RNP

GT MESH

Rede Mesh de Acesso Universitário Faixa Larga Sem Fio

Universidade Federal Fluminense

RT3 - Relatório técnico de avaliação dos resultados do protótipo

Autores

Célio Vinicius Neves de Albuquerque – IC/UFF

Débora Christina Muchaluat Saade – TET/UFF

Diego Gimenez Passos – IC/UFF

Douglas Vidal Teixeira – TET/UFF

Julius Leite – IC/UFF

Luciana Esteves Neves – IC/UFF

Luiz Cláudio Schara Magalhães – TET/UFF

Data

31 de agosto de 2006

Sumário

1. Apresentação do protótipo desenvolvido 5
1.1 Componentes do protótipo 5
1.1.1 Montagem do Módulo POE 11
1.2 Arquitetura e esquema de endereçamento 13
1.3 Protocolo de Roteamento 15
1.3.1 OLSR original 16
1.3.2 OLSR adaptado pelo GT-Mesh: OLSR-ML 18
1.4 Topologia atual 19
1.4.1 Rede interna 19
1.4.2 Rede externa 21
1.5 Gerência da Rede Mesh 24
1.5.1 Visualização da qualidade dos enlaces 25
1.5.2 Visualização de informações dos nós 26
1.5.3 Visualização de estatísticas dos acessos dos usuários 27
1.5.4 Gerência administrativa dos nós e dos usuários 28
1.6 Módulo de Autenticação e gerência dos acessos: o *wifidog* 30
1.7 Framework de Suporte a QoS 42
1.7.1 Framework nos roteadores 42
1.7.2 Framework no cliente e servidor 44
2. Resultados dos testes 46
2.1 Resultados dos testes na rede interna 46
2.1.1 Variações da Métrica ETX 46
2.1.2 Estabilidade das Rotas 47
2.1.3 Taxas de Perda de Pacotes 48
2.1.4 Retardo 48
2.1.5 Vazão 49
2.2 Resultados dos testes na rede externa 51
2.2.1 Vazão 51
2.2.2 Perda e atraso 55
2.2.3 Problemas encontrados e soluções: utilização de diferentes antenas e a questão do uso de diversidade espacial 56
2.3 Estatísticas dos acessos 59
2.3.1 Estatísticas dos acessos na rede interna 60
2.3.2 Estatísticas dos acessos na rede externa 63
2.4 Resultados do *Framework* de QoS 65
2.4.1 Problemas encontrados 65
2.4.2 Testes e resultados 66
3. Avaliação 76
Referências 78
Apêndice 1 - Código Fonte do Protótipo 78
Apêndice 2 - Documentação do Protótipo: Manual de Instalação 110
Apêndice 3 - Documentação do Protótipo: Manual do Usuário 118

Lista de Figuras

Figura 1: Roteador WRT54G da Linksys 6
Figura 2: Caixa hermética para proteção do roteador 7
Figura 3: Base para a haste da antena 8
Figura 4: Haste de 2 metros para suporte da antena e do roteador 8
Figura 5: Antenas omni-direcional e puramente direcional 9
Figura 6: Cabo RGC 213 9
Figura 7: Conectores RP-TNC e N-Macho 9
Figura 8: Suporte metálico para prender a caixa ao suporte 10
Figura 9: Módulo POE 10
Figura 10: Protótipo completo montado em ambiente externo 11
Figura 11: Conector RJ45. O POE utiliza os fios 4 e 5 para um fio de energia e 7 e 8 para o outro 13
Figura 12: Topologia da rede interna 20
Figura 13: Qualidade dos enlaces da rede interna 20
Figura 14: Topologia da rede externa 22
Figura 15: Visão da rede externa por foto do Google Earth 23
Figura 16: ETX entre os enlaces da rede externa 24
Figura 17: Ferramenta desenvolvida pelo GT que permite a visualização da topologia da rede em tempo real através do servidor web 25
Figura 18: Informações de cada um dos nós disponíveis através do servidor web 26
Figura 19: Visualização de estatísticas de acessos disponível no servidor web 28
Figura 20: Tela de entrada do *wifidog* para usuários que foram desabilitados temporariamente. 30
Figura 21: *Wifidog* – cliente inicia uma conexão 31
Figura 22: *Wifidog* - Processo de autenticação e liberação 32
Figura 23: *Wifidog* - cliente autenticado e navegação liberada 32
Figura 24: Esquematização da instalação do wifidog na rede Mesh da UFF 34
Figura 25: Wifidog - Autenticação de um cliente sem fio 34
Figura 26: Página de entrada do servidor de autenticação 36
Figura 27: Página de gerência com as opções à esquerda 37
Figura 28: Wifidog - Criação de um novo nó 38
Figura 29: Wifidog- Arquivo de configuração *wifidog.conf* localizado dentro dos roteadores no diretório /etc 39
Figura 30: Wifidog - A página de abertura foi personalizada para exibir o logo do GT 40
Figura 31: Usuário entrando com o mesmo login em duas máquinas diferentes. A máquina anterior será agora desabilitada. 40
Figura 32: Interface de gerência do Wifidog 41
Figura 33: Configuração das qdisc e classes em cada roteador 44

Figura 34:O ambiente interno de testes do projeto ReMesh 46
Figura 35: Taxas de perda de pacotes na comunicação entre os nós 3 e 6. 48
Figura 36: Retardos de ida e volta na comunicação entre os nós 3 e 6..... 49
Figura 37: Vazão medida com IPERF em comunicações começando pelo nó 3. 50
Figura 38:Vazão medida com TTCP em comunicações começando pelo nó 3. 51
Figura 39: Vazão entre os nós 10.151.1.1 e 10.151.7.1 52
Figura 40: Vazão entre os nós 10.151.4.1 e 10.151.7.1 53
Figura 41: Vazão entre os nós 10.151.11.1 e 10.151.7.1 53
Figura 42: Vazão entre os nós 10.151.13.1 e 10.151.7.1 54
Figura 43: Vazão entre os nós 10.151.14.1 e 10.151.7.1 54
Figura 44: Lóbulos de irradiação da antena omni-direcional adotada 56
Figura 45: Nó mesh montado com duas antenas 57
Figura 46: Desempenho da antena omni-direcional - Média de 3.53 Mbits/seg 58
Figura 47: Desempenho da antena direcional - Média de 5.09 Mbits/seg 58
Figura 48: Desempenho do modo de diversidade - Média de 3.77 Mbits/seg 59
Figura 49: Número de novos usuários registrados nas redes interna e externa 60
Figura 50: Número acumulativo de usuários cadastrados nas redes interna e externa..... 60
Figura 51: Cenário dos testes do protótipo de suporte a QoS..... 67
Figura 52: *Jitter* e Taxa de envio em 4 experimentos da cena 1 com QoS. 68
Figura 53: Perda de pacotes e taxa de envio em 4 experimentos da cena 1 com QoS..... 69
Figura 54: *Jitter* e Perda de pacotes em 4 experimentos da cena 1 sem QoS. 69
Figura 55: *Jitter* e Taxa de envio em 4 experimentos da cena 2 com QoS. 70
Figura 56: Perda de pacotes e taxa de envio em 4 experimentos da cena 2 com QoS..... 70
Figura 57: *Jitter* e Perda de pacotes em 4 experimentos da cena 2 sem QoS. 71
Figura 58: *Jitter* e Taxa de envio em 4 experimentos da cena 3 com QoS. 72
Figura 59: Perda de pacotes e taxa de envio em 4 experimentos da cena 3 com QoS..... 73
Figura 60: *Jitter* e Perda de pacotes em 4 experimentos da cena 3 sem QoS. 73
Figura 61: *Jitter* e Taxa de envio em 4 experimentos da cena 4 com QoS. 74
Figura 62: Perda de pacotes e taxa de envio em 4 experimentos da cena 4 com QoS..... 74
Figura 63: *Jitter* e Perda de pacotes em 4 experimentos da cena 3 sem QoS. 75

1. Apresentação do protótipo desenvolvido

O GT-Mesh – Grupo de Trabalho em Redes Mesh – desenvolveu um protótipo de roteador mesh com o objetivo de demonstrar na forma de prova de conceito a viabilidade de uma rede de acesso universitário faixa larga sem fio. Além dos componentes de hardware foram desenvolvidas soluções em software para gerência da rede mesh e autenticação de usuários e um framework para qualidade de serviço. Esta seção apresenta em detalhes os componentes de hardware do protótipo, a solução de endereçamento empregada, o protocolo de roteamento desenvolvido pelo GT-Mesh, as ferramentas de gerência e autenticação, o status atual da rede e o framework de QoS. Duas redes mesh independentes foram implantadas, uma rede interna às instalações do Prédio da Engenharia da UFF e outra externa na vizinhança de São Domingos, próxima ao campus da Praia Vermelha da UFF. Na Seção 2 os resultados dos testes e a análise de desempenho das redes mesh são apresentados. E na Seção 3 uma avaliação do protótipo é apresentada indicando seu potencial para suporte a um serviço alternativo de conectividade faixa larga sem fio para a comunidade acadêmica. Nos apêndices deste documento encontram-se os manuais de instalação, do usuário e o código fonte do protótipo.

1.1 Componentes do protótipo

O protótipo do ponto de acesso a rede mesh desenvolvido pelo GT-Mesh deve funcionar adequadamente em ambientes externos, por isso não somente o roteador mas todos os outros itens presentes em sua montagem devem ser considerados.

O roteador utilizado é o WRT54G, da Linksys, em suas versões 1 até 4 (Figura 1). Trata-se de um roteador wireless com 4 MB de memória flash (permanente) e 8 MB de memória RAM. Além de ponto de acesso, ele rotea os clientes ligados a ele tanto pela interface sem fio como pelas suas cinco portas ethernet presentes. O roteador vem de fábrica com um sistema operacional da própria Linksys que possui uma interface de administração via web. Contudo, o roteador da Linksys deve ser adaptado para nossa funcionalidade e ser transformado em um roteador mesh. Para tanto instalamos dentro dele uma imagem gerada através do sistema operacional OpenWRT e substituímos o *firmware* original.

Um problema ocorrido foi que nas versões atuais dos roteadores WRT54G (a partir da versão 5 em diante) eles passaram a vir com sistema operacional fornecido pela VxWorks, o que impossibilitava a inserção do firmware gerado pelo Openwrt. Além disso, eles passaram a ter metade do tamanho disponível tanto para a memória RAM como para a memória flash. Apesar da questão da inserção do firmware do Openwrt ter sido superada com a utilização de um *bootloader* da própria VxWorks, o problema do tamanho ainda continuou. A solução adotada para esta questão foi a adoção do roteador WRT54GS também da Linksys. Estes roteadores são mais caros do que o anterior, porém os que estão disponíveis no mercado até o

momento ainda não apresentam nenhuma limitação de inserção do firmware e ainda possuem o dobro de espaço disponível (8 MB para flash e 16 MB de RAM).

Recentemente a Linksys anunciou o lançamento do roteador WRT54GL (Linux), onde seria possível fazer novamente a inserção do firmware gerado pelo Openwrt, o que seria uma outra alternativa além da utilização do WRT54GS, porém este outro hardware não foi testado ainda pelo GT-Mesh.

Figura 1: Roteador WRT54G da Linksys

O OpenWRT (http://www.openwrt.org) é uma distribuição peculiar do Linux criada especialmente para gerar imagens que possam substituir o sistema operacional original em hardwares especializados. O OpenWRT pode ser baixado para qualquer máquina utilizando sistema operacional Linux. Ele possui um interface amigável para a geração da imagem bem semelhante à interface existente para compilação do kernel no Linux. Nela pode ser escolhido o processador alvo (no caso do WRT54G é um processador do tipo mipsel) e os pacotes que deverão estar presentes. Todas as ferramentas básicas do sistema operacional (comandos e aplicativos básicos) estão presentes em um grande arquivo binário executável chamado *busybox*. O restante das ferramentas é disponibilizado como arquivos independentes através de módulos a serem inseridos na memória e arquivos executáveis.

Uma vez decidido quais serão os componentes do sistema operacional, basta escolher o sistema de arquivos para a geração da imagem. Atualmente são produzidos dois tipos: jffs e squash. A direrença prática entre os dois é desempenho e tamanho ocupado. O jffs ocupa mais espaço, porém é mais rápido e foi o escolhido para a geração da imagem utilizada no nosso protótipo. A inserção da imagem no roteador é simples e pode ser feita através da interface web original do roteador na seção “Upgrade Firmware”.

Além do sistema operacional, o protótipo utiliza ferramentas adicionais como a nossa versão do protocolo OLSR, o módulo binário do agente de autenticação (*wifidog)* e uma ferramenta de medição de vazão interna (*iperf)*. Todas os binários a serem executados dentro do roteador devem ser gerados utilizando o compilador e o *linker* do OpenWRT presentes na sua distribuição.

Para a montagem do protótipo completo, pronto para ser instalado em ambientes externos, utilizamos ainda os seguintes componentes:

- Caixa hermética da CEMAR (Figura 2)
- Base de ferro galvanizado (Figura 3)
- Haste de ferro galvanizado de 2 metros (Figura 4)
- Antena omni-direcional de 18,5 dbi de ganho ou alternativamente antena direcional de 24 dbi de ganho, dependendo do enlace (Figura 5)
- Cabo RGC 213 de 1m (Figura 6)
- Conectores RP-TNC para ligação do cabo com a saída RF do roteador (Figura 7)
- Conectores N-Macho para ligação do cabo com a antena omni-direcional e N-Fêmea para ligação com a antena direcional (Figura 7)
- Suporte metálico para suporte da caixa com o roteador na haste (Figura 8)
- Módulo POE desenvolvido pelo próprio GT-Mesh (Figura 9), cuja montagem é detalhada na Seção 1.1.1.
- Cabo de rede (CAT5), de preferência com capa protetora, para ligação do roteador ao cliente.

Figura 2: Caixa hermética para proteção do roteador

Figura 3: Base para a haste da antena

Figura 4: Haste de 2 metros para suporte da antena e do roteador

Figura 5: Antenas omni-direcional e puramente direcional

Figura 6: Cabo RGC 213

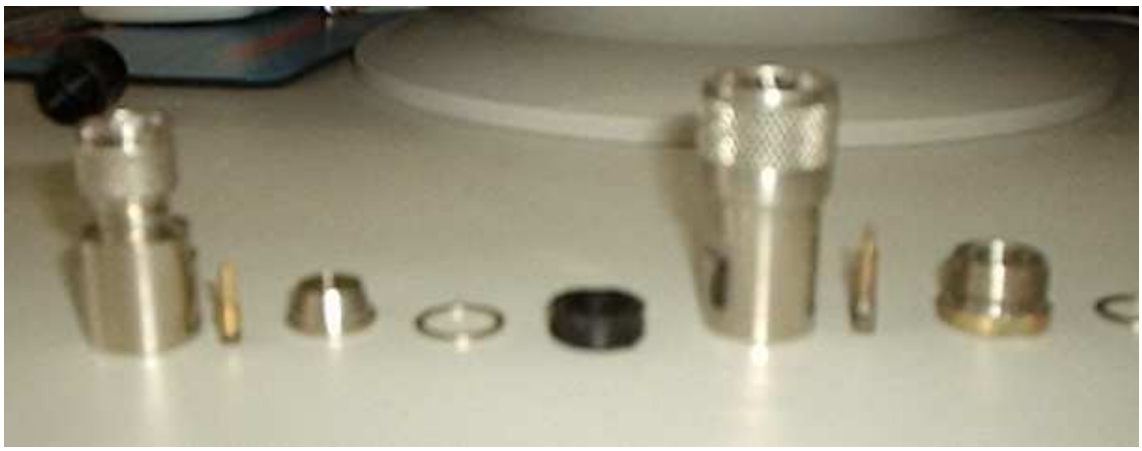


Figura 7: Conectores RP-TNC e N-Macho

Figura 8: Suporte metálico para prender a caixa ao suporte

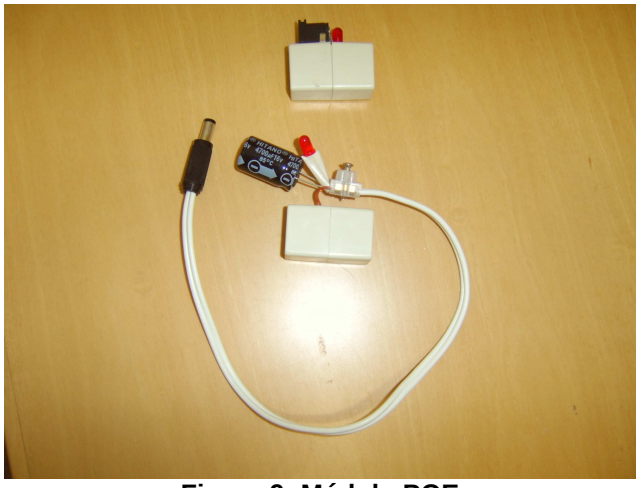


Figura 9: Módulo POE

Montando-se todos os itens listados anteriormente temos o protótipo completo, ilustrado na Figura 10.

Figura 10: Protótipo completo montado em ambiente externo

Além do roteador e das peças de montagem para o ambiente externo, é necessária uma máquina para fazer o serviço de autenticação e coleta de estatísticas de acesso (servidor de autenticação instalado com o software *wifidog*). Adicionalmente utilizamos outra máquina como servidor web, onde executamos as ferramentas de gerência dos nós e da qualidade dos links. Nada impede que apenas uma máquina realize ambos os serviços desde que sejam identificados como servidores diferentes através da utilização da diretiva *virtual host* do apache ou de qualquer outro método. A diferenciação é necessária porque o servidor de autenticação também utiliza o serviço *http* para a apresentação da página de *login* dos usuários.

1.1.1 Montagem do Módulo POE

O módulo de POE desenvolvido pelo GT (veja Figura 9) não é POE padrão, pois não há injetores de tensão que medem o comportamento do equipamento remoto, e desligam a tensão se o equipamento não for capaz de receber alimentação via POE. Nosso módulo simplesmente usa os fios "ociosos" do padrão ethernet (4,5,7 e 8) para levar tensão DC do próprio adaptador do equipamento, colocado remotamente, ao equipamento em si. Além de ser uma solução barata diminui o trabalho adicional da passagem de um cabo somente para

energia e dos transtornos que poderiam ser ocasionados se a alimentação fosse suprida pelas dependências do prédio onerando os condôminos.

No projeto original usa-se leds e um capacitor para diminuir a carga indutiva do fio, mas atualmente simplificamos o projeto para o mínimo possível.

Lista de peças para confecção do adaptador POE:

- 1 par macho/fêmea de conectores de força iguais ao do seu equipamento (é possível eliminar até estes, cortando o fio do adaptador, mas neste caso a adaptação é permanente). De preferência, use o modelo de fêmea com os contatos radiais em vez de axiais (isto é, saindo do lado em vez de em baixo do conector).
- conectores (extensores) de rede RJ45. Aqui, vários destes extensores vieram com conexões erradas, sem seguir qualquer padrão de cores, requerendo atenção dobrada ao montar, e alguns não eram usáveis porque as conexões estavam trocadas.
- 20 cm de fio paralelo.

Adicionalmente, pode-se adicionar ao projeto:

- 2 leds vermelhos
- 2 resistores apropriados (470R dá mais de 20 mA em 12V, o que acende bem o led).
- 1 capacitor eletrolítico grande, de voltagem apropriada (ex, 470KuF, 16V para fontes de 12V)

Montagem mínima:

Marque e faça três furos no topo de um dos extensores de rede, usando o conector de fonte fêmea como modelo. No nosso módulo, o conector fica com o contato central alinhado com o centro do extensor, de forma a não impedir a conexão dos cabos. Os furos devem ser feitos com atenção para não danificar os fios. Abra os extensores de rede. Corte os fios 4, 5, 7 e 8, com cuidado para não cortar os outros pares. Solde o par 4 e 5 no conector central, e o par 7 e 8 no conector da ponta. Solde um par de fios nos contados (um em cada, obviamente) do conector macho e siga o mesmo padrão (7 e 8 no centro, 4 e 5 no corpo). É necessário colocar a capa antes de soldar os fios, ou você terá de fazer a solda novamente. Além disso, deve-se isolar bem as soldas utilizando "*shrink tubing"* ou simplesmente com canudo e cola quente. Fita isolante falha com o tempo.

Montagem avançada:

Faça mais dois furos em cada um dos extensores para encaixar os led's. Solde o resistor no led, e no par 4 e 5, e o outro contato do led no par 7 e 8, cuidando da polaridade (a marca do led vai para o negativo). O capacitor pode ser soldado diretamente nos pares do conector macho.

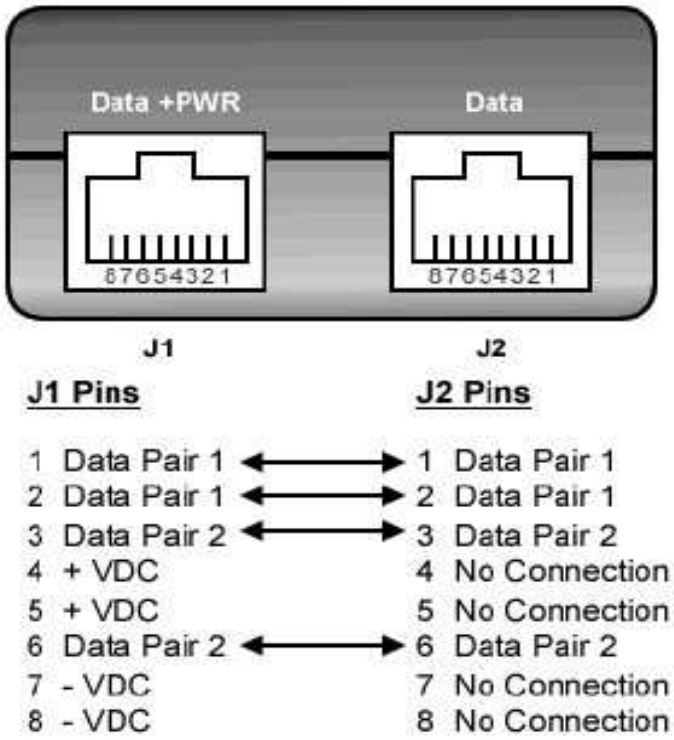


Figura 11: Conector RJ45. O POE utiliza os fios 4 e 5 para um fio de energia e 7 e 8 para o outro

Para utilização no protótipo, são necessários três cabos ethernet. Um vai do equipamento ao primeiro adaptador POE (fêmea), que recebe a alimentação do conversor do equipamento. O segundo vai do primeiro adaptador POE ao segundo POE (macho). Do segundo POE sai o cabo de alimentação para o equipamento remoto. O terceiro cabo de rede vai do segundo POE ao equipamento remoto, e provavelmente é bem curto.

1.2 Arquitetura e esquema de endereçamento

Para conexão da rede mesh à Internet, é necessária a escolha de um nó mesh para ser o gateway da rede. O nó gateway deve ter acesso à Internet e deve estar na mesma subrede do servidor web e do servidor de autenticação. Na nossa arquitetura optamos por não interligar clientes por fio no gateway, pois estes não passam por nenhum tipo de autenticação, tendo o acesso liberado. Os demais roteadores mesh não precisam estar ligados à Internet e escoam o tráfego até o gateway através dos múltiplos saltos presentes em seus respectivos percursos.

Os roteadores possuem duas interfaces, a ethernet e a wireless. A interface ethernet ainda é dividida dentro dos roteadores pelo sistema operacional entre LAN e WAN. A interface WAN

corresponde a apenas uma porta ethernet e é utilizada efetivamente apenas pelo roteador gateway.

A rede mesh adotou um esquema de endereçamento que divide basicamente a rede em duas: a rede com fio e a rede sem fio. Cada roteador irá definir duas sub-redes que serão particulares a ele, uma para os clientes com fio e outra para os clientes sem fio.

Foi adotada uma nomenclatura específica para a geração dos endereços. Cada roteador é chamado por um número inteiro iniciando de 0 até 253, tal número o identifica como "*ID*". A interface wireless terá os seus parâmetros configurados de acordo com os valores presentes nas variáveis da *nvram* do roteador. Estes valores são configurados através da execução do script "Configure_manet" seguido do *ID* do roteador. A interface wireless terá um ESSID definido neste script de configuração e a princípio estará ativa no canal 6 e operando no modo *ad-hoc.* O script fornecerá também o endereço das interfaces sem fio e da interface ethernet LAN. No caso do gateway um endereço para a interface ethernet WAN também deverá ser fornecido.

O endereço da interface sem fio do nó é dado pelo esquema: 10.151.*ID*+*1.*1 com máscara 255.255.0.0. Por exemplo, o nó de "ID" 0 terá o endereço 10.151.1.1. Já a interface LAN terá o endereço dado o seguinte esquema:

10.152.Y.Z onde

Y ={(*ID**32)}mod(256)

Z = {(*ID**32)mod (256)} + 2, com máscara 255.255.255.224

Onde *mod* se refere à operação módulo.

Esses serão os endereços internos do roteador. Os clientes deverão ter os seguintes endereçamentos que serão entregues pelo servidor de DHCP do próprio roteador. Os clientes sem fio receberão endereços dentro do seguinte escopo:

Início → 10.151.*ID*+1.2

Final da faixa→ 10.151.*ID* +1.28

Máscara → 255.255.255.224

E os clientes com fio receberão:

Início →10.152.Y.Z , onde

Y = {(*ID**32) mod 65536} mod 256

Z = {(*ID**32) mod 256} + 2

Final da faixa → 10.152.Y.Z , onde

Y = {(*ID**32) mod 65536} div 256

Z = {(*ID**32) mod 256}+30

Máscara → 255.255.255.224

Onde *div* é o resultado da divisão inteira.

As faixas das sub-redes entregues pelos roteadores também terão que ser exportadas pelo protocolo de roteamento exigindo, portanto, que elas sejam especificadas na sua configuração.

Percebe-se que os roteadores possuem uma máscara diferente para o endereço da interface sem fio comparado ao endereço dado aos clientes também sem fio. Isto tem que ser feito pois os clientes sem fio, para se associarem à rede, terão que estar operando em modo *ad-hoc* também. Caso a máscara não defina bem a sub-rede que o nó "serve", um determinado cliente poderá se comunicar simultaneamente com dois nós distintos da rede, o que trará problemas na entrega dos pacotes. Além disso, os roteadores têm que possuir uma máscara aberta (255.255.0.0) para que eles possam falar entre si e ao mesmo tempo com os seus clientes. O que garantirá o roteamento correto dos pacotes para os clientes sem fio será a configuração correta da sub-rede sem fio no protocolo de roteamento OLSR. Ele será o responsável para dizer a todos os nós participantes da rede mesh que uma determinada sub-rede será alcançada somente através do determinado nó que a anunciou.

1.3 Protocolo de Roteamento

Atualmente o protótipo utiliza um protocolo de roteamento próprio, o OLSR-ML, que é uma derivação do protocolo OLSR original. O arquivo binário executável do protocolo original é parte integrante da imagem gerada pelo OpenWRT, contudo, ao longo dos testes foi verificado que ele trazia uma grande instabilidade devido a forma com que ele calculava as métricas para eleição das melhores rotas. Em vista disso, alterando-se a forma original com a qual o protocolo calculava as métricas geramos o novo protocolo que trouxe uma maior estabilidade no processo de eleição das rotas e, conseqüentemente, no tráfego da rede. Os parâmetros de configuração do protocolo também foram testados para obtenção dos atuais que acreditamos serem os melhores possíveis para o propósito de uma rede mesh.

Nas próximas duas subsecções explicaremos o funcionamento do protocolo original e o que foi desenvolvido. Na seção 2 demonstramos os testes que foram realizados na rede interna onde podemos comparar as melhoras obtidas.

1.3.1 OLSR original

A métrica mais comum utilizada para calcular rotas ótimas na maioria dos protocolos de roteamento ad-hoc existentes, incluindo a própria especificação padronizada do OLSR, é a quantidade de saltos mínima. Entretanto, minimizar a quantidade de saltos somente não é suficiente em um ambiente sem fio, pois quando a rede é densa, podem existir diversas rotas com o mesmo comprimento e diferente qualidade em cada caminho. A decisão arbitrária feita pelos algoritmos que minimizam saltos pode não selecionar a melhor rota disponível. A implementação do OLSR utiliza uma métrica que mede a qualidade dos enlaces, chamada ETX (*expected transmission count*) (OLSR-ETX). Essa extensão tenta encontrar caminhos com o menor número de transmissões necessárias para entregar um pacote ao seu destino final.

A métrica ETX de um enlace é calculada usando as taxas de recepção de pacotes em ambos os sentidos do enlace. A taxa de recepção é a probabilidade com que um pacote de dados chegue no próximo nó do caminho. A probabilidade esperada com que uma transmissão seja recebida com sucesso e seja reconhecida é o produto entre taxa de recepção de ida (d_f) e a taxa de recepção de volta (d_r) do enlace. Logo, o valor da métrica ETX é dado por:

$$ETX = 1 / (d_f \times d_r)$$

As taxas de recepção (d) são medidas utilizando pacotes HELLO do protocolo OLSR modificados, enviados a cada t segundos ($t=2$, por default). Cada nó calcula o número de HELLOs recebidos em um período de w segundos ($w=20$, por default) e divide pelo número de HELLOs que deveriam ter sido recebidos no mesmo período (10, por default). Cada pacote HELLO modificado informa o número de HELLOs recebidos pelo vizinho durante os últimos w segundos, para permitir que cada vizinho calcule a taxa de recepção no outro sentido do enlace. Quanto pior a qualidade do enlace, maior o valor de ETX. O valor mínimo de ETX é 1 em um enlace perfeito e pode ser 0 somente se nenhum pacote HELLO for recebido durante o período de w segundos.

No caso de um caminho com múltiplos saltos, o valor de ETX da rota completa é dado pelo soma do valor de ETX de cada salto. Logo, em uma rota do nó A ao nó C, passando pelo nó B, o valor final de ETX é dado por:

$$ETX_{AC} = ETX_{AB} + ETX_{BC}.$$

O protocolo OLSR seleciona como melhor rota, de uma origem a um destino específico, aquela que tem o menor valor de ETX. Entretanto, através de testes de desempenho, observou-se que o uso do protocolo original OLSR-ETX poderia resultar em (1) instabilidades nas tabelas de rotas e (2) altas taxas de perda de pacotes (mesmo que menores do que as obtidas com algoritmos que minimizam saltos).

Para ilustrar uma fonte potencial de instabilidade, considere uma janela *w* de 22 segundos (11 pacotes HELLO) e o estado desta janela em três enlaces como ilustrado na Tabela 1. A Tabela 1 é uma abstração da janela ETX e considera que os enlaces entre os nós N1 e N2 e entre os nós N1 e N3 são perfeitos. O enlace direto entre os nós N1 e N3 tem 50% de probabilidade de perda e, nos intervalos pares, um pacote HELLO é perdido. Quando selecionamos uma rota entre os nós N1 e N3, o valor de ETX do caminho N1-N2-N3 é sempre igual a 2, já que cada salto tem ETX igual a 1. Por outro lado, valor de ETX do enlace direto N1-N3 varia entre 1,83 e 2,20, dependendo do estado da janela ETX deste enlace.

Tabela 1. Exemplo de Janela ETX

N1-N2	1	1	1	1	1	1	1	1	1	1	1
N2-N3	1	1	1	1	1	1	1	1	1	1	1
N1-N3	1	0	1	0	1	0	1	0	1	0	1

Neste exemplo hipotético, as rotas entre N1 e N3 mudariam a cada 2 segundos, alternando entre o enlace direto com perdas e o caminho perfeito com 2 saltos. Apesar de ser um exemplo hipotético, não foi raro constatar esse comportamento instável nos testes realizados. A respeito da alta taxa de perda, apesar de se basear na probabilidade de sucesso em um único enlace, OLSR-ETX visa minimizar o número de transmissões ao logo do caminho e não minimizar a robabilidade de perda ao longo do caminho. O exemplo exibido na Tabela 1 mostra que OLSR-ETX selecionaria freqüentemente a rota com quase 50% de taxa de perda ao invés do caminho alternativo sem perda. Na verdade, com um janela *w* de 20 segundos, o ETX do enlace direto N1-N3 seria exatamente 2, e esse enlace com 50% de taxa de perda seria a rota selecionada entre N1-N3, já que o critério usado por OLSR-ETX é selecionar o caminho com o menor número de saltos dentre os caminhos com o menor valor de ETX.

Com isso, é possível que OLSR-ETX escolha uma rota em que a taxa de perda seja tão alta que degrade a vazão da rede, levando-o a não atingir o seu objetivo. Isso pode ser concluído se notarmos que, com uma janela de *w* pacotes de tamanho, o incremento em ETX para cada pacote perdido como uma função de *r* pacotes recebidos é dado por:

$$\delta = \frac{w}{r(r-1)}.$$

Este incremento δ é tipicamente muito pequeno em enlaces de boa qualidade e só chega perto de 1 quando a taxa de perda se aproxima de metade da janela. Como já

comentado anteriormente, para um enlace perfeito, o valor mínimo de ETX é 1, então incremento de selecionar uma rota com mais um salto é no mínimo 1. OLSR-ETX acaba selecionando caminhos mais curtos com taxas de perda mais altas que caminhos mais longos com menores taxas de perda.

1.3.2 OLSR adaptado pelo GT-Mesh: OLSR-ML

Usando o protocolo OLSR com a extensão ETX original, vivenciamos instabilidade nas tabelas de rotas e altas taxas de perda de pacotes na rede do projeto, como será visto na Seção 2.1. Para melhorar o desempenho da rede, este trabalho propõe uma maneira alternativa para calcular a qualidade de um dado caminho com o objetivo de selecionar a rota com a menor probabilidade de perda de pacotes.

Interpretando as taxas de recepção dos enlaces como probabilidades, como foram originalmente definidas, a probabilidade de transmissão com sucesso de A até B é:

$$P_{AB} = (d_f \times d_r).$$

Em um caminho com múltiplos saltos, a probabilidade de transmissão com sucesso em todo o caminho deveria ser o produto das probabilidades de cada salto. Com isso, em uma rota de A até C, passando por B, a probabilidade total de transmissão com sucesso no caminho é dada por:

$$P_{AC} = P_{AB} \times P_{BC}.$$

Dessa forma, em nossa proposta, o valor da medida de qualidade de um caminho com múltiplos saltos é dado pelo produto do valor da medida de qualidade de cada salto e não pela soma de todos eles, como na proposta original. Como estamos usando a probabilidade P e não seu valor inverso (ETX), a melhor rota de uma origem a um destino específico é aquela com a maior probabilidade (P) de transmissão com sucesso, ou seja, aquela com a menor probabilidade de perda de pacotes.

Usando esta abordagem, poderíamos questionar que se todas as medidas de qualidade fossem iguais a 1, então a probabilidade de transmissão com sucesso em um caminho com múltiplos saltos, entre dois nós, seria mesma que a probabilidade em um enlace direto entre eles. Isto é verdade, porém a implementação do OLSR já fornece uma solução para esse cenário. Quando múltiplas rotas com a mesma medida de qualidade são encontradas, aquela com a menor quantidade de saltos é a escolhida. Então, o enlace direto seria escolhido neste caso.

Por outro lado, se um caminho com múltiplos saltos tem probabilidade de transmissão com sucesso maior (ou probabilidade de perda de pacotes menor) que um caminho direto, o

caminho com mais saltos é escolhido. Resultados de testes de desempenho, documentados na Seção 4, indicam que a nova proposta melhorou o desempenho da rede mesh em relação à estabilidade das rotas, taxas de perda de pacotes, retardos de ida e volta e até em vazão em alguns casos.

Os proponentes do ETX afirmam que a vazão máxima é conseguida minimizando o número de transmissões por pacote sobre o meio compartilhado, e demonstram melhoras como o dobro da vazão sobre a métrica tradicional baseada em número de saltos mínimo. Neste trabalho, nós estendemos esse argumento, sugerindo que caminhos com a menor taxa de perda (ou a maior probabilidade de transmissão com sucesso) também levam a melhoras na vazão, com o benefício adicional de rotas mais estáveis e taxas de perda de pacotes mais baixas.

1.4 Topologia atual

Atualmente o GT-Mesh dispõe de duas redes, uma localizada dentro das dependências do campus da Praia Vermelha da UFF (campus dos cursos de Engenharia e Ciência da Computação) constituída por 7 nós mesh com suas antenas originais dentro do mesmo prédio. A rede interna é muito útil para desenvolvimento e realizações de testes em um ambiente protegido. A segunda rede é a externa onde já contamos com um total de 6 roteadores, incluindo o gateway localizado no topo do Bloco E do campus, e mais 5 nos topos dos prédios de alunos voluntários, cobrindo a região no entorno do campus.

1.4.1 Rede interna

A rede interna constituída por 7 nós mesh, funcionando com suas antenas originais, estão espalhados ao longo dos 4° e 3° andares do bl oco “E” do campus da Praia vermelha da UFF. Suas configurações internas são:

- Potência de saída: 255mW
- Ambas as antenas em funcionamento
- Canal 1
- ESSID: gt_mesh_interna
- Controle de taxa no modo automático

O propósito da rede interna é o desenvolvimento de novos serviços e aplicativos, bem como o provimento de acesso sem fio à Internet para funcionários e alunos. A topologia atual basicamente é a exibida na Figura 11 (as distâncias são valores estimados e aproximados).

