TRABALHO DE GRADUAÇÃO

AVALIAÇÃO DE QUALIDADE DE SERVIÇO DE VIDEOCONFERÊNCIA NA REDUNB COM O PARADIGMA DE REDES DEFINIDAS POR SOFTWARE

Igor Müller de Moraes

Brasília, julho de 2015

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

TRABALHO DE GRADUAÇÃO

AVALIAÇÃO DE QUALIDADE DE SERVIÇO DE VIDEOCONFERÊNCIA NA REDUNB COM O PARADIGMA DE REDES DEFINIDAS POR SOFTWARE

Igor Müller de Moraes

Relatório submetido como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação.

Banca Examinadora

Prof. Dr. André Costa Drummond, CIC/UnB _____
Orientador

Prof. Dr. Marcelo Menezes de Carvalho, ENE/UnB _____
Examinador

Prof. Dr. Rafael Timóteo de Sousa Jr. ENE/UnB _____
Examinador

Dedico este trabalho a minha família,
professores e amigos.

Agradecimentos

À minha família, em especial, à minha irmã por estar sempre ao meu lado e a meus pais, por todo apoio e educação que me deram ao longo da vida.

Ao meu orientador, professor André Drummond, pelo incentivo, apoio, inspiração e por orientar os caminhos deste trabalho.

A meus colegas de vida e universidade, em especial a meus grandes amigos, Ighor O., Rodrigo C., Camila L., Carlos G., Eduardo L., Pedro Vitor C., Yuri C., Marta K., Tatiane F.,Ricardo M., Cosmin N., Raiane B., Jorge M., por todo companheirismo e suporte.

A todos aqueles que, de alguma forma, colaboraram para que este trabalho pudesse existir.

RESUMO

A atual arquitetura das redes de computadores traz um desafio para os pesquisadores da área, pela dificuldade na realização de experimentos de novas tecnologias em redes reais. A fim de contornar tal questão, novas tecnologias e novos protocolos são testados em simuladores com cenários que simplificam a rede real.

O paradigma de Redes Definidas por Software (*Software-Defined Network* – SDN) e a arquitetura OpenFlow oferecem um caminho para este problema e constituem uma promissora resposta aos desafios enfrentados pelos pesquisadores da área e por provedores de serviço de redes flexíveis, escalonáveis, e gerenciáveis dinamicamente, possibilitando os testes em redes reais.

Outro desafio enfrentado atualmente é relacionado à Qualidade de Serviço (*Quality of Service* – QoS). O QoS é responsável por garantir uma qualidade à um serviço oferecido. No entanto, as implementações voltadas para QoS se chocam com a atual arquitetura da rede, pelo fato que qualquer mecanismo de reserva de banda ou mudança de prioridade nos pacotes irá afetar o princípio do melhor esforço. As soluções mais utilizadas, capazes de garantir o QoS para alguns serviços, sem interferir na arquitetura de rede atual são o IntServ e o DiffServ.

O presente trabalho apresenta os problemas existentes nas redes de computadores atuais, assim como, os impasses enfrentados com a provisão de Qualidade de Serviço e apresenta o paradigma de Redes Definidas por Software como uma das principais propostas para a viabilização da Internet do Futuro, além de oferecer por meio dele, a avaliação de QoS para videoconferência em um ramo da REDUnB utilizando parâmetros reais da rede.

A simulação de videoconferência com parâmetros do codificador H.261 na plataforma *Mininet* em um ramo do cenário real da rede metropolitana da UnB com os parâmetros reais do tráfego da rede mostraram que é possível realizar uma videoconferência na rede, com baixa carga, sem a necessidade de provisão de QoS mas já com a rede em sobrecarga se faz necessário a provisão de QoS a fim de garantir o serviço de videoconferência.

ABSTRACT

The current architecture of computer networks brings a challenge for researchers in the field, because of difficulty in carrying out new technologies in real networks experiments. In order to overcome this issue, new technologies and new protocols are tested on simulators with scenarios that simplify a real network.

The paradigm of Software Defined Networks (SDN) and OpenFlow architecture provide a path for this problem and constitute a promising response to the challenges faced by researchers and flexible , scalable, and dynamically manageable network service providers, making possible tests in real network.

Another challenge faced today is related to the Quality of Service (QoS). The QoS is responsible for ensuring quality in one offered service. However, QoS implementations clash with the current network architecture with the fact that, any mechanism of bandwidth reservation or packets priority changes, will affect the best effort principle. Currently, one of the most widely used solutions, able to ensure QoS for some services, without interfering with the current network architecture are IntServ and DiffServ.

This paper presents the today's problems with computer networks, as well as the dilemmas faced with the provision of Quality of Service and presents the paradigm of Software Defined Networks as one of the main proposals for the viability for the Future of Internet, as well as offer through it, the evaluation of QoS for videoconferencing on a branch of REDUnB using actual parameters of the network.

The videoconferencing simulation with H.261 encoder parameters in Mininet platform on a branch of metropolitan network of UnB with the actual parameters of network traffic showed that it is possible to hold a videoconferencing on the network with low load, without the need of QoS provision but with the network overhead the QoS provision is necessary to ensure the videoconferencing service.

SUMÁRIO

Sumário

LISTA DE FIGURAS

Lista de Figuras

LISTA DE TABELAS

Índice de tabelas

LISTA DE SIGLAS E SÍMBOLOS

Siglas

BER	*Bit Error Rate*
CoS	*Class of Service*
cRTP	*Compressed Real-time Transport Protocol*
D-ITG	*Distributed Internet Traffic Generator*
DCCP1	*Datagram Congestion Control Protocol*
DNS	*Domain Name System*
DS	*Differentiated Services*
FTP	*File Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDT	*Inter Departure Time*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IP ECN	*Internet Protocol Explicit Congestion Notificaion*
ITU	*International Telecommunication Union*
ITU-T	*Telecommunication Standardization Sector*
LAN	*Local Area Network*
LLT	*Time to Live*
MAC	*Media Access control*
MAN	*Metropolitan Area Network*
NAT	*Address Translation*
ONF	*Open Networking Foundation*
PS	*Packet Size*
QoS	*Quality of Service*
RSVP	*Resource Reservation Protocol*
SCTP	*Stream Transmission Control Protocol*
SLA	*Service Level Agreement*
TCP	*Transmission Control Protocol*
TI	*Tecnologia da Informação*
ToS	*Typer of Service*
UDP	*User Datagram Protocol*
UnB	Universidade de Brasília
VLAN	*Virtual Local Area Network*

VLAN PCP	*Virtual Local Area Network Priority Code Point*
VM	*Virtual Machine*
VoD	*Video on Demand*
VoIP	*Voice over IP*
WAN	*Wide Area Network*

Sobrescritos

bps	*Bits per second*
DiffServ	*Differentiated Services*
InteServ	*Integrated Services*
kbps	*Kilo bits per second*
max	Máximo
Mbps	*Mega bits per second*
min	Mínimo
ms	*Millisecond*
pps	*Packet per second*
TB	*Terabyte*

Símbolos

k (kilo)	10^3
M (mega)	10^6
m (mili)	10^{-3}
σ	Desvio padrão do atraso
d_i	Atraso do pacote *i*
d'	Média de atraso dos pacotes
S_i	Tempo de transmissão
R_i	Tempo de recepção

Abreviação referente ao *host*

hX – .X – *host* X	Referente ao Host X com IP 10.0.0.X

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

A Internet evoluiu de forma extremamente rápida. Em 2005 ela atingiu 1 bilhão de usuários, em 2010 atingiu 2 bilhões e estima-se que este ano atinja cerca de 3 bilhões de usuários [1]. Desta forma, a internet desempenha um papel fundamental nas sociedades modernas, sendo responsável não mais, apenas, pela troca de informações, propriamente dita, mas, por proporcionar diversos serviços, entretenimento, comunicação e por se tornar uma das principais responsáveis pela união de pessoas, pela transmissão de culturas e ideias, por facilitar nossas tarefas cotidianas, em geral e por ditar o ritmo da sociedade atual.

