       gnuplot3.SetTerminal("png");
234
235       gnuplot3.SetLegend("Flow", "Delay");
236
237       Gnuplot2dDataset dataset3;
```

```
238       dataset3.SetTitle(dataTitle3);
239       dataset3.SetStyle(Gnuplot2dDataset::LINES_POINTS);
240
241       DelayMonitor(&fmHelper, allMon, dataset3);
242
243
244   Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report,
        this);
245   Simulator::Stop (Seconds (m_totalTime));
246
247 uint64_t maximo = 10000000000000000;
248         AnimationInterface anim ("manualallocation.xml");
249   anim.SetMaxPktsPerTraceFile(maximo);
250
251
252   Simulator::Run ();
253
254   gnuplot.AddDataset (dataset);
255   std::ofstream plotFile (plotFileName.c_str());
256   gnuplot.GenerateOutput (plotFile);
257   plotFile.close ();
258
259
260   gnuplot3.AddDataset(dataset3);;
261   std::ofstream plotFile3 (plotFileName3.c_str());
262   gnuplot3.GenerateOutput(plotFile3);
263   plotFile3.close();
264
265
266   Simulator::Destroy ();
267   return 0;
268 }
269 void
270 MeshTest::Report ()
271 {
272   unsigned n (0);
273   for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i
        != meshDevices.End (); ++i, ++n)
274     {
275       std::ostringstream os;
276       os << "mp-report-" << n << ".xml";
277       std::cerr << "Printing mesh point device #" << n << "
          diagnostics to " << os.str () << "\n";
278       std::ofstream of;
279       of.open (os.str ().c_str ());
280       if (!of.is_open ())
```

```
281         {
282           std::cerr << "Error: Can't open file " << os.str () <<
                "\n";
283           return;
284         }
285       mesh.Report (*i, of);
286       of.close ();
287     }
288 }
289 int
290 main (int argc, char *argv[])
291 {
292   MeshTest t;
293   t.Configure (argc, argv);
294   return t.Run ();
295 }
296
297 void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<
    FlowMonitor> flowMon,Gnuplot2dDataset DataSet)
298   {
299         double localThrou=0;
300     std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon
        ->GetFlowStats();
301     Ptr<Ipv4FlowClassifier> classing = DynamicCast<
        Ipv4FlowClassifier> (fmhelper->GetClassifier());
302     for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
         stats = flowStats.begin (); stats != flowStats.end (); ++
        stats)
303     {
304                       if(stats->first == 2){
305       Ipv4FlowClassifier::FiveTuple fiveTuple = classing->
          FindFlow (stats->first);
306       std::cout<<"Fluxo      : " << stats->first <<" ; "<<
          fiveTuple.sourceAddress <<" -----> "<<fiveTuple.
          destinationAddress<<std::endl;
307       std::cout<<"Transmitidos = " << stats->second.txPackets<<
          std::endl;
308       std::cout<<"Recebidos = " << stats->second.rxPackets<<std::
          endl;
309       std::cout<<"Duração:    " <<  (stats->second.
          timeLastRxPacket.GetSeconds()-stats->second.
          timeFirstTxPacket.GetSeconds())<<std::endl;
310       std::cout<<"Ultimo pacote recebido  : "<< stats->second.
          timeLastRxPacket.GetSeconds()<<" Seconds"<<std::endl;
311       std::cout<<"Vazão: " << stats->second.rxBytes * 8.0 / (
          stats->second.timeLastRxPacket.GetSeconds()-stats->second
```

```
          .timeFirstTxPacket.GetSeconds())/1024/1024  << " Mbps"<<
          std::endl;
312          localThrou=(stats->second.rxBytes * 8.0 / (stats->
                second.timeLastRxPacket.GetSeconds()-stats->second.
                timeFirstTxPacket.GetSeconds())/1024/1024);
313
314          DataSet.Add((double)Simulator::Now().GetSeconds(),(
                double) localThrou);
315      std::cout<<"
          ----------------------------------------------------------------
          "<<std::endl;
316          }
317 }
318      Simulator::Schedule(Seconds(1),&ThroughputMonitor, fmhelper
          , flowMon,DataSet);
319
320      {
321   flowMon->SerializeToXmlFile ("ThroughputMonitor.xml", true,
       true);
322      }
323
324   }
325
326
327   void  DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor
       > flowMon, Gnuplot2dDataset Dataset3)
328   {
329     double localDelay=0;
330
331     std::map<FlowId, FlowMonitor::FlowStats> flowStats3 = flowMon
          ->GetFlowStats();
332     Ptr<Ipv4FlowClassifier> classing3 = DynamicCast<
          Ipv4FlowClassifier> (fmHelper->GetClassifier());
333     for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
          stats3 = flowStats3.begin(); stats3 != flowStats3.end(); ++
          stats3)
334     {
335                  if(stats3->first == 2){
336          Ipv4FlowClassifier::FiveTuple fiveTuple3 = classing3
                ->FindFlow (stats3->first);
337         std::cout<<"Flow ID : "<< stats3->first <<";"<<
              fiveTuple3.sourceAddress <<"------>" <<fiveTuple3.
              destinationAddress<<std::endl;
338          localDelay = stats3->second.timeLastRxPacket.
                GetSeconds()-stats3->second.timeLastTxPacket.
                GetSeconds();
```

```
339             std::cout<<"Atraso: "<< localDelay <<std::endl;
340           Dataset3.Add((double)Simulator::Now().GetSeconds(), (
                 double) localDelay);
341           std::cout<<"
                 ----------------------------------------------------------------
                 "<<std::endl;
342       }
343     }
344     Simulator::Schedule(Seconds(1),&DelayMonitor, fmHelper,
          flowMon, Dataset3);
345     {
346       flowMon->SerializeToXmlFile("DelayMonitor.xml", true, true
            );
347     }
348   }
```