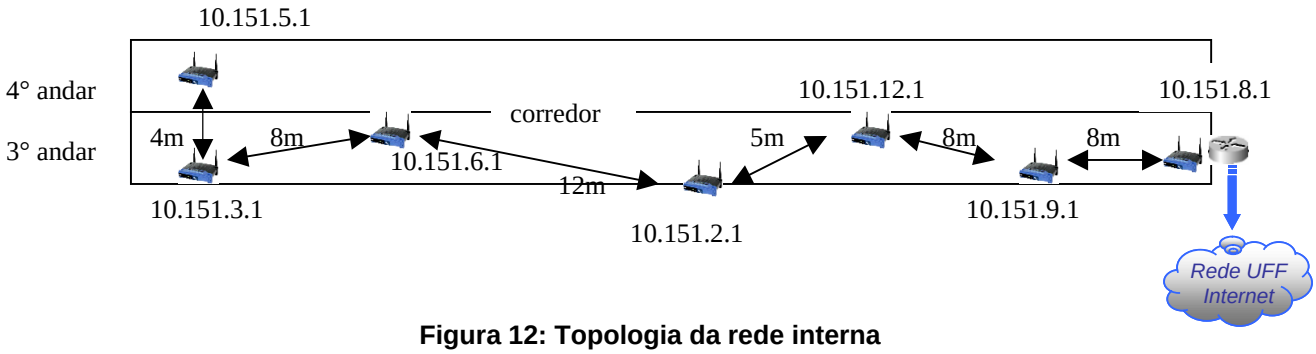


Figura 12: Topologia da rede interna

O valor da métrica ETX de cada enlace pode ser visualizado na Figura 13.

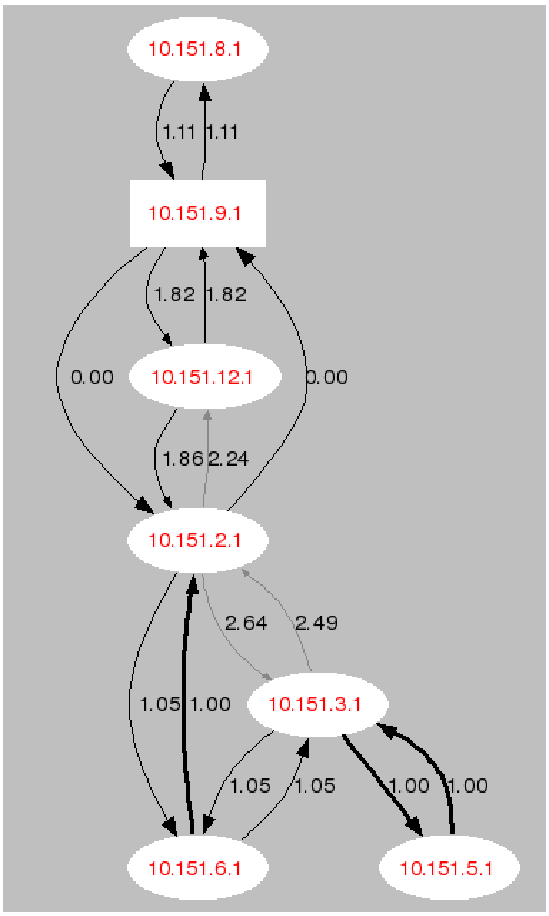


Figura 13: Qualidade dos enlaces da rede interna

A rede interna foi instalada segundo a viabilidade de localização em termos de segurança, energia e qualidade dos links. Procurou-se montar uma topologia com links perfeitos adjacentes e com influência de sinais não desejados. Em vista disso, na rede interna, na maioria das vezes os nós falam entre si pelos caminhos desejados.

1.4.2 Rede externa

A rede externa foi criada para a validação do protótipo como um produto pronto para ser colocado em produção. O planejamento foi feito ao longo dos seis primeiros meses do projeto onde diversos anúncios foram colocados através de diferentes meios (painéis, panfletos, internet e comunicados aos professores).

Em resposta à divulgação, foi montado um banco de voluntários que foram mapeados e analisados através de visitas nos locais. Nas visitas eram avaliados a viabilidade de espaço para instalação, visada direta para algum outro ponto, estado dos dutos para passagem de cabos, viabilidade de passagem do cabo pelo lado de fora e autorização do síndico e condôminos. A segunda fase consistiu na instalação do protótipo. Vários eventos não esperados ocorreram durante o processo como danos nos cabos de RF (RGC213) devido aos ventos fortes, falta de comunicação entre o nós desejados, excesso de sinal interferente operando na mesma faixa e até mesmo sinais refletidos por outros pontos que acabavam se tornando prejudiciais à topologia esperada.

Em vista destes fenômenos, o projeto que antes contava com um modelo utilizando apenas antenas omni-direcionais passou a usar antenas direcionais, porém colocadas de maneira a não interferir na possibilidade de crescimento da rede, que é o propósito de uma rede mesh.

Existem atualmente 6 pontos instalados nos topos dos prédios de seis alunos voluntários. O primeiro ponto instalado foi o gateway localizado no topo do prédio do Bloco E do campus da Praia Vermelha da UFF. A partir de abril novos pontos foram colocados com a rede em funcionamento começando pelos pontos 1 e 3 no mapa (vide Figura 14) que são os principais pontos de saída da UFF, sendo eles os responsáveis para que os outros pontos sejam alcançados. Hoje todos os pontos possuem plena conectividade com o gateway, permitindo a todos os voluntários trafegarem na Internet com taxas mínimas de 500 Kbps e máxima de até 12Mbps.

A topologia da rede externa pode ser visualizada nas Figuras 14 e 15.

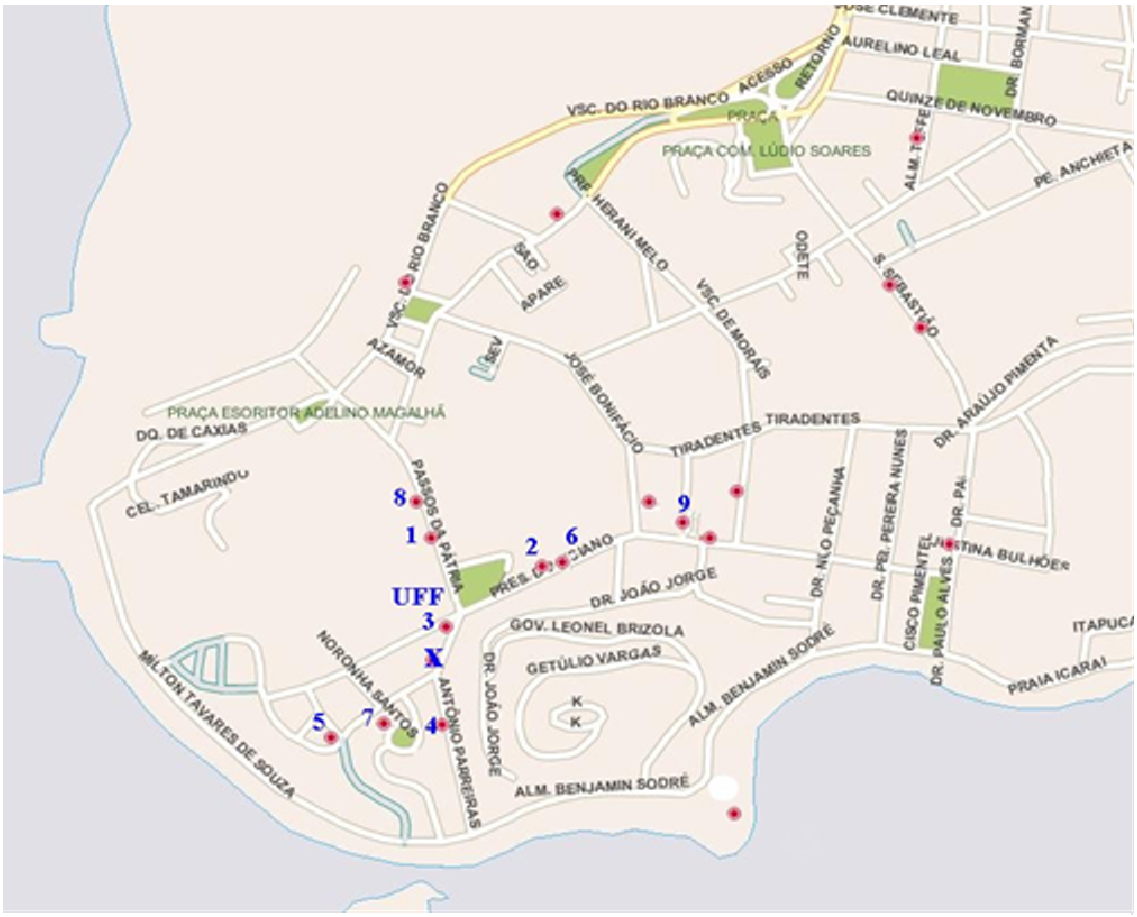


Figura 14: Topologia da rede externa

Na Figura 14 podem ser vistos todos os voluntários selecionados para participarem da rede mesh externa. Atualmente as instalações estão nos seguintes pontos:

- UFF: Gateway, 10.151.7.1
- Ponto 1 no mapa: Endereço 10.151.4.1
- Ponto 3 no mapa: Endereço 10.151.1.1
- Ponto 8 no mapa: Endereço 10.151.11.1
- Ponto 5 no mapa: Endereço 10.151.14.1
- Ponto 2 no mapa: Endereço 10.151.13.1

Na Figura 15 podemos ver os pontos em foto do Google Earth e as ligações mais freqüentes que ocorrem entre os nós.

Figura 15: Visão da rede externa por foto do Google Earth

A Figura 16 exibe a topologia da externa mostrando os valores da métrica ETX para cada enlace.

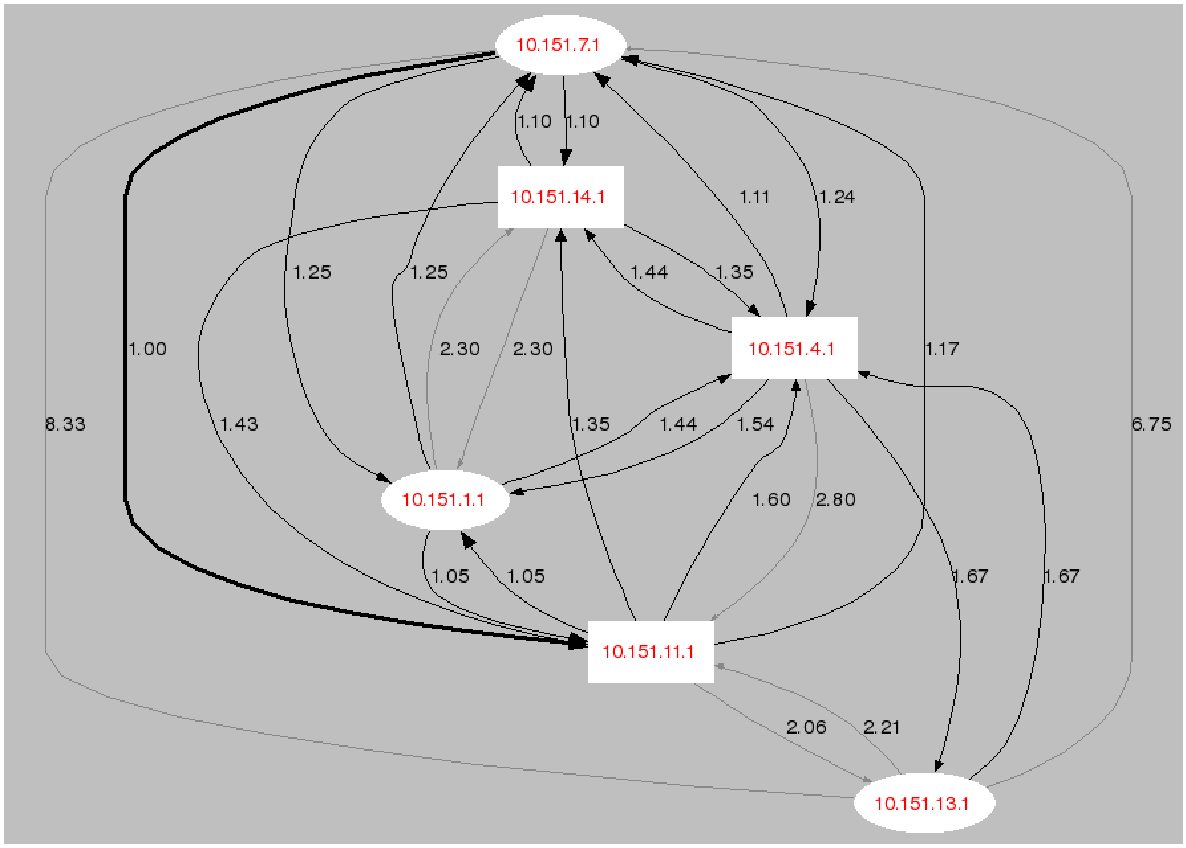


Figura 16: ETX entre os enlaces da rede externa

1.5 Gerência da Rede Mesh

Atualmente o projeto conta com quatro dispositivos básicos para a realização dos processos de gerência:

- Visualização da qualidade dos enlaces entre os nós,
- Visualização de informações internas dos nós,
- Visualização de estatísticas dos acessos dos usuários,
- Obtenção de dados filtrados dos acessos e gerência dos nós e usuários cadastrados.

Todas as ferramentas estão disponíveis no servidor web através da ação de cgi's em conjunto com os plugins do olsr e da ferramenta de autenticação *wifidog* instalada no servidor de autenticação.

1.5.1 Visualização da qualidade dos enlaces

O olsr possui diversos *plugins* para ações diversas. Um destes plugins é o "olsrd_dot_draw". Este plugin age abrindo uma porta dentro do nó que o executa para acesso via telnet e transferência de dados relativos à qualidade dos enlaces e das sub-redes exportadas por cada roteador mesh.

O plugin realiza apenas a abertura da porta e o envio das informações. Aliado ao plugin, utilizamos um programa escrito em *perl* chamado "olsr-topology-view.pl". É um programa simples e de código aberto que pode ser obtido na url http://meshcube.org/meshwiki/OlsrTopologyVisualization. Este programa age juntamente com os aplicativos Linux *GraphViz* e *ImageMagick,* ambos de código aberto e disponíveis na Internet (www.graphviz.org e www.imagemagick.org). O programa em perl recebe os dados do roteador que executa o plugin e gera figuras da topologia da rede mesh que são atualizadas automaticamente.

O GT customizou o programa em perl para exibir as informações dos nós sem as suas sub-redes e com alterações visuais na criação dos desenhos. Aliado a estas ferramentas, o GT desenvolveu um cgi para que os gráficos sejam apresentados via web (Figura 16).

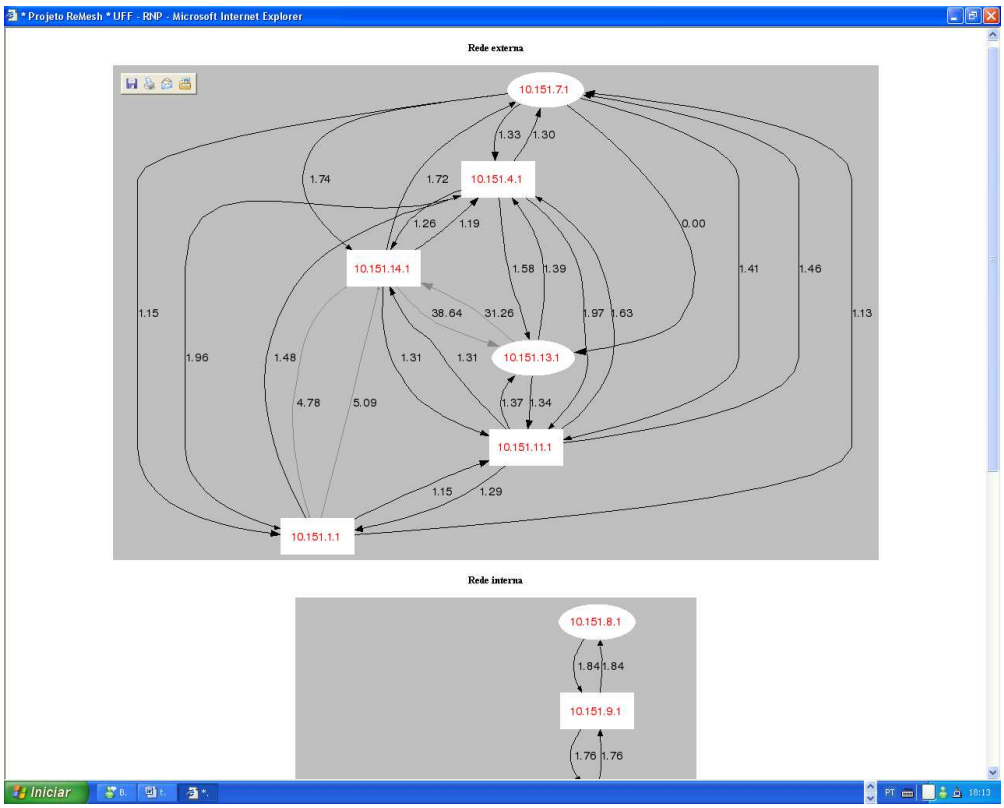


Figura 17: Ferramenta desenvolvida pelo GT que permite a visualização da topologia da rede em tempo real através do servidor web

Através desta ferramenta é possível um acompanhamento em tempo real das redes interna e externa. Através dela podemos saber se o nó está ativo, com quais nós ele está se

comunicando e a qualidade do enlace. Além disso, é muito útil no momento de novas instalações onde, a partir do momento em que o nó é ativado, ele entra automaticamente na topologia permitindo a verificação do estado do novo nó dentro da rede.

1.5.2 Visualização de informações dos nós

Através de um outro plugin do olsr, o *"olsrd_http_info"*, cada nó exibe suas informações através de acesso web em qualquer porta que seja configurada. As informações disponibilizadas são: sub-redes que o nó exporta, seus vizinhos, seus MPR's, a qualidade da comunicação dele com os outros nós, os endereços de suas interfaces (sem fio e ethernet) e a sua tabela de rotas (Figura 17).

Para permitir que tais informações se tornassem acessíveis através do servidor web, duas medidas foram tomadas: Primeiramente, criamos uma série de regras de roteamento através da ferramenta *iptables* dentro do nó gateway para que os roteadores pudessem ser acessados de fora da rede mesh. Em seguida, como o acesso deveria ser feito através de portas altas (não reservadas a serviços básicos), não era possível que um usuário conseguisse visualizar a página a partir de sua casa devido ao firewall da UFF. Para contornar isto, o GT criou um outro cgi que atua dentro do servidor web, ou seja, dentro da UFF e não bloqueado pelo firewall, abrindo uma conexão interna aos nós nas portas altas e exibindo as informações na porta 80 (serviço http) normalmente.

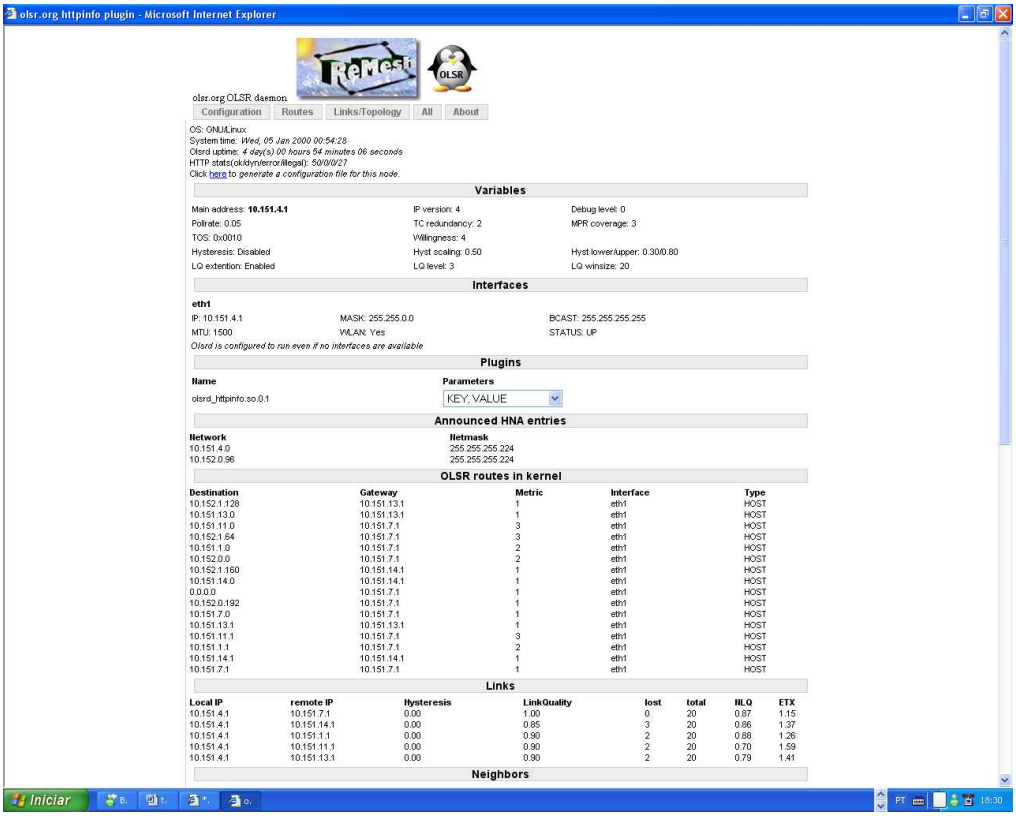


Figura 18: Informações de cada um dos nós disponíveis através do servidor web

1.5.3 Visualização de estatísticas dos acessos dos usuários

A ferramenta *wifidog* possui uma interface de gerência de acesso exclusivo ao gerente do servidor de autenticação. Esta interface permite a visualização de estatísticas sobre os acessos bem como a criação e exclusão dos nós da rede. Com o objetivo de tornar determinadas informações públicas e de fácil acesso, foi criado uma pequena página em PHP dentro do servidor de autenticação que fornece somente os dados que desejamos informar. Para tanto, quando acessada diretamente a página não disponibiliza informação nenhuma, porém, dentro do servidor web, colocamos a URL com os valores desejados das variáveis do programa PHP para que as informações fossem exibidas.

As informações disponibilizadas são diferenciadas entre rede interna e rede externa. Elas são:

- Os 10 usuários que mais consomem banda
- Os 10 usuários mais freqüentes
- Os 10 usuários que mais acessaram a rede por nós distintos
- Gráficos da quantidade de acessos por hora, dia da semana e mês
- Os nós mais visitados
- Gráficos do número de novos usuários registrados por mês e acumulativo
- Número de novos usuários por nó de acesso

A Figura 19 ilustra as estatísticas sobre usuários e acessos à rede mesh.

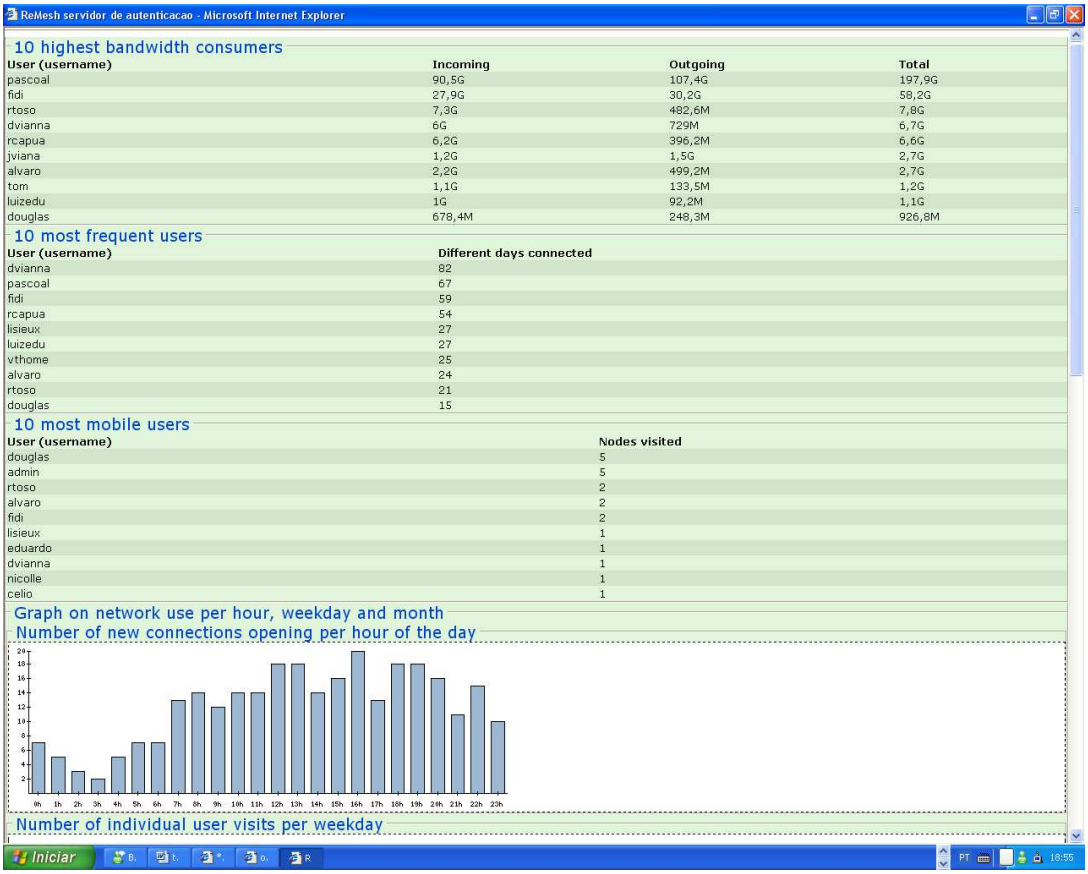


Figura 19: Visualização de estatísticas de acessos disponível no servidor web

1.5.4 Gerência administrativa dos nós e dos usuários

Como dito anteriormente, o servidor de autenticação está instalado com o aplicativo *wifidog* (explicado detalhadamente na seção 1.6). Além de controlar o acesso à rede, o *wifidog* é o responsável pela coleta das estatísticas dos acessos e pela gerência dos nós participantes da rede mesh. Em sua interface administrativa, o *wifidog* permite a adição, a exclusão e a edição dos nós participantes da rede.

Quanto ao acesso aos usuários, o *wifidog* permite em sua interface de administração a adição de novos usuários e o acesso às estatísticas de acesso filtradas por usuário, nó e/ou data. Além das informações já descritas no subitem anterior, é possível obter informações extras como os usuários que estão *online* e a quantidade de dados trafegados na conexão atual. É possível também acessar o *log* de todas as outras conexões realizadas, fornecendo o usuário, data e duração do acesso e a quantidade de dados trafegados.

Quanto ao acesso direto aos usuários como exclusão ou suspensão de uma dada conexão, o nosso módulo do *wifidog* instalado no servidor de autenticação não fornece uma interface web direta para a realização destas ações. Para tanto, quando é necessário realizar

tais ações administrativas, o GT age diretamente no banco de dados do servidor de autenticação. As duas ações mais freqüentes são: suspensão temporária de usuários e de conexões ativas. Elas são feitas através dos seguintes comandos SQL:

1. Primeiro deve-se acessar o banco no servidor de autenticação: */usr/local/pgsql/bin/psql wifidog.* Onde “wifidog” é o nome do banco criado pelo servidor de autenticação.

2. Dentro do banco digite:

    a. “`UPDATE USERS SET ACCOUNT_STATUS=0 WHERE USERNAME!='admin';`”. Este comando, por exemplo, desabilita todos os usuários diferentes de admin. Para habilitar novamente basta executar o mesmo comando mudando o valor da coluna ACCOUNT_STATUS para 1.

    b. “`UPDATE CONNECTIONS SET TOKEN_STATUS='USED' WHERE TOKEN_STATUS='IN_USE' AND USER_ID=(SELECT USER_ID FROM USERS WHERE USERNAME!='admin');`”. Este comando irá desabilitar todas as conexões ativas que não estejam sendo realizadas pelo usuário “admin”.

3. O módulo do *wifidog* foi customizado pelo GT para que, quando um usuário for apenas desabilitado, porém não excluído, ele visualize a tela exibida na Figura 20 ao tentar se conectar.

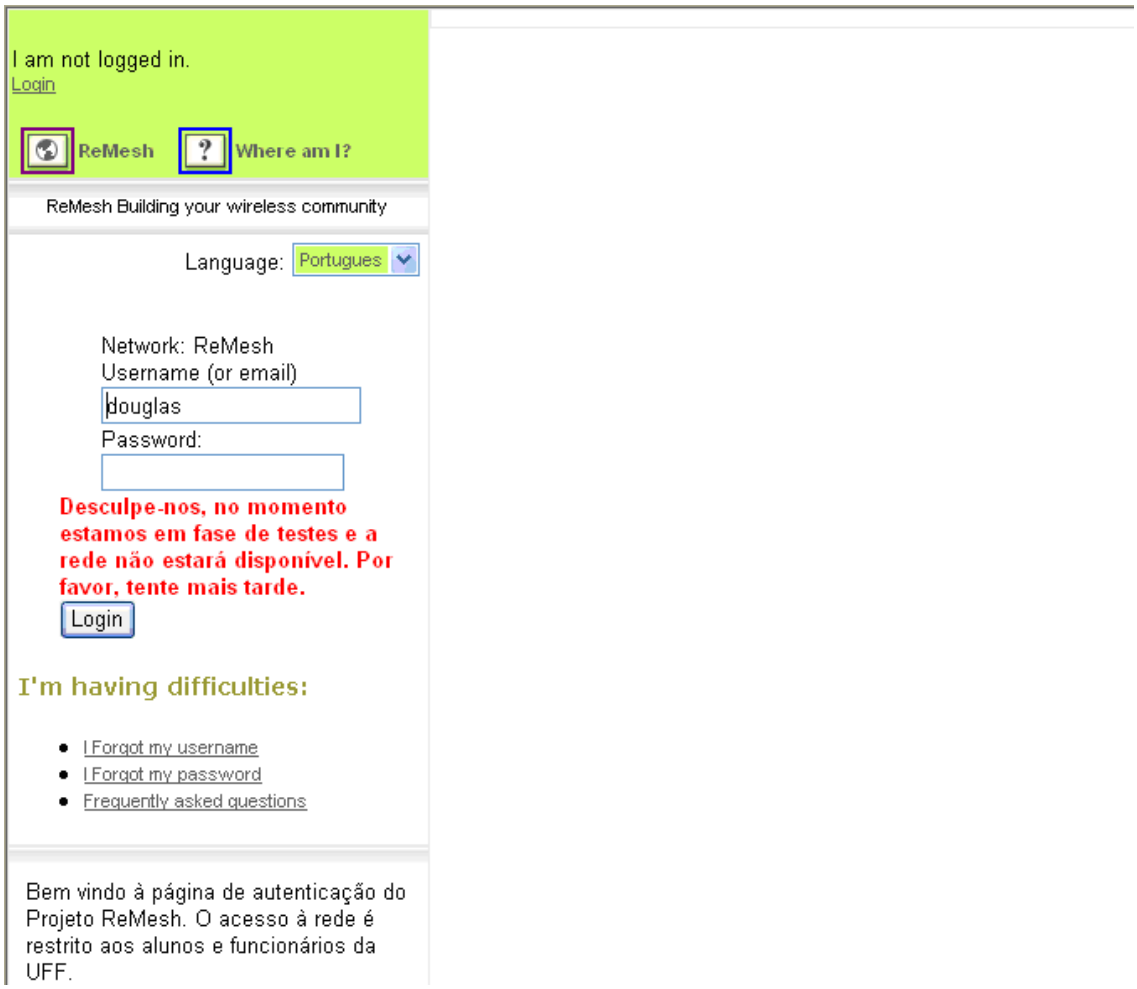


Figura 20: Tela de entrada do *wifidog* para usuários que foram desabilitados temporariamente.

Obviamente, demais ações, como exclusão permanente de usuários, podem ser feitas através de simples comandos SQL diretamente no banco de dados do servidor.

1.6 Módulo de Autenticação e gerência dos acessos: o *wifidog*

A rede mesh externa montada pelo GT-Mesh proporciona conectividade aos indivíduos voluntários do projeto cujos apartamentos possuem um ponto instalado. Contudo, sendo uma rede sem fio e sem criptografia (WEP), a princípio qualquer indivíduo com uma placa de rede sem fio poderia se associar a navegar utilizando o *link* da UFF. Com o objetivo de proporcionar segurança e controle ao acesso, foi instalada a solução *wifidog* no protótipo.

O *wifidog* é um framework que proporciona um esquema de segurança ao acesso mediante autenticação. O *wifidog* é um software aberto que pode ser encontrado em www.wifidog.org. Ele utiliza a técnica de *Captive Portal* para realizar a autenticação. A técnica de *Captive Portal* consiste na inserção de um *firewall* entre o usuário e o gateway de acesso. Toda vez que um pacote é encaminhado para o gateway, ele passa pela barreira. Um usuário não autenticado terá o seu pacote bloqueado pelo firewall.

Para que os pacotes do usuário possam fluir livremente até o gateway, ele deverá executar um browser de Internet e fazer uma requisição a uma URL qualquer. O pacote da requisição será bloqueado no gateway que deverá, em resposta a este bloqueio, redirecionar o cliente ao servidor de autenticação e, ao mesmo tempo, encaminhar o endereço IP e o MAC do usuário que está fazendo a requisição ao servidor de autenticação. Este servidor é encarregado de armazenar um banco de dados de todos os clientes cadastrados. Assim que o usuário chega à página inicial do servidor de autenticação, ele deverá entrar com um login e senha autorizados. O servidor liberará ou não o acesso deste cliente encaminhando uma mensagem ao gateway de acesso para abrir as portas do firewall para o determinado endereço IP e MAC. Com o firewall do gateway liberado, o cliente pode navegar sem interrupções em seu acesso. Além da funcionalidade de segurança, o módulo de autenticação pode ainda armazenar dados sobre o acesso do determinado cliente.

Existem hoje disponíveis diversas soluções para implantação de um esquema de *captive portal*. Na rede Mesh optamos por utilizar o *Wifidog* por ser um software *open source* (licensa GNU GPL). Ele é divido em dois módulos: um binário escrito em C, presente no gateway que deve ser inicializado junto com o sistema; e o outro que consiste em uma página dinâmica escrita em PHP presente no servidor de autenticação.

O funcionamento de *wifidog* pode ser resumido pelas Figuras 21, 22 e 23.