Ao mesmo tempo em que ocorre essa expansão, a Internet vem sofrendo transformações, tornando-se uma plataforma mais robusta e confiável. No entanto, tais transformações são enraizadas em razão de que ocorreram baseadas na matriz da internet. Isto ocorre devido à dificuldade para implantação de novos protocolos e tecnologias, pois a grande quantidade de equipamentos e protocolos, já consagrados, restringe o lançamento de ideias inovadoras [2], visto que novas propostas, em geral, requerem mudanças em todos os milhares de equipamentos de rede instalados [3], e geralmente os experimentos precisam ser feitos com tráfego real de produção [2]. Hoje já há um consenso, na comunidade acadêmica, de que a Internet está "ossificada" [4].

As redes de computadores são parte da infraestrutura crítica do nosso dia a dia. Mudanças tornam-se obstáculos e acabam sendo descartadas pelos administradores de rede. Todas essas barreiras dificultam a implantação de novas ideias, que acabam não sendo testadas em ambientes reais [3].

As redes virtuais programáveis permitem que pesquisadores possam experimentar novas arquiteturas e protocolos de redes. O paradigma de Redes Definidas por Software e o protocolo OpenFlow foram desenvolvidos com esse propósito, sendo uma maneira de testar novas propostas em redes reais sem afetar as redes de produção. Dessa forma, além de atender os problemas de inflexibilidade das redes de computadores, o paradigma SDN terá a esperança de atender às necessidades das novas aplicações de rede, que deverão surgir em um futuro próximo [3].

O protocolo OpenFlow é um padrão em desenvolvimento para administração de redes corporativas, como Redes Locais (*Local Area Network* – LAN), Redes de Longa Distância (*Wide Area Network* – WAN) e Redes de área Metropolitana (*Metropolitan Area Network* –

MAN). A maioria das redes corporativas, utiliza equipamento que necessitam de administração manual, o que ocasionam os problemas encontrados pelos pesquisadores, ao tentarem experimentar novas arquiteturas e protocolos de redes.

Através do OpenFlow é possível modificar o comportamento de dispositivos de rede remotamente. O OpenFlow é um padrão aberto, que permite implantar novos protocolos de roteamento em uma rede, abstraindo componentes da mesma. Dessa forma, pesquisadores podem realizar experimentos em redes físicas ou virtuais e distribuir novas aplicações em ambiente de produção [2]. Isso também possibilita uma aplicação muito importante, que é a garantia de Qualidade de Serviço (*Quality of Service* – QoS) na rede.

Com o aumento de usuários, também houve um aumento dos serviços prestados na Internet, com o aumento de tráfego de vídeo e serviços, que dependem de Qualidade de Serviço. Um exemplo disso é que em 2012 metade do tráfego da Internet era tráfego de vídeo [1]. QoS refere-se a capacidade da rede em prestar um melhor serviço, permitindo o transporte de pacotes com requisitos especiais.

Diferentes das aplicações elásticas (ex: Email, Downloads, Compartilhamento de arquivos) [5], serviços que dependem de QoS são sensíveis à Variação do Atraso (*Jitter*), Atraso (*Delay*), Perda de Pacotes (*Packet Loss*), Taxa de Bits (*BitRate*), Latência (*Latency*), pacotes fora de ordem. Exemplos de serviços que dependem de QoS são *Voice over IP* (*VoIP*), Videoconferência [5], *Video on Demand* (*VoD*).

Os principais fabricantes de *switches Ethernet* comerciais estão adotando SDN através da implementação do protocolo OpenFlow [6]. Diante deste cenário de Internet do Futuro, há uma gama de possibilidades a serem exploradas e a motivada para este trabalho é a avaliação de qualidade de serviço de videoconferência em uma rede metropolitana específica, a REDUnB, com o paradigma de Redes Definidas por Software.

1.2. OBJETIVOS

Este trabalho apresenta, como objetivo geral, a avaliação do desempenho de aplicações de videoconferência na rede metropolitana da UnB, via o paradigma de Redes Definidas por Software.

Incluso, no objetivo geral, estão alguns objetivos específicos que possibilitam o cumprimento do mesmo. São eles:

º Estudo sobre as Redes Definidas por Software

º Introdução sobre QoS e seus parâmetros para videoconferência

º Simulação de videoconferência utilizando um ramo da REDUnB para dois cenários com dados do tráfego real da rede (obtidos pelo *NAGIOS*):

º Quando a rede está com carga baixa, denominada de Carga Baixa

º Quando a rede está com sobrecarga, denominada de Carga Alta

º Avaliação dos requisitos de QoS para videoconferência

1.3. ORGANIZAÇÃO DO TEXTO

O capítulo 2 apresenta o paradigma das Redes Definidas por Software, assim como as ferramentas utilizadas para a virtualização de redes e controle da rede, o protocolo OpenFlow e os controladores SDN, respectivamente.

O capítulo 3 aborda o conceito de Qualidade de Serviço (QoS), os parâmetros necessários para analisar o desempenho de QoS, dando ênfase à videoconferência.

O capítulo 4 apresenta a proposta do trabalho, além, da metodologia utilizada.

O capítulo 5 aborda as ferramentas e programas utilizados, D-ITG (gerador de tráfego) e plataformas e controladores SDN.

O capítulo 6 apresenta a avaliação de desempenho. Simulações e resultados.

O capítulo 7 conclui o trabalho, apresentando as principais conclusões acerca do tema e sugerindo propostas de trabalhos futuros.

2. REDES DEFINIDAS POR SOFTWARE

2.1. INTRODUÇÃO

Toda a evolução das redes de computadores, no âmbito de penetração, dispersão e aplicações, que permitiram a Internet como ela é hoje, se deu graças a sua arquitetura baseada no controle distribuído, que tornou possível tal progresso. No entanto, esta arquitetura não prosperou da mesma forma que a Internet e acabou se tornando incapaz de atender os novos requisitos da Internet.

Nesse modelo de controle distribuído, o processamento funciona por comutação no hardware (*switches* e roteadores) , por onde trafegam os pacotes de dados e suas funções de controle, que se encontram dentro deste mesmo hardware [7].

Diante de tal impasse, foram feitas pesquisas para a Internet do Futuro. Estas pesquisas visam resolver problemas como os de associação ao endereçamento, à mobilidade, ao crescimento em escala, ao gerenciamento e à qualidade de serviço [3].

O paradigma de Redes Definidas por Software constitui uma promissora resposta para tais problemas. Ao contrário do funcionamento dos equipamentos de redes tradicionais, onde, como dito anteriormente, as tomadas de decisões ocorrem no interior dos próprios equipamentos por algoritmos muitas vezes fechados e de difícil ou impossível modificação [3], a ideia central das Redes Definidas por Software consiste em mover as funções de controle de rede para fora dos elementos que constituem a estrutura física da rede [7].

Com tal abstração, entre o plano de controle da rede e a estrutura física da rede, torna-se possível a virtualização da mesma, além do que, a inteligência lógica da rede é centralizada em um único ponto, sendo o controlador SDN responsável por este ponto.

Como o próprio nome diz, controladores são softwares responsáveis pelo controle da rede, permitindo, dessa forma, que uma rede inteira seja controlada por um único componente. Uma das principais vantagens disso é que, como os controladores podem ser programados, as redes passam a ser flexíveis para lidar com requisitos e problemas, tanto atuais como futuros, e as Redes Definidas por Software garantem essa possibilidade de programação.

2.2. VIRTUALIZAÇÃO

A implantação de novos protocolos e serviços no núcleo da Internet sofre rejeição de

grande parte dos provedores de serviços devido ao grande risco que essas mudanças representam para a rede. Uma das propostas para conciliar o desenvolvimento de inovações sem prejudicar o tráfego de produção é a virtualização de redes [8], assim como a virtualização das funções de rede NFV (*Network Functions Virtualization*).

A virtualização de rede (NFV) é uma iniciativa para virtualizar funções de rede anteriormente realizadas por hardware dedicado e proprietário. Como se propõe, a virtualização das funções de redes deve diminuir a quantidade de hardware proprietário necessário para implantar e operar serviços de rede [7].

Objetivando reduzir a dependência de dispositivos proprietários, assim como a melhora da flexibilidade de serviço por meio de um *framework* baseado em *software* mais ágil para a construção de recursos e serviços, o propósito do NFV tem como foco, a redução dos custos de implantação dos serviços [7].

Algumas razões pelas quais se deve optar pela virtualização da rede são, por exemplo:

-**Compartilhamento**, pois caso um único usuário utilize um recurso muito grande é possível dividi-lo em várias partes virtuais;

-**Isolamento**, visto que usuários que fazem uso de um componente virtual não devem ser capazes de monitorar/interferir em outros usuários;

-**Agregação**, que viabiliza, no caso de um recurso muito pequeno, a possibilidade de construí-lo de forma bem maior na rede virtual ;

-**Dinamismo**, pois propicia grande vantagem com a possibilidade de rápida realocação de recursos ;

-**Facilidade de Migração**, visto que permite a possibilidade de migrar da rede virtual para a física.

Isolamento

A virtualização de redes permite que seja compartilhem sua capacidade, de maneira que realize, simultaneamente, múltiplas funções, estabelecendo infraestruturas lógicas distintas e mutuamente isoladas, provendo um método para que múltiplas arquiteturas de rede distintas compartilhem o mesmo substrato físico [3].

Uma rede virtual tem como característica seu isolamento, garantindo assim a possibilidade de experimentos, visto que o tráfego experimental não influenciará o tráfego de produção. A Figura 2.1 exemplifica uma rede virtualizada. Uma das principais abordagens

para as redes de computadores do futuro consiste na ideia de que cada rede virtual é isolada e possui sua própria pilha de protocolos e seu gerenciamento individual

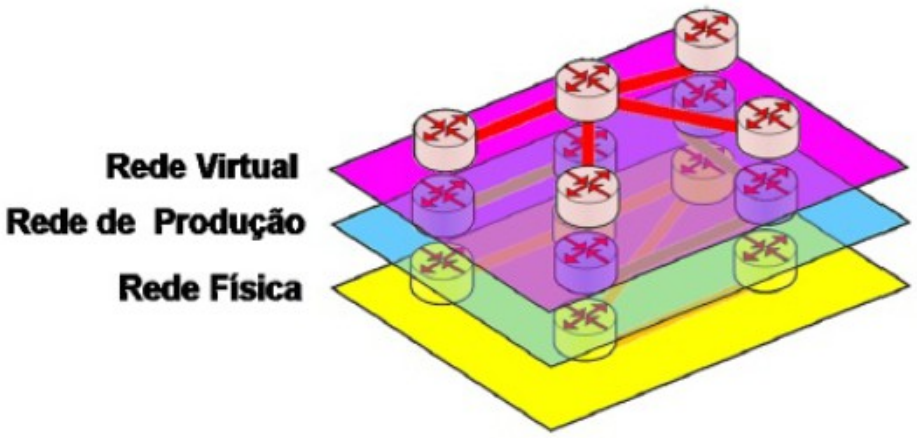

Figura 2.1: Rede virtualizada, com duas redes compartilhando o mesmo substrato físico [8]

De fato, virtualização não é um conceito novo e ela já esta presente há algumas décadas em nossas vidas, como por exemplo as *Virtual* LANs (VLANs). Uma VLAN torna possível, que vários departamentos de uma empresa compartilhem uma LAN física com isolamento.

Migração

Com experimentos em redes virtualizadas, sem a interferência na rede local, é possível realizar a migração sem interferir no correto funcionamento da própria. Sem a possibilidade de migração seria essencial, por exemplo, o desligamento de alguns nós caso fosse necessária a manutenção dos próprios. Isso geraria quebra de conexões e, no caso de os nós serem roteadores, o desligamento poderia gerar perda de adjacências dos protocolos de roteamento, introduzindo um atraso de convergência, até que as rotas fossem reorganizadas em todos os nós [8].

Para tal exemplo, uma solução seria a migração do nó virtual para o nó físico que esteja em funcionamento. Desta maneira, as redes virtuais garantem que a topologia lógica não seja alterada, fazendo com que as rotas permaneçam válidas.

Uma importante técnica de virtualização de redes com suporte à migração é oferecida pelo protocolo OpenFlow [8]. O OpenFlow é um protocolo padrão aberto, que permite

modificar o comportamento dos dispositivos de rede através de um conjunto de instruções, definidas por um controlador remoto. Com isso, a virtualização é uma das maiores vantagens de SDN [10], garantindo isolamento e com isso, possibilitando testes de novos protocolos e arquiteturas com tráfegos reais além de facilitar a implementação da rede testada (em ambiente virtual) para a rede real, por meio da migração. Mais à frente, será feita uma abordagem mais detalhada sobre o protocolo em questão.

2.3. REDES PROGRAMÁVEIS

Em um ambiente tradicional de TI, a configuração da rede é feita dispositivo por dispositivo. Esta abordagem apresenta alguns problemas, como a questão de tempo, no caso de muitos dispositivos da rede precisarem de uma configuração, e também possibilidade de erros, visto que serão realizadas várias configurações.

Esta abordagem tem como característica o fato de ser estática. Este caráter estático leva a uma redução da performance da rede, visto que a rede é pouco flexível e não garante a possibilidade de rápidas tomadas de decisões.

A arquitetura atual dos roteadores de rede, mostrada na Figura 2.2, é formada, basicamente, por duas camadas distintas: o *software* de controle e o *hardware* dedicado ao encaminhamento dos pacotes.

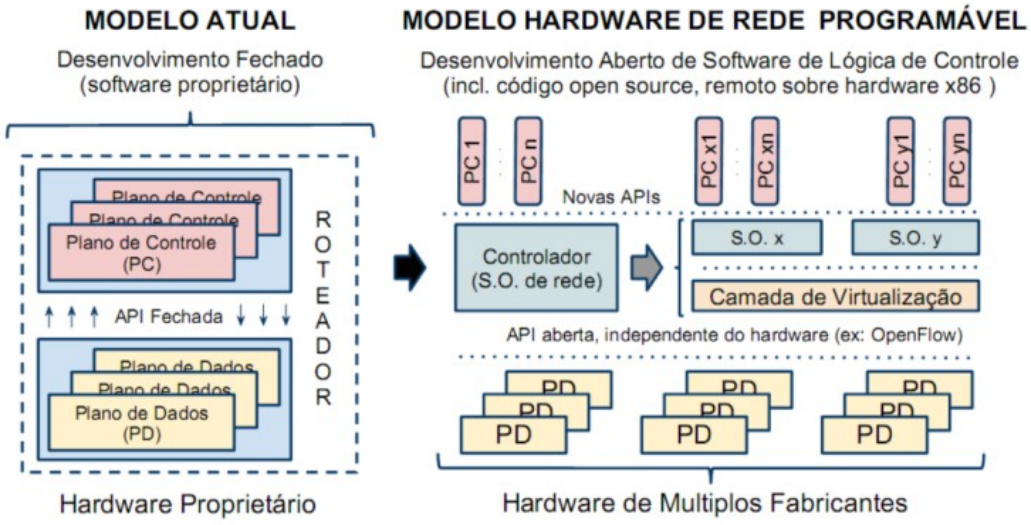

Figura 2.2: Arquitetura de roteadores: modelo atual (API fechada) e modelo programável (API aberta) [9].