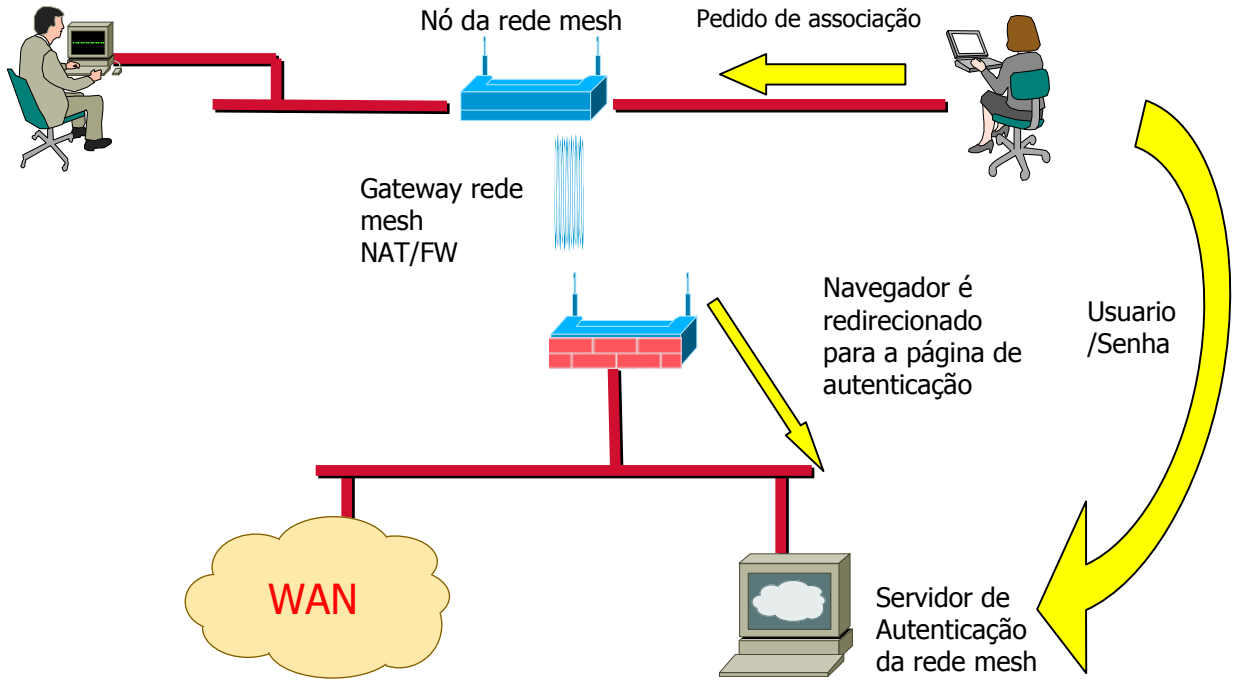


Figura 21: *Wifidog* – cliente inicia uma conexão

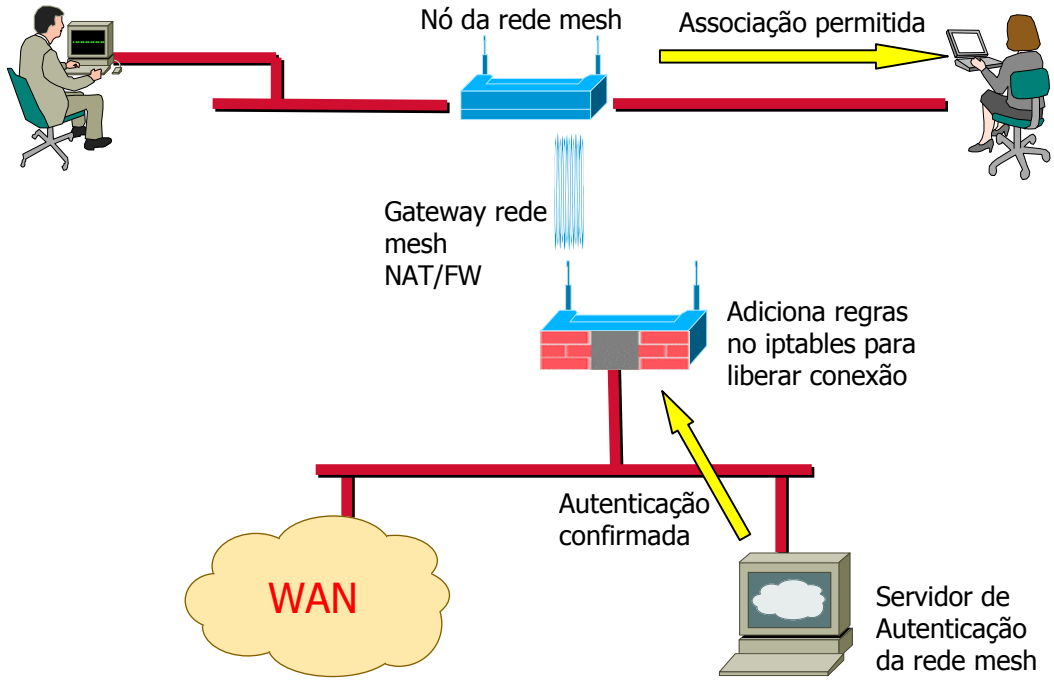


Figura 22: *Wifidog* - Processo de autenticação e liberação

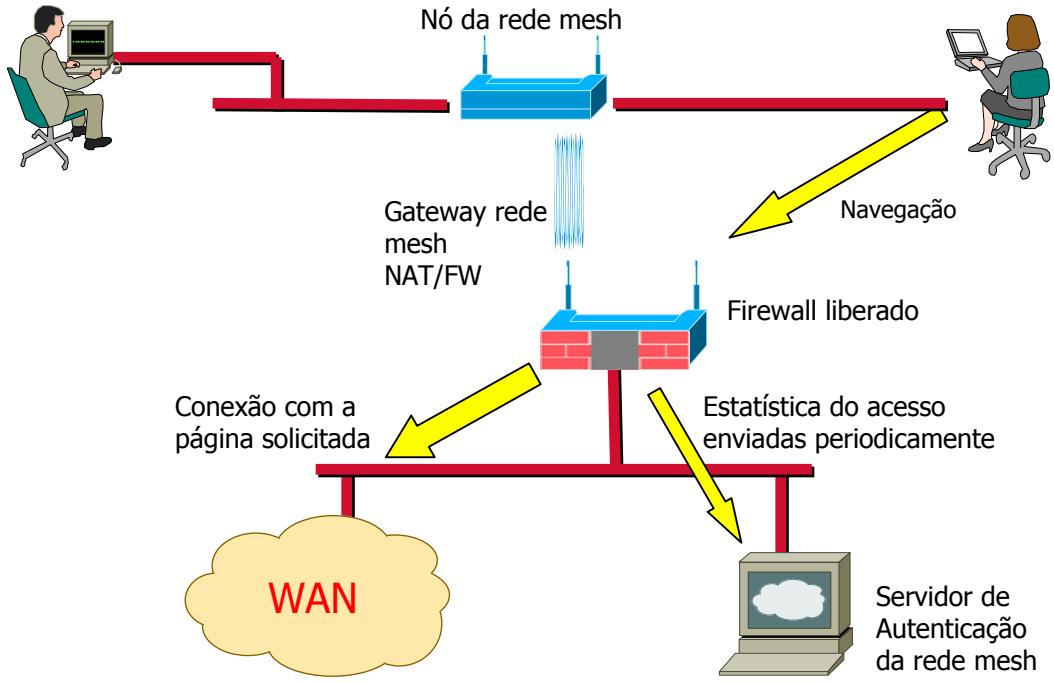


Figura 23: *Wifidog* - cliente autenticado e navegação liberada

Na Figura 21 podemos ver o início do processo. Um cliente recebe o endereço por DHCP, neste momento qualquer serviço na rede estará bloqueado. Assim que ele executar um

browser e tentar navegar para qualquer URL ele será direcionado ao servidor de autenticação. Assim que é autenticado (Figura 22) o servidor de autenticação envia ao gateway a informação de que para aquele ip e MAC todas as portas estarão liberadas. O cliente recebe a notificação e poderá então navegar diretamente na Internet sem ser bloqueado (Figura 23), ainda, ao longo da conexão o módulo binário do *wifidog* envia dados do acesso para que estatísticas do uso da rede possam ser geradas. Esse envio de informação é transparente para o cliente.

No protótipo desenvolvido, deveríamos conseguir autenticar não somente os clientes que estão ligados à rede por fio através dos nós instalados nos prédios, mas também possibilitar a autenticação dos clientes que se ligam à rede sem fio. O *wifidog* autentica apenas uma interface de entrada para uma interface de saída, ou seja, se estamos autenticando clientes que se ligam ao roteador pela interface *ethernet*, não podemos autenticar os que se conectam pela interface *wifi.* Para resolver isto, tivemos que propor uma arquitetura particular ao projeto.

Primeiramente devemos dividir os nós da rede entre nós clientes e o nó *gateway*. Os nós clientes são aqueles que serão instalados nos apartamentos possibilitando a ligação por fio dos computadores através das portas *ethernet* do roteador. O nó gateway é o que encaminha todo o tráfego da rede mesh para a Internet, ou seja, é o que deve ter a sua porta *ethernet* ligada a WAN. O gateway é o nó que está ligado ao provedor de acesso, ou seja, a WAN. Nele não ligamos computadores clientes nas portas *ethernet*, não exigindo, portanto, a realização de autenticação para esta interface. Desta forma, podemos eleger o gateway como o nó que fará a autenticação dos clientes sem fio enquanto que os outros nós farão a autenticação dos clientes por fio. Isto é feito da seguinte maneira: cada nó cliente possui uma instalação do módulo binário original do *wifidog* configurado para autenticar a porta de entrada *ethernet* para a porta de saída *wifi*; o nó gateway possui uma versão alterada do módulo binário do *wifidog* configurado para autenticar a interface de entrada *wifi* para a interface de saída *ethernet* WAN. A alteração do módulo (descrita no Apêndice 1) não filtra os clientes pelo MAC de origem, pois um cliente sem fio que se autentica em um nó cliente da rede não terá o seu MAC estampado no pacote roteado. Além disso, o nó gateway terá que ter regras específicas para não barrar os pacotes provenientes de clientes da rede com fio que já foram autenticados nos seus nós de origem.

A Figura 24 representa esquematicamente a arquitetura.

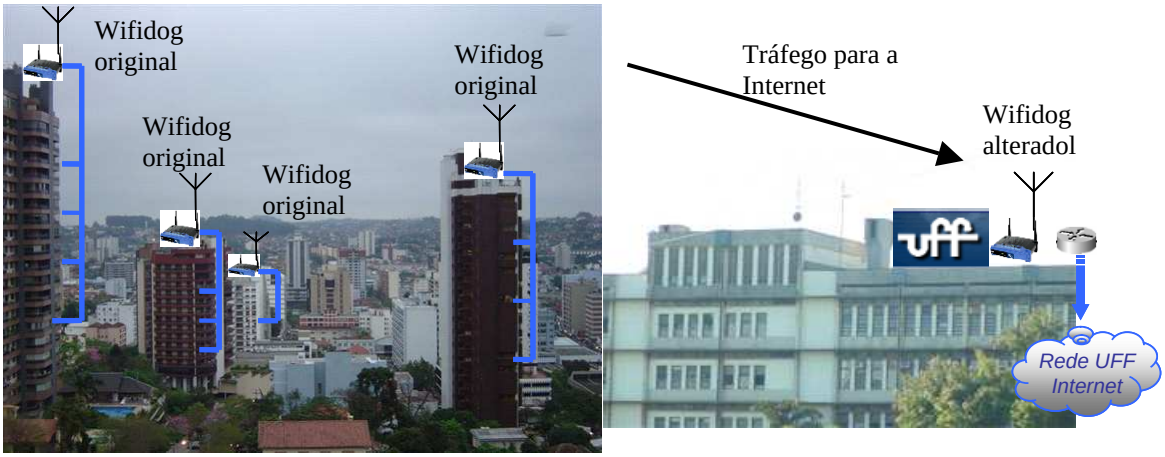


Figura 24: Esquematização da instalação do wifidog na rede Mesh da UFF

Para um cliente sem fio, o esquema representado na Figura 21 é um pouco diferente. Ao pedir um IP para um nó próximo por DHCP, ele estará habilitado a trafegar dentro da rede mesh, ou seja, ele pode ter acesso a qualquer máquina da rede pois não foi necessária qualquer autenticação para isto. Contudo, ao sair da rede mesh para a Internet, ele deverá passar pelo gateway, neste momento ele será encaminhado para o servidor de autenticação conforme a Figura 25.

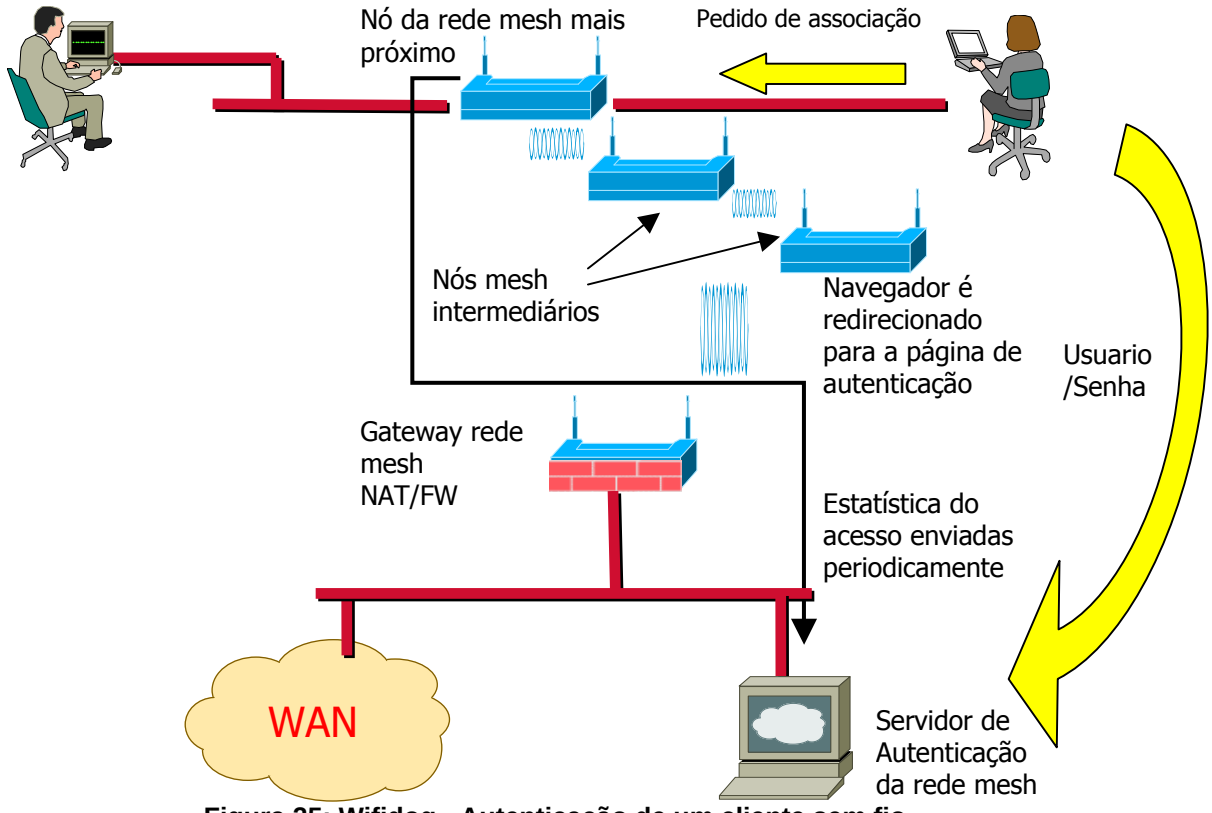


Figura 25: Wifidog - Autenticação de um cliente sem fio

O módulo binário instalado nos nós clientes da rede (nós que não são gateway) consiste em:

- Binário executável: /usr/bin/wifidog
- Scripts de inicialização: /etc/init.d/S65wifidog e /usr/bin/wifidog-init
- Arquivo de configuração: /etc/wifidog.conf onde a interface de entrada é a ethernet LAN e a de saída é a wifi
- Bibliotecas necessárias:/lib/libpthread.so.0; /usr/lib/libhttpd.so.0.0.0; /usr/lib/stdc++.so.6.

O módulo binário instalado no nó gateway da rede consistem em:

- Binário executável alterado: /usr/bin/wifidog
- Scripts de inicialização: /etc/init.d/S65wifidog e /usr/bin/wifidog-init; o primeiro é alterado e o segundo é o mesmo que o original.
- Arquivo de configuração: /etc/wifidog.conf onde a interface de entrada é a wifi e a de saída é a ethernet WAN
- Bibliotecas necessárias (iguais):/lib/libpthread.so.0; /usr/lib/libhttpd.so.0.0.0; /usr/lib/stdc++.so.6.
- Arquivo /etc/wifidog_extra; este arquivo é executado pelo S65wifidog e escreve nas tabelas corretas as permissões para que os clientes com fio não sejam bloqueados.
- Arquivo /etc/init.d/S45firewall devidamente configurado. Este arquivo adiciona regras extras no firewall do gateway para que dados necessários às ferramentas de gerência da rede (tais como as informações geradas pelo módulo *httpinfo* do olsr) não sejam bloqueadas, porém permitidas somente nas portas em que são requeridas e somente para a máquina que as exibe (no caso é o servidor web).

O módulo PHP deverá ser instalado em uma máquina comum com sistema operacional Linux. Para o seu funcionamento ele depende dos seguintes itens:

- PHP versão 5.xx
- Banco de dados Postgres
- Servidor http Apache versão 2.xx
- Módulo php_xml para suporte a dialetos XML (RSS)
- Módulo php_gettext para suporte a múltiplas línguas
- Módulo php_mbstring para suporte ao esquema de autenticação e para RSS
- Módulo php_mcrypt, php_xml e php_mhash para suporte ao RADIUS (opcional)
- Módulo php_radius para utilização de RADIUS (opcional)

- Módulo php_gd, php_png e php_jpg para construção dos gráficos (opcional)
- Conjunto das páginas desenvolvidas pelo GT com as alterações realizadas
- Configuração das páginas para definição do banco de dados e da senha de administração.

O módulo do *wifidog* em PHP consiste em um conjunto de páginas dinâmicas que devem ser instaladas no servidor de autenticação. Elas não somente permitem a autenticação dos usuários, mas também a gerência dos mesmos e dos diversos nós instalados. Uma vez instalado no servidor com todas as aplicações em funcionamento, os nós da rede deverão ser cadastrados para que a gerência possa diferenciar os usuários de acordo com os nós de acesso (Figura 26 e Figura 27).

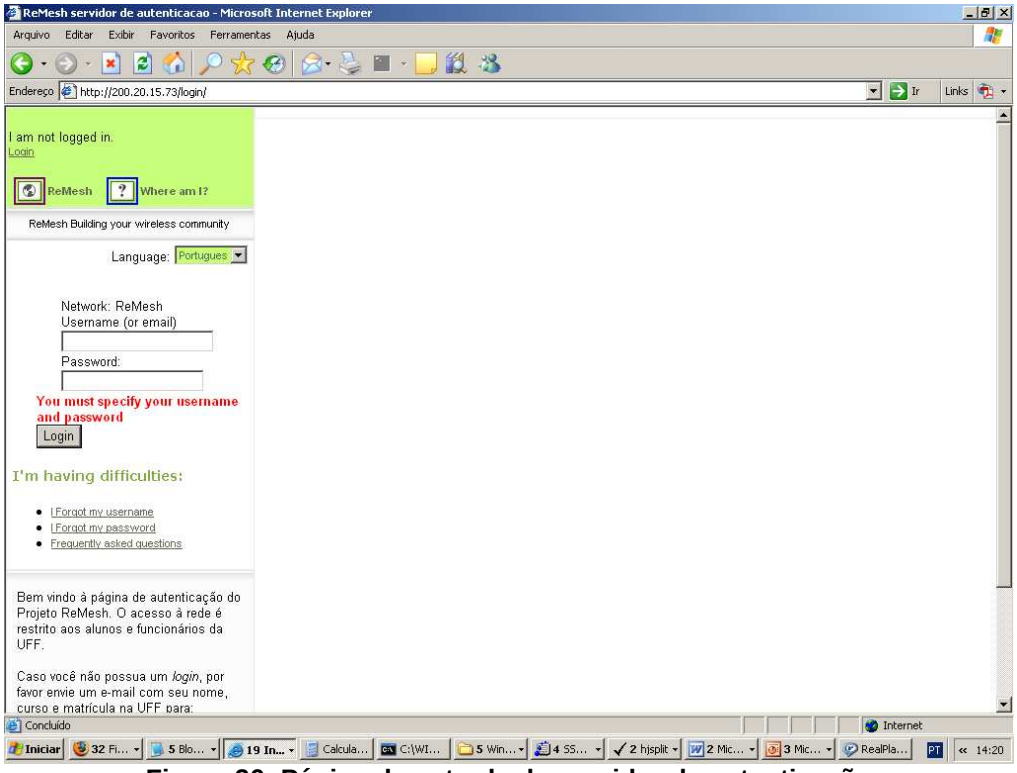


Figura 26: Página de entrada do servidor de autenticação

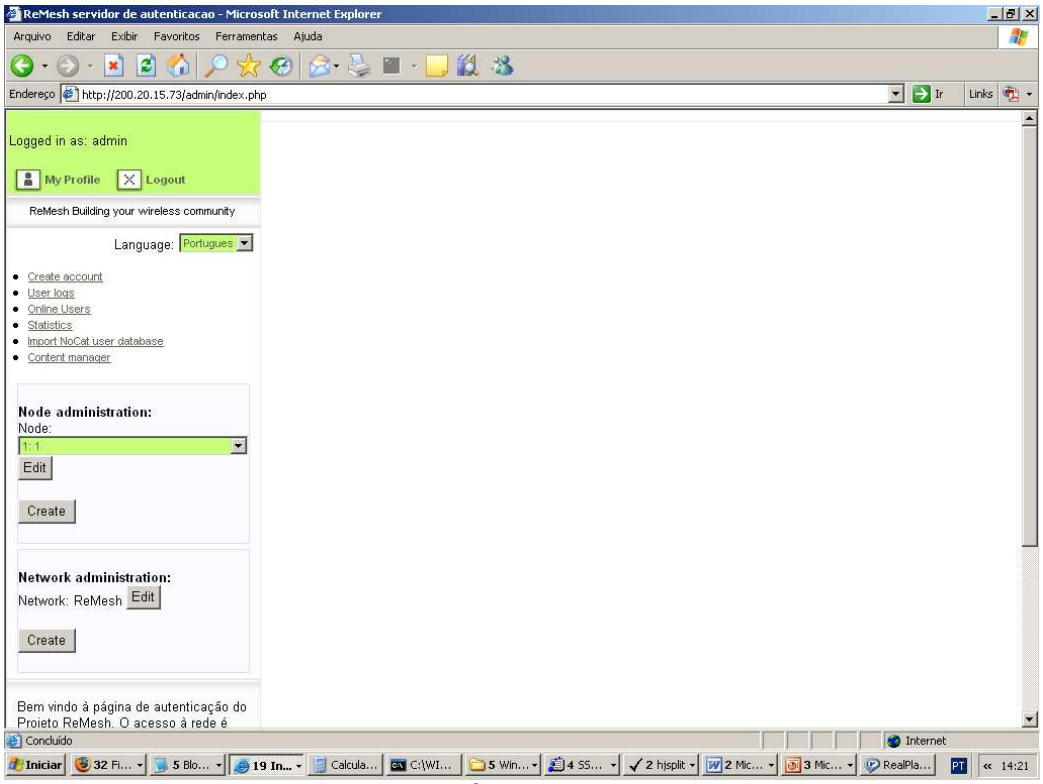


Figura 27: Página de gerência com as opções à esquerda

Ao se *logar* como administrador, dentre as opções disponíveis está a criação de nós. Cada nó da rede Mesh deverá ser cadastrado. Nós utilizamos como identificador o terceiro *byte* do endereço do nó, por exemplo o nó 10.151.20.1 será identificado como "*ID* 20" (Figura 28). Esta identificação é importante, pois é o primeiro parâmetro a ser configurado no arquivo *wifidog.conf* do módulo binário que é executado dentro dos roteadores (Figura 29). Assim que o nó for criado, ele poderá ser alterado na mesma página. O cadastro de usuários também é feito nesta página e as informações necessárias são o login, a senha e email, que não pode ser repetido.

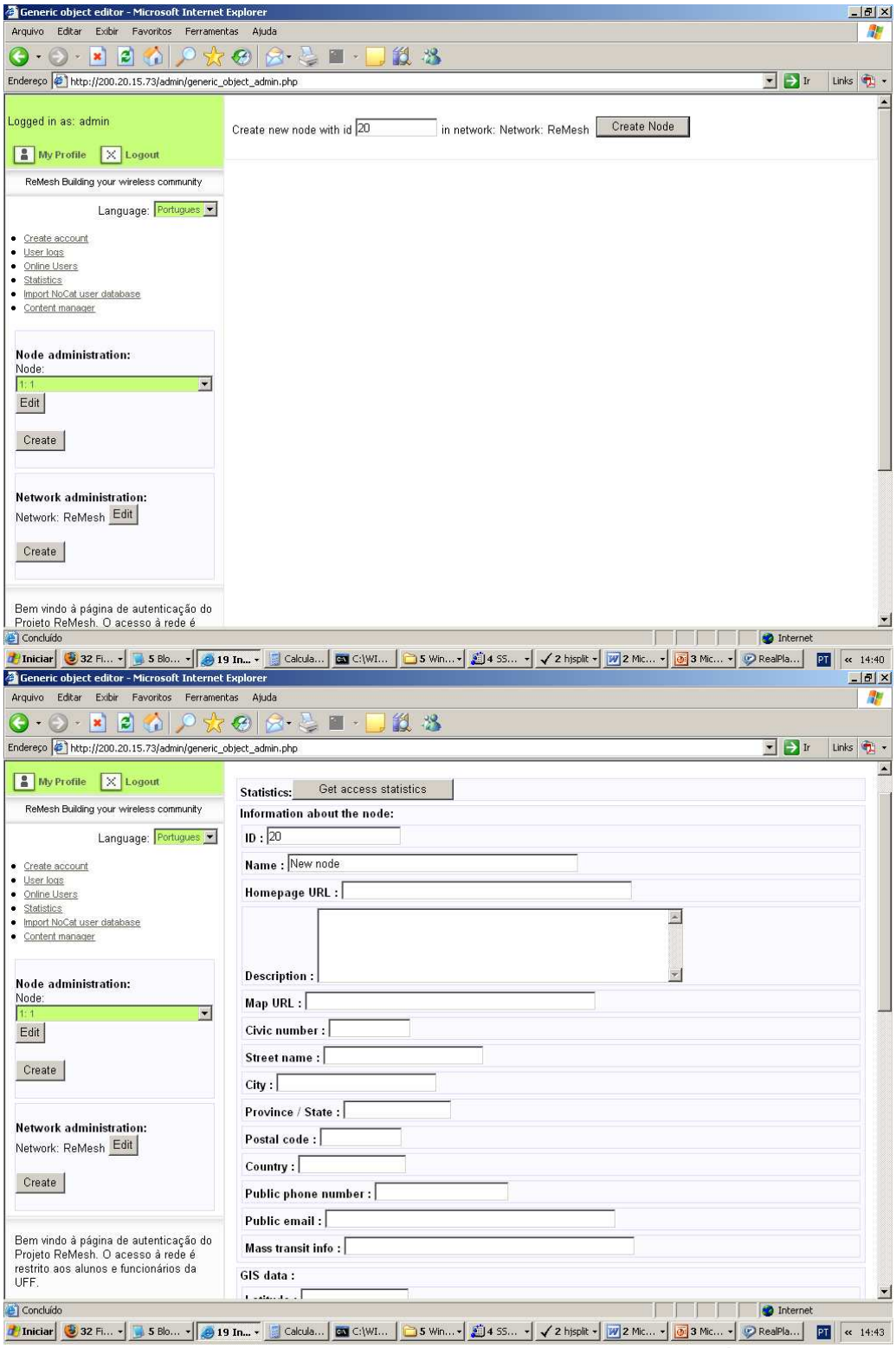


Figura 28: Wifidog - Criação de um novo nó

```
# WiFiDog Configuration file

# Parameter: GatewayID
# Default: default
# Optional but essential for monitoring purposes
#
# Set this to the template ID on the auth server
# this is used to give a customized login page to the clients
# If none is supplied, the default login page will be used.

GatewayID 20

# Parameter: ExternalInterface
# Default: NONE
# Optional
#
# Set this to the external interface.  Typically vlan1 for OpenWrt, and eth0 or ppp0 otherwise

ExternalInterface vlan1

# Parameter: GatewayInterface
# Default: NONE
# Mandatory
#
# Set this to the internal interface.    Typically br0 for OpenWrt, and eth1 otherwise

GatewayInterface eth1

# Parameter: GatewayAddress
# Default: Find it from GatewayInterface
# Optional
#
```

Figura 29: Wifidog- Arquivo de configuração *wifidog.conf* localizado dentro dos roteadores no diretório /etc

Originalmente, as páginas do módulo em PHP do *wifidog* possuem uma dinâmica diferente da que fora adotada na construção da rede Mesh. Abaixo listamos nossas modificações:

- Logotipo de exibição foi trocado para exibirmos o do GT (Figura 30)
- Qualquer indivíduo cadastrado pode cadastrar novos usuários. Na nossa versão somente o administrador pode fazer isto.
- Cada novo usuário ao ser cadastrado tem 20 minutos para acessar o seu correio eletrônico e validar o email cadastrado. Na nossa versão o usuário cadastrado não recebe email é está plenamente autorizado para utilização da rede.
- Um mesmo *login* pode ser usado em diferentes máquinas ao mesmo tempo. Na nossa versão, o usuário que entra com um *login* que já está em utilização desabilita o anterior e recebe uma notificação na página de entrada (Figura 31).
- Dados sobre estatísticas dos usuários podem ser visualizados somente pelo administrador. Na nossa versão nós criamos uma página em PHP que age em conjunto com o servidor web do projeto para que algumas informações possam ser disponibilizadas livremente.

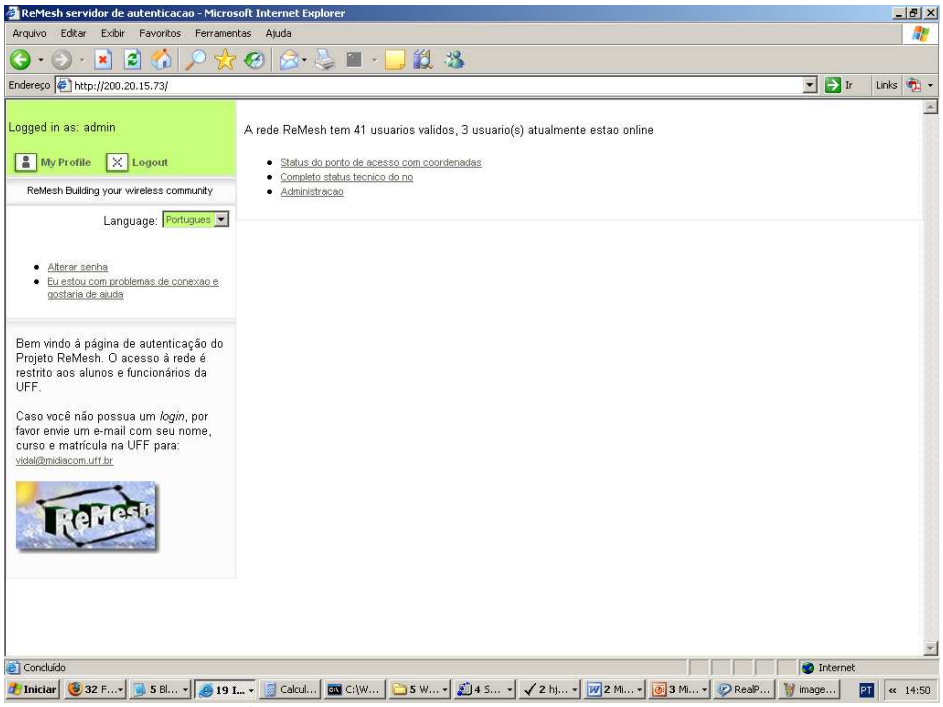


Figura 30: Wifidog - A página de abertura foi personalizada para exibir o logo do GT

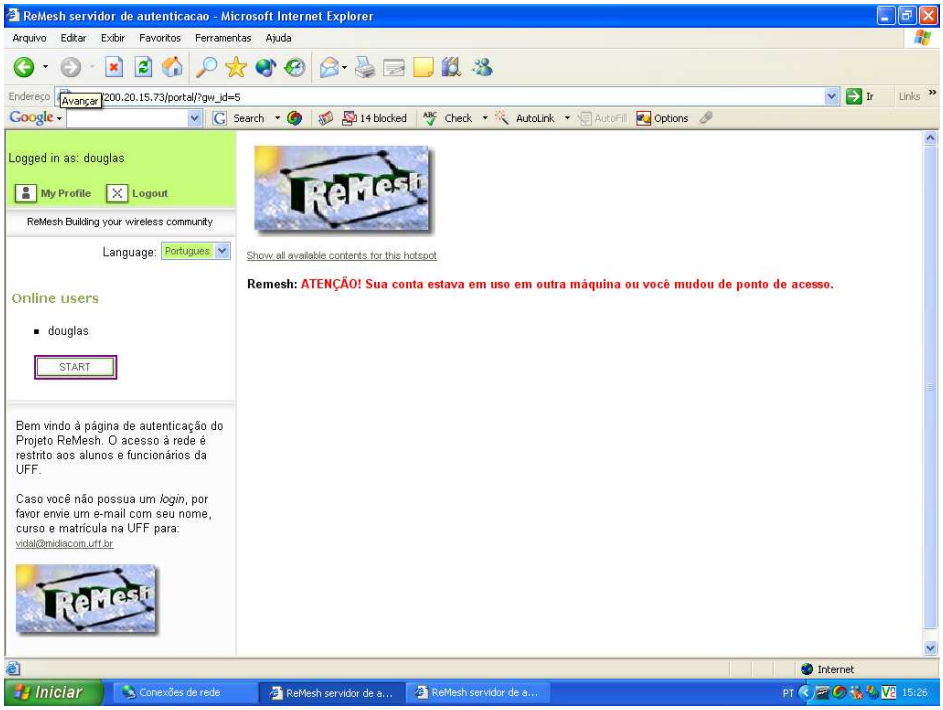


Figura 31: Usuário entrando com o mesmo login em duas máquinas diferentes. A máquina anterior será agora desabilitada.

Além disso, o *wifidog* permite a visualização dos *logs* dos usuários, visualização dos usuários que estão *online* e a quantidade de dados que estão trafegando e a visualização de estatísticas de uso (Figura 32).

As estatísticas de uso podem ser filtradas por usuário, nó e data. São dados disponíveis apenas para o administrador. Contudo, na nossa versão ampliamos a funcionalidade de visualização de determinados dados para qualquer pessoa que acesse a página web do projeto.

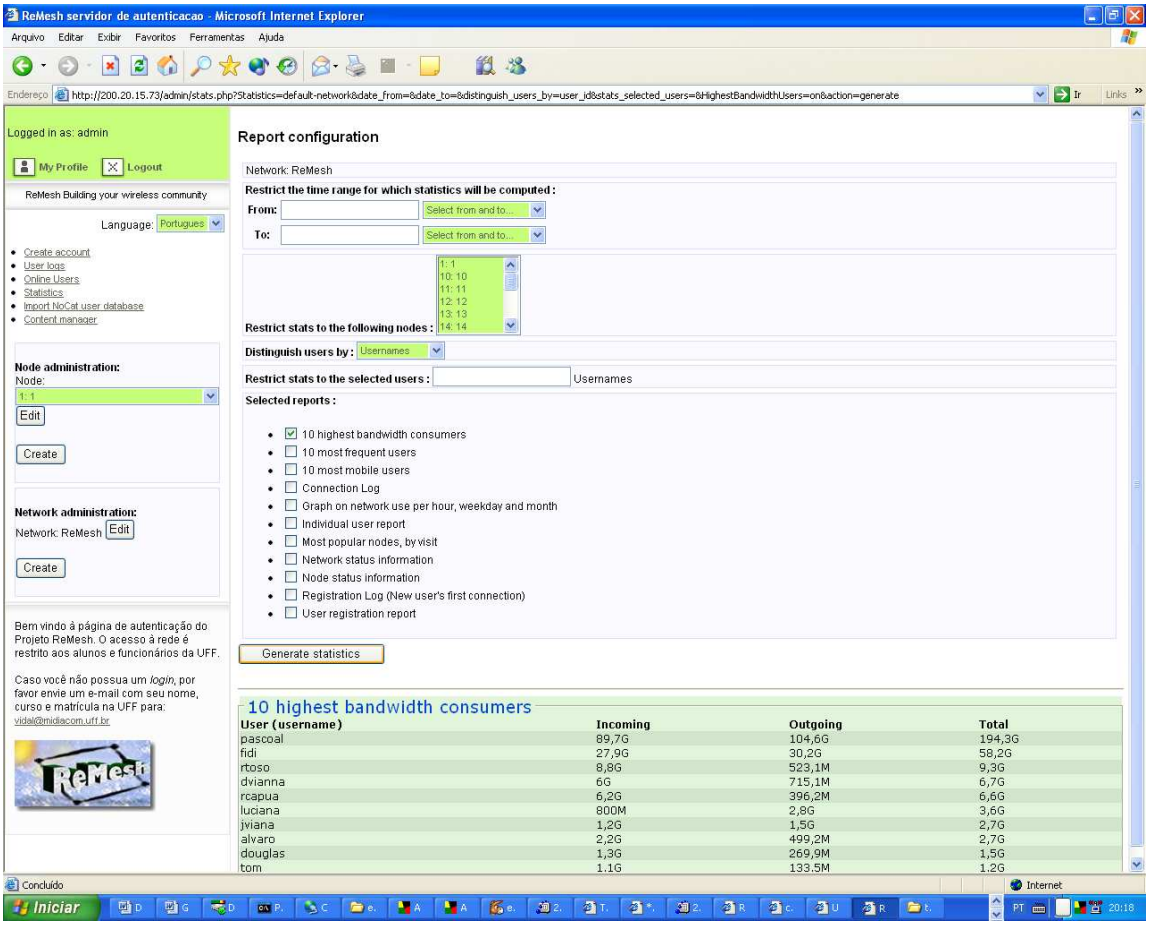


Figura 32: Interface de gerência do Wifidog

1.7 Framework de Suporte a QoS

Como descrito no relatório RT2, o *framework* de suporte a QoS em redes *Mesh* utiliza reserva de recursos no *backbone* de roteadores sem fio, e medições de perda de pacotes e de *jitter* no cliente. As medições desses parâmetros permitem que o processo de adaptação seja acionado, caso a qualidade da transmissão seja deteriorada. O processo de adaptação possui níveis diferenciados, e consiste em baixar a taxa de transmissão no servidor. Cada nível de adaptação corresponde a uma taxa de transmissão. Quanto menor o nível, maior a taxa.

Pelo fato dos roteadores utilizarem a técnica de reserva e o cliente realizar medições, o protótipo desenvolvido é divido em duas partes distintas, mas que trabalham em conjunto. Primeiro será explicado como são feitas as reservas nos roteadores, e depois como são medidos os parâmetros no cliente e acionada a adaptação.

1.7.1 Framework nos roteadores

Os roteadores tiveram o seu *firmware* alterado para o OpenWRT, que é um sistema operacional baseado em Linux e desenvolvido para dispositivos com limitações de processamento e memória, como os roteadores sem fio. O OpenWRT possui algumas ferramentas que o Linux possui, entre elas a ferramenta *Traffic Control*, ou simplesmente tc.

A ferramenta tc faz parte do pacote iproute2. O *site* [LARTC 2005] apresenta uma comunidade que dá suporte ao uso desse pacote e das ferramentas existentes nele. Nesse *site* também foi encontrado o documento [Hubert et al. 2003], que foi muito importante para o entendimento da ferramenta tc e de seus métodos.

O tc permite criar disciplinas de filas (*Queueing Disciplines* ou *qdisc*) para o gerenciamento de largura de banda em um nó. Essas filas são criadas mais usualmente para organizar as informações que são transmitidas por um nó, apresentando poucas funcionalidades para lidar com os pacotes que estão chegando no nó.

As várias classes de uma *classful qdisc* classificam os diferentes tipos de tráfego que passam por um dispositivo. Essas classes podem conter outras *qdisc* ou ainda outras classes. Assim, uma classe possui como "pai" uma *qdisc* ou outra classe. Uma classe folha é aquela que não contém filhos. Essa classe filha contém uma *qdisc* anexada a ela. Essa *qdisc* anexada a uma classe folha é responsável por enviar os dados provenientes dessa classe. Quando se é criada uma classe, uma *fifo qdisc* é anexada a ela por padrão. Se uma classe filha for anexada a ela, essa *qdisc* é removida. Para as classes folhas, essa *fifo qdisc* pode ser substituída por qualquer outra *qdisc*.

Para que o entendimento da ferramenta tc se torne mais claro é necessário explicar alguns termos que serão utilizados. Como já foi dito, uma *Queueing Discipline*, ou *qdisc*, é um algoritmo usado para o gerenciamento de filas de um dispositivo, podendo ser filas de entrada ou de saída de dados. Com uma *qdisc* é possível alterar o modo com que os dados que chegam ou que partem de um nó serão tratados. A *qdisc* anexada diretamente no dispositivo é chamada de *root qdisc*. As *qdisc* podem ser classificadas como *classless* ou *classful*. Uma

classless qdisc é uma disciplina sem subdivisões internas que podem ser configuráveis. Já uma *classful qdisc* contém múltiplas classes internas.

Cada *classful qdisc* precisa determinar para que classe ela precisa enviar um pacote que chegou ao dispositivo e deve ser transmitido na rede. Isso é feito usando um classificador. Essa classificação pode ser feita usando filtros. Um filtro contém um determinado número de condições que devem ser consultadas quando um pacote precisa ser classificado e, se coincidirem com os dados do pacote, o filtro aponta uma classe para o qual o pacote deve ser enviado.

Uma *qdisc* pode, com a ajuda de um classificador, decidir que alguns pacotes precisam ser transmitidos mais cedo do que outros. Esse processo é chamado de escalonamento. Também é possível que seja necessário atrasar alguns pacotes antes que eles sejam transmitidos na rede, para que o tráfego apresente uma taxa máxima configurada. Esse processo é chamado de modelagem, e é feito nos pacotes que estão saindo de um nó. O processo de descartar pacotes para diminuir a velocidade do tráfego também é freqüentemente chamado de modelagem. Porém, o processo de atrasar ou descartar pacotes a fim de manter o tráfego abaixo da largura de banda configurada é chamado de policiamento. No Linux, o policiamento é feito apenas descartando pacotes, e não os atrasando.

Dentre todas as disciplinas apresentadas em [Hubert et al. 2003], foi escolhida a *classful qdisc Hierarchical Token Bucket*, ou HTB, por ser a abordagem mais apropriada quando se tem uma quantidade fixa de largura de banda que deseja-se dividir entre diferentes propósitos, dando a cada um uma largura de banda garantida. O HTB também permite que cada classe definida abaixo do *root qdisc* possa pegar emprestado largura de banda de outras classes que não estejam usam a banda que lhe foi reservada. Quando uma classe utiliza uma quantidade menor de banda do que a designada para ela, o restante, ou seja, o excesso pode ser distribuído para outras classes que desejam banda extra.

Cada classe folha implementa uma fila do tipo *Stochastic Fairness Queueing*, ou SFQ, que é um algoritmo simples da família das “filas uniformes”. Ele é menos preciso do que outros tipos de filas, porém necessita de menos cálculos, enquanto o seu desempenho é justo para com todas as filas.

No SFQ, uma conversação ou fluxo corresponde a uma sessão TCP ou a um *stream* UDP. O tráfego é dividido num determinado número de filas FIFO, uma para cada conversação. O tráfego então é enviado como *Round Robin*, dando a cada sessão uma chance de enviar seus dados por vez. Isso leva a um comportamento justo para todas as filas e não permite que um único fluxo apenas transmita.

O SFQ é chamado de *stochastic* porque ele não aloca realmente uma fila para cada sessão. Ele possui um algoritmo que divide os fluxos entre um número limitado de filas usando um algoritmo *hash*. Por causa desse algoritmo *hash*, mais de uma sessão pode ir parar na mesma fila, o que faz reduzir a chance de cada uma dessas sessões enviar um pacote, reduzindo assim a velocidade efetiva de transmissão. Para prevenir que esta situação se torne perceptível, o SFQ muda o algoritmo *hashing* freqüentemente, para que duas sessões que compartilhem a mesma fila não fiquem muito tempo nessa situação.

Para fazer a divisão da banda disponível nos roteadores entre os diferentes tipos de tráfego, para efeito de teste, foram criadas três classes HTB: uma classe para pacotes de

controle enviados pelo *framework* de suporte a QoS, uma classe para *streams* de vídeo, e uma classe que irá servir para qualquer outro tipo de tráfego. A classe de vídeo é dividida em outras quatro classes, que serão os níveis de adaptação disponíveis no *framework*. Como já foi dito, cada nível de adaptação representa uma taxa de transmissão. A Figura 33 apresenta a configuração de cada roteador com suas *qdisc* e classes.

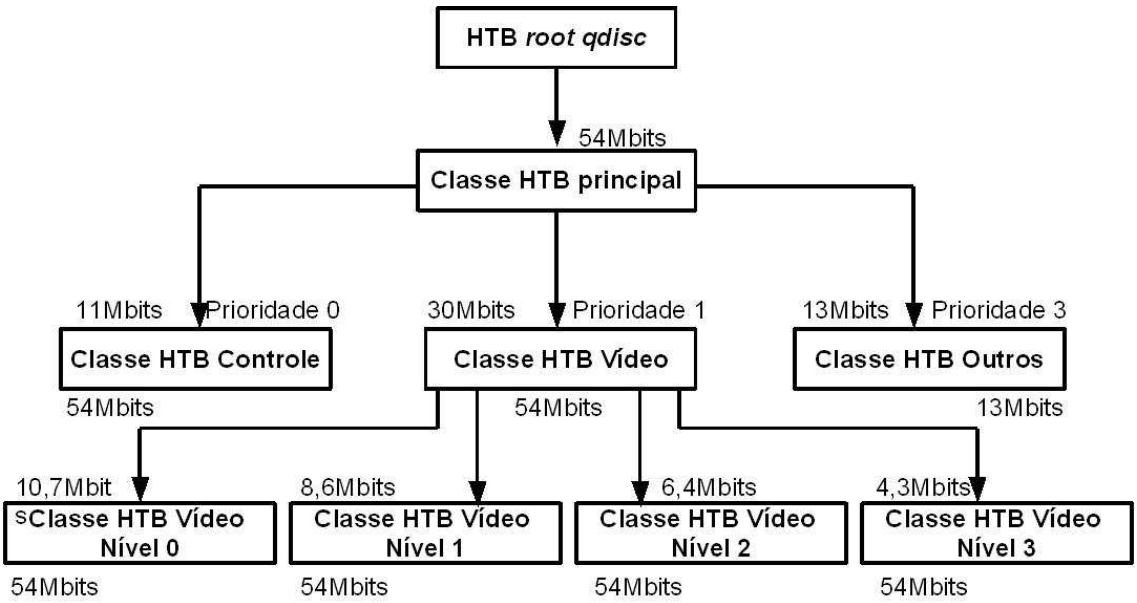


Figura 33: Configuração das qdisc e classes em cada roteador

A largura de banda indicada na parte superior das classes corresponde à banda reservada. A banda indicada na parte inferior apresenta o valor máximo que a classe poderá pegar emprestado. É importante observar que a classe HTB Outros não pode pegar banda emprestada de suas classes irmãs, já que o valor máximo de banda é igual à quantidade reservada para ela. Porém, ela poderá emprestar banda sobrando para outra classe que necessitar.

Cada classe folha possui uma prioridade. As classes de vídeo possuem a mesma prioridade da classe pai. As prioridades afetam em como o excesso de largura de banda será distribuído entre as classes "irmãs". A regra é que para as classes com prioridades mais altas é oferecido o excesso de banda primeiro. As classes com prioridades mais baixas ou não serão contempladas com alguma banda extra, ou receberão uma fatia pequena.

Para que os fluxos sejam apontados para as suas respectivas classes foram criados filtros que analisam os dados dos pacotes e fazem uma combinação. Caso um pacote não seja da classe controle e nem de uma das classes de vídeo, ele será enviado para a classe outros.

1.7.2 Framework no cliente e servidor

As partes cliente e servidor não tomam conhecimento das reservas feitas nos roteadores. A ferramenta tc não distingue o quanto de banda efetivamente foi dado a cada fluxo de uma classe. A banda reservada para uma determinada classe está garantida para todos os pacotes que passarem por ela. Caso a taxa de transferência de uma classe exceda a banda reservada, ela irá pegar banda emprestada com suas classes irmãs. O que pode acontecer, caso muitos fluxos disputem banda em uma mesma classe, é que alguns pacotes se atrasem

ou sejam descartados. Nesse caso, o cliente irá perceber essa degradação e irá acionar o processo de adaptação, diminuindo a taxa de transferência.

Ao diminuir a quantidade de dados a ser transmitida, o fluxo também terá seu nível de adaptação alterado. Conseqüentemente, quando os pacotes daquele fluxo passarem pelos roteadores, eles serão encaminhados para outra classe, diferente da que estavam anteriormente. Isso faz com que a classe anterior utilize menos banda, favorecendo os outros fluxos que estão passando por ela.

O nó cliente realiza uma monitoração dos dados periodicamente. A cada 100 pacotes recebidos, ele verifica o *jitter* e a perda de pacotes nesse grupo. Se os valores medidos estiverem acima de um limite máximo definido pela aplicação, o processo de adaptação é acionado. Com isso, o nível de adaptação é incrementado e um relatório de QoS é enviado para o servidor. Ao receber essa mensagem, o servidor percebe que precisa então diminuir a taxa de transferência e alterar o nível de adaptação dos pacotes. Quando o primeiro pacote com o nível de adaptação alterado chega no cliente, ele então percebe que o relatório de QoS chegou com sucesso ao servidor e o nível de adaptação foi alterado. Caso o cliente detecte que o relatório de QoS não chegou ao servidor, ele então o envia novamente.

A monitoração continua periodicamente. Caso seja detectado que o *jitter* e a perda de pacotes medidos apresentam valores abaixo do limite mínimo definido pela aplicação, o processo de adaptação é acionado novamente, fazendo o nível de adaptação diminuir. Mais uma vez o cliente envia um relatório de QoS para o servidor avisando-o que pode aumentar a taxa de transferência. Ao receber este relatório, o servidor percebe que as condições da rede melhoraram e que pode enviar mais dados.

Este *framework* foi testado com um programa de videoconferência desenvolvido pelo aluno Robson Hilario da Silva, mestrando em Computação do Instituto de Computação (IC). Este programa possui uma parte servidora que capta as imagens de uma câmera e as envia para um cliente. Além de transmitir dados ao vivo, o programa também pode gravar as imagens captadas em um arquivo e depois enviar os quadros desse arquivo.

2. Resultados dos testes

2.1 Resultados dos testes na rede interna

Os testes de desempenho da rede interna foram feitos conectando sete laboratórios dentro de um dos campi da UFF. Estações finais foram conectadas aos pontos de acesso da rede ReMesh através de Ethernet cabeada. O ambiente interno de testes consistiu de sete roteadores e algumas estações finais posicionadas em dois pisos adjacentes de um mesmo prédio.

A Figura 34 ilustra o ambiente interno de testes do projeto ReMesh. Os nós estão numerados de 0 a 6 e indicam os respectivos laboratórios onde foram posicionados. O nó 6 fica no quarto andar do prédio e os outros ficam no terceiro andar. Os enlaces L1 a L20 foram construídos monitorando a topologia fornecida pelo OLSR em cada roteador, através de um *plug-in* do *daemon* OLSR. As linhas pontilhadas indicam enlaces de qualidade baixa enquanto que as linhas contínuas indicam enlaces de melhor qualidade.

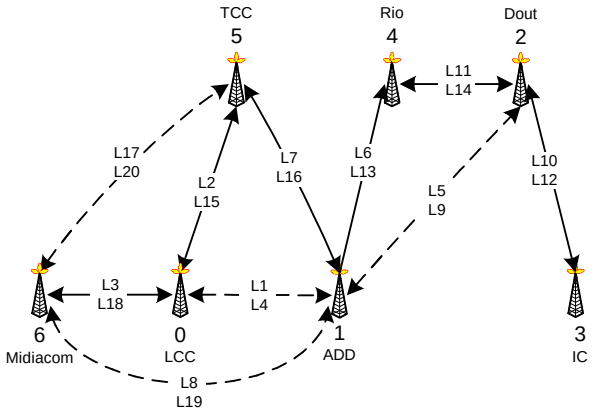


Figura 34:O ambiente interno de testes do projeto ReMesh

No cenário descrito, observamos instabilidade nas rotas e altas taxas de perda com o protocolo OLSR usando a extensão ETX original. Isso motivou a necessidade de utilizar um algoritmo de roteamento diferente.

2.1.1 Variações da Métrica ETX

A Tabela 2 mostra as medidas da métrica ETX para o ambiente de testes internos do projeto ReMesh ilustrado na Figura 34. Foram monitorados a média, o mínimo, o máximo e o desvio padrão do valor de ETX para cada um dos 20 enlaces por um período de 24 horas. As linhas em negrito na Tabela 2 indicam enlaces de boa qualidade e valor médio de ETX menor que 1,20. Entretanto, observe que mesmo os enlaces de boa qualidade, tais como L3 e L18, entre os nós 0 e 6, ocasionalmente vivenciam momentos de altas perdas, tendo registrado um

valor máximo de ETX de 141,67. Um fenômeno similar é observado nos enlaces L7, L14, L15 e L16. De maneira interessante, enlaces de baixa qualidade com valor de ETX maior que 6,10, tais como L4, L5, L9 e L17, ocasionalmente não vivenciam nenhuma perda e reportam um valor mínimo de ETX de 1. O desvio padrão (σ) indica a variação da métrica ETX de cada enlace.

A variação da métrica ETX resultou em mudanças freqüentes de rotas no ambiente de testes do projeto. Considere, por exemplo, uma comunicação entre os nós 1 e 6. Existe um enlace direto L8 com um valor médio de ETX de 2,40. Existe também um caminho alternativo através de três enlaces, L7, L15 e L3, (ou através dos nós 1-5-0-6) com valor médio de ETX de 3.29 (= 1.13 + 1.04 + 1.12). Pequenas flutuações na qualidade do enlace L8 podem resultar na mudança da rota para o caminho alternativo apontado. Note que os enlaces L7, L15 e L3 são mais estáveis que o enlace L8. Note também que a probabilidade de perda de L8 é 58,3% enquanto que a probabilidade de perda ao longo do caminho L7-L15-L3 é 24%.

Tabela 2:Medidas de ETX

L	S	D	Avg	Min	Max	σ
L1	0	1	9.40	1.05	71.30	7.03
L2	**0**	**5**	**1.06**	**1.00**	**1.97**	**0.07**
L3	**0**	**6**	**1.12**	**1.00**	**51.00**	**2.07**
L4	1	0	10.09	1.00	53.12	8.02
L5	1	2	90.91	1.00	451.56	72.11
L6	**1**	**4**	**1.07**	**1.00**	**2.21**	**0.09**
L7	**1**	**5**	**1.13**	**1.00**	**13.42**	**0.17**
L8	1	6	2.40	1.00	104.04	4.08
L9	2	1	199.60	1.00	451.56	180.58
L10	**2**	**3**	**1.02**	**1.00**	**1.32**	**0.03**
L11	**2**	**4**	**1.07**	**1.00**	**1.39**	**0.06**
L12	**3**	**2**	**1.01**	**1.00**	**1.24**	**0.03**
L13	**4**	**1**	**1.06**	**1.00**	**2.28**	**0.09**
L14	**4**	**2**	**1.05**	**1.00**	**68.45**	**0.42**
L15	**5**	**0**	**1.04**	**1.00**	**30.44**	**0.19**
L16	**5**	**1**	**1.20**	**1.00**	**451.56**	**4.19**
L17	5	6	6.10	1.00	51.00	3.54
L18	**6**	**0**	**1.10**	**1.00**	**141.67**	**2.16**
L19	6	1	2.25	1.00	106.25	2.17
L20	6	5	8.21	1.05	425.00	6.58

2.1.2 Estabilidade das Rotas

Para verificar a estabilidade das rotas, foram enviados 18.000 pacotes de *ping*, registrando a rota percorrida, entre os nós 3 e 6, durante um período entre 10:00 e 15:00 horas de um dia de trabalho normal. Foram observadas 426 mudanças de rotas usando o protocolo OLSR-ETX. Quando OLSR-ML foi instalado, nenhuma mudança de rota foi observada durante o mesmo período em um outro dia de trabalho normal, através do mesmo teste.

2.1.3 Taxas de Perda de Pacotes

Para medir a taxa de perda de pacotes, foram realizados experimentos durante um período de 12 horas transmitindo 43.200 pacotes entre os nós 3 e 6 como o objetivo de comparar os protocolos OLSR-ETX e OLSR-ML. Os experimentos foram realizados durante as mesmas horas do dia, porém em dias de trabalho diferentes para cada algoritmo de roteamento. Como esperado, a Figura 35 exibe um queda significativa nas taxas de perda de pacotes com o OLSR-ML, com reduções variando entre 59,8% e 97,0%.

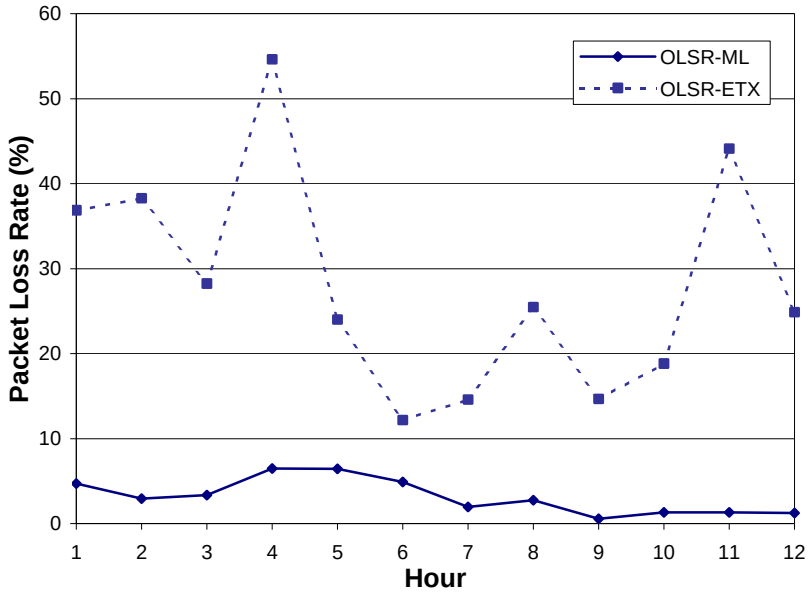


Figura 35: Taxas de perda de pacotes na comunicação entre os nós 3 e 6.

2.1.4 Retardo

Para selecionar caminhos com menor taxa de perda, OLSR-ML normalmente seleciona caminhos com maior número de saltos, comparado ao OLSR-ETX. Esse aumento na quantidade de saltos poderia resultar em retardos maiores na rede. De maneira interessante, a Figura 36 mostra que OLSR-ML também se comporta melhor em termos de retardo. Os testes também foram executados transmitindo 43.200 pacotes entre os nós 3 e 6, durante períodos de 12 horas para medir o retardo de ida e volta de cada pacote.

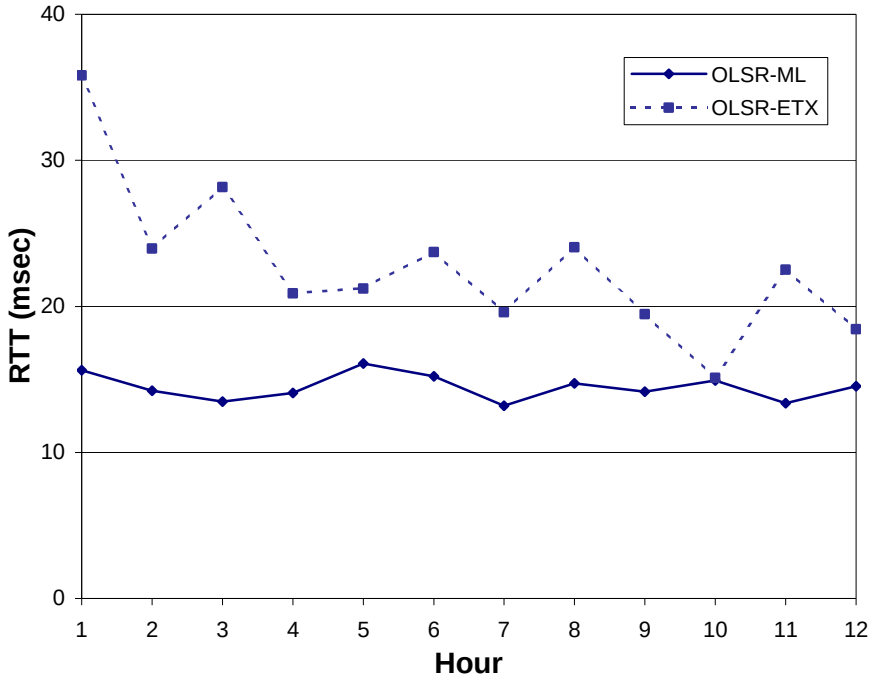


Figura 36: Retardos de ida e volta na comunicação entre os nós 3 e 6

Para entender como caminhos com maior número de saltos têm retardos de ida e volta (RTT - *round trip time*) menor que caminhos com menor número de saltos, estatísticas da camada 2 foram investigadas. Os resultados iniciais indicam menos retransmissões da camada 2 ao longo do caminho com menor probabilidade de perda, selecionado pelo OLSR-ML, em comparação com o número de retransmissões de camada 2 ao longo do caminho selecionado por OLSR-ETX. O número máximo de tentativas de retransmissão para cada quadro foi setado como 7 na camada 2, de acordo com o padrão IEEE 802.11. A redução nos valores de RTT, usando OLSR-ML, variou entre 1,34% e 56,3%.

2.1.5 Vazão

Como explicado anteriormente, o protocolo OLSR-ML proposto tipicamente escolhe caminhos com taxas de perda menores e maior número de saltos quando comparado ao OLSR-ETX. Cada salto adicional em transmissões através de múltiplos saltos sobre o meio compartilhado aumenta a probabilidade de contenção e colisão, e pode ter um impacto negativo significativo sobre a vazão total da rede.

Para medir a vazão da rede mesh, um transmissor conectado ao nó 3 rodando a ferramenta de medição IPERF foi configurado para enviar pacotes para cada nó do ambiente interno de testes. Um total de 300 medidas IPERF por nó de destino foi realizado. A Figura 37 mostra um pequeno aumento na vazão usando OLSR-ML para comunicações com 3, 4 e 5 saltos.

Um segundo experimento para medir a vazão foi realizado transferindo arquivos de 10Mbytes com a ferramenta de medição TTCP. Um desempenho similar foi observado, como exibido pela Figura 38.

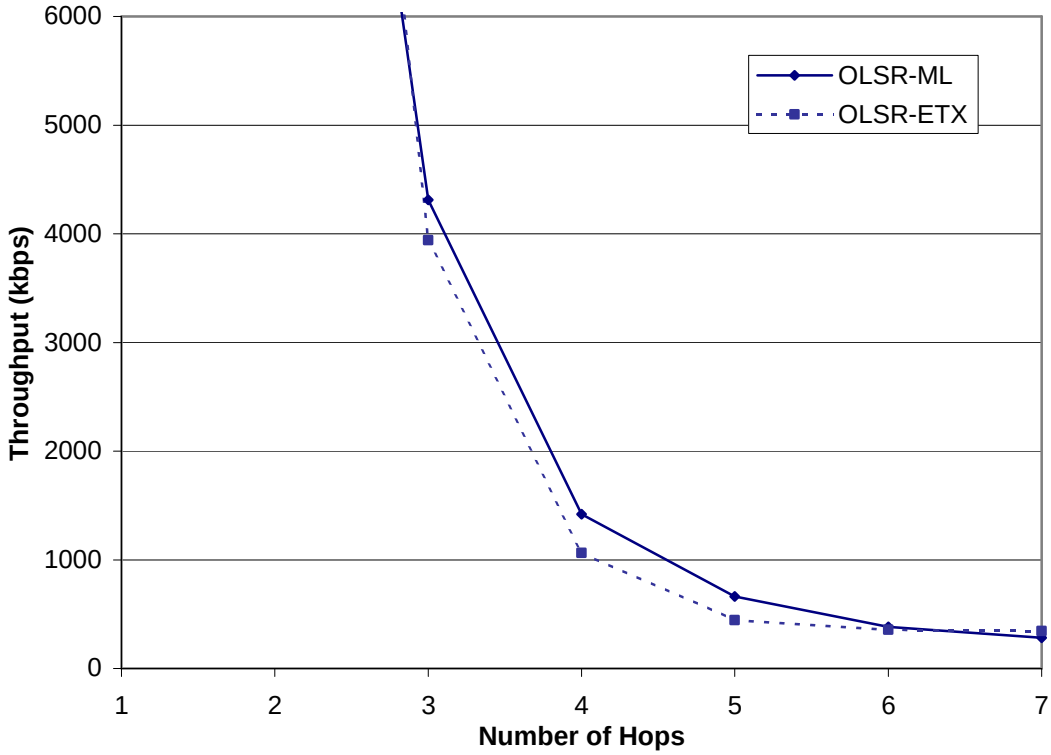


Figura 37: Vazão medida com IPERF em comunicações começando pelo nó 3.

A queda exponencial na vazão, à medida que o número de saltos aumenta, mostrada na Figura 37 e na Figura 38, é consistente com o comportamento de outras redes mesh e ad-hoc. RoofNet exibe um comportamento similar em termos de vazão, reportando valores de 2,45 Mbps para comunicações com 1 salto caindo para 181Kbps para comunicações com 7 saltos. A vazão da rede ReMesh é maior que a da RoofNet porque estamos usando o padrão IEEE 802.11g, enquanto que RoofNet utiliza 802.11b. Além disso, a medida de qualidade dos enlaces de RoofNet é feita com a métrica ETT (*Expected Transmission Time*), que prevê o tempo total para enviar um pacote ao longo de uma rota, considerando a taxa de transmissão máxima de cada enlace e a probabilidade de recepção usando essa taxa. O protocolo de roteamento usado por RoofNet escolhe a rota de menor ETT.

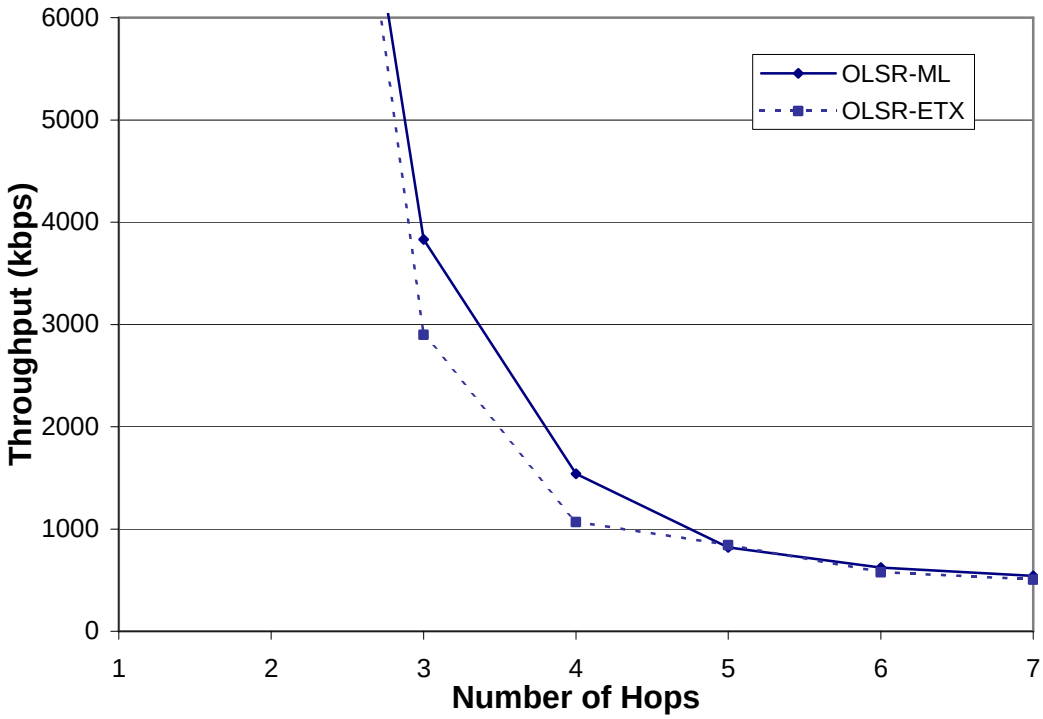


Figura 38:Vazão medida com TTCP em comunicações começando pelo nó 3.

2.2 Resultados dos testes na rede externa

2.2.1 Vazão

Para avaliação da vazão, foi utilizada a ferramenta *iperf* sem a presença de tráfego dos usuários. A ferramenta foi utilizada a partir do nó gateway da rede externa disparando 10 minutos de tráfego TCP para cada um dos nós participantes. O teste foi realizado nas duas direções entre os nó gateway e nó cliente simultâneamente. A escolha em realizar o teste com tráfego TCP e não UDP foi feita em vista de refletir as taxas obtidas pela maioria das aplicações utilizadas pelos usuários.

O primeiro nó avaliado é o roteador de endereço ip 10.151.1.1. Este nó se encontra na maior parte do tempo a um salto de distância do gateway (nó 10.151.7.1).

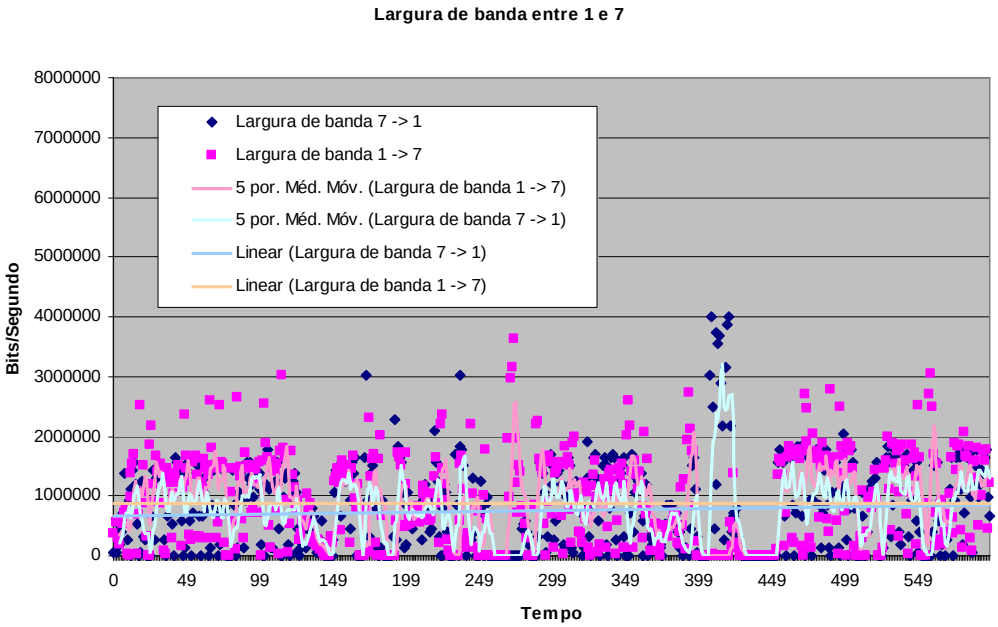


Figura 39: Vazão entre os nós 10.151.1.1 e 10.151.7.1

Onde:

- Largura de banda é a taxa do tráfego TCP obtida naquele instante.
- “5 por Média Movente” é a média de janela movente a cada 5 segundos de medição. Ela fornece uma tendência da variação da taxa ao longo do tempo.
- Linear é a regressão linear dos valores obtidos para demonstrar a tendência da vazão. Pode ser considerada como uma média da banda ao longo do tempo.

Outro nó adjacente ao gateway é o roteador de endereço 10.151.4.1. Ambos constituem, na maior parte do tempo, os principais nós de saída para o escoamento do tráfego gerado na rede mesh.

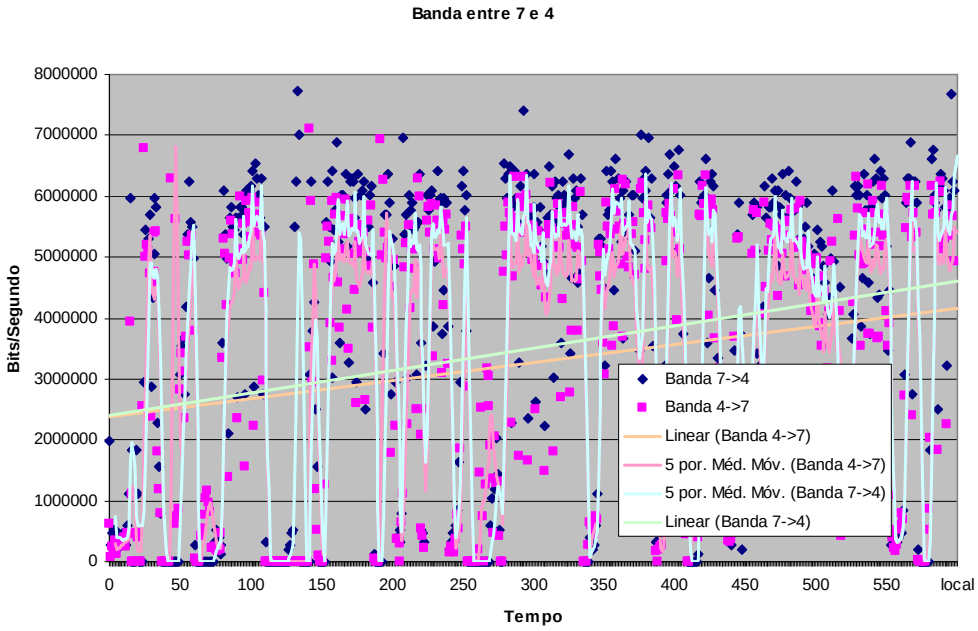


Figura 40: Vazão entre os nós 10.151.4.1 e 10.151.7.1

Os próximos 3 nós avaliados constituem pontos que se alcançam na maior parte do tempo através de dois saltos: 10.151.11.1, 10.151.13.1 e 10.151.14.1

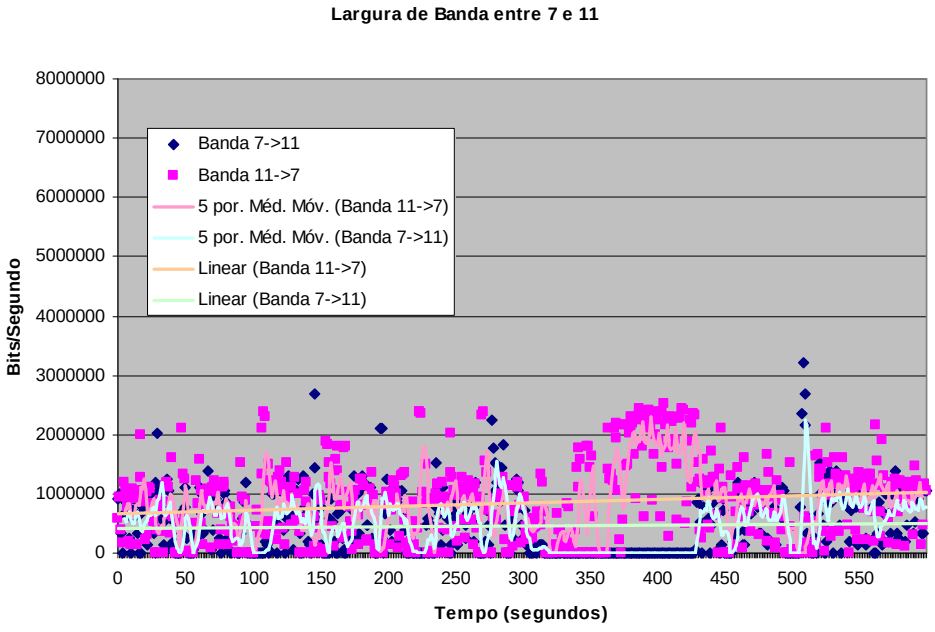


Figura 41: Vazão entre os nós 10.151.11.1 e 10.151.7.1

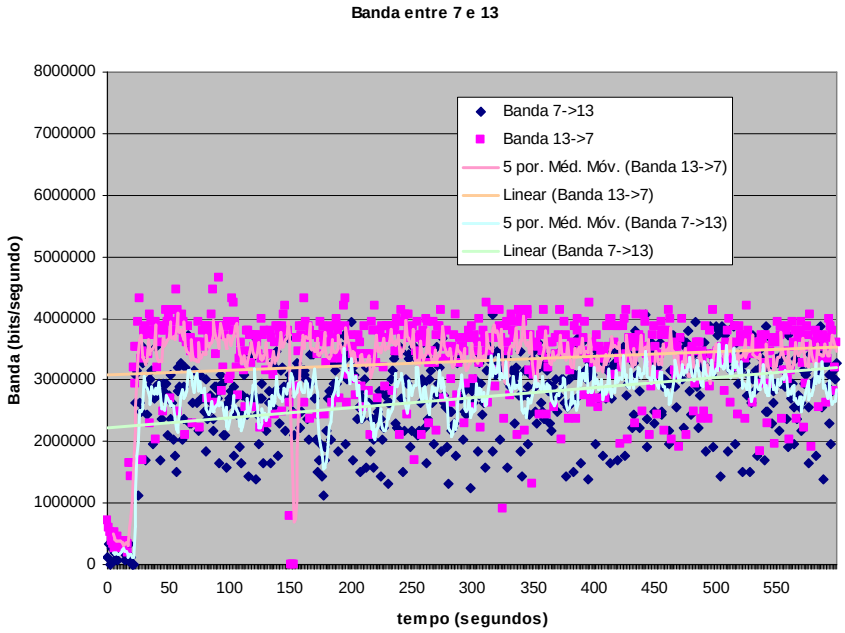


Figura 42: Vazão entre os nós 10.151.13.1 e 10.151.7.1

E por fim

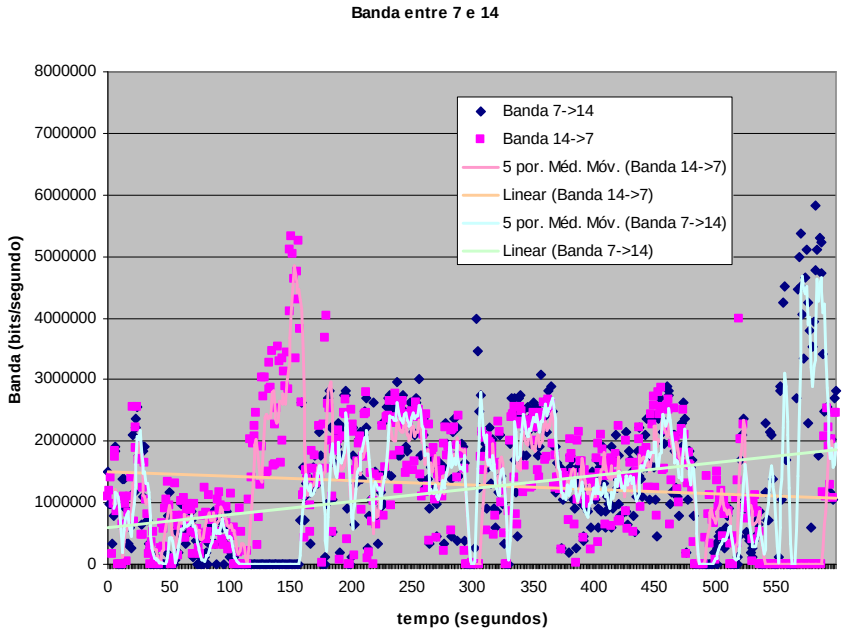


Figura 43: Vazão entre os nós 10.151.14.1 e 10.151.7.1

Os gráficos demonstram a banda real para aplicações que os usuários mesh dispõem atualmente. Isto demonstra que, mesmo com um alto número de usuários conectados por nó, poderemos fornecer conexão à Internet com banda larga facilmente devido às altas taxas alcançadas. Ainda, os resultados acima demonstram que a rede pode ser extendida para uma quantidade maior de nós desde que se mantenha a qualidade dos enlaces.