Redes Definidas por Software, ou redes programáveis, são redes cujo substrato físico é composto por equipamentos de propósito geral e a função de cada equipamento, ou conjunto de equipamentos, é realizada por um *software* especializado [8].

As Redes Definidas por Software são baseadas na separação entre o plano de controle e o plano de dados. O plano de dados consiste nos componentes de repasse, que promovem uma simples comutação por rótulos. É também chamado de plano de encaminhamento [3], por cuidar da comutação e repasse dos pacotes da rede. O plano de controle está preocupado com funções de coordenação, como roteamento e sinalização, e é responsável pelos protocolos e pelas tomadas de decisões que resultam nas tabelas de encaminhamento.

Como já mencionado, nos equipamentos comuns o plano de controle e o de dados são implementados dentro do próprio equipamento, o que, de fato, impossibilita qualquer tomada de decisão extra que não esteja determinada no equipamento.

Para que seja possível implementar algo flexível, possibilitando uma tomada de decisão extra, é necessário que o plano de controle e de dados funcionem fora desse equipamento, em outras palavras, é preciso que o equipamento de rede se comunique com o meio externo, encaminhando pacotes por meio de protocolos e tais implementações externas sejam abrigadas em uma máquina externa, física ou virtual.

Arquitetura

O princípio básico das SDN é essa extração das funções de controle de rede da estrutura física, separando o plano de controle do plano de dados, possibilitando assim, a comunicação com o meio externo. Além disso, tal mecanismo também favorece a centralização da inteligência da rede, como ilustrado na Figura 2.3. O encarregado por esse envio de informações “externas”, assim como recebimento das informações “internas” da rede, e responsável pela inteligência desta é o controlador SDN.

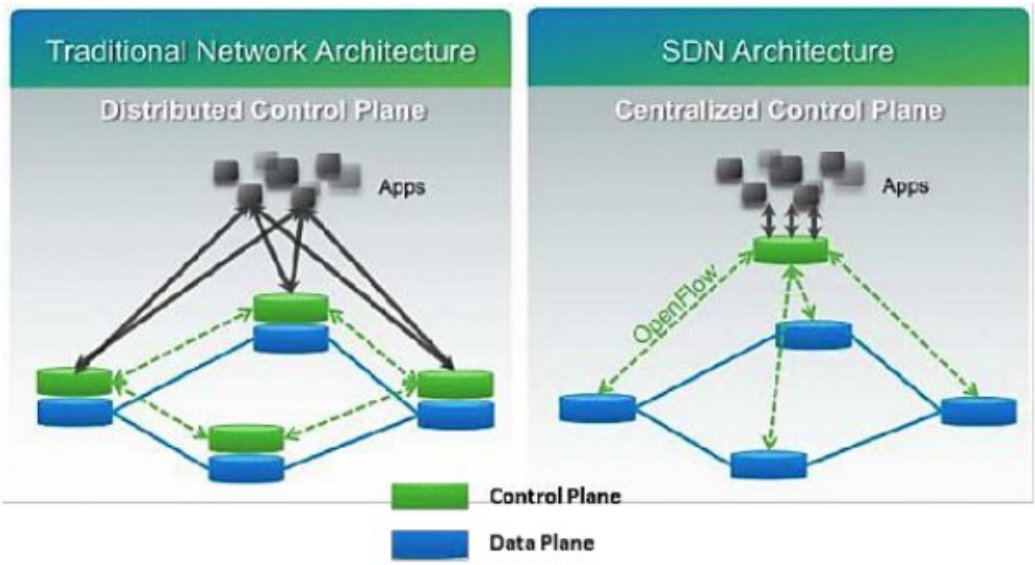

Figura 2.3: Arquitetura de rede tradicional VS Arquitetura SDN [7].

Em uma rede SDN, os responsáveis por sua administração têm total controle sobre ela, podendo moldar o tráfego, mudar as rotas e alterar as regras de qualquer *switch* de rede a partir deste console de controle centralizado sem a necessidade de configurar cada switch da rede. Todo o mecanismo é realizado através deste controlador SDN, que detém a inteligência e total controle da rede, garantindo, por exemplo, maior flexibilidade, eficiência, agilidade nas tomadas de decisões e mais controle sobre o fluxo de tráfego na rede, para os responsáveis por sua gerência.

Com a inteligência lógica da rede centralizada nos controladores SDN, e a separação entre o plano de controle e o plano de dados, a rede passa a ser dividida em camadas [2] como é possível observar na Figura 2.4.

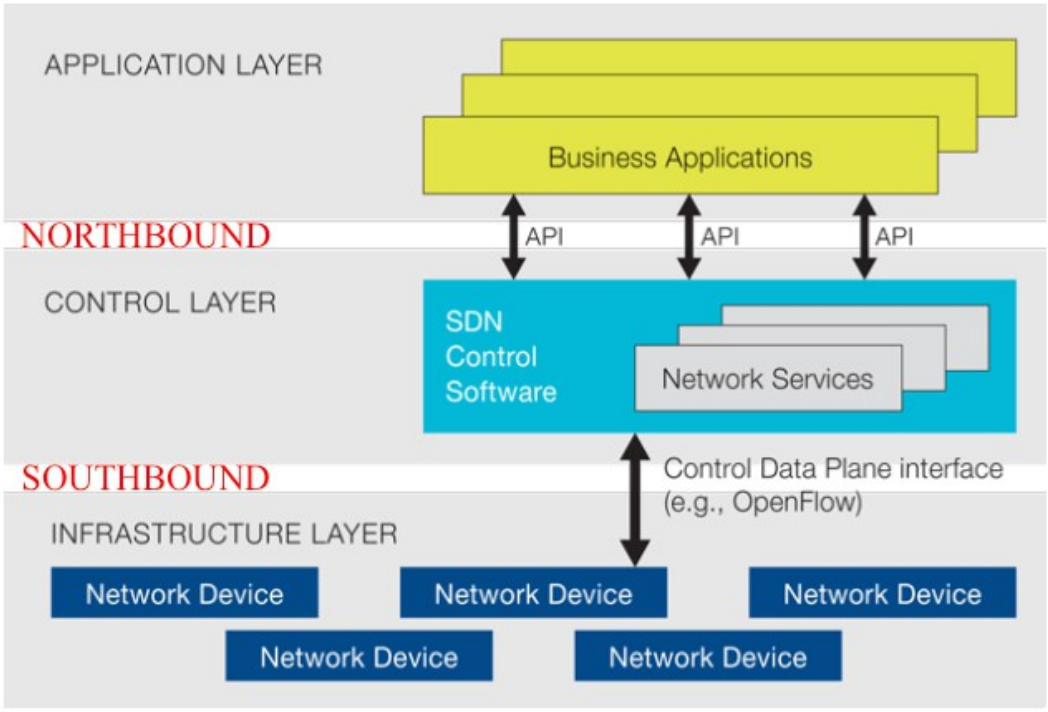

Figura 2.4: Adaptada – Arquitetura de Redes Definidas por Software (SDN) [2]

Pelo fato de a arquitetura SDN estar em camadas, é preciso que exista uma comunicação entre a camada de controle e a camada de infraestrutura (Interface *southbound*). Para que tal comunicação seja possível, tanto o controlador, como os dispositivos de rede, precisam se "entender" e isto é feito por meio de um protocolo padronizado.

O protocolo utilizado para esta interface de comunicação, neste trabalho, é o OpenFlow, definido pela ONF (*Open Networking Foundation*) [12]. Porém, este não é o único padrão SDN, visto que Grandes grupos como *IBM* e *Intel Corporation* adotaram o OpenFlow como solução SDN, mas outras empresas, como a *Cisco Systems* e a *Juniper Networks,* preferiram adotar estratégias diferentes, utilizando suas próprias soluções SDN [11].