2.2.2 Perda e atraso

Para a avaliação das perdas foi utilizado um teste com PINGs para cada nó participante, a partir de uma máquina ligada diretamente ao gateway. Foram disparados para cada nó participante 3600 pings durante 6 horas, totalizando em um total de 21600 PINGS para cada nó. Deste teste foram obtidos os dados de perdas e do atraso da rede.

O resultado do teste é apresentado em tabelas separadas para cada nó. Cada tabela fornece os atrasos máximo, médio e mínimo, correlacionados com a quantidade de pacotes enviados com os TTL's mais freqüentes. Esta correlação foi feita para demonstrar que o atraso está diretamente relacionado com o número do TTL: quanto menor o TTL, maior será o número de saltos que o pacote percorreu até alcançar o destino. A variação do TTL pode ser inferida como o número de vezes em que ocorreram mudanças de rotas.

Tabela 1: Nó 10.151.1.1. Atrasos por TTL. Perda de 1.81%.

TTL	Num. pacotes	Máximo(ms)	Médio(ms)	Mínimo(ms)
61	292	4520	339.966	4.02
62	2540	3800	222.608	3.19
63	17456	3757	28.5426	2.20

Tabela 2: Nó 10.151.4.1. Atrasos por TTL. Perda total de 3.73%

TTL	Num. pacotes	Máximo(ms)	Médio(ms)	Mínimo(ms)
61	2015	3716	604.494	3.61
62	3388	3680	567.034	3.08
63	14661	9454	469.665	2.16

Tabela 3: Nó 10.151.14.1 - Atrasos por TTL. Perda de 12.58%

TTL	Num. pacotes	Máximo (ms)	Médio(ms)	Mínimo(ms)
61	5531	7763	555.606	3.29
62	4856	8176	481.936	2.64
63	5511	6279	325.061	2.04

Tabela 4:Nó 10.151.13.1 - Atrasos por TTL. Perda de 8%

TTL	Num. pacotes	Máximo (ms)	Médio(ms)	Mínimo(ms)
60	1872	4154	130.436	4.81
61	9808	8353	663.894	4.02
62	7057	8330	140.965	3.11
63	7	465	867.114	7.52

Tabela 5: Nó 10.151.11.1 - Atrasos por TTL. Perda de 6.72%

TTL	Num. pacotes	Máximo (ms)	Médio(ms)	Mínimo(ms)
61	1203	2019	270.789	3.78
62	12357	3110	152.231	3.22
63	3686	1082	148.596	2.66

2.2.3 Problemas encontrados e soluções: utilização de diferentes antenas e a questão do uso de diversidade espacial

O roteador WRT54G da Linksys vem equipado com duas saídas de RF para instalação de duas antenas com o propósito de utilizar diversidade espacial para recepção e envio do sinal. O roteador apresenta, nas suas configurações de hardware (parâmetros da *nvram*), três modos para a escolha da antena em operação: ou a da esquerda, ou a da direita ou ambas no modo de diversidade. O modo de diversidade possui funcionamentos distintos para a recepção e para o envio do sinal. Na recepção o roteador troca constantemente de antenas para eleger qual delas recebeu o sinal mais recente com maior intensidade. Para a transmissão, o roteador escolhe a última antena que recebeu um bom pacote de nível 2.

Originalmente, o propósito do protótipo da rede mesh era a utilização de apenas uma das saídas de RF para a colocação de uma antena omni-direcional de ganho de 18,5 dbi. Ao longo do projeto, foi visto que, para as instalações externas, a utilização de antenas omni-direcionais, apesar de permitirem o crescimento não planejado da rede a cada novo ponto instalado, apresenta limitações inerentes à natureza do sinal irradiado.

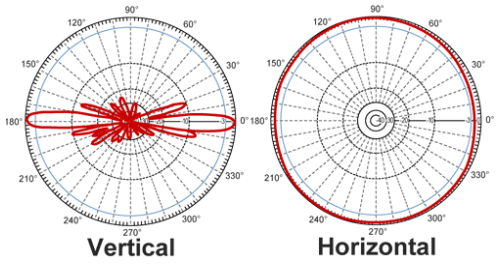


Figura 44: Lóbulos de irradiação da antena omni-direcional adotada

Como pode ser visto na Figura 44, a antena adotada pelo GT apresenta no diagrama de irradiação vertical dois grandes lóbulos concentrados nos ângulos 0 e 180°. Isto significa que a antena irradia com grande intensidade diretamente para cima e para baixo, porém com baixa intensidade para as outras direções. Isto significa que as componentes diagonais do sinal irradiado são fracas, fazendo com que antenas localizadas em alturas e distâncias horizontais diferentes não se comuniquem muito bem. Nas vizinhanças do campus escolhido para a implantação da rede os prédios apresentam alturas diferentes, variando de 12 até a 60 metros, por isso problemas nas comunicações começaram a surgir.

Enlaces relativamente curtos, como foi o caso entre o Gateway e o nó “10.151.4.1”, apresentavam ETX com média de 2.5 (o que é relativamente bom), porém baixa vazão. Isto ocorria pelo fato dos roteadores estarem operando no modo automático de seleção de taxa. Por causa disso, enlaces ruins forçavam os roteadores a se re-configurarem quanto às suas taxas de transmissão para que as perdas de pacotes diminuíssem. Com isso o ETX melhorava, porém a vazão diminuía, tornando a navegação dos usuários muito lenta.

Para a resolução deste problema foram adotas antenas direcionais com ganho maior. Esta solução resolveu o problema de comunicação nos enlaces diagonais, contudo apresentou um novo problema que foi a limitação da região irradiada. Usuários sem fio que moravam em prédios laterais à UFF não puderam mais acessar a rede, pois o sinal não os alcançava.

Como, a princípio, o objetivo era a qualidade do enlace entre os nós, este problema se tornou secundário. Contudo, para tentar solucionar isto, adotamos o uso de ambas as antenas no mesmo ponto com o roteador operando no modo de diversidade de antenas. Uma dúvida freqüente a esta abordagem é a interferência que as antenas possam ocasionar entre si. Na verdade, como são duas antenas, porém apenas um rádio, o sinal é recebido ou enviado por apenas uma das antenas por vez. Ou seja, as únicas interferências que podem ocorrer são as que são causadas pelos efeitos de borda e reflexões causadas por suas localizações.

Figura 45: Nó mesh montado com duas antenas

Porém, como explicado no início desta seção, o módulo de diversidade do roteador escolhe para transmitir a antena que recebeu um bom pacote pela última vez. Em uma rede mesh que funciona por saltos, nem sempre a antena a ser usada para transmitir algum pacote foi a mesma que recebeu o anterior. Isto tornou o uso da diversidade do roteador pouco eficiente. Para demonstrar isto, abaixo colocamos um teste de vazão realizado durante 20 segundos, com a ferramenta *iperf,* entre o nó gateway e o nó adjacente “10.151.1.1”, sem o olsr em funcionamento, para cada um dos modos de operação do uso das antenas:

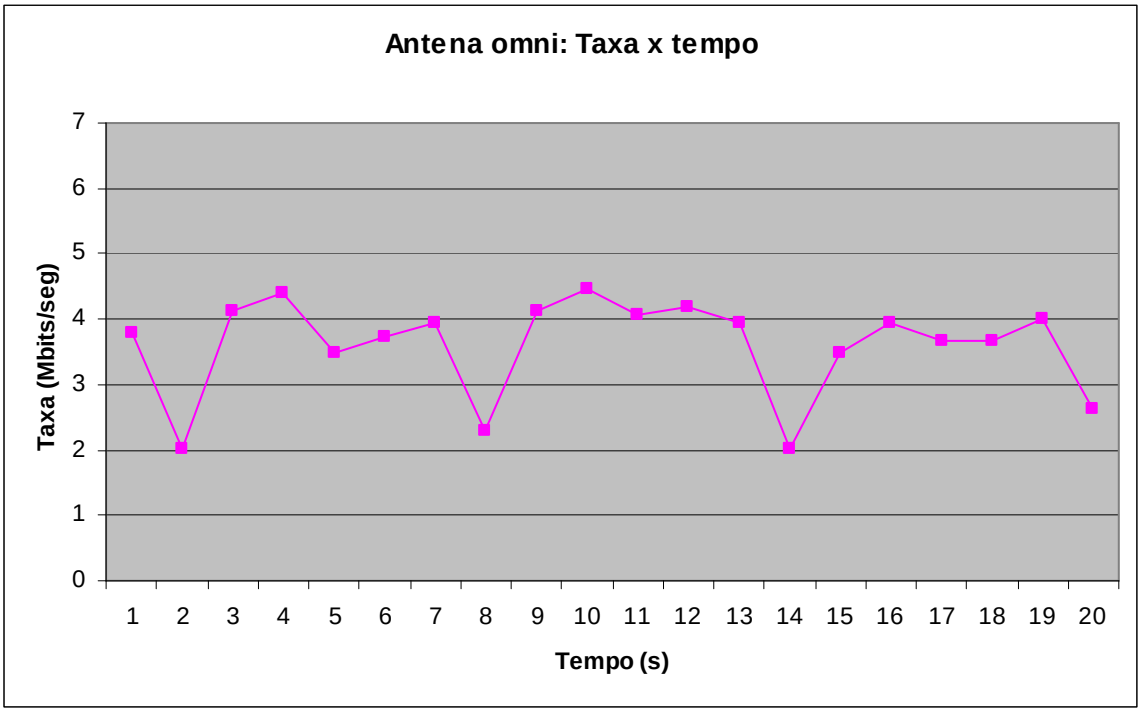


Figura 46: Desempenho da antena omni-direcional - Média de 3.53 Mbits/seg

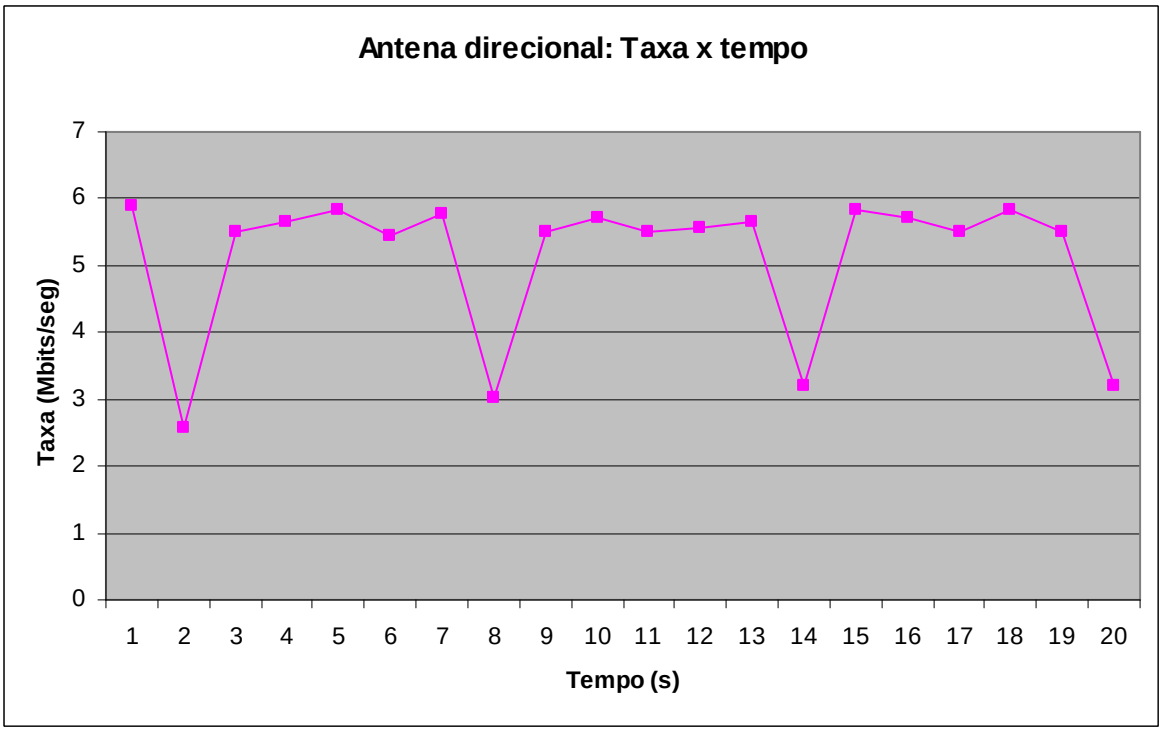


Figura 47: Desempenho da antena direcional - Média de 5.09 Mbits/seg

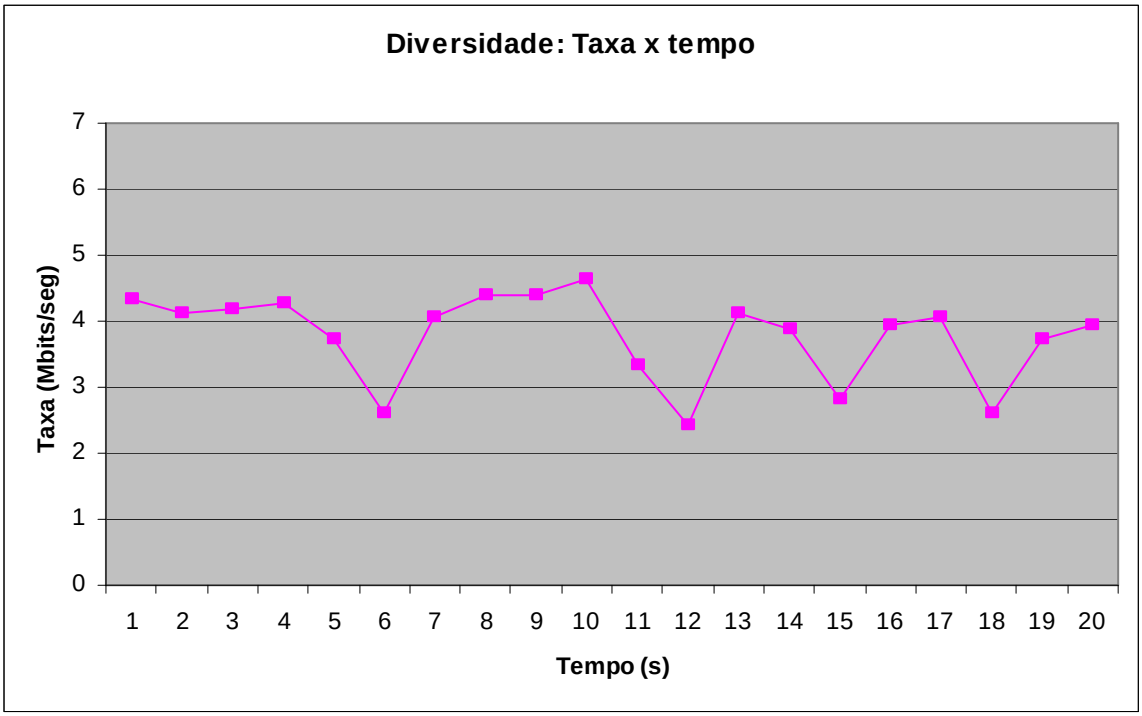


Figura 48: Desempenho do modo de diversidade - Média de 3.77 Mbits/seg

Analisando os gráficos das Figuras 40, 41 e 42, percebemos que a antena direcional apresentou maiores valores de taxa do que os outros dois modos, que apresentaram valores bastante semelhantes. O modo de diversidade apresentou uma ligeira melhora na média do usando apenas a omni, contudo ficou bem abaixo do resultado obtido com a direcional. Percebemos que, atuando com ambas as antenas, o modo de diversidade deveria obter valores próximos aos obtidos com o uso da melhor. Contudo, o resultado demonstrou que ocorreu o contrário.

Em vista disto, o GT começou a desenvolver um módulo a ser implantado no kernel do sistema operacional para o uso mais eficiente da escolha das antenas, que ainda está em desenvolvimento.

2.3 Estatísticas dos acessos

A rede mesh já está em utilização com o módulo de autenticação desde abril de 2006. A rede é aberta aos alunos e funcionários da UFF, bastando enviar email com dados de cadastro na UFF para o gerente da rede.

A rede interna abrange praticamente os blocos D e E inteiramente do campus da Praia Vermelha da UFF, possibilitando acesso à Internet principalmente sem fio aos interessados. A rede externa já conta com seis pontos externos instalados nas redondezas do campus onde, com exceção do gateway cujo acesso cabeado é evitado (por não realizar autenticação), todos atuam como pontos de acesso cabeado e sem fio à rede da UFF. Um cliente cadastrado na

rede mesh pode acessá-la tanto pela rede interna como pela rede externa. As estatísticas de novos usuários cadastrados pode ser vista nas figuras 43 e 44.

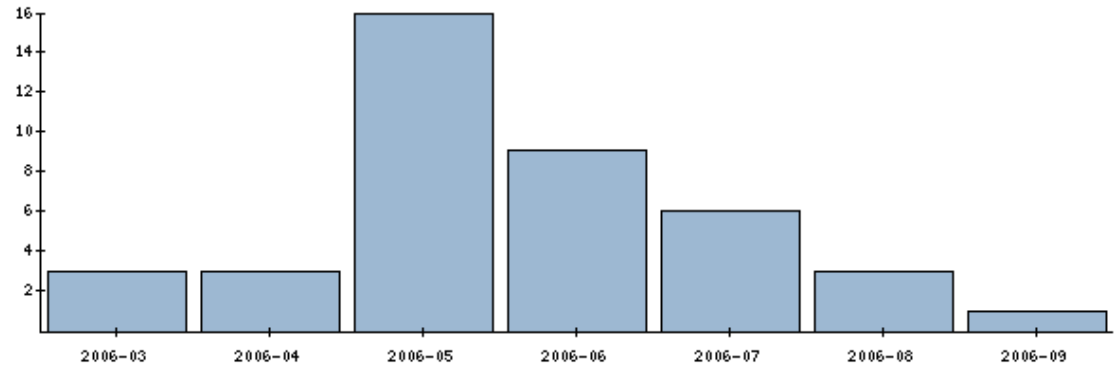


Figura 49: Número de novos usuários registrados nas redes interna e externa

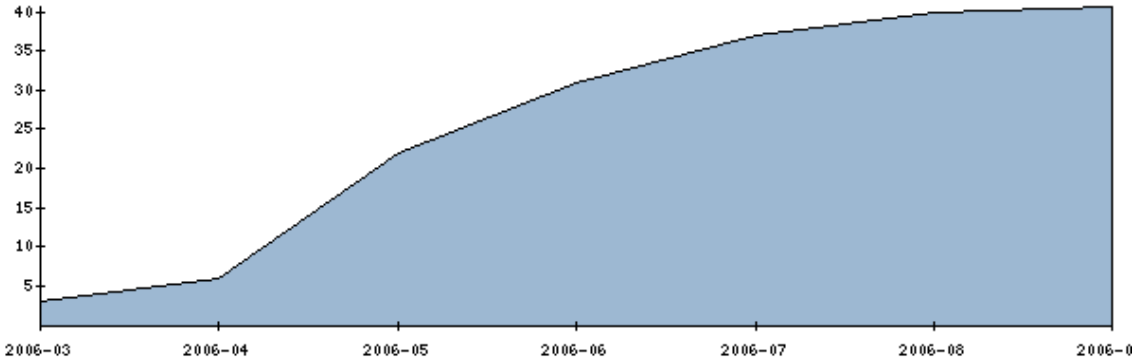


Figura 50: Número acumulativo de usuários cadastrados nas redes interna e externa

Abaixo, demonstramos algumas estatísticas de acessos específicas das redes interna e externa, coletadas até o dia 8 de setembro de 2006.

2.3.1 Estatísticas dos acessos na rede interna

10 usuários que mais consomem banda

Usuário	*Incoming*	*Outgoing*	Total
luciana	800M	2,8G	3,6G
rtoso	1,5G	40,5M	1,5G
admin	112,2M	687,1M	799,2M
douglas	627,3M	21,4M	648,8M
vthome	496,9M	28,1M	524,9M
kadusarruf	341,4M	10,9M	352,2M
higor	171,7M	11,4M	183M
fabiana_mp	114,4M	25,5M	139,9M
eduardo	77,3M	10,5M	87,8M
tiago	77M	5,3M	82,3M

10 usuários mais freqüentes

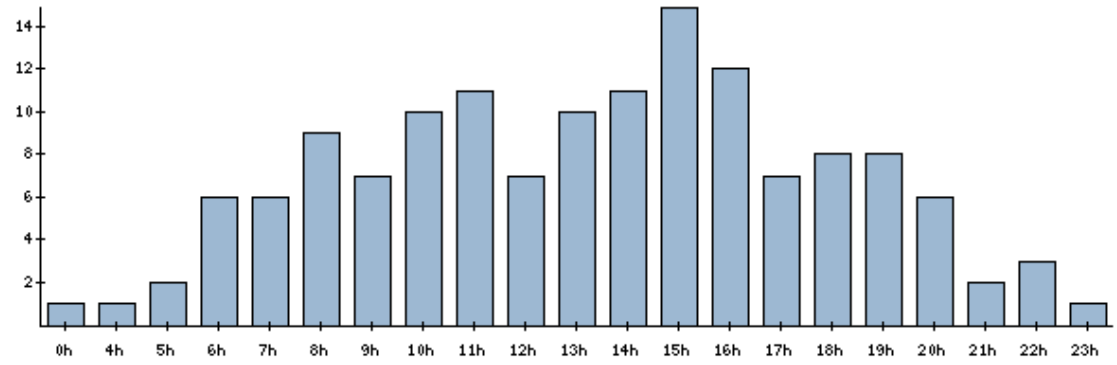
Usuário	Dias diferentes conectado
douglas	25
admin	22
fabiana_mp	20
luciana	18
rtoso	10
vthome	7
celio	7
higor	7
lpessoa	5
eduardo	5

10 usuários com maior mobilidade entre nós

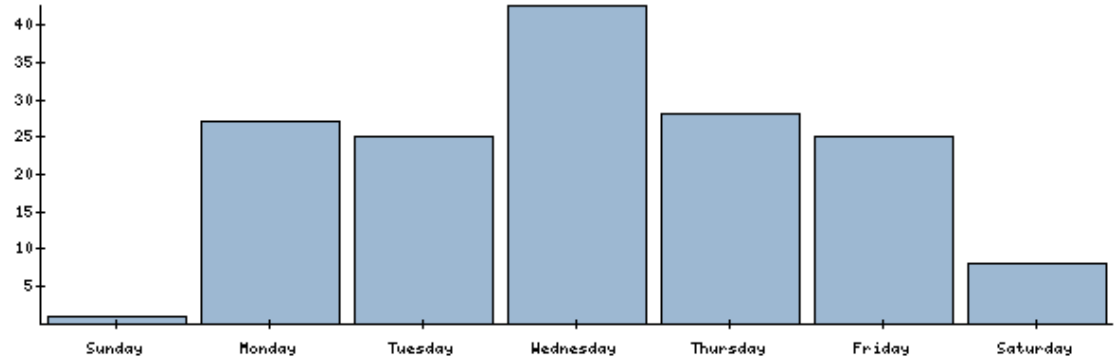
Usuário	Nós visitados
admin	4
teste	3
luciana	3
celio	3
douglas	3
diego	2
fmay	2
rtoso	1
janine	1
eduardo	1

Gráfico do uso da rede por hora, dia da semana e mês.

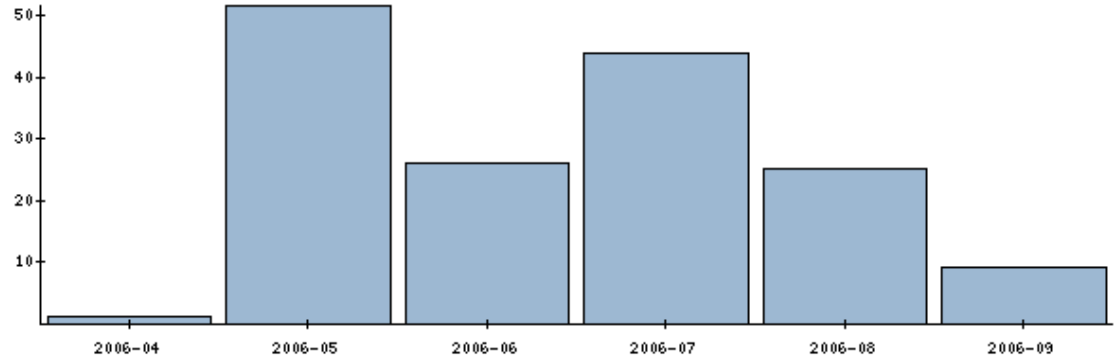
Número de novas conexões abertas por hora do dia



Número de visitas individuais dos usuários por dia da semana



Número de visitas individuais dos usuários por mês



Nós mais populares, por visita:

Nó	Visitas
8	95
6	25
5	19
9	16
3	1
Total:	**156**

Observação: Uma visita é uma contagem de conexões, porém considerando apenas uma conexão por dia para cada usuário em um determinado nó.

Primeira conexão por nó:

Nó	Número de primeiras conexões de novos usuários
8	10
6	1
5	1
Total:	**12**

Observação: Esta é realmente uma lista da quantidade de primeiras conexões ocorridas para cada novo usuário para cada nó.

2.3.2 Estatísticas dos acessos na rede externa

10 usuários que mais consomem banda

Usuário	***Incoming***	***Outgoing***	**Total**
pascoal	90,6G	107,6G	198,1G
fidi	27,9G	30,2G	58,2G
rtoso	7,3G	482,6M	7,8G
dvianna	6G	729,9M	6,7G
rcapua	6,2G	396,2M	6,6G
jviana	1,2G	1,5G	2,7G
alvaro	2,2G	499,2M	2,7G
tom	1,1G	133,5M	1,2G
luizedu	1,1G	93,1M	1,2G
douglas	678,4M	248,3M	926,8M

10 usuários mais freqüentes

Usuário	**Dias diferentes conectado**
dvianna	82
pascoal	67
fidi	59
rcapua	54
luizedu	28
lisieux	27
vthome	25
alvaro	24
rtoso	21
douglas	15

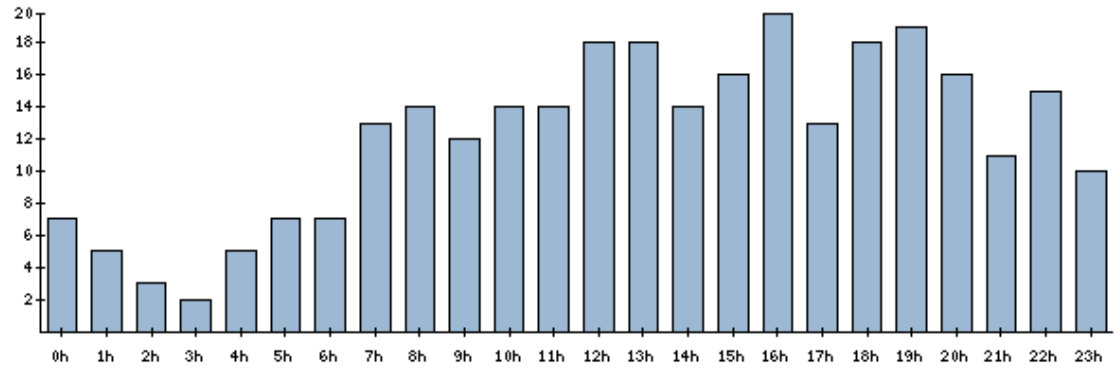
10 usuários com maior mobilidade entre nós

Usuário	**Nós visitados**
douglas	5
admin	5
rtoso	2
alvaro	2
fidi	2
lisieux	1
eduardo	1

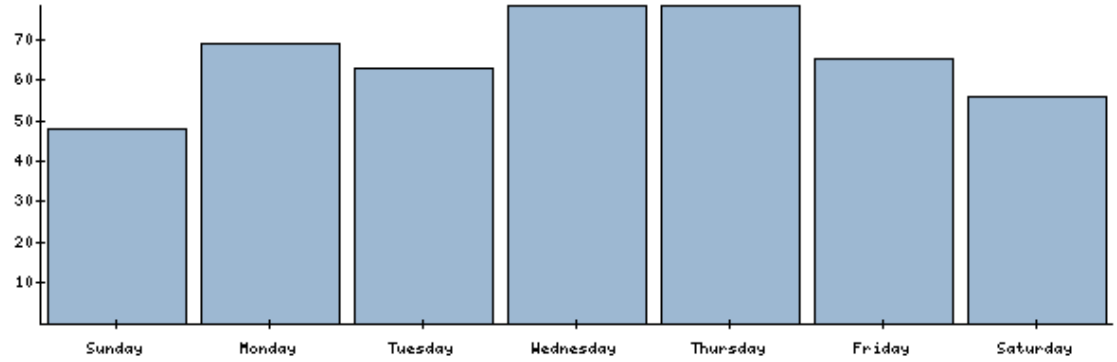
Usuário	Nós visitados
dvianna	1
nicolle	1
celio	1

Gráfico do uso da rede por hora, dia da semana e mês.

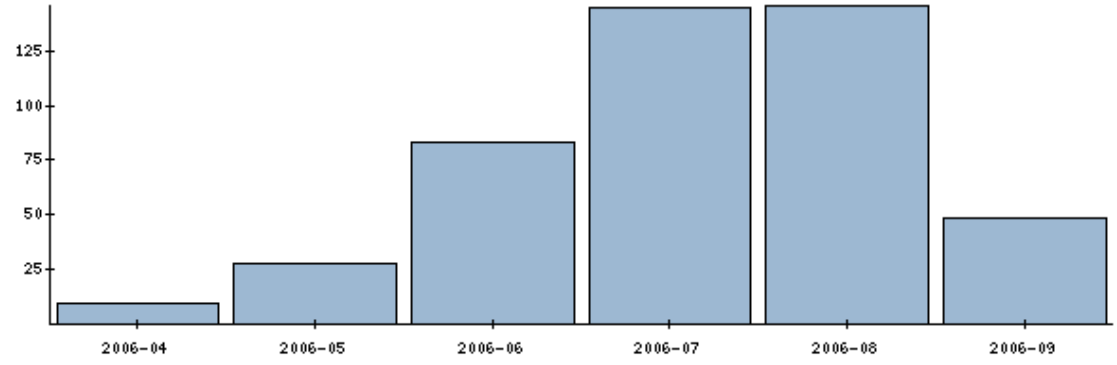
Número de novas conexões abertas por hora do dia



Número de visitas individuais dos usuários por dia da semana



Número de visitas individuais dos usuários por mês



Nós mais populares, por visita

Nó	Visitas
4	157
7	149

Nó	Visitas
1	78
14	54
11	16
13	3
Total:	**457**

Observação: Uma visita é uma contagem de conexões, porém considerando apenas uma conexão por dia para cada usuário em um determinado nó.

Primeira conexão por nó

Nó	Número de primeiras conexões de novos usuários
7	6
11	3
4	2
14	2
1	1
13	1
Total:	**15**

Observação: Esta é realmente uma lista da quantidade de primeiras conexões ocorridas para cada novo usuário para cada nó.

As estatísticas comprovam que ambas as redes têm sido bastante utilizadas, principalmente a externa. Deve-se considerar que o endereço IP disponibilizado para os usuários da rede externa não passa pelo firewall da UFF, permitindo a utilização de aplicativos *peer-to-peer* como *bittorrent* e *emule*.

2.4 Resultados do *Framework* de QoS

Esta seção será divida em duas partes. Primeiramente, serão descritos os problemas encontrados na codificação do *framework*, tanto nos roteadores quanto nas extremidades, bem como as soluções utilizadas. A segunda parte descreve os testes realizados e os resultados obtidos.

2.4.1 Problemas encontrados

Inicialmente, a proposta era testar o *framework* usando nós móveis, sendo esses nós dispositivos pequenos, como *Pocket PC* e grandes, como *Laptop.* Para que fosse possível trabalhar com sistemas *off-the-shelf*, foi escolhido o Microsoft Visual Studio .NET, que permite a implementação de aplicações para *Pocket PC* em C# e Visual Basic. A linguagem escolhida foi o C#.

Apesar da parte cliente poder ser executada no *Pocket PC*, não foi possível realizar os testes com esse tipo de dispositivo. Até o fim da implementação e dos testes, foi encontrada apenas uma versão do protocolo de roteamento OLSR para *Pocket PC*, desenvolvida por Carlos Calafate [Calafate 2002], mas que não é compatível com a versão do OLSR que executa nos roteadores. Por esta razão, a implementação disponibilizada e os testes realizados utilizaram um *Laptop*, executando uma versão do OLSR compatível com os roteadores.

A idéia inicial do *framework* é que o cabeçalho das mensagens, com as informações importantes para os roteadores e para as extremidades, fosse carregado no campo Opções do cabeçalho IP. Isso permitiria que essas informações ficassem desacopladas dos dados, tornando o *framework* mais transparente para a aplicação. Porém, não foi possível ter acesso às camadas mais baixas da mensagem no ambiente de desenvolvimento utilizado. Tanto no envio quanto no recebimento, as classes e métodos disponíveis apenas possibilitavam o acesso aos dados da mensagem, e não aos cabeçalhos, como o IP.

A solução encontrada para esse problema foi colocar o cabeçalho do *framework* no início de cada pacote transmitido. Cada quadro a ser enviado é fragmentado em pedaços de tamanho menores ou iguais ao MTU de uma rede *Ethernet*, para que todos os pacotes que trafeguem nos roteadores tenham o cabeçalho do *framework* anexado no início da mensagem. No cliente, também foi implementado um sistema que junta os pedaços para formar um quadro pronto para ser exibido.

O cabeçalho do *framework* então é verificado pelos filtros que residem nos roteadores e desta forma são classificados para suas classes correspondentes. O pacote que não tiver o cabeçalho do *framework* no início da mensagem será classificado para a classe Outros, pois não se encaixa como um fluxo multimídia com suporte a QoS.