2.4. PROTOCOLO OPENFLOW

O protocolo OpenFlow é o responsável por realizar a comunicação na interface *southbound* (entre a camada de controle e a camada de infraestrutura). Essa comunicação permite modificar o comportamento dos dispositivos de rede através de um conjunto de instruções definidas pelo controlador [2]. Dessa forma, o protocolo garante a comunicação entre as camadas, permitindo o controle e gerenciamento das tabelas de encaminhamento dos *switches* de rede, roteadores e pontos de acesso [7].

O OpenFlow é um protocolo de virtualização de redes baseada na comutação de fluxos [3]. Ele procura oferecer uma opção controlável e programável, sem atrapalhar o fluxo de produção. A fim de garantir tal opção de isolamento de fluxos, o comutador OpenFlow utiliza o conceito de Redes Definidas por Software, cujo substrato físico é composto pela parte que cuida do tráfego de produção da rede, e por outra parte que cuida do tráfego experimental das novas propostas de rede [3].

O *switch* OpenFlow garante ainda a possibilidade de realizar várias pesquisas ao mesmo tempo. Isto é possível graças a virtualização do tráfego experimental da rede, tornando viável o compartilhamento dos fluxos. O *switch* OpenFlow encaminha os pacotes do tráfego experimental de acordo com regras definidas por um controlador centralizado. Mais à frente será realizada uma explicação mais aprofundada a respeito do *switch* OpenFlow.

Outra vantagem do paradigma OpenFlow é que, graças a generalização do plano de dados, qualquer modelo de encaminhamento de dados que se baseie na tomada de decisão por algum valor no cabeçalho dos pacotes pode ser suportado [3].

Com tal viabilidade, torna-se possível a gerência de QoS (*Quality of Service*) em que, pacotes sensíveis a atrasos, latência, perda de pacotes, dentre outros, recebam um valor específico em seus cabeçalhos, para que possam ser identificados e tratados de forma diferente na rede. Com a aplicação de QoS, é possível a identificação destes pacotes e também a provisão de medidas específicas. Além do controle QoS, garantia de QoS outras aplicações viáveis são a coleta de estatísticas e também o *Firewall*. No capítulo 3 o QoS será abordado de forma mais específica.

O OpenFlow possibilita também que os fabricantes possam adicionar suas funcionalidades em seus equipamentos (como roteadores, *switches*) sem a necessidade de expor o projeto de *software* e *hardware* [3]. Isto garante a não necessidade de exposição do projeto dos fabricantes em suas plataformas, além de gerar uma implementação de baixo custo e alto desempenho.

A especificação OpenFlow na sua versão 1.3.3 foi lançada em dezembro de 2013 pela Open Networking Foundation – ONF [7] (sua versão 1.5.0 encontra-se em progresso durante desenvolvimento deste trabalho [15])

Componentes de uma Rede OpenFlow

Uma Rede OpenFlow possui uma característica própria, e para que a mesma exista são

necessários alguns componentes. Basicamente, para que uma rede programável com OpenFlow exista, é necessária a presença de quatro componentes. É necessário que equipamentos habilitados tenham a capacidade de alterar suas tabelas de encaminhamento conforme as decisões de um controlador em *software* a eles conectados por um canal seguro [3].

Tabela de Fluxos - Cada entrada na tabela de fluxos do *hardware* de rede é composta por regras, ações e controles de estatística. A regra é formada pela definição dos valores de um ou mais campos do cabeçalho do pacote e é por meio dela que é determinado o fluxo. As ações então ficam associadas ao fluxo e vão determinar como os pacotes devem ser processados, para onde vão ser encaminhados ou se serão descartados [3]. Os controles estatísticos consistem em manter estatísticas de utilização e remoção de fluxos inativos.

Protocolo OpenFlow – Protocolo responsável pela troca de mensagens entre os esquipamentos da rede e o controlador da rede. Essas mensagens podem ser simétricas, assíncronas ou ainda iniciadas pelo controlador [13].

Controlador – O controlador fica responsável por tomar as decisões adicionando e removendo entradas na tabela de fluxos de acordo com uma política de encaminhamento [3].

Canal seguro – Garante que as trocas de mensagens sejam confiáveis e com baixa taxa de erros, além, é claro, de proteção contra ataques mal intencionados. A interface recomendada pelo projeto OpenFlow é o SSL (*Secure Socket Layer*), mas existem alternativas como o TCP e o PCAP (*Packet Capture*).

Um *switch* OpenFlow

Como mencionado anteriormente, a inteligência da rede está centralizada no controlador SDN. É deste ponto que as informações e funções são enviadas para os equipamentos da rede. Nesta seção será dado ênfase ao *switch* OpenFlow, que foi o equipamento de rede utilizado neste trabalho.

Com o uso do protocolo OpenFlow abrem-se muitas portas para a área de pesquisa, sendo uma delas, como já mencionado, a possibilidade de executar seus experimentos sem que os fabricantes tenham que expor a implementação interna dos seus equipamentos. Tal vantagem, garante ao OpenFlow a possibilidade de implementar uma solução de baixo custo mas com alto desempenho.

Outra vantagem é a flexibilidade para se programar, de forma independente, o tratamento

de cada fluxo da rede e como ele deve ou não ser encaminhado pela rede. De uma maneira mais prática, o OpenFlow determina como um fluxo pode ser definido, quais serão as ações que podem ser realizadas para cada pacote pertencente a este fluxo, e qual é o protocolo de comunicação entre o controlador e os comutadores utilizados para realizar as definições de fluxo e ações [3].

Tendo essas grandes vantagens, diferentes *switches* comerciais estão chegando ao mercado com as funcionalidades de um *switch* comum, mas com o recurso de OpenFlow disponível e pronto para ser ativado. Esses *switches* são conhecidos como *switches* híbridos [2] existindo também a possibilidade de que outros modelos sejam puramente OpenFlow.

Um *switch* de rede comum é um dispositivo capaz de analisar o cabeçalho dos pacotes e tomar decisões sobre como encaminhá-los [14]. Quando um pacote chega ao *switch* OpenFlow, seu cabeçalho é verificado e comparado com a tabela de fluxos. Se a tabela de fluxo contiver informação para o pacote analisado, dizemos que ocorreu uma combinação (*match*), e uma ação em relação a tal pacote é tomada. Por exemplo, o pacote será encaminhado para a porta específica ou será descartado.

Quando um *switch* OpenFlow recebe um pacote e não ocorre a combinação com a tabela de fluxos, ele encapsula o pacote em questão e o envia para o controlador SDN. O controlador, então, irá decidir qual ação tomar para o pacote e notificará todos os demais *switches* necessários para tal fluxo para descartá-lo ou criar uma nova entrada na tabela de fluxos para suportar o novo fluxo.

Como citado anteriormente, existem quatro componentes básicos para a existência de uma rede OpenFlow, e os elementos básicos para um *switch* OpenFlow são os mesmos, exceto, pelo componente controlador. Um *switch* OpenFlow necessita de uma tabela de fluxo, um canal seguro e o protocolo OpenFlow. É possível visualizar sua arquitetura na Figura 2.5.

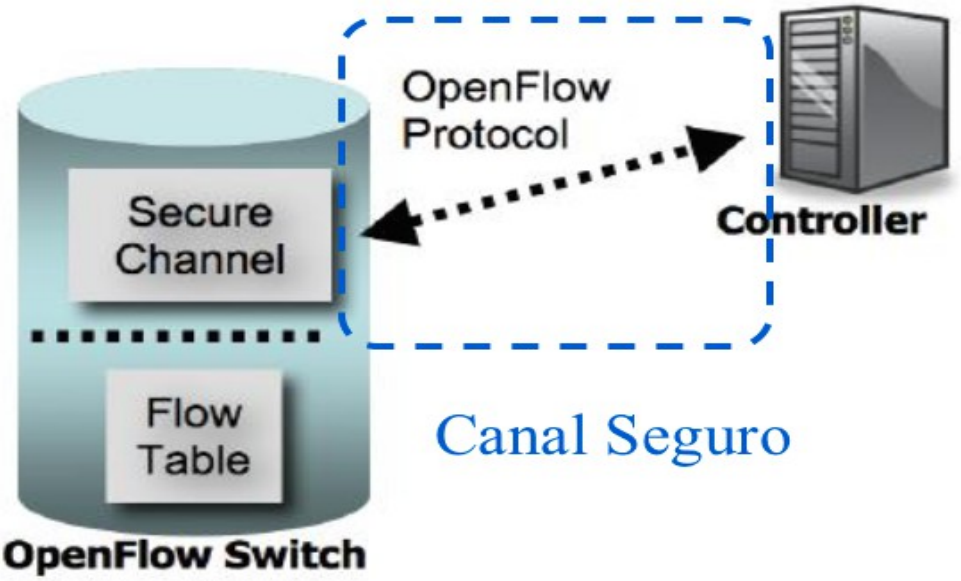

Figura 2.5: Switch OpenFlow comunicando-se com um controlador por um canal seguro [2].

O modelo de encaminhamento de um *switch* OpenFlow consiste no que é conhecido como Tabela de Fluxos (*flow table*). Estas tabelas são as responsáveis por conter as ações a serem tomadas para cada entrada de fluxo (*flow entry*), e pode ser vista na Tabela 2.1 os campos que formam uma tabela.

º Campos de Combinação (*Match Fields*): Campos responsáveis por fazerem a análise dos pacotes que chegam ao *switch* e consultar à tabela de fluxo para verificar se possuem informações sobre tal pacote. Caso existam, ocorre a combinação.

º Contadores (*Counters*): Existem contadores específicos para tabelas, fluxos, portas ou filas que fornecem informações estatísticas. Por exemplo: o tempo passado desde (duração) que uma regra de fluxo foi instalada no switch, quantidade de pacotes ou bytes recebidos em uma determinada porta, ou por um determinado fluxo [2].

º Ações (Actions): Ações e instruções a serem tomadas com os pacotes que atenderam à regra de fluxo especificada. Caso não exista uma especificação, o pacote é descartado.

Tabela 2.1: Formato de entrada de fluxo (flow entry) [2].

CAMPO DE COMBINAÇÃO	CONTADORES	AÇÕES

A Tabela 2.2 apresenta os campos do cabeçalho usados pela tabela de fluxo para fazer a verificação.

Tabela 2.2: Campos do cabeçalho usados para a combinação pela tabela de fluxo [2].

In Port	Ethernet			VLAN		IP				TCP/UDP	
	src	dst	type	id	pcp	src	dst	proto	tos	srcp	dstp

Os seguintes campos constituem são [2]:

º *In Port*: Porta de entrada do *switch;*

º *Ethernet Source*: Endereço MAC de origem;

º *Ethernet Destination*: Endereço MAC de destino;

º *Ethernet Type:* Tipo de quadro (*frame*) *Ethernet;*

º VLAN ID: Número de identificação da VLAN (*Virtual* LAN);

º VLAN PCP (*Priority Code Point*): Nível de prioridade;

º IP *Source*: Endereço IP de origem;

º IP *Destination*: Endereço IP de destino;

º IP *Protocol*: Protocolo IP;

º IP ToS (*Type of Service*): Tipo de serviço;

º TCP/UDP *Source Port*: Porta de origem do protocolo TCP/UDP

º TCP/UDP *Destination Port*: Porta de destino do protocolo TCP/UDP

As principais ações que um *switch* OpenFlow deve possuir são definidas por padrão, sendo elas [3]:

º Encaminhar o pacote para uma determinada porta;

º Encapsular o pacote e encaminhar para o controlador;

º Alterar parte do cabeçalho do pacote;

º Descartar o pacote;

Caso esteja em uso o *switch* OpenFlow híbrido, é possível a realização, também, de ações comuns da mesma forma em que um *switch* comum faria [2].

Um fluxo é descrito por seus campos de correspondência (de acordo com a especificação OpenFlow 1.3) na Figura 2.6. Nesta figura, para cada fluxo, o *switch* aplica as ações correspondentes caso uma entrada correspondente já esteja definida em qualquer um do seu fluxo de mesas. Por outro lado, caso um fluxo não possua nenhuma entrada correspondente em qualquer uma de suas mesas, o detector passa a procurar uma ação na entrada da tabela-*miss* e se, não existir, ocorre o descarte do pacote.

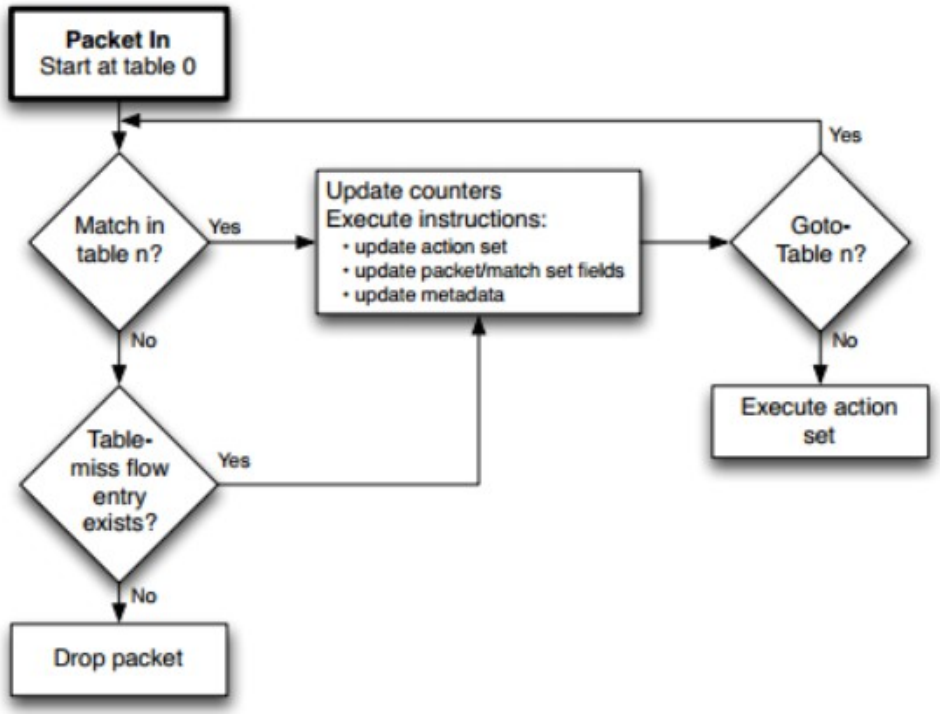

Figura 2.6: Fluxograma especificando o fluxo de pacotes através de um switch OpenFlow. [7]

2.5. CONTROLADOR

O controlador, como abordado nas seções anteriores, é o principal dispositivo da rede, assim como o grande responsável por manter todas as regras da rede e distribuir as devidas instruções para os dispositivos da rede [14]. Nas Redes Definidas por Software, a inteligência da rede está concentrada em um único componente, e todas as funções que irão ditar o funcionamento dela partem deste ponto.

Esse componente é o controlador SDN, que é o responsável por concentrar a comunicação com todos os elementos programáveis da rede e oferecer uma visão completa e unificada da mesma.

Existem diversos controladores SDN que já foram desenvolvidos, cada um possui uma qualidade própria. A Tabela 2.3 apresenta algumas características dos principais controladores OpenFlow.