Para facilitar os testes, ao invés de utilizar um vídeo ao vivo, ou seja, conforme a câmera vai capturando os quadros a aplicação vai enviando-os, foi utilizado um vídeo gravado em arquivo e carregado para um *buffer* no servidor. A próxima seção irá descrever os cenários de teste e os resultados obtidos.

2.4.2 Testes e resultados

Como já foi dito, a aplicação utilizada para teste foi uma cliente/servidor, onde o servidor é uma máquina *Desktop* conectada a rede *Mesh* através de uma cabo na porta *Ethernet* de um dos roteadores e o nó cliente é um *Laptop* ligado a *Mesh* por uma interface sem fio e que executa o OLSR para comunicação e roteamento entre os roteadores.

Por causa da natureza das redes sem fio, alguns testes do *framework* no protótipo da rede *Mesh* não puderam ser feitos totalmente sem interferência, tendo em vista que outras redes sem fio presentes no *campus* também disputam o meio e que o simples movimentar de pessoas e elevadores pode causar interferências. Mesmo assim, foram montados cenários sem interferência controlada e com interferência de dispositivos sem fio ou cargas extras nos roteadores. A topologia dos cenários está sendo mostrada na Figura 51.

Figura 51: Cenário dos testes do protótipo de suporte a QoS.

As linhas tracejadas formam a rota escolhida pelo protocolo de roteamento OLSR. O servidor está ligado por um cabo a um dos roteadores *Mesh*. O cliente sem fio pode ficar parado na extremidade oposta ao servidor, fazendo que os dados trafeguem por 6 saltos da origem até o destino. Em outros testes ele poderá ficar se movendo, tanto se aproximando do servidor como se afastando. No meio da rede também existe um computador fixo que injeta tráfego na rede, congestionando o meio.

Os cenários utilizados são de acordo com a figura anterior:

- Cenário 1: nó sem fio parado recebendo dados multimídia do servidor;
- Cenário 2: nó sem fio se movimentando, inicialmente se afastando do servidor e depois voltando para perto dele;
- Cenário 3: nó sem fio parado recebendo dados multimídia do servidor, tendo interferência de outro nó móvel no meio do caminho e de tráfegos gerados por outro computador ligado em algum roteador;
- Cenário 4: nó sem fio se movimentando, inicialmente se afastando do servidor e depois voltando para perto dele, tendo interferência de outro nó móvel no meio do caminho e de tráfegos gerados por outro computador ligado em algum roteador;

Para cada um desses foram executados 8 experimentos, sendo 4 com a proposta de suporte a QoS e 4 sem o *framework*. A comparação dos resultados com e sem QoS tem como objetivo avaliar o desempenho da proposta. A taxa de envio dos testes sem QoS se manteve em 25fps, e dos testes com QoS variou entre 25fps e 10 fps, dependendo do nível de adaptação em que o fluxo se encontrava.

É importante observar que os testes em cada cenário não foram executados nos mesmo dias e horários. Logo, muitas vezes as diferenças entre cenas e experimentos se deve a instabilidade ou não que a rede *Mesh* estava sofrendo no momento dos testes.

A **Error! Reference source not found.** apresenta as medidas de *jitter* ao longo da transmissão com QoS nos 4 experimentos da cena 1, juntamente com a taxa de envio dos dados em *frames* por segundo. As linhas azul e laranja representam os limites máximo e mínimo, respectivamente, de *jitter* usados para acionar a adaptação. Logo em seguida estão apresentados os gráficos com as medidas de perda de pacotes para os mesmos experimentos,

junto com a taxa de envio em fps. Novamente as linhas azul e laranja representam os limites máximo e mínimo para acionar a adaptação. A taxa de transmissão é alterada conforme a qualidade de *jitter* e perda de pacotes é degradada ou melhorada.

De acordo com os gráficos, podemos perceber que a transmissão se manteve estável. Por ser um cenário em que o nó cliente estava parado, os picos maiores de *jitter* e perda foram causados por alguma interferência não controlada ou instabilidade da rede. Comparando com a Figura 54, que mostra os resultados dos 4 experimentos sem QoS, o *framework* desempenhou um papel importante de estabilizar a qualidade da transmissão. A melhora de desempenho também foi visível na reprodução do vídeo. Nos momentos de maior degradação da rede, o vídeo que estava sendo recebido sem QoS apresentou muitas falhas, além de ser perceptível a grande quantidade de pacotes que chegaram fora de ordem.

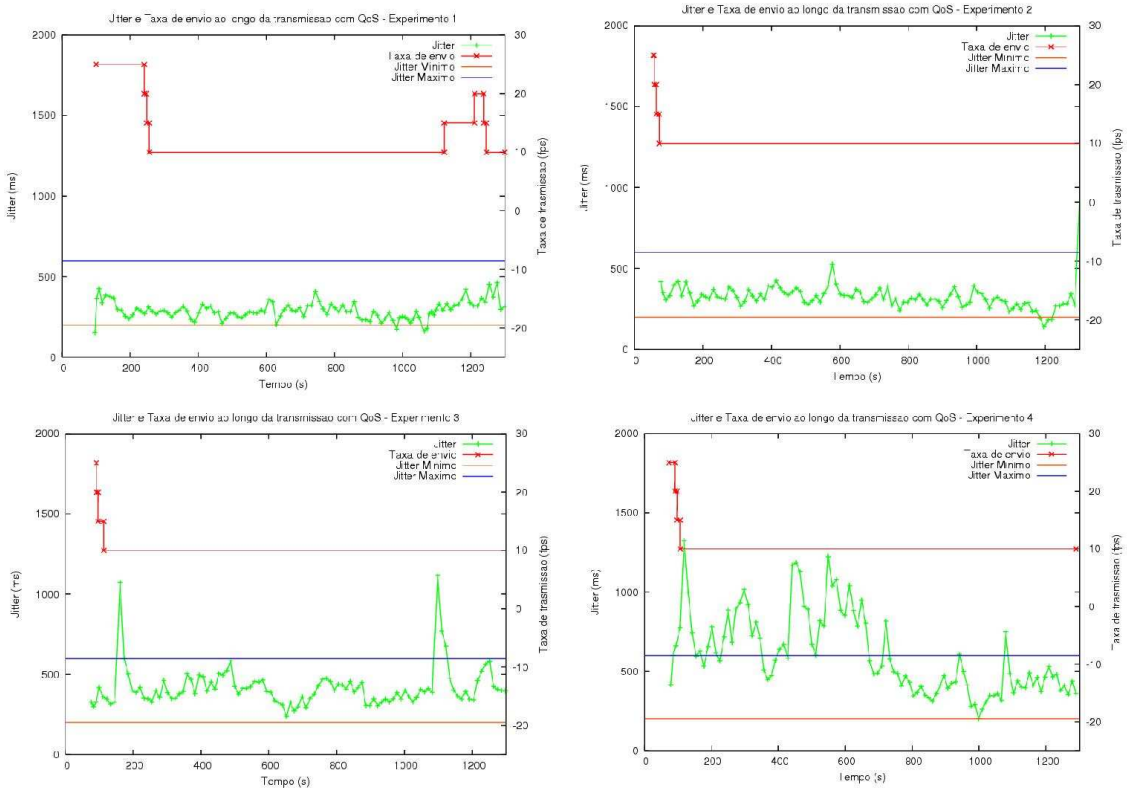


Figura 52: *Jitter* e Taxa de envio em 4 experimentos da cena 1 com QoS.

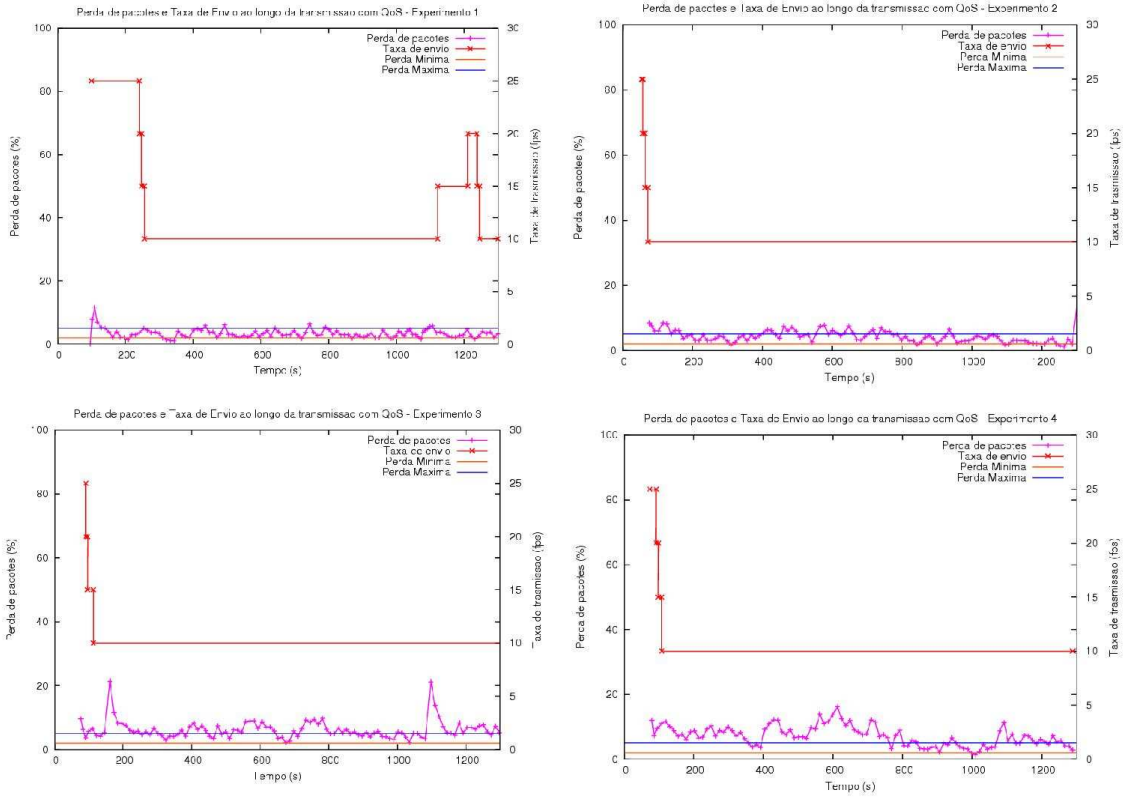


Figura 53: Perda de pacotes e taxa de envio em 4 experimentos da cena 1 com QoS.

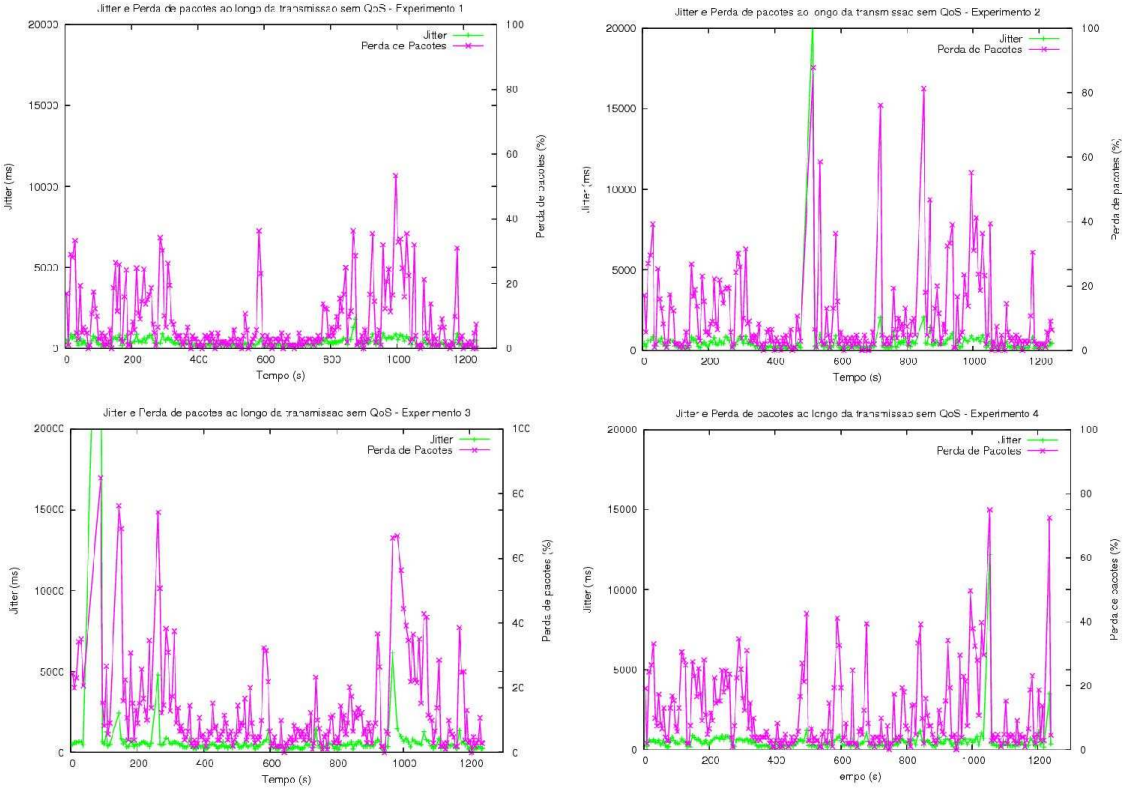


Figura 54: *Jitter* e Perda de pacotes em 4 experimentos da cena 1 sem QoS.

Os gráficos das Figura 55 e Figura 56 apresentam os resultados dos 4 experimentos com QoS da cena2, em que o nó cliente se movimenta inicialmente se afastando do servidor e ao final voltando a se aproximar do nó origem.

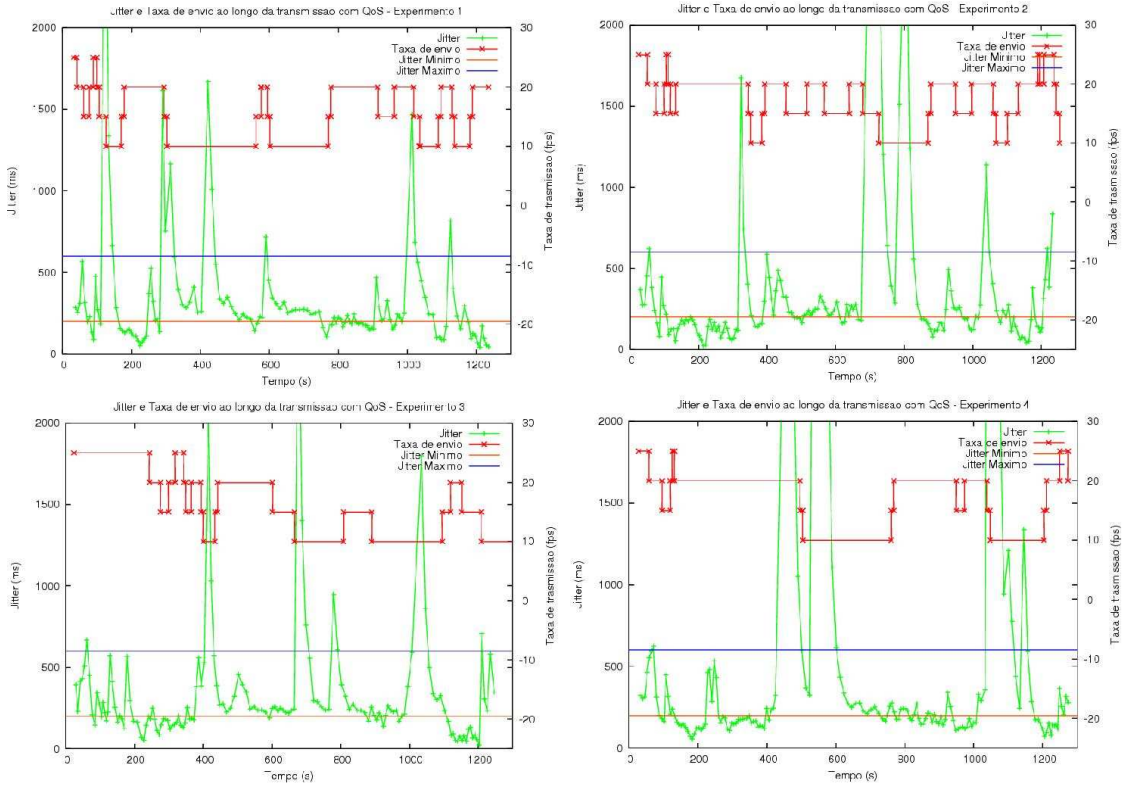


Figura 55: *Jitter* e Taxa de envio em 4 experimentos da cena 2 com QoS.

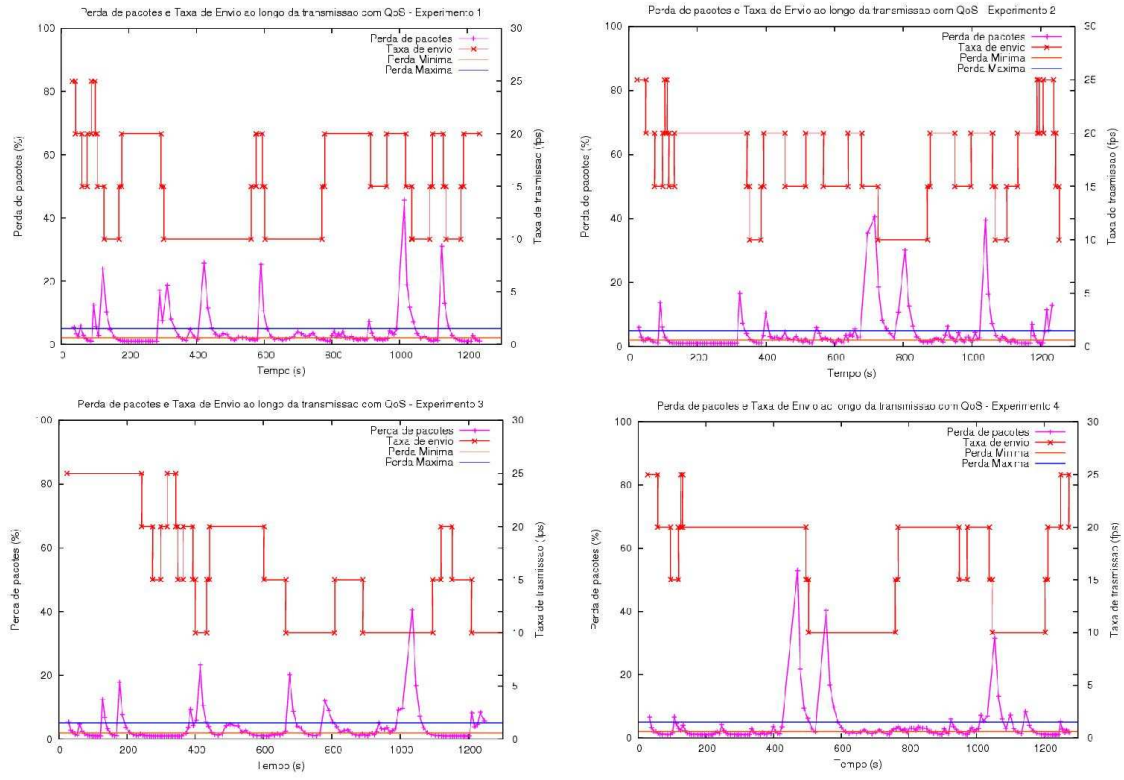


Figura 56: Perda de pacotes e taxa de envio em 4 experimentos da cena 2 com QoS.

Esses gráficos apresentam maiores picos de *jitter* e perda que os anteriores pois o fato do nó cliente estar se movimentando faz com que novas rotas precisem ser encontradas para que a transferência dos dados possa continuar. É interessante observar, principalmente nos experimentos 1 e 4 o mecanismo de estabilização agindo. Esse mecanismo analisa se o fluxo já esteve em um determinado nível de adaptação muitas vezes, e que não conseguiu se manter nele. Caso isto aconteça, o *framework* não permite que o fluxo volte para este determinado nível por um período de tempo.

No experimento 1, por volta do segundo 200, tanto a perda de pacotes como o *jitter* apresentaram valores abaixo do limite mínimo. Isso significa que ele deveria subir mais um nível de adaptação, enviando a uma taxa de 25fps. Porém, como o fluxo já esteve nesse nível por duas vezes e não conseguiu se manter, o *framework* não voltou a ele, mantendo a qualidade desejável para a aplicação. Os picos seguintes que fizeram o nível descer novamente foram causados por mudanças de rota. No experimento 4 podemos observar o mesmo comportamento entre os segundos 200 e 400, e também entre 800 e 1000. Porém, quase no final do teste, o *framework* permitiu que o fluxo voltasse ao nível 0, com taxa de transmissão de 25fps, para que fosse possível observar se a rede já estava em condições de suportar essa taxa.

Os gráficos da Figura 57 apresentam os resultados de 4 experimentos da cena 2 sem o *framework* de suporte a QoS. Como pode-se perceber, as métricas medidas apresentaram em muitos momentos maiores taxas e variações que os testes com QoS.

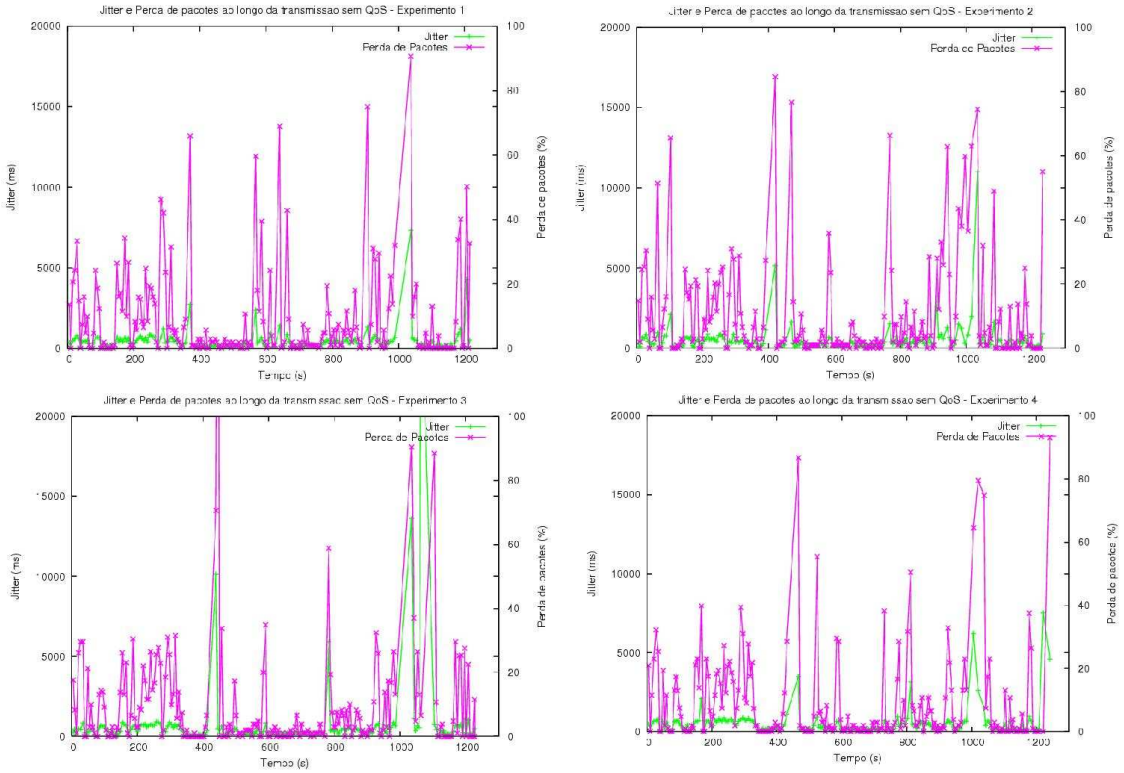


Figura 57: *Jitter* e Perda de pacotes em 4 experimentos da cena 2 sem QoS.

Apresentamos agora os resultados do cenário 3, que consiste na transferência de vídeo com o cliente parado, mas sendo injetado tráfego extra na rede. Foram gerados dois tipos de tráfego extra: com 1 minuto de teste, se iniciou um tráfego FTP de um arquivo de 23Mbytes, e após dois minutos do fim da sua transferência, se iniciou outro FTP e assim por diante até o fim da execução; aos 10 minutos, o nó cliente fez a requisição de um vídeo na Internet para ser reproduzido no próprio nó.

Os resultados mais interessantes não aparecem nos gráficos e foram obtidos a partir da observação dos dois vídeos sendo visualizados no nó cliente. Nos testes sem QoS, onde os dois tráfegos estavam disputando a banda da classe HTB Outros, a visualização dos dois vídeos ficou prejudicada. Já nos testes com QoS, onde cada tráfego estava sendo encaminhado para uma classe diferente, um não afetou a transmissão do outro.

As Figura 58 e Figura 59 apresentam os gráficos dos testes com QoS e injeção de tráfego. A Figura 60 apresenta os gráficos dos testes sem QoS e também injeção de tráfego extra.

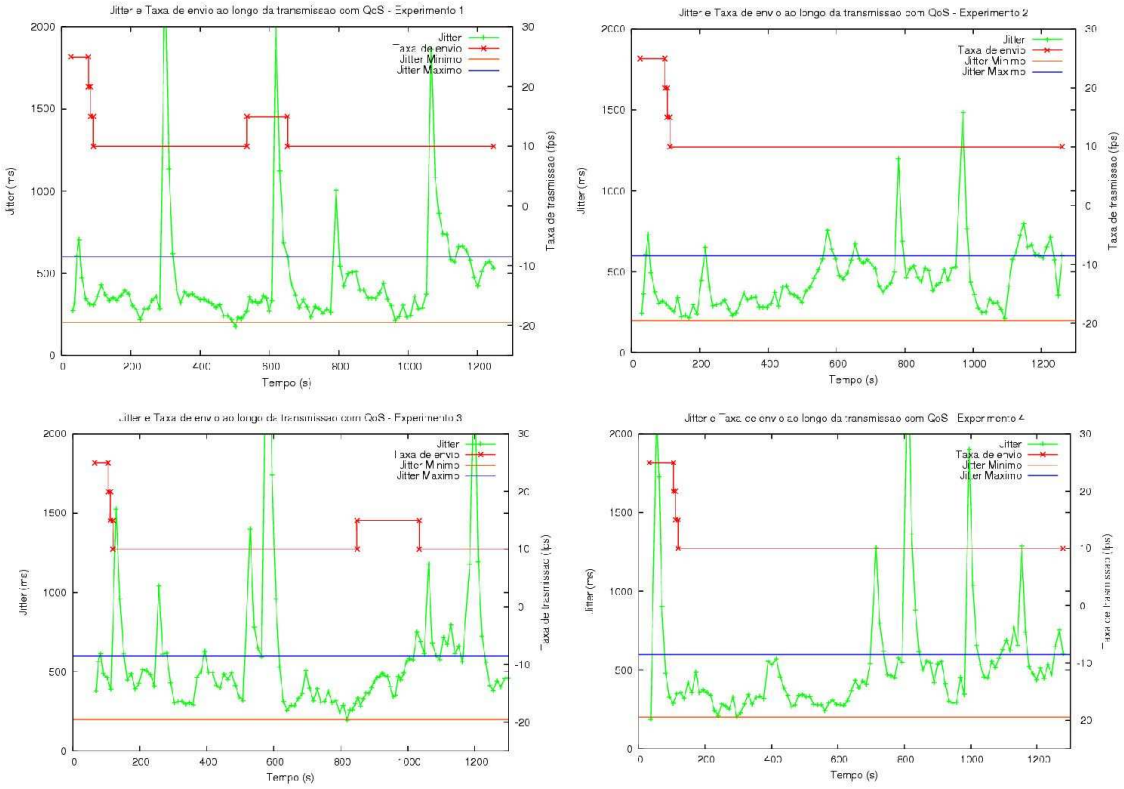


Figura 58: *Jitter* e Taxa de envio em 4 experimentos da cena 3 com QoS.

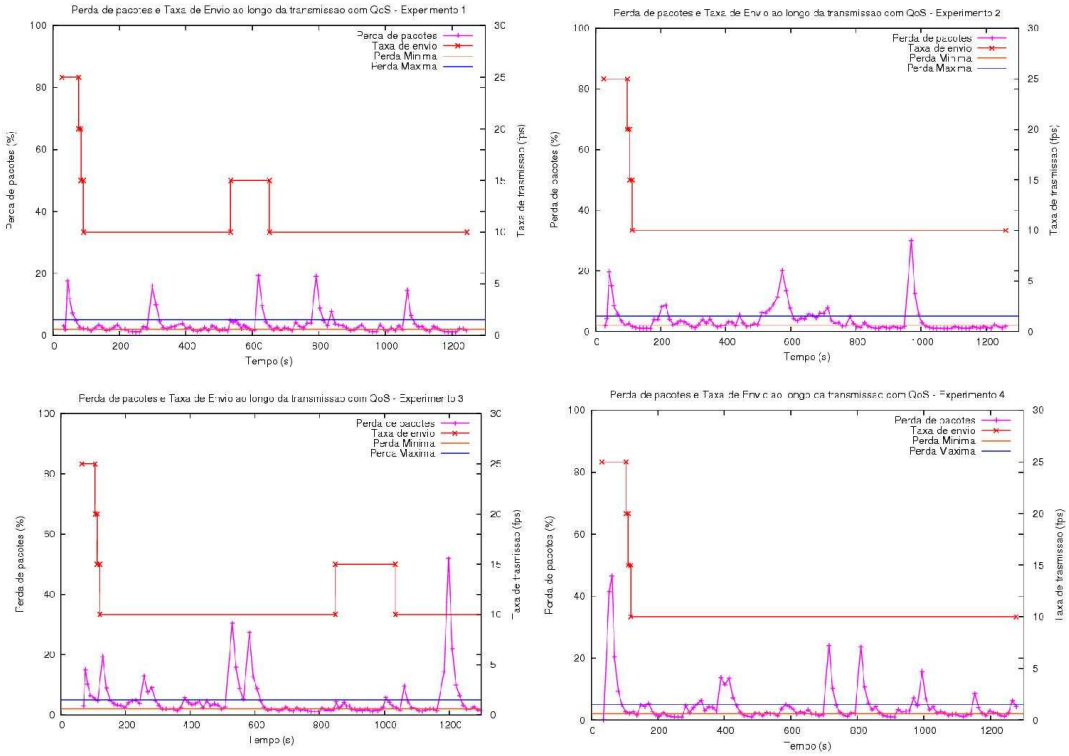


Figura 59: Perda de pacotes e taxa de envio em 4 experimentos da cena 3 com QoS.

Nos experimentos sem QoS, além da grande taxa de perda de pacotes e *jitter*, podemos observar no experimento 1 grandes períodos de desconexão, como entre 200 e 600 segundos, o que geraram taxas bem elevadas das métricas.

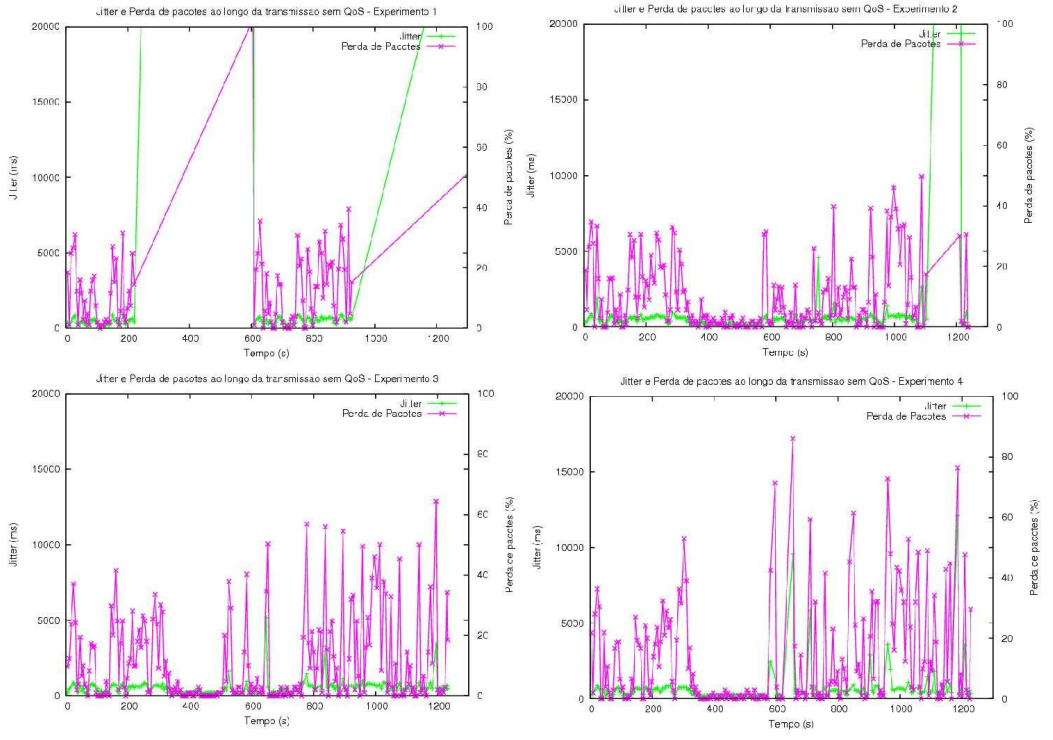


Figura 60: *Jitter* e Perda de pacotes em 4 experimentos da cena 3 sem QoS.

Os experimentos do cenário 4 foram executados em momentos em que a rede *Mesh* apresentava grande instabilidade. Como nesse cenário o nó cliente se move, a geração de novas rotas se tornou ainda mais lenta, gerando períodos de desconexão. A avaliação dos resultados observados foram as mesmas do cenário anterior.

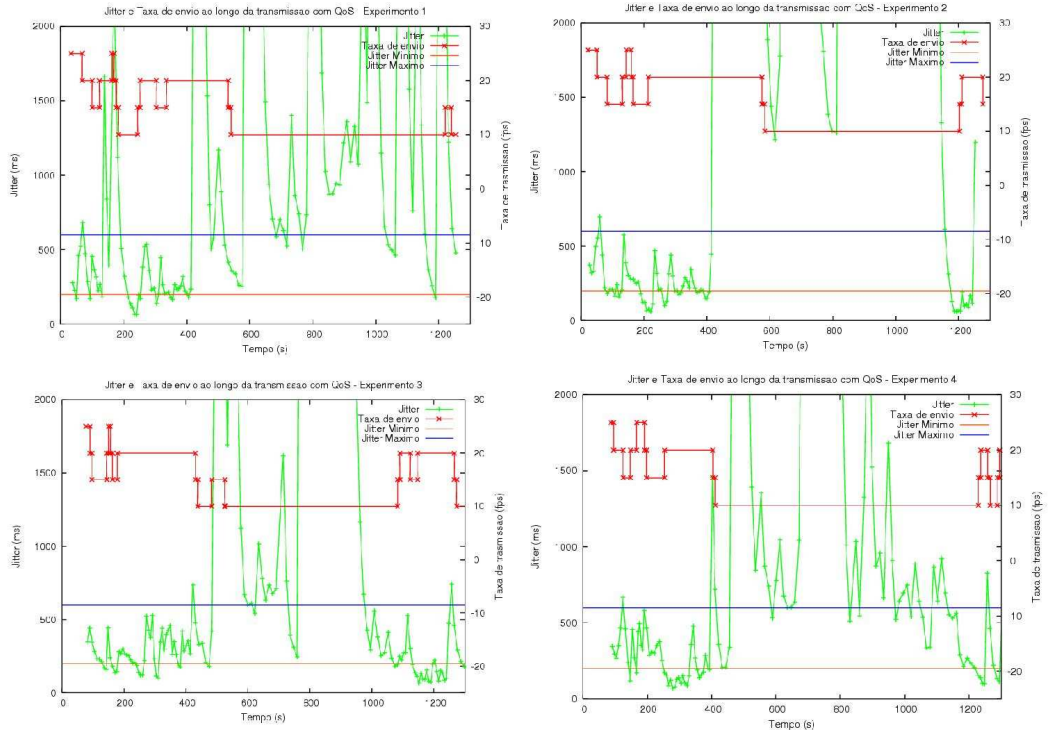


Figura 61: *Jitter* e Taxa de envio em 4 experimentos da cena 4 com QoS.

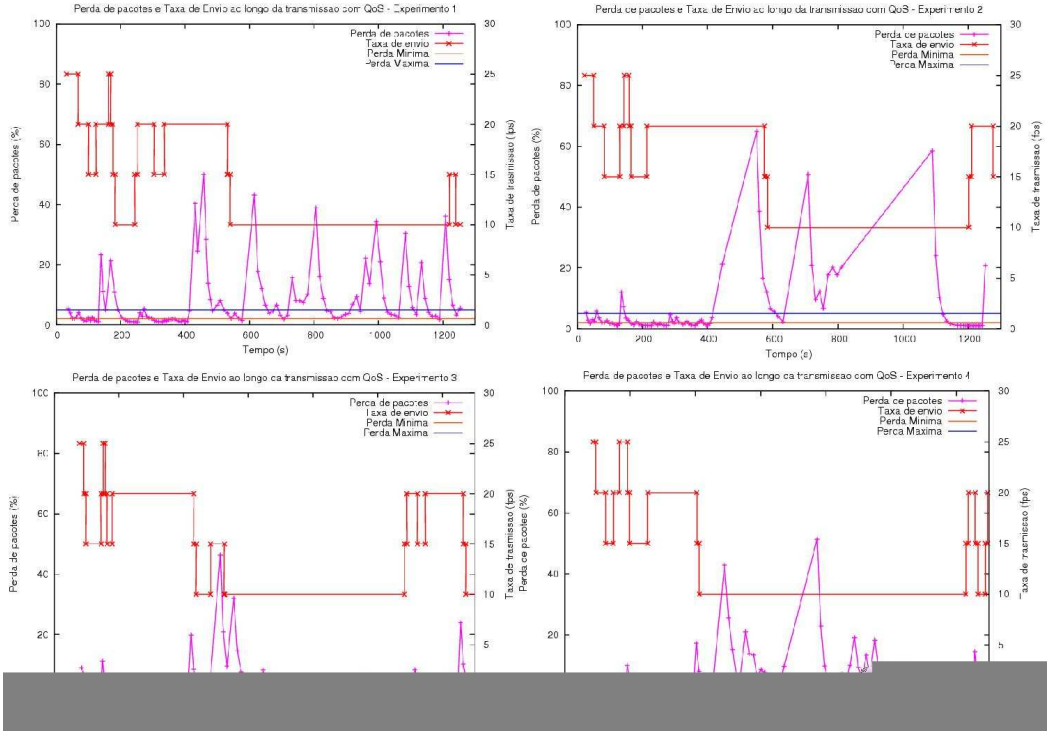


Figura 62: Perda de pacotes e taxa de envio em 4 experimentos da cena 4 com QoS.