Tabela 2.3: Principais controladores SDN

NOME	LINGUAGEM	PLATAFORMA	CARACTERÍSTICA
NOX	Python, C++	Linux	Controlador de referência do OpenFlow
POX	Python, C++	Windows, Mac, Linux	Evolução do NOX
Maestro	Java	Windows, Mac, Linux	Explora o paralelismo
Floodlight	Java	Mac, Linux	Pode ser integrado a redes que não utilizam OpenFlow
OpenDaylight	Java	Linux	Arquitetura bem definida em camadas
RYU	Python	Linux	Suporta várias versões do OpenFlow

Durante o desenvolvimento deste trabalho, utilizou-se o controlador OpenDaylight.

Controlador OpenDaylight

Logo no começo do desenvolvimento deste trabalho foi descoberta uma máquina virtual (*virtual machine* , VM) totalmente voltada para SDN. No Apêndice B.1. é apresentada sua instalação. A VM SDNHub já traz instalada na própria VM as principais ferramentas de SDN, assim como o controlador OpenDaylight. Foi utilizado o programa VirtualBox versão 4.3.26 para executar a máquina virtual. No site oficial do SDNHub [16] é possível encontrar o tutorial completo sobre a utilização do controlador OpenDaylight.

O controlador OpenDaylight é um controlador novo que utiliza a linguagem Java e vem recebendo um grande incentivo e patrocínio de grandes empresas no ramo de TI como: Linux Foundation, Cisco, Citrix, Dell, Ericsson, HP, Intel, Oracle, Microsoft, dentre outras [40]. O controlador também possui uma plataforma bem interativa com o usuário e tornou viável que

trabalho fosse possível de uma forma menos complexa.

O OpenDaylight é uma plataforma modular, com a maioria dos módulos reutilizando alguns serviços e interfaces comuns [17]. A idéia com a criação de aplicativos na plataforma OpenDaylight é alavancar a funcionalidade em outros pacotes de plataformas, cada uma das quais exportam serviços importantes através de interfaces Java. Muitos destes serviços são construídos em um modelo de provedor para o consumidor, através de uma camada de adaptação chamada MD-SAL [17]. A Figura 2.7 apresenta a arquitetura da versão atual, Helium.

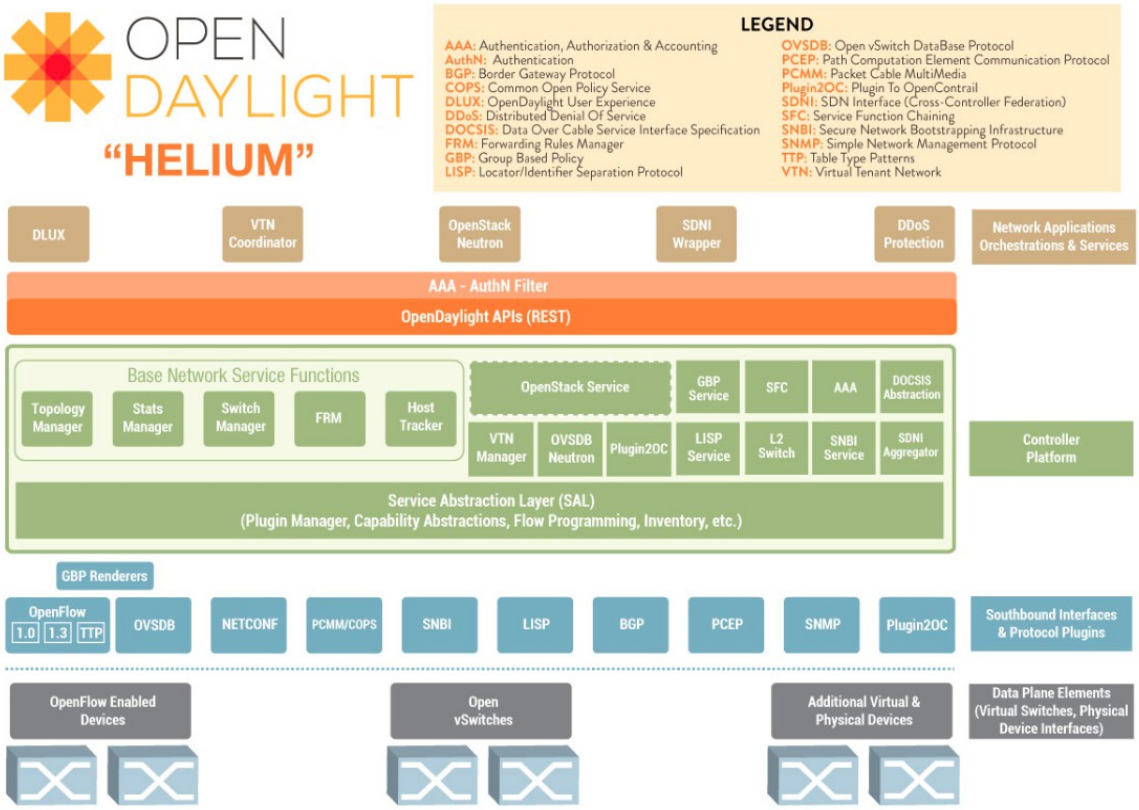

Figura 2.7: Arquitetura interna do OpenDaylight "Helium" (versão atual) [17].

O OpenDaylight Helium oferece uma interface gráfica. Para acessá-la basta digitar no seu *browser* "localhost:8080", ou então o IP do seu controlador (neste exemplo, 127.0.0.1:8080). Primeiramente, aparecerá uma tela de login padrão. Mostrado na Figura 2.8. Após o login é possível ter acesso à API do controlador, Figura 2.9. Neste exemplo, apenas o controlador está ligado, por isso é possível ver a mensagem "*No Network Elements Connected*". No Capítulo 5 será apresentada a questão das simulações realizadas para este trabalhado com o controlador OpenDaylight, e assim será explicado de forma mais

aprofundada o seu funcionamento durante a simulação.

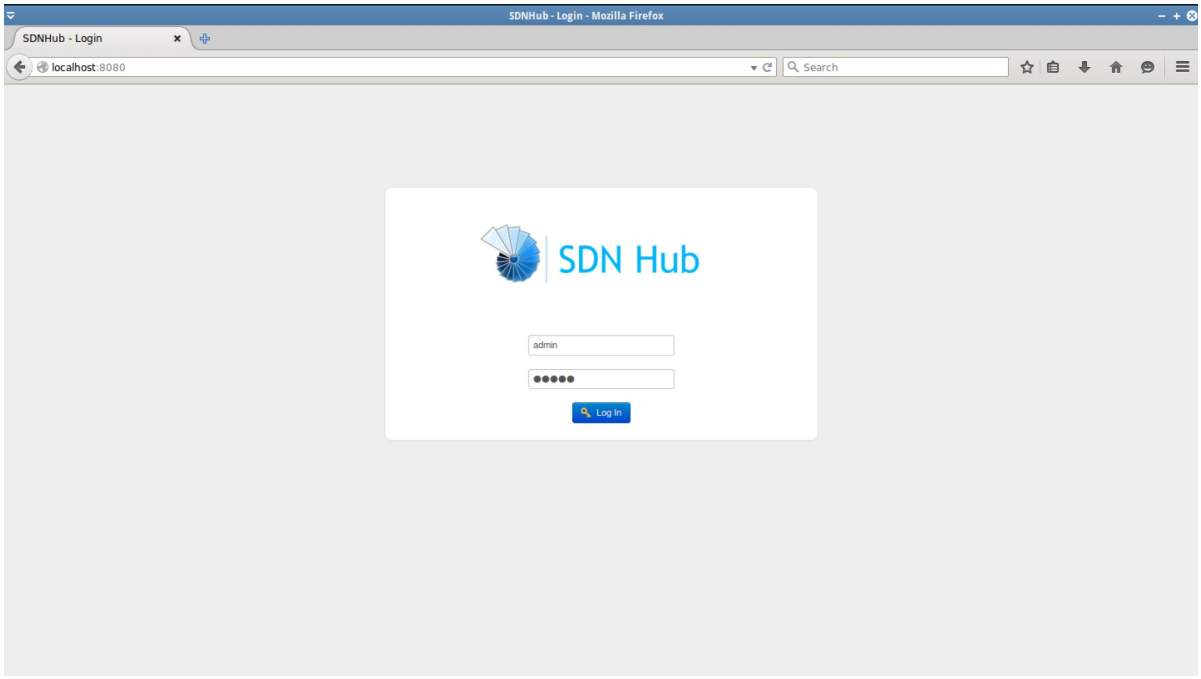

Figura 2.8: Página de login para acesso à interface web do controlador OpenDaylight.