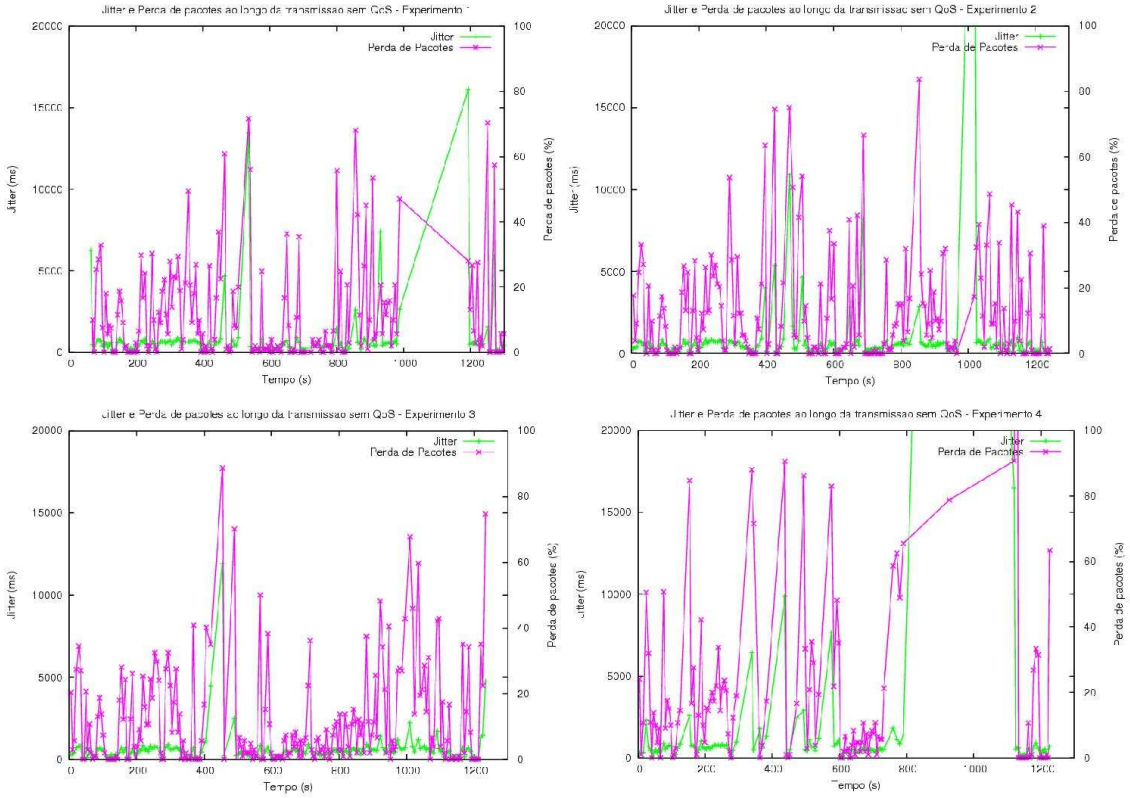


Figura 63: *Jitter* e Perda de pacotes em 4 experimentos da cena 3 sem QoS.

Os resultados obtidos com os testes do protótipo ficaram de acordo com os resultados gerados a partir da simulação feita anteriormente como um estudo preliminar. Os dois resultados possuem algumas distinções, devido a diferença de protocolo de roteamento utilizado e também pelo fato de que nos testes com o protótipo não é possível controlar interferências externas. Já na simulação o ambiente pode ser controlado. Os resultados do protótipo mostraram que a proposta atingiu o objetivo de minimizar perdas e fornecer qualidade a transmissão de vídeo.

3. Avaliação

O GT Mesh desenvolveu um prototótipo de roteador mesh sem fio e já implantou um primeiro projeto piloto de rede de acesso banda larga utilizando o protótipo desenvolvido nas proximidades do campus Praia Vermelha da UFF em Niterói, RJ. O GT atingiu todos os seus objetivos principais e demonstrou a viabilidade do uso de redes mesh para construção de redes de acesso banda larga de baixo custo, sendo a primeira experiência universitária brasileira bem sucedida na construção de redes mesh.

A rede de acesso universitária da UFF está em operação desde o final de março de 2006, tendo atualmente seis roteadores sem fio em operação na rede externa. Além da rede externa, o GT mantém também uma rede interna de testes em funcionamento, onde estão sendo realizados testes com novos protocolos, incluindo o framework de QoS. Desde abril de 2006, a rede mesh externa já foi usada para a transferência de 300GB de informação mostrando-se bastante estável e atendendo às expectativas dos usuários. Atualmente, possui 41 usuários cadastrados que utilizam a rede diariamente.

Um aspecto importante que diferencia o protótipo construído pelo GT Mesh das outras soluções comerciais é o custo de cada roteador mesh. Nas soluções de grandes fabricantes, como a Cisco, por exemplo, um roteador mesh custa alguns milhares de dólares, enquanto que cada ponto mesh instalado pelo GT tem um custo de menos de R$ 1.000,00. Por isso, além do aspecto científico e tecnológico, este projeto se enquadra perfeitamente como solução de acesso visando à inclusão social e digital em universidades brasileiras e outras comunidades. Desta forma, o seu potencial para utilização por outras instituições e POPs da RNP é bastante expressivo. A realidade brasileira mostra que grande parte dos usuários da comunidade universitária e das instituições de ensino em geral não possui condições de arcar com os altos custos de uma conexão faixa larga tradicional do tipo ADSL ou cabo. O desenvolvimento e implantação de redes de acesso faixa larga sem fio do tipo mesh em POPs e instituições ligadas à RNP é portanto uma alternativa de acesso de baixo custo altamente desejável.

Outro potencial de redes mesh, além da implantação de redes universitárias de acesso à Internet é a construção de cidades digitais, que são cidades que oferecem conectividade a todos os seus cidadãos. Algumas cidades brasileiras, como Tiradentes, e cidades estrangeiras como Taipei, já tiveram experiências desse tipo usando soluções mesh comerciais da Cisco e Nortel respectivamente. Entretanto, como já salientado, as soluções comerciais apresentam custo elevado. Um outro potencial do protótipo desenvolvido pelo GT Mesh é sua aplicação para construção de cidades digitais em todo o país.

Durante todo o ano de desenvolvimento do projeto, principalmente após o início da implantação da rede de acesso externa, o GT teve ampla repercussão na mídia e despertou o interesse de várias empresas, prefeituras e outras universidades para uso do protótipo desenvolvido. Esse interesse da comunidade demonstra a necessidade atual de soluções de acesso não tradicionais e de baixo custo, reforçando o potencial de utilização do protótipo em outras instituições e localidades brasileiras.

A seguir são listadas as principais repercussões dos trabalhos realizados pelo GT:

- Apresentação do Prof. Schara sobre “Redes em Malha para Inclusão Digital” no 7º. Fórum Internacional de Software Livre, em Porto Alegre – 22/04/06
- Citação do GT Mesh na reportagem sobre acesso sem fio a Internet – Jornal O DIA – 27/04/2006
- Entrevista da Prof. Débora para o programa Bom Dia Paraná sobre o trabalho desenvolvido pelo GT Mesh – 30/05/2006
- Apresentação do Prof. Schara sobre “Conectividade: Redes em Malha, Banda Larga para Todos” na 5ª. Oficina para Inclusão Digital, em Porto Alegre – 05/06/2006
- Palestra do Prof. Célio sobre “Redes Mesh: Estendendo os limites de redes Wi-Fi tradicionais" na Escola Superior de Redes da RNP – 20/06/2006
- Palestra do Prof. Célio sobre Redes Mesh no evento “NCE Debate TI” a ser realizada no Núcleo de Computação Eletrônica da UFRJ em 19/10/2006
- Reportagem “Redes Mesh ampliam o alcance do Wi-Fi” na Revista TI de 28/06/2006
- Reportagem na Revista Fórum PCs de 09/06/2006
- Citação da UFF no site da Microsoft Research sobre universidades que desenvolvem pesquisas em redes Mesh.
- Contato e grupos interessados em replicar a solução – LNCC, UFPA, UTF-PR
- Contato de empresas interessadas em usar a solução – TBE (Transmissoras Brasileiras de Energia) entre outras
- Contato da prefeitura de Rio das Ostras para utilização da solução
- Contato do Ministério do Planejamento para testes da solução desenvolvida pelo GT Mesh na Vila Planalto em Brasília – agosto/2006

A continuação dos trabalhos do GT Mesh será fundamental para a ampliação do uso do protótipo desenvolvido, refinamento das ferramentas de gerência da rede e oferecimento de novos serviços sobre a rede mesh, tais como VoIP e TV Digital Interativa. O status atual estável do protótipo, seu bom desempenho, sua comprovada utilização, a satisfação dos usuários, o baixo custo e a repercussão positiva do projeto são fatores que indicam seu alto potencial para suporte a um serviço de conectividade banda larga sem fio a ser oferecido pela RNP a comunidade acadêmica de diversas instituições brasileiras.

Referências

[Calafate 2002] Calafate, C. M. T. OLSR for Windows 2000 and Pocket PC. 2002. Disponível em: <http://www.grc.upv.es/calafate/olsr/olsr.htm>.

[Hubert et al. 2003] Hubert, B.; Graf, T.; Maxwell, G.;van Mook, R.;van Oosterhout Paul B Schroeder, M.; Spaans, J. e Larroy, P. Linux Advanced Routing & Traffic Control HOWTO. 2003. Disponível em: <http://www.lartc.org/>.

[LARTC 2005] LARTC. Linux Advanced Routing & Traffic Control. 2005. Disponível em: <http://www.lartc.org/>.

Apêndice 1 - Código Fonte do Protótipo

A1.1: Fontes executáveis

A1.1.1: Olsrd

As mudanças feitas no código fonte do protocolo olsr são baseadas na versão 0.4.9. Dois arquivos foram alterados.

A1.1.1.1: olsr_cfg.h

Na linha 78, é definida uma constante:

```
#define MAX_LQ_LEVEL        2
```

O seu valor foi alterado de 2 para 3 para permitir mais uma possibilidade de configuração do parâmetro *LinkQualityLevel.* Com isso, esta linha ficará:

```
#define MAX_LQ_LEVEL        3
```

A1.1.1.2: lq_route.c

Na linha 267 deste arquivo, existe a seguinte função:

```
267: static void relax(struct dijk_vertex *vert)
268: {
269:  struct dijk_edge *edge;
270:  float new_etx;
271:  struct list_node *node;
272:
273:  node = list_get_head(&vert->edge_list);
274:
275:  // loop through all edges of this vertex
276:
277:  while (node != NULL)
278:   {
```

```
279:     edge = node->data;
280:
281:     // total quality of the path through this vertex to the
282:     // destination of this edge
283:
284:     new_etx = vert->path_etx + edge->etx;
285:
286:     // if it's better than the current path quality of this
287:     // edge's destination, then we've found a better path to
288:     // this destination
289:
290:     if (new_etx < edge->dest->path_etx)
291:     {
292:       edge->dest->path_etx = new_etx;
293:       edge->dest->prev = vert;
294:     }
295:
296:     node = list_get_next(node);
297:   }
298: }
```

Nela, 4 linhas foram adicionadas. A função ficará como listada abaixo (as linhas adicionadas estão indicadas no código):

```
267: static void relax(struct dijk_vertex *vert)
268: {
269:   struct dijk_edge *edge;
270:   float new_etx;
271:   struct list_node *node;
272:
273:   node = list_get_head(&vert->edge_list);
274:
275:    if  (olsr_cnf->lq_level  ==  3)  vert->path_etx  =  1;  //linha
adicionada
276:
277:   // loop through all edges of this vertex
278:
279:   vert->path_etx = 1;
280:   while (node != NULL)
281:   {
282:     edge = node->data;
283:
284:     // total quality of the path through this vertex to the
285:     // destination of this edge
286:
287:    if (olsr_cnf->lq_level == 3) //linha adicionada
289:      new_etx = vert->path_etx * edge->etx; //linha adicionada
290:    else  //linha adicionada
291:      new_etx = vert->path_etx + edge->etx;
292:
293:     // if it's better than the current path quality of this
294:     // edge's destination, then we've found a better path to
295:     // this destination
296:
297:     if (new_etx < edge->dest->path_etx)
298:     {
299:       edge->dest->path_etx = new_etx;
300:       edge->dest->prev = vert;
```

```
301:     }
302:
303:     node = list_get_next(node);
304:   }
305: }
```

A1.1.2: wifidog

Como explicado anteriormente, o wifidog é dividido em dois módulos: um binário que deve ser executado dentro do roteador e um em PHP 5, que constitui um conjunto de páginas dinâmicas instaladas dentro do servidor de autenticação.

A1.1.2.1: Módulo binário do wifidog (roteador gateway)

O módulo binário do wifidog a ser inserido nos nós comuns da rede (repetidores) não sofreu alteração. Contudo, no nó gateway ele teve que modificado para operar como agente de autenticação dos clientes móveis. As alterações realizadas foram para permitir que o módulo não filtre os clientes pelo MAC. Isto deve ser feito pois, no caso de clientes móveis, eles serão roteados a partir dos seus nós de origem até o nó gateway. Desta forma, o gateway não os haverá em sua tabela MAC, impossibilitando a verificação. As alterações realizadas no pacote original foram:

Arquivo src/fw_iptables.c

Linha 431 passou de:

```
iptables_do_command("-t mangle -A " TABLE_WIFIDOG_OUTGOING " -s %s -m
mac --mac-source %s -j MARK --set-mark %d", ip, mac, tag);
```

Para

```
iptables_do_command("-t mangle -A " TABLE_WIFIDOG_OUTGOING " -s %s -j
MARK --set-mark %d", ip, tag);
```

Linha 435 passou de:

```
iptables_do_command("-t mangle -D " TABLE_WIFIDOG_OUTGOING " -s %s -m
mac --mac-source %s -j MARK --set-mark %d", ip, mac, tag);
```

Para

```
iptables_do_command("-t mangle -D " TABLE_WIFIDOG_OUTGOING " -s %s -j
MARK --set-mark %d", ip, tag);
```

Arquivo src/firewall.c

Após a linha 138 “*fclose(proc)*” na função *iptables_fw_access*, adicionar o seguinte trecho:

```
if (!reply) {
        for (i = 0; i < 18; i++) {
            if ((i != 0) && (!(i & 1))) mac[i] = ':';
            else {
                if (j < 16) {
                    if (ip[j] != '.') mac[i] = ip[j];
                    j++;
                }
                else mac[i] = '0';
            }
        }
    }
    reply = safe_strdup(mac);
    return reply;
} //Fim da função
```

Para gerar o binário executável no roteador, serão necessários os pacotes do wifidog e do *openwrt* instalados na mesma máquina:

- Entre no diretório do wifidog aonde se encontra o arquivo Makefile
- Digite "make CC=<*caminho_para_openwrt*>/staging_dir_mipsel/bin/mipsel-linux-gcc LD=<*caminho_para_openwrt*>/staging_dir_mipsel/bin/mipsel-linux-ld
- Haverá um erro de compilação já esperado, execute então
- <*caminho_para_openwrt*>/staging_dir_mipsel/bin/mipsel-linux-ranlib <*caminho_para_wifidog*>/libhttpd/.libs/libhttpd.a
- E, em seguinda execute o make novamente:

```
make CC=<caminho_para_openwrt>/staging_dir_mipsel/bin/mipsel-linux-gcc
LD=<caminho_para_openwrt>/staging_dir_mipsel/bin/mipsel-linux-ld
```

A1.1.2.2: Módulo PHP do wifidog (servidor de autenticação)

As alterações no módulo PHP foram feitas para customizar o servidor para os interesses do projeto. Para cada objetivo, um conjunto de alterações foram realizadas. Abaixo estão listadas (considere o local de modificação a pasta *wifidog-auth-1.0.0_m2/wifidog/* como raiz dos diretórios listados abaixo*)*:

- Alteração para o não haver o período de validação onde, na versão original, o servidor permite a conexão do novo usuário por 20 minutos para que ele acesse seu email, enviado pelo servidor, validando a sua entrada.

Arquivo admin/signup.php

Trocar a linha:

```
//throw new Exception ("Somente administradores podem criar contas.");
```

Comentar as linhas:

```
//$created_user->sendValidationEmail();
```

```
//$smarty->assign('message', _('An email with confirmation instructions was sent to your email address.  Your account has been granted 15 minutes of access to retrieve your email and validate your account.  You may now open a browser window and go to any remote Internet address to obtain the login page.'));
```

```
//$smarty->display("templates/validate.html");
```

Colocar no lugar da última:

```
$smarty->assign('message', _('Conta criada.'));
```

Arquivo classes/User.php

Linha 176 mudou de

```
$status = $ACCOUNT_STATUS_VALIDATION;
```

Para

```
$status = '1'; // ACCOUNT_STATUS_VALIDATION;
```

- Alteração para não permitir múltiplos usuários com o mesmo login e mostrar uma tela de notificação

  Primeiro deve-se criar no diretório raiz um arquivo chamado “msgfile.txt” com permissão de escrita para todos;

  Em seguida altere o arquivo classes/AuthenticatorLocalUser.php:

  Abaixo da linha 66 (“$db->ExecSqlUniqueRes($sql, $user_info, false”);, apague tudo até a linha 101 (“/** Start accounting traffic for the user */”) insira neste trecho excluído:

  /*Modificacoes para nao pemitir que o mesmo login seja usado em duas autenticacoes ao mesmo tempo!!!*/

```
        $sql = "SELECT conn_id FROM connections WHERE (user_id = '{$user_info['user_id']}' AND token_status = 'INUSE');\n";
        $conn_rows = null;
        $db->ExecSql($sql, $conn_rows, false);
        /*Fim*/

        if ($conn_rows != null)
        {
            foreach ($conn_rows as $conn_row)
            {
                parent :: acctStop($conn_row['conn_id']);
            }
        }
        if ($user_info != null)
        {

            $user = new User($user_info['user_id']);
            if ($user->isUserValid($errmsg))
            {
                $retval = & $user;
                User::setCurrentUser($user);
                if ($conn_rows == null)
                    $errmsg = _("Login successfull");
                else{
                    $mensagem = _(" <br><br><strong>Remesh: </strong><font color=\"#FF0000\"><strong>ATEN&Ccedil;&Atilde;O! Sua conta estava em uso em outra m&aacute;quina ou voc&ecirc; mudou de ponto de acesso.</strong><br><br></font>");
                    fwrite($fh,$mensagem);

                }
            }           else
            {
                $retval = false;
                //Reason for refusal is already in $errmsg
            }
        }
        else
        {
            $user_info = null;              /* This is only used to discriminate if the problem was a non-existent user of a wrong password. */ $db->ExecSqlUniqueRes("SELECT * FROM users WHERE (username='$username' OR email='$username') AND account_origin='".$this->getNetwork()->getId()."'", $user_info, false);
```

```
                if ($user_info == null)
                {
                        $errmsg = _('Unknown username or email');
                }
                else
                {
                        if ($conn_rows != null)
                        {
                                $errmsg = _('Username already in use.');
                        }
                        else                                { $errmsg =
_('Incorrect password (Maybe you have CAPS LOCK on?)');
                        }
                }
                $retval = false;
        }
        return $retval;
    fclose($fh);
    }

    /** Start accounting traffic for the user */
    function acctStart($info, & $errmsg = null)
    {
        parent :: acctStart($info);
        return true;
    }

    /** Update traffic counters */      function acctUpdate($info, $incoming, $outgoing,
& $errmsg = null)
    {
                                                                    133,2-9       83%
                $user_info = null;
                /* This is only used to discriminate if the problem was a non-existent user
of a wrong password. */
                $db->ExecSqlUniqueRes("SELECT * FROM users WHERE
(username='$username' OR email='$username') AND account_origin='".$this-
>getNetwork()->getId()."'", $user_info, false);
                if ($user_info == null)
                {
                        $errmsg = _('Unknown username or email');
                }
                else
                {
                        if ($conn_rows != null)
                        {
                                $errmsg = _('Username already in use.');
                        }
                        else
                        {
                                $errmsg = _('Incorrect password (Maybe you have CAPS
LOCK on?)');
                        }
                }
                $retval = false;
        }
        return $retval;
    fclose($fh);
    }
```

- Alteração para não permitir que usuários comuns cadastrem novos usuários

Simplesmente apague ou retire as permissões de leitura e execução do arquivo /signup.php. Lembre-se que o arquivo admin/signup.php deverá ser mantido para que o administrador consiga inserir novos usuários.

- Alteração para exibição de mensagem para clientes temporariamente desativados

  No arquivo classes/User.php troque a linha

  $errmsg = _("Sorry, your account is not valid:
").$account_status_to_text[$account_status];

  Por

  $errmsg = _("Desculpe-nos, no momento estamos em fase de testes e a rede não estará disponível. Por favor, tente mais tarde.").$account_status_to_text[$account_status];

- Criação de página para visualização aberta das principais estatísticas de uso da rede. Esta página deverá ser acessada através de um *link* especificado no servidor web para que funcione corretamente

  Para a visualização das estatísticas no servidor web foram criadas duas novas páginas adicionais no conjunto das páginas do wifidog localizada no diretório raiz.

  Arquivo */stats.php:*

```
 /**@file stats.php
  * @author Copyright (C) 2005 Philippe April
  */
  /*
ini_set('error_reporting', E_ALL);
ini_set('display_errors', true);
*/
//douglas -- alterou basepath -- este arquivo nao pertence a este diretorio
define('BASEPATH','./');
require_once BASEPATH.'admin/admin_common.php';
require_once BASEPATH.'classes/MainUI.php';
require_once BASEPATH.'classes/Utils.php';

$current_user = User::getCurrentUser();

$statistics = new Statistics();
      if(!empty($_REQUEST['action']) &&  $_REQUEST['action'] == 'generate')
      {
$statistics -> processAdminUI();
      }
try {
   if (!empty($_REQUEST['node_id'])) {
         $node_id = $db->EscapeString($_REQUEST["node_id"]);
      $nodeObject = Node::getObject($node_id);
      $stats_title = _("Connections at") . " '" . $nodeObject->getName() . "'";
   } elseif (isset($_REQUEST['user_id'])) {
      $user_id = $db->EscapeString($_REQUEST["user_id"]);
      $userObject = User::getObject($user_id);
      $stats_title = _("User information for") . " '" . $userObject->getUsername() .
"'";
```

```
    } elseif (isset($_REQUEST['user_mac'])) {
        $user_mac = $db->EscapeString($_REQUEST["user_mac"]);
        $stats_title = _("Connections from MAC") . " '" . $user_mac . "'";
    } elseif (isset($_REQUEST['network_id'])) {
            $network_id = $db->EscapeString($_REQUEST["network_id"]);
        $networkObject = Network::getObject($network_id);
        $stats_title = _("Network information for") . " '" . $networkObject->getName() . "'";
    }

    $html = '';


    if (isset($stats_title)) {
        $html .= "<h2>{$stats_title}</h2>";
            }
        $html .= "<form>";
//        $html .= $statistics->getAdminUI();

        if (isset($_REQUEST['node_id'])) {
            $html .= "<input type='hidden' id='node_id' name='node_id' value='{$_REQUEST['node_id']}'>";
        } elseif (isset($_REQUEST['user_id'])) {
            $html .= "<input type='hidden' id='user_id' name='user_id' value='{$_REQUEST['user_id']}'>";
        } elseif (isset($_REQUEST['user_mac'])) {
            $html .= "<input type='hidden' id='user_mac' name='user_mac' value='{$_REQUEST['user_mac']}'>";
        } elseif (isset($_REQUEST['network_id'])) {
            $html .= "<input type='hidden' id='network_id' name='network_id' value='{$_REQUEST['network_id']}'>";
        }

        $html .= "<input type='hidden' name='action' value='generate'>";


 //       $html .= "<input type='submit' value='" . _("Generate statistics") . "'>";
        $html .= "</form>";
        $html .= "<hr>";
$html .= $statistics->getReportUI();

    //$sql = "select user_mac,count(user_mac) as nb,max(timestamp_in) as last_seen,substract(timestamp_in, timestamp_out) as time_spend from c
onnections where node_id='{$node_id}' group by user_mac order by nb desc";
/*
    if (isset($node_id) && ($current_user->isSuperAdmin() || $current_user->isOwner())) {
        include "stats_node.inc.php";
    } elseif (isset($user_id) && $current_user->isSuperAdmin()) {
        include "stats_user_id.inc.php";
    } elseif (isset($user_mac) && $current_user->isSuperAdmin()) {
        include "stats_user_mac.inc.php";
    } elseif (isset($network_id) && $current_user->isSuperAdmin()) {
        include "stats_network.inc.php";
    } else if($current_user->isSuperAdmin()) {
        include "stats_all_networks.inc.php";
    }*/


} catch (exception $e) {
```

```
    $html = "<p class='error'>";
    $html .= $e->getMessage();
    $html .= "</p>";
}
$ui=new MainUI();
//$ui->setToolSection('ADMIN');
//$ui->setMainContent($html);
//$ui->display();
printf('<html>');
printf('<head>');
printf('<meta http-equiv="Content-Type" content="text/html; charset=utf-8">');
printf('<meta http-equiv="Pragma" CONTENT="no-cache">');
printf('<meta http-equiv="Expires" CONTENT="-1">');
printf('<html>');
printf('<head>');
printf('<title>ReMesh servidor de autenticacao</title>');
printf('<style type=\'text/css\'>');
$filename="local_content/default/stylesheet.css";
$fh=fopen($filename,'r');

$t='';
$t=fread($fh,filesize($filename));
fclose($fh);
print($t);


printf('</style> </head>');
printf($html);
```

Arquivo */stats_show_graph.php:*

```
/**@file graph_registrations_cumulative.php
  * parameters:
  * $_REQUEST['graph_class']: The name of the class for which we want the graph
  * displayed.
  * @author Copyright (C) 2005 Philippe April
  */
//douglas mudou basepath -- este arquivo nao pertence a este diretorio
define('BASEPATH',"./");

if(empty($_REQUEST['graph_class']))
{
      echo "You must specify the class of the graph you want";
}
else
{
      $classname = $_REQUEST['graph_class'];
require_once BASEPATH.'classes/StatisticGraph/'.$classname.'.php';
$graph = call_user_func(array ($classname, 'getObject'), $classname);
$graph->showImageData();
```

Ainda, comente todo o trecho demonstrado abaixo no arquivo classes/Statistics.php:

```
if ($user->isSuperAdmin())
      //    {
                  $sql_join = '';
```

```
//      }
//      else
//      {
//              $user_id = $db->EscapeString($user->getId());
//                              $sql_join = " JOIN node_stakeholders ON
(nodes.node_id=node_stakeholders.node_id AND user_id='$user_id') ";
//      }
```

Para que o servidor web exiba as informações corretamente, use como argumentos no link:

http://<*endereço_servidor_autenticação*>/stats.php?Statistics=default-network&date_from=&date_to**=&statistics_selected_nodes%5B%5D=12&statistics_selected_nodes%5B%5D=2&statistics_selected_nodes%5B%5D=3&statistics_selected_nodes%5B%5D=5&statistics_selected_nodes%5B%5D=6&statistics_selected_nodes%5B%5D=8&statistics_selected_nodes%5B%5D=9&**distinguish_users_by=user_id&stats_selected_users=&HighestBandwidthUsers=on&MostFrequentUsers=on&MostMobileUsers=on&ConnectionGraphs=on&MostPopularNodes=on&UserRegistrationReport=on&action=generate', 'Topologia','height=800,width=900,scrollbars=yes,resizable=yes');

O endereço acima exibirá as informações do conjunto de nós da rede interna (em negrito): 8, 9, 12, 2, 6 e 5. Para alterar o conjunto de nós a serem exibidos, utilize a variável *statistics_selected_nodes****%5B%5D=(id do nó segundo o cadastro do wifidog).***

A1.2: Ferramentas de Gerência

A1.2.1: httpinfo.c

CGI criado para buscar as páginas geradas pelo plugin httpinfo e exibi-las, alterando partes do código html. Isso é feito para permitir o acesso de usuários externos a rede da UFF, sujeitos ao *firewall* que bloqueia portas altas, através do servidor web do projeto (interno a rede da uff). Para compilar, utilize:

gcc httpinfo.c –o httpinfo

É importante ressaltar que é necessário alterar na 18º linha da função main() o endereço IP do *gateway* da Rede Mesh (endereço da interface *wan*).

```
#include      <netdb.h>
#include      <netinet/in.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <stdio.h>
#include      <signal.h>
```

```
#include	<string.h>
#include	<stdlib.h>

//Função que modifica o código html da página original para alterar as referências dos links e //arquivos auxiliares.
void limpa_pagina(char *pag, char *port, char *tipo)
{
	char	*ch1, *ch2;
	int	tam;

	ch1 = strstr(pag, "<");
	ch2 = strstr(ch1,"\"httpinfo.css\"");

	for (; ch1 < (ch2); ch1++) printf("%c",*ch1);
	printf("\"http://mesh.ic.uff.br/topologia/httpinfo.css\"");
	ch1=ch2+14;
	ch2 = strstr(ch1, "<img");

	for (; ch1 < (ch2); ch1++) printf("%c",*ch1);
	printf(">    <img src=\"/imagens/novo_logo.jpg\">");

	ch2 = strstr(ch1, "\"tabnav\">");
	for (; ch1 < (ch2 + 10); ch1++) printf("%c",*ch1);
	printf("<li><a		href=\"http://mesh.ic.uff.br/cgi-bin/httpinfo.cgi?%s+config\"
>Configuration</a></li>\n", port);
	printf("<li><a		href=\"http://mesh.ic.uff.br/cgi-bin/httpinfo.cgi?%s+routes\"
>Routes</a></li>\n", port);
	printf("<li><a		href=\"http://mesh.ic.uff.br/cgi-bin/httpinfo.cgi?%s+nodes\"
>Links/Topology</a></li>\n", port);
	printf("<li><a		href=\"http://mesh.ic.uff.br/cgi-bin/httpinfo.cgi?%s+all\"
>All</a></li>\n", port);
	printf("<li><a		href=\"http://mesh.ic.uff.br/cgi-bin/httpinfo.cgi?%s+about\"
>About</a></li>\n</ul>\n</td></tr>\n", port);
	ch1 = strstr(ch2, "<tr><td>");
	if ((!(strcmp(tipo, "config"))) || (!(strcmp(tipo, "all"))))
	{
		for (; *ch1 != 'C'; ch1++) printf("%c", *ch1);
		printf("Click		<a		href=\"http://mesh.ic.uff.br/cgi-
bin/httpinfo.cgi?%s+cfgfile\">here</a> to <i>generate a configuration file for this node</i>.\n", port);
```

```
            ch1 = strstr(ch1, "\n");
            ch1++;
        }
        for (; *ch1 != '\0'; ch1++) printf("%c", *ch1);
}

main(int argc,char *argv[])
{
        struct  sockaddr_in     sock;
        int     sd;

        char    request[8];

        char    pagina[90000];

        char    buff[BUFSIZ];

        char    ipaddr[16];
        int     porta;

        int     i,l,nrv;

        if (argc < 3) strcpy(request, "all");
        else strcpy(request, argv[2]);
//Inicio do processo de abertura de conexão TCP
        porta = atoi(argv[1]);
        strcpy(ipaddr, "200.20.15.238");   //IP do gateway da Rede Mesh

        sock.sin_family = AF_INET;
        sock.sin_port = htons(porta);
        sock.sin_addr.s_addr = inet_addr(ipaddr);
        bzero(&(sock.sin_zero), 8);

        sd = socket(AF_INET,SOCK_STREAM,0);
        if(sd == 0)     exit(1);
        if(connect(sd,(struct sockaddr *) &(sock),sizeof(struct sockaddr)) == -1)
        {
            perror("Connection failed");
            exit(1);
        }
```

```
//Aqui montamos o pedido http a ser enviado
      sprintf(buff,"GET %s HTTP/1.0\r\nConnection: close\r\n\r\n",
                  request);
      l = strlen(buff);
//Enviamos o pedido
      write(sd,buff,l);

      pagina[0] = 0;
      l = 0;
      do
      //Aqui lemos a página e a armazenamos no nosso buffer
         {
           nrv = read(sd,buff,BUFSIZ);
           l += nrv;
           if(nrv > 0) strcat(pagina, buff);
           else break;
      } while(1);
      close(sd);
      printf("Content-Type:text/html;charset=iso-8859-1%c%c\n",13,10);
      pagina[l] = 0;
      limpa_pagina(pagina, argv[1], request);
}
```

A1.2.2: S45firewall

Arquivo de configuração do *firewall* dos roteadores. Este arquivo deve ser executado apenas no *gateway*.

```
#!/bin/sh

${FAILSAFE:+exit}

. /etc/functions.sh
WAN=$(nvram get wan_ifname)
LAN=$(nvram get lan_ifname)
WIFI=$(nvram get wifi_ifname)

## AS 4 LINHAS ABAIXO SÃO PARA A INICALIZAÇÃO: LIMPA AS TABELAS

for T in filter nat; do
```

```
  iptables -t $T -F
  iptables -t $T -X
done

## AS 6 LINHAS ABAIXO REALIZAM A TRADUÇÃO DO ENDEREÇO E PORTA DE ACESSO ##ÀS PÁGINAS DO PLUGIN HTTPINFO DOS ROTEADORES PARA O ENDEREÇO IP ##"REAL" DO GATEWAY. EX.:(O ENDEREÇO E PORTA 200.20.15.217:8081 IRÁ ##ACESSAR O ROTEADOR 10.151.1.1 NA PORTA 8080)
  iptables -t nat -A PREROUTING -p tcp --dport 8081 -j DNAT --to 10.151.1.1:8080
  iptables -t nat -A PREROUTING -p tcp --dport 8087 -j DNAT --to 10.151.7.1:8080
  iptables -t nat -A PREROUTING -p tcp --dport 8084 -j DNAT --to 10.151.4.1:8080
  iptables -t nat -A PREROUTING -p tcp --dport 8091 -j DNAT --to 10.151.11.1:8080
  iptables -t nat -A PREROUTING -p tcp --dport 8093 -j DNAT --to 10.151.13.1:8080
  iptables -t nat -A PREROUTING -p tcp --dport 8094 -j DNAT --to 10.151.14.1:8080

##A LINHA ABAIXO POSSIBILITA QUE O FIREWALL DO WIFIDOG NÃO BLOQUEIE AS ##REQUISIÇÕES PROVENIENTES DO SERVIDOR WEB PARA AS PÁGINAS ##FORNECIDAS PELO PLUGIN HTTPINFO

iptables -A FORWARD -i $WAN -o $WIFI -j ACCEPT

##A LINHA ABAIXO POSSIBILITA QUE O FIREWALL DO WIFIDOG NÃO BLOQUEIE AS ##RESPOSTAS PROVENIENTES DOS ROTEADORES AO SERVIDOR WEB PARA AS ##REQUISIÇÕES ÀS PÁGINAS DO PLUGIN HTTPINFO

iptables -A FORWARD -i $WIFI -p tcp -m multiport --source-ports 8080,8081,8084,8087, 8091,8093,8094 -o $WAN -d 200.20.15.217 -j ACCEPT

## A LINHA ABAIXO REALIZA A CONVERSÃO DOS ENDEREÇOS INTERNOS PARA OS ##EXTERNOS, POSSIBILITANDO A SAÍDA DO TRÁFEGO DA REDE MESH PARA A I##NTERNET

  iptables -t nat -A POSTROUTING -o $WAN -j MASQUERADE
```

A1.2.3: topologia.c

CGI que gera e exibe as figuras da topologia da rede, baseadas nas informações do *plugin dotdraw* do olsrd. Vale notar que nas linhas 12 e 13 deve-se substituir <diretório_script> pelo diretório que contém o script *olsr-topology-view.pl*. Para compilar, utilize o comando:

g++ topologia.c –o topologia

```
#include <iostream>
using namespace std;
int main()
{
  // Cabecalho do CGI, necessario para que as informacoes HTML sejam interpretadas pelo browser
  cout << "Content-type:text/html" << endl << endl;

  // Remove qualquer topologia previamente gerada
  system("rm /tmp/topologia*.jpg");

  // Executa o script para pegar os dados dos roteadores e gerar os graficos
  system("<diretório_script>/olsr-topology-view.pl -server 200.20.15.238 -name topologia-id6");
  system("<diretório_script>/ olsr-topology-view.pl -server 200.20.15.72 -name topologia-id7");

  // Codigo em HTML que o CGI ira gerar
  cout << "<html>" << endl;
  cout << "   <head>" << endl;
  cout << "     <title>* Projeto ReMesh * UFF - RNP</title>" << endl;
  cout << "   </head>" << endl;
  cout << "   <body>" << endl;
  cout << "     <br>" << endl;
  cout << "     <center>" << endl;
  cout << "       <strong> Rede externa</strong>" << endl;
  cout << "        <br>" << endl;
  cout << "        <br>" << endl;
  cout << "        <img src=\"/tmp/topologia-id6.png\">" << endl;
  cout << "        <br>" << endl;
  cout << "        <br>" << endl;
  cout << "       <strong> Rede WRNP</strong>" << endl;
  cout << "        <br>" << endl;
  cout << "        <br>" << endl;
  cout << "        <img src=\"/tmp/topologia-id7.png\">" << endl;
  cout << "        <br>" << endl;
  cout << "        <br>" << endl;
  cout << "     </center>" << endl;
```

```
  cout << "      <script language=\"JavaScript\">" << endl;
  cout << "        window.setTimeout('window.location.reload()', 8000)" << endl;
  cout << "      </script>" << endl;
  cout << "   </body>" << endl;
  cout << "</html>" << endl;

  // Remove as informacoes geradas pelo plugin olsrd_dot_draw
     system("rm /tmp/topologia*.dot");

}
```

A1.2.4: olsr-topology-view.pl

Script Perl que obtém que acessa o *plugin dotdraw* do *gateway* da Rede Mesh e, a partir das informações obtidas, chama o programa *dot* (do pacote *graphviz*) para gerar as imagens da topologia.

```
#!/usr/bin/perl

# a hack to display OLSR topology information
#
# copyleft 2004 Bruno Randolf <br1@subnet.at>
# licensed under the GPL

# Carrega o modulo socket para conexao remota com os roteadores
use IO::Socket;
# Para o menu de opcoes
use Getopt::Long;

# Local aonde serao armazenadas as figuras e arquivos do mapa
$TOPPATH = "/tmp";

# Prefixo dos nomes das figuras e arquivo de mapa
$NAME = "topologia";


# Define o formato da figura a ser utilizado
$EXT = "png";

# Endereco do roteador que contem o plugin olsrd_dot_draw
$SERVER = "localhost";
```

```
# Porta do servidor a conectar
$PORT = "2004";


$HELP = 0;
$KEEP = 0;
$BGCOLOR = "grey";

# Programa que ira gerar o grafico, neste caso o dot
$SIZE = "10,10";
$ROOTWIN = 0;
$ONCE = 1;
$GRAPH = 1;
$FONTNAME = "Arial";
$FONTSIZE = 12;
$LINEWIDTH = 1;
$LINECOLOR = 1;
$RESOLV = 1;


# Aloca os parametros passados no script em suas correspondentes variaveis
GetOptions ("name=s" => \$NAME,
        "server=s" => \$SERVER,
        "port=s" => \$PORT,
        "help!" => \$HELP,
        "keep!" => \$KEEP,
        "bgcolor=s" => \$BGCOLOR,
        "fontname=s" => \$FONTNAME,
        "fontsize=s" => \$FONTSIZE,
        "size=s" => \$SIZE,
        "once!" => \$ONCE,
        "graph!" => \$GRAPH,
        "linewidth!" => \$LINEWIDTH,
        "linecolor!" => \$LINECOLOR,
        "resolv" => \$RESOLV,
         );

# Define o caminho completo do arquivo do mapa
$FILENAME = "$TOPPATH/$NAME.dot";
```

```
# Se foi solicidada ajuda
if ($HELP) {
	print << "EOF";
uso: $0 [ --server SERVER ] [--port PORT] [--fullscreen] [--keep]

um utilitario para mostrar informacoes sobre a topologia do OLSR

opcoes:
	--server SERVER    conectar no noh OLSR (padrao: localhost)
	--port PORT        conectar na porta (padrao: 2004)
	--bgcolor          cor de fundo (padrao: grey)
	--fontname         nome da fonte (padrao: Courier)
	--fontsize         tamanho da fonte (padrao: 12)
	--size X,Y         tamanho da imagem, em polegadas, para o graphviz (padrao: 10,10)
	--[no]graph        gerar graficos (padrao: on)
	--[no]once         rodar somente uma vez e sair (padrao: rodar uma vez)
	--[no]linewith      mudar o comprimento da linha de acordo com a metrica (padrao:
off)
	--[no]linecolor    mudar a cord da linha de acordo com a metrica (padrao: off)
	--[no]resolv       resolver nomes (padrao: on)
	--[no]keep         manter os arquivos .dot com a data em /tmp (padrao: off)

Requer o pacote "graphviz" instalado
e o plugin "olsrd_dot_draw" configurado no noh olsr

EOF
	exit;
}

# Cria o arquivo que contera a figura
`touch $TOPPATH/$NAME.$EXT`;

# Faz a conexao com o roteador
$remote = IO::Socket::INET->new(
		Proto    => "tcp",
		PeerAddr => $SERVER,
		PeerPort => $PORT,
	)
```

```
            or die "nao foi possivel conectar na porta $PORT no noh $SERVER!\no plugin olsrd_dot_draw esta carregado e configurado para permitir conexoes desse host?\n";

$f;


# Define a linha de comando completa que gerara o grafico
      $DOT_CMD = "dot -Tpng -Timap -Gsize=${SIZE} -Elen=2 -Ncolor=gray -Nstyle=filled -Nfillcolor=white -Nfontname=${FONTNAME} -Nfontsize=${FONTSIZE} -Efontname=${FONTNAME} -Efontsize=${FONTSIZE} -Nfontcolor=red -Nwidth=1 -Gbgcolor=$BGCOLOR $FILENAME -o $TOPPATH/$NAME.new";

# Recupera as informacoes do roteador
while ( <$remote> )
{
      $line = $_;
      if ($RESOLV) {
            $line = resolv( $line );
      }
      if ($LINEWIDTH || $LINECOLOR) {
            $line = draw_thicker_metric_lines( $line );
      }
      $f = $f . $line;


      # fim de um grafico
      if ( $line =~ /}/i ) {
            #imprime "* ";
            open DOTFILE, ">$FILENAME";
            print DOTFILE $f;
            close DOTFILE;
            $f = "";

            if ($GRAPH) {
                  `$DOT_CMD`;
                  `mv $TOPPATH/$NAME.new $TOPPATH/$NAME.$EXT`;
            }
            if ($KEEP) {
                  `cp $TOPPATH/$NAME.dot $TOPPATH/$NAME-\$(date +'%Y-%m-%d-%H-%M-%S').dot`;
```

```
            }
            exit if $ONCE;
        }
}

print "connection closed\n";

# Procedimento para resolver os nomes dos IPs
sub resolv {
        my $l = shift;
        if ( $l =~ /\"(.*)\" -> "([^[]*)"(.*)/) {
                my $from = $1;
                my $to = $2;
                my $rest = $3;
                $from =~ s/\"//g;
                $to =~ s/\"//g;
                my $iaddrf = inet_aton($from);
                my $fromn  = gethostbyaddr($iaddrf, AF_INET);

                my $iaddrt = inet_aton($to);
                my $ton  = gethostbyaddr($iaddrt, AF_INET);
                $fromn = $from if ! $fromn;
                $ton = $to if ! $ton;
                $l = "\"$fromn\" -> \"$ton\"$rest\n";
        }
        return $l;
}

# Procedimento para desenhar mais grossas as linhas com melhores metricas e colorir as linhas
sub draw_thicker_metric_lines {
        my $l = shift;
        if ($l =~ /.*label="[0-9].*/){     # reconhece o padrao
                @n=split (/"/,$l);         # divide o string para pegar o valor da metrica
                if ($LINECOLOR) {
                        if ( $n[5]>2 ) {       # colore de cinza as metricas maiores que 2
                                $c="888888";
                        }
                        else {                 # colore de preto as metricas menores que 2
                                $c="000000";
```

```
            }
            $setcol = "color=\"#$c\",";
        }
        if ($LINEWIDTH) {
            if ($n[5]>0 && $n[5]<10) {  # linhas mais grossas se 10 > metrica > 0
                $lt=2/$n[5];
                $at=$lt/2;
            }
            else {                  # senao desenha uma fina linha
                $lt=1;
                $at=$lt;
            }
            $setwidth = "style=\"setlinewidth($lt)\", arrowsize=$at";
        }
        $l =~ s/\]/, $setcol $setwidth]/g; # substitui o padrao
    }
    return $l;
}

__END__
```

A1.3: Arquivos de configuração

A1.3.1: olsrd.conf

Arquivo de configuração do olsrd.

```
#
# olsr.org OLSR daemon config file
#
# Lines starting with a # are discarded
#
# This file was shipped with olsrd 0.4.9
#

# Debug level(0-9)
# If set to 0 the daemon runs in the background

DebugLevel      0

# IP version to use (4 or 6)
```

```
IpVersion       4

# Clear the screen each time the internal state changes

ClearScreen     no

# HNA IPv4 routes
# syntax: netaddr netmask
# Example Internet gateway:
# 0.0.0.0 0.0.0.0

Hna4
{
#   Internet gateway:
   0.0.0.0      0.0.0.0
#   more entries can be added:
#   192.168.1.0  255.255.255.0
10.152.0.192 255.255.255.224
10.151.7.0 255.255.255.224
}

# HNA IPv6 routes
# syntax: netaddr prefix
# Example Internet gateway:
Hna6
{
#   Internet gateway:
#   ::              0
#   more entries can be added:
#   fec0:2200:106:: 48
}


# Should olsrd keep on running even if there are
# no interfaces available? This is a good idea
# for a PCMCIA/USB hotswap environment.
# "yes" OR "no"

AllowNoInt      yes
```

```
# TOS(type of service) value for
# the IP header of control traffic.
# If not set it will default to 16

#TosValue       16

# The fixed willingness to use(0-7)
# If not set willingness will be calculated
# dynamically based on battery/power status
# if such information is available

Willingness     4

# Allow processes like the GUI front-end
# to connect to the daemon.

IpcConnect
{
    # Determines how many simultaneously
    # IPC connections that will be allowed
    # Setting this to 0 disables IPC

    MaxConnections  0

    # By default only 127.0.0.1 is allowed
    # to connect. Here allowed hosts can
    # be added

    Host            127.0.0.1
    #Host            10.0.0.5

    # You can also specify entire net-ranges
    # that are allowed to connect. Multiple
    # entries are allowed

    #Net            192.168.1.0 255.255.255.0
}

# Wether to use hysteresis or not
```

```
# Hysteresis adds more robustness to the
# link sensing but delays neighbor registration.
# Used by default. 'yes' or 'no'

UseHysteresis   no

# Hysteresis parameters
# Do not alter these unless you know
# what you are doing!
# Set to auto by default. Allowed
# values are floating point values
# in the interval 0,1
# THR_LOW must always be lower than
# THR_HIGH.

HystScaling     0.50
HystThrHigh     0.80
HystThrLow      0.30


# Link quality level
# 0 = do not use link quality
# 1 = use link quality for MPR selection
# 2 = use link quality for MPR selection and routing
# 3 = use probabilities
# Defaults to 0

LinkQualityLevel        3

# Link quality window size
# Defaults to 10

LinkQualityWinSize      100

# Polling rate in seconds(float).
# Default value 0.05 sec

Pollrate   0.05
#0.05
```

```
# TC redundancy
# Specifies how much neighbor info should
# be sent in TC messages
# Possible values are:
# 0 - only send MPR selectors
# 1 - send MPR selectors and MPRs
# 2 - send all neighbors
#
# defaults to 0

TcRedundancy    2

#
# MPR coverage
# Specifies how many MPRs a node should
# try select to reach every 2 hop neighbor
#
# Can be set to any integer >0
#
# defaults to 1

MprCoverage     3

# Olsrd plugins to load
# This must be the absolute path to the file
# or the loader will use the following scheme:
# - Try the paths in the LD_LIBRARY_PATH
#   environment variable.
# - The list of libraries cached in /etc/ld.so.cache
# - /lib, followed by /usr/lib

# Example plugin entry with parameters:

#LoadPlugin "olsrd_dyn_gw.so.0.3"
#{
    # Here parameters are set to be sent to the
    # plugin. Theese are on the form "key" "value".
    # Parameters ofcause, differs from plugin to plugin.
    # Consult the documentation of your plugin for details.
```

```
    # Example: dyn_gw params

    # how often to check for Internet connectivity
    # defaults to 5 secs
#   PlParam     "Interval"   "40"

    # if one or more IPv4 addresses are given, do a ping on these in
    # descending order to validate that there is not only an entry in
    # routing table, but also a real internet connection. If any of
    # these addresses could be pinged successfully, the test was
    # succesful, i.e. if the ping on the 1st address was successful,the
    # 2nd won't be pinged
#   PlParam     "Ping"       "141.1.1.1"
#   PlParam     "Ping"       "194.25.2.129"
#}

LoadPlugin "olsrd_httpinfo.so.0.1"
{
PlParam        "Port"  "8087"
PlParam        "Net"   "0.0.0.0 0.0.0.0"
}

LoadPlugin "olsrd_dot_draw.so.0.3"
{
PlParam        "accept"       "200.20.15.217"
}

# Interfaces and their rules
# Omitted options will be set to the
# default values. Multiple interfaces
# can be specified in the same block
# and multiple blocks can be set.

# !!CHANGE THE INTERFACE LABEL(s) TO MATCH YOUR INTERFACE(s)!!
# (eg. wlan0 or eth1):

Interface "eth1"
{

    # IPv4 broadcast address to use. The
```

```
    # one usefull example would be 255.255.255.255
    # If not defined the broadcastaddress
    # every card is configured with is used

    # Ip4Broadcast            255.255.255.255

    # IPv6 address scope to use.
    # Must be 'site-local' or 'global'

    # Ip6AddrType             site-local

    # IPv6 multicast address to use when
    # using site-local addresses.
    # If not defined, ff05::15 is used

    # Ip6MulticastSite        ff05::11

    # IPv6 multicast address to use when
    # using global addresses
    # If not defined, ff0e::1 is used

    # Ip6MulticastGlobal      ff0e::1

    # Emission intervals.
    # If not defined, RFC proposed values will
    # be used in most cases.

    # Hello interval in seconds(float)
    HelloInterval    2.0
#   HelloInterval    2.0

    # HELLO validity time
     HelloValidityTime  60.0
#    HelloValidityTime 6.0

    # TC interval in seconds(float)
    TcInterval       5.0
#   TcInterval       5.0

    # TC validity time
```

TcValidityTime 90.0
 # TcValidityTime 10.0

 # MID interval in seconds(float)
MidInterval 5.0
 # MidInterval 5.0

 # MID validity time
MidValidityTime 90.0
 # MidValidityTime 10.0

 # HNA interval in seconds(float)
HnaInterval 5.0
 # HnaInterval 5.0

 # HNA validity time
HnaValidityTime 90.0
 # HnaValidityTime 10.0

 # When multiple links exist between hosts
 # the weight of interface is used to determine
 # the link to use. Normally the weight is
 # automatically calculated by olsrd based
 # on the characteristics of the interface,
 # but here you can specify a fixed value.
 # Olsrd will choose links with the lowest value.

Weight 0

}

A1.3.2: wifidog.conf

Arquivo de configuração do wifidog (nos roteadores).

 # WiFiDog Configuration file
 # Parameter: GatewayID
 # Default: default
 # Optional but essential for monitoring purposes
 # Set this to the template ID on the auth server
 # this is used to give a customized login page to the clients

 # If none is supplied, the default login page will be used.

GatewayID 7

 # Parameter: ExternalInterface

 # Default: NONE

 # Optional

 # Set this to the external interface. Typically vlan1 for OpenWrt, and eth0 or ppp0 otherwise

ExternalInterface vlan1

 # Parameter: GatewayInterface

 # Default: NONE

 # Mandatory

 # Set this to the internal interface. Typically br0 for OpenWrt, and eth1 otherwise

GatewayInterface eth1

 # Parameter: GatewayAddress

 # Default: Find it from GatewayInterface

 # Optional

 # Set this to the internal IP address of the gateway

GatewayAddress 10.151.7.1

 # Parameter: AuthServMaxTries

 # Default: 1

 # Optional

 # Sets the number of auth servers the gateway will attempt to contact when a request fails.

 # this number should be equal to the number of AuthServer lines in this

 # configuration but it should probably not exceed 3.

 # AuthServMaxTries 3

 # Parameter: AuthServer

 # Default: NONE

 # Mandatory

 # Set this to the hostname or IP of your auth server, the path where

 # WiFiDog-auth resides and optionally as a second argument, the port it

 # listens on.

#AuthServer {

 # Hostname (Mandatory; Default: NONE)

 # SSLAvailable (Optional; Default: no; Possible values: yes, no)

 # SSLPort 443 (Optional; Default: 443)

 # HTTPPort 80 (Optional; Default: 80)

 # Path wifidog/ (Optional; Default: /wifidog/ Note: The path must be both prefixed and suffixed by /. Use a single / for server root.)

#}

```
AuthServer {
Hostname 200.20.15.73
SSLAvailable no
Path /
}
#AuthServer {
#    Hostname auth2.ilesansfil.org
#    SSLAvailable yes
#    Path /
#}
#AuthServer {
#    Hostname auth3.ilesansfil.org
#    SSLAvailable yes
#    Path /
#}
# Parameter: Daemon
# Default: 1
# Optional
#
# Set this to true if you want to run as a daemon
Daemon 1
# Parameter: GatewayPort
# Default: 2060
# Optional
# Listen on this port
# GatewayPort 2060
# Parameter: HTTPDName
# Default: WiFiDog
# Optional
# Define what name the HTTPD server will respond
# HTTPDName WiFiDog
# Parameter: HTTPDMaxConn
# Default: 10
# Optional
# How many sockets to listen to
# HTTPDMaxConn 10
# Parameter: CheckInterval
# Default: 60
# Optional
# How many seconds should we wait between timeout checks
```

```
CheckInterval 60
# Parameter: ClientTimeout
# Default: 5
# Optional
#
# Set this to the desired of number of CheckInterval of inactivity before a client is logged
out
# The timeout will be INTERVAL * TIMEOUT
ClientTimeout 5

# Parameter: FirewallRuleSet
# Default: none
# Mandatory
#
# Groups a number of FirewallRule statements together.

# Parameter: FirewallRule
# Default: none
#
# Define one firewall rule in a rule set.

# Rule Set: global
#
# Used for rules to be applied to all other rulesets except locked.
# This is the default config for the Teliphone service.
FirewallRuleSet global {
FirewallRule allow udp to 69.90.89.192/27
FirewallRule allow udp to 69.90.85.0/27
FirewallRule allow tcp port 80 to 69.90.89.205
#douglas inseriu abaixo
#FirewallRule allow tcp to 10.151.0.1
}
# Rule Set: validating-users
# Used for new users validating their account
FirewallRuleSet validating-users {
   FirewallRule block tcp port 25
   FirewallRule allow to 0.0.0.0/0
}
# Rule Set: known-users
#
```

```
# Used for normal validated users.
FirewallRuleSet known-users {
   FirewallRule allow to 0.0.0.0/0
}
# Rule Set: unknown-users
#
# Used for unvalidated users, this is the ruleset that gets redirected.
#
# XXX The redirect code adds the Default DROP clause.
FirewallRuleSet unknown-users {
FirewallRule allow udp port 53
FirewallRule allow tcp port 53
FirewallRule allow udp port 67
FirewallRule allow tcp port 67
#FirewallRule allow tcp to 0.0.0.0/0
}
# Rule Set: locked-users
#
# Used for users that have been locked out.
FirewallRuleSet locked-users {
   FirewallRule block to 0.0.0.0/0
}
```

A1.3.3: S65wifidog

Localizado em /etc/init.d e é responsável pelo início do serviço de autentiação. Ele executa o wifidog-init localizado em /usr/sbin e em seguida executa o script wifidog_extra localizado /etc

```
#!/bin/ash

/usr/bin/wifidog-init start

sleep 5

/etc/wifidog_extra
```

A1.3.4: wifidog_extra

Responsável pelo desbloqueio de todos os clientes com fio que já foram autenticados localmente nos seus roteadores de acesso e que foram roteados por múltiplos saltos até o gateway. Não utilizando o wifidog_extra todos os clientes da rede mesh conectados por fio seriam obrigados a se autenticarem duas vezes.

```
#!/bin/sh
```

iptables -A WiFiDog_Global -s 10.152.0.0/16 -d 0.0.0.0/0 -j ACCEPT
iptables -t nat -A WiFiDog_Global -s 10.152.0.0/16 -d 0.0.0.0/0 -j ACCEPT
iptables -t filter -A WiFiDog_Global -s 10.152.0.0/16 -d 0.0.0.0/0 -j ACCEPT

#iptables -t filter -A WiFiDog_Known -i eth1 -o vlan1 -j ACCEPT
#iptables -t filter -A WiFiDog_Known -i vlan1 -o eth1 -j ACCEPT

#iptables -t nat -A WiFiDog_Outgoing -i eth1 -o vlan1 -j ACCEPT
#iptables -t nat -A WiFiDog_Outgoing -i vlan1 -o eth1 -j ACCEPT

A1.4: Scripts

A1.4.1: S71iperf

Script de inicialização do lado servidor da ferramenta de medição *iperf*.

#!/bin/ash

iperf -s -D

Apêndice 2 - Documentação do Protótipo: Manual de Instalação

A2.1: Instalação do Roteador Mesh

A2.1.1: Pré-requisitos

Algumas informações são necessárias durante a instalação de um roteador:

1. ID do roteador: cada roteador deverá ter o seu id, um número de 0 a 253.
2. IP do servidor de DNS.
3. IP da máquina usada como servidor de autenticação.

A rede é composta de pontos de acesso e de um roteador específico que funcionará como *gateway.* No caso da instalação do ponto que será usado como *gateway* da Rede Mesh, ainda existe um quarto requisito: saber o gateway padrão da rede a qual será conectada a porta *wan* do roteador.

A2.1.2: Atualização da Imagem

O primeiro passo na configuração dos roteadores *linksys wrt54g* é a mudança do *firmware.* O original deve ser substituído pela imagem que se encontra no cd anexado ao relatório.

Para isso, é possível utilizar a interface web do roteador, através do endereço **http://192.168.1.1** (assumindo as configurações originais de fábrica). Nesta interface, existe a opção de atualização de *firmware*. Na página de atualização, deve-se indicar o arquivo "openwrt-wrt54g-jffs2.bin", localizado no CD, que contém a nova imagem. **É fundamental que o roteador e o pc estejam ligados a um no-break, pois este processo não deve ser interrompido em hipótese alguma.**

Se a atualização for concluída com sucesso, o roteador será reiniciado automaticamente. É preciso esperar esta inicialização ser encerrada (o *led* "pwr" irá parar de piscar) e, em seguida, deve-se reiniciar novamente o dispositivo.

Neste momento, o roteador já pode ser acessado pelo comando:

> telnet 192.168.1.1

Uma tela de boas-vindas do OpenWrt será exibida. Neste ponto, o próximo passo é alterar a senha de root (único usuário do sistema):

> passwd

Com a nova senha cadastrada, deve-se reiniciar novamente:

> reboot

Para utilizar o roteador simplesmente como um ponto de acesso à Rede Mesh, prossiga a configuração como descrito no item A2.1.2. Para utilizá-lo também como gateway da Rede Mesh para a internet, prossiga a configuração conforme o item A2.1.3.

A2.1.2: Configuração do Roteador como Ponto de Acesso à Rede Mesh

Algumas configurações devem ser feitas antes que o ponto de acesso possa ser utilizado. O primeiro passo é copiar seguintes arquivos do diretório "config_router", localizado no CD, para o roteador. Este diretório contém aplicativos (executáveis e bibliotecas) e scripts de configuração. Em uma máquina linux, estando no diretório "config_router" do CD, execute o script "config_mesh_router.sh":

> ./config_mesh_router.sh

Este script copiará todos os arquivos necessários. Um detalhe a ser destacado é o fato de que a senha do roteador será pedida 7 vezes.

Deve-se, em seguida, acessar o roteador de forma segura, através do comando:

> ssh 192.168.1.1

A tela de boas-vindas do OpenWrt será exibida. Neste momento, o script de configuração dos endereços das interfaces e das sub-redes, “Configure_manet”, copiado anteriormente deverá ser executado:

> **cd /www**

> **./Configure_manet #*id***

No comando acima, *#id* é o id do roteador, que deve ser um valor entre 0 e 253. Após configurar os endereços de rede, deve-se informar o endereço IP do servidor de DNS:

> nvram set wan_dns=#ip_servidor_dns

> nvram commit

> **reboot**

No primeiro comando, *#ip_servidor_*dns deve ser substituído pelo ip do servidor de dns. A partir de agora, o roteador responderá por um novo endereço, fazendo com que seja necessário obter um novo endereço de ip na máquina local, na mesma sub-rede do endereço de IP do roteador. Isso pode ser feito através de DHCP. No linux, o comando dhcpcd pode ser usado.

Com isso, o roteador pode ser novamente acessado, através de ssh:

> **ssh 10.151.x.1**

Onde x = id + 1.

Antes de executar o próximo script é preciso fazer uma pequena alteração no arquivo “/etc/olsrd.conf”. A alteração consiste em adicionar a sub-rede da interface *lan* do roteador. Para saber o endereço da sub-rede, basta saber o ip da interface:

> **ifconfig vlan0**

O endereço da sub-rede será o endereço anterior ao da interface. Por exemplo, se o endereço da interface é 10.152.0.193, o endereço da sub-rede será 10.152.0.192.

De posse deste endereço, é possível editar o arquivo:

> **vi /etc/olsrd.conf**

A linha 32 tem um ip seguido de uma máscara de sub-rede:

10.152.0.192 255.255.255.224

É preciso mudar o ip, para o da sub-rede. A máscara já está correta. Após salvar o arquivo e sair do editor, é possível executar o segundo script para possibilitar DHCP através da interface wireless:

> **cd /www**

> **./wireless_dhcp.sh #*ip***

Aqui, #ip denota o terceiro byte do endereço ip da interface wireless. Ou seja, id + 1.

É necessário também modificar o arquivo “wifidog.conf”, para configurar a autenticação:

> **vi /etc/wifidog.conf**

Na linha 12, é necessário alterar o *GatewayID* para id + 1. Na linha 36, deve-se modificar a opção *GatewayAddress* para o endereço para o ip da interface vlan0. Por último, na linha 64, na seção *AuthServer*, o parâmetro *Hostname* deve ser configurado com o endereço ip da máquina usada como servidor de autenticação.

A2.1.3: Configurações do Roteador como *Gateway* para Internet

De modo geral, o *gateway* é configurado como um ponto de acesso. Porém, existem algumas diferenças. Passo a passo, o processo (bastante semelhante ao anterior) começaria também pela cópia dos mesmos arquivos necessários aos pontos de acesso. Isso pode ser feito através do script “config_mesh_router_gw.sh”:

> **./config_mesh_router_gw.sh**

É importante notar que é preciso estar no diretório “config_router” do CD. Vale destacar ainda que será necessário digitar a senha do roteador 7 vezes.

Neste momento, podemos acessar novamente o roteador:

> ssh 192.168.1.1

Novamente, a tela de boas-vindas do OpenWrt será exibida e o script de configuração dos endereços das interfaces e das sub-redes, “Configure_manet.sh”, deverá ser executado:

> **cd /www**

> **./Configure_manet #*id***

No comando acima, *#id* é o id do roteador, que deve ser um valor entre 0 e 253. Após configurar os endereços de rede, deve-se informar o endereço ip do servidor de DNS e o endereço de ip do gateway padrão da rede onde a interface *wan* do roteador será conectada:

> nvram set wan_dns=#ip_servidor_dns

> nvram set wan_gateway=#ip_gateway_padrao

> nvram set wan_ipaddr=#ip_wan

> nvram commit

> **reboot**

No primeiro comando, *#ip_servidor_*dns deve ser substituído pelo ip do servidor de dns, assim como *#ip_gateway_padrao* deve ser substituído pelo ip do gateway padrão da *wan*. No terceiro comando, #ip_wan é o ip da interface *wan*. A partir de agora, o roteador responderá por um novo endereço, fazendo com que seja necessário pegar um novo endereço de ip na máquina local, na mesma sub-rede da ip do roteador. Isso pode ser feito através de DHCP. No linux, o comando dhcpcd pode ser usado.

Com isso, o roteador pode ser novamente acessado, através de ssh:

> **ssh 10.151.x.1**

Onde x = id + 1.

Antes de executar o próximo script é preciso fazer uma pequena alteração no arquivo "/etc/olsrd.conf". A alteração consiste em adicionar a sub-rede da interface *lan* do roteador. Para saber o endereço da sub-rede, basta saber o ip da interface:

> **ifconfig vlan0**

O endereço da sub-rede será o endereço anterior ao da interface. Por exemplo, se o endereço da interface é 10.152.0.193, o endereço da sub-rede será 10.152.0.192.

De posse deste endereço, é possível editar o arquivo:

> **vi /etc/olsrd.conf**

A linha 32 tem um ip seguido de uma máscara de sub-rede:

10.152.0.192 255.255.255.224

É preciso mudar o ip, para o da sub-rede. A máscara já está correta. Além disso, nesta mesma seção, é necessário adicionar a seguinte entrada na linha seguinte (linha 33):

0.0.0.0 0.0.0.0

Isso fará com que o endereço ip do gateway padrão seja exportado para os demais nós da rede, ou seja, exporta a informação de que este é será o roteador responsável pelo escoamento do tráfego para a Internet.

Mais abaixo, existe uma seção chamada *LoadPlugin "olsrd_httpinfo.so.0.1".* Deve-se retirar os comentários de todas as linhas da seção, bem como da próxima (*LoadPlugin "olsrd_dot_draw.so.0.3")*.

Após salvar o arquivo e sair do editor, é possível executar o segundo script para possibilitar DHCP pela interface wireless:

> **cd /www**

> **./wireless_dhcp.sh #*ip***

Aqui, #ip denota o terceiro byte do endereço ip da interface wireless. Ou seja, id + 1.

É também necessário modificar o arquivo "wifidog.conf" para configurar a autenticação:

> **vi /etc/wifidog.conf**

Na linha 12, é necessário alterar o *GatewayID* para id + 1. Na linha 20, *ExternalInterface* deve ser vlan1, assim como na linha 28 *GatewayInterface* será eth1. Na linha 36, deve-se modificar a opção *GatewayAddress* para o endereço para o ip da interface eth1 (10.151.x.1, onde x = id + 1). Por último, na linha 64, na seção *AuthServer*, o parâmetro *Hostname* deve ser configurado com o endereço ip da máquina usada como servidor de autenticação.

A2.1.4: Mudança de Canal

Caso seja necessário mudar o canal dos roteadores, pode-se executar o seguinte comando:

> **nvram set wl0_channel=#canal**

> **nvram commit**

No primeiro comando, #canal denota o novo canal de comunicação.

Observação: esse comando deve ser executado em todos os roteadores e simultaneamente. Isso significa que eles deverão ser reinicidos ao mesmo tempo. Se isso não for feito, ao ser reiniciado, um roteador poderá identificar outros pontos na sua rede no canal antigo e, com isso, a mudança não surtirá efeito.

A2.2: Instalação do Servidor de Autenticação com módulo de gerencia de usuários

O servidor de preencher todos os requisitos descritos no item 1 (php versão 5 ou maior, servidor Apache versão 2.0 ou maior, banco de dados Postgres e todos os módulos php listados).

Com o servidor pronto, bastará apenas seguir os passos:

1) Copiar do CD o diretório “wifidog-auth-1.0.0_m2” com as permissões originais para a máquina que será o servidor de autenticação. Embaixo deste diretório existe um outro que se chama “wifidog”. Este é o diretório que contém as páginas php a serem servidas.

2) Especifique no “httpd.conf” o diretório onde se encontra a pasta “wifidog” mencionada acima, como a diretiva DocumentRoot do apache.

3) Inicie o apache normalmente e verifique se o diretório “wifidog” possue permissão de execução para todos (chmod a+x <wifidog/>).

4) Uma vez iniciado, acesse a página “install.php” no servidor de autenticação (<endereço>/install.php). Nesta página, estarão listados todos os passos para a verificação da instalação dos módulos necessários para o servidor.

5) Uma vez instalado o servidor e a senha de administrador do banco de dados definada, os nós deverão ser adicionados. Para tanto, basta acessar o servidor de autenticação normalmente e se *logar* como administrador. Assim que se *logar* clique no link “administração” e, na próxima página, haverá na aba esquerda da tela o botão “Create”. Para cada nó, o identificador utilizado é o terceiro byte do endereço ip. Por exemplo, para adicionar o nó 10.151.1.1, crie o “node” 1.

A2.3: Instalação do servidor web com os módulos de gerencia dos roteadores

O servidor web deverá ter instalados os pacotes GraphViz (http://www.graphviz.org) e ImageMagick (http://www.imagemagick.org), além de um interpretador Perl. Estes pacotes são necessários para a geração das figuras da topologia.

É necessário compilar os CGI’s apresentados no apêndice 1 e colocar os executáveis no diretório cgi-bin do servidor. Além disso, deve-se verificar se esses arquivos têm permissão de execução para todos os usuários.

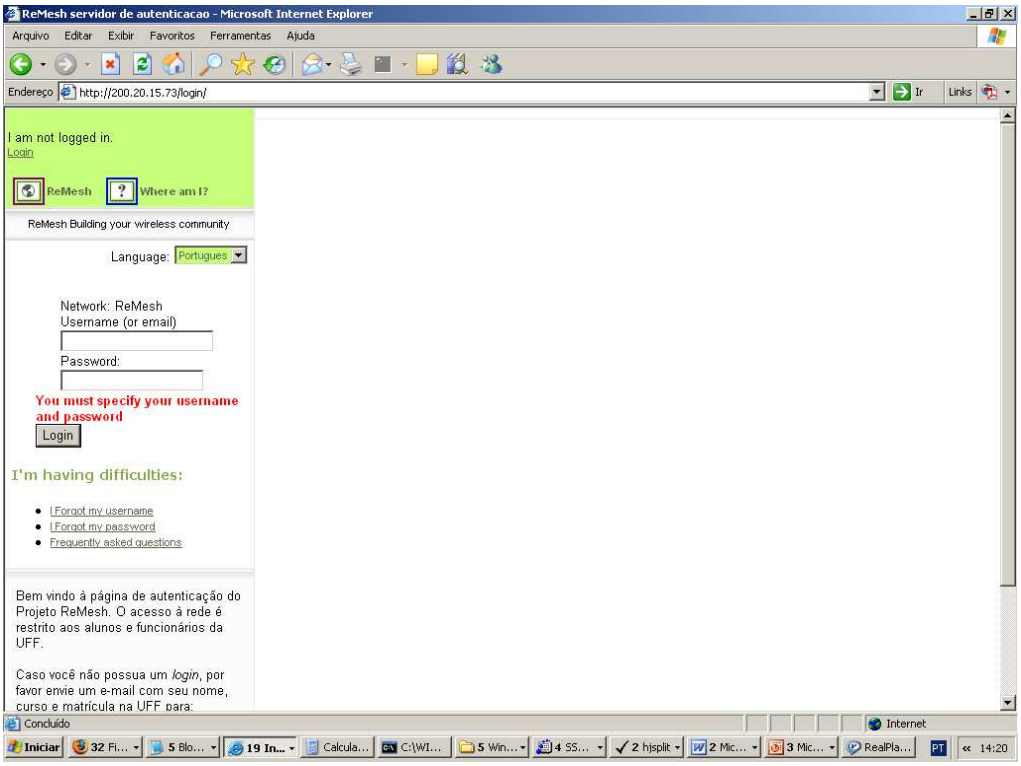
É importante também copiar o script *olsr-topology-view.pl* para o diretório definido em tempo de compilação no CGI *topologia.c*, como explicado no apêndice 1. Em seguida, deve-se criar um *alias* para o diretório “/tmp/”, pois é o local onde serão geradas a figura deste CGI e para exibi-las, o servidor web deverá busca-las neste diretório. Após estes passos, basta colocar links para os CGI’s.

Vale ressaltar que o CGI httpinfo precisa de pelo menos um parâmetro de entrada: a porta a ser acessada no *gateway* da Rede Mesh. Como explicado anteriormente, a porta define qual roteador está sendo acessado. Esta porta depende das configurações feitas no arquivo *S45firewall* (vide apêndice 1).

Apêndice 3 - Documentação do Protótipo: Manual do Usuário

A3.1: Para conexões com fio:

O usuário deve obter as configurações de rede por DHCP e executar qualquer navegador web. No momento em que se tentar acessar alguma página, o navegador será automaticamente redirecionado para a página de autenticação, como mostrado abaixo:



O usuário deve digitar seu *login* e senha, o selecionar o botão *login*. Caso a autenticação tenha sido efetuada com sucesso, o navegador abrirá uma janela com a opção de logout e a janela atual será redirecionada para o site originalmente desejado.

A3.2: Para conexões sem fio:

O usuário deve se associar a Rede Mesh sem fio e prosseguir com os mesmos passos do item **A3.1.**