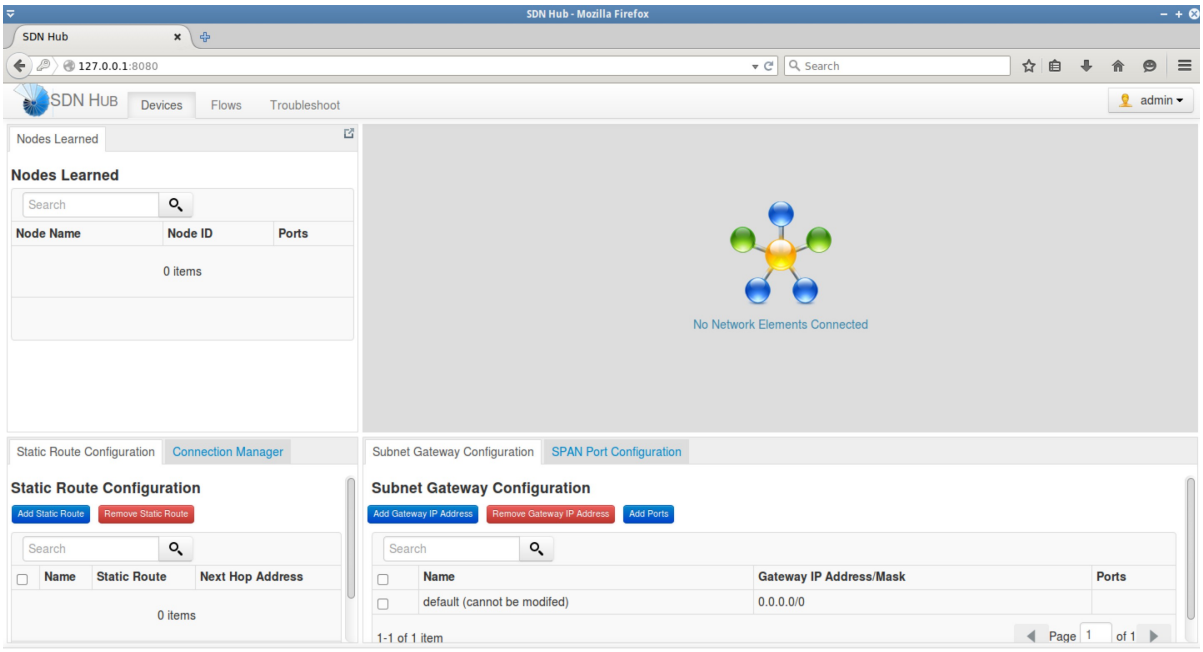

Figura 2.9: Página dos “Dispositivos” do controlador OpenDaylight na interface web.

2.6. RESUMO CONCLUSIVO

A Internet atualmente está “travada”, pois a mesma tecnologia que a fez evoluir é a responsável por impedir a continuação de tal progresso. As Redes Definidas por Software, como apresentado neste capítulo, possuem diversos atributos capazes de superar este impasse e fazer com que a Internet continue evoluindo. Várias destas soluções possibilitam, no geral, que mudanças, razoavelmente simples e de baixo custo, possam ser implementadas nas redes e em seus equipamentos, garantindo uma maior agilidade, flexibilidade e desempenho. Tal promessa, deve-se em grande parte, ao protocolo OpenFlow, responsável por padronizar e realizar a comunicação entre os componentes da rede e seus controladores. No capítulo seguinte será abordada a Qualidade de Serviço (QoS), serviço de bastante importância não só para Internet mas também para seus usuários.

3. QUALIDADE DE SERVIÇO

3.1. INTRODUÇÃO

A Internet cresceu muito, e junto a isso é possível perceber novas exigências tanto de usuários como de aplicações no que diz respeito à eficiência e melhor utilização de recursos disponíveis [28]. Atualmente a Internet é capaz de oferecer diversas informações, serviços, entretenimento além de comunicação. Com toda essa gama de oportunidades veio a necessidade de garanti-las com uma qualidade mínima sendo este serviço Qualidade de Serviço (*Quality of Service* – QoS).

A Qualidade de Serviço é definida pela União Internacional de Telecomunicações (*International Telecommunication Union* – ITU) como o efeito global da qualidade de funcionamento de um serviço, que determina o grau de satisfação de um usuário para tal serviço [20]. Desta forma, a qualidade de serviço é algo muito subjetivo, pois, depende diretamente do grau de satisfação de cada usuário. Existem tecnologias capazes de medir tais serviços, mas, a satisfação do serviço é avaliada pelo usuário. Por exemplo, ao ver um vídeo no Youtube em 360 pixels um usuário pode qualifica-lo como "Excelente", e já outro usuário pode considerá-lo como "Mediano". Outro exemplo seria uma videoconferência entre dois usuários. Para o mesmo serviço, um usuário poderia expor uma nota três, em uma escala de zero a dez, para o serviço, por conta de atrasos, telas congeladas, falha na voz, dentre outros, enquanto o outro usuário, participante da mesma videoconferência, poderia não achar tais falhas tão significativas e classificar o serviço com nota seis.

Logo, por se tratar de uma medição subjetiva, não é possível criar um algoritmo capaz de qualificar tais serviços com precisão. Mas é, sim, factível, a criação de algoritmos e métricas capazes de garantir um mínimo de qualidade para que serviços dependentes de QoS possam existir.

As principais métricas de QoS são [2]:

º Largura de Banda (*Bandwidth*): Taxa de fluxo de dados suportada pela rede, normalmente medida em *bits* por segundo (bps)

º Latência (*Latency*): Tempo de propagação;

º Atraso (*Delay*): Todo o tempo de atraso entre o transmissor e o receptor, a latência está contida no atraso;

º *Jitter*: Variação do atraso de pacotes sucessivos; Influencia diretamente no QoS para

serviços em tempo real, como voz e vídeo;

º Perda de pacotes (*Packet Loss*): Taxa de pacotes perdidos durante a transmissão, muito usada em tráfegos de tempo real, como voz e vídeo;

º Vazão (*Thoughput*): Taxa média de dados entregues com sucesso para determinado canal de comunicação.

Tais metodologias, métricas e algoritmos são amplamente estudados pela comunidade científica, a fim de garantir a Qualidade de Serviço para vários serviços na Internet. No entanto, tal garantia é uma tarefa muito complicada. Isto ocorre pelo contexto da atual arquitetura da Internet, que acaba impondo diversas restrições.

Implementações voltadas ao suporte de QoS se chocam com a atual estrutura da rede, pelo fato de que, qualquer mecanismo de reserva de banda ou mudança de prioridade nos pacotes, irá afetar o princípio do melhor esforço [3].

Diante de tal impasse, para se garantir o QoS na arquitetura atual, é necessário que haja um acordo entre os vários provedores de serviço, visto que, para alcançar um serviço QoS é preciso garantir certos parâmetros entre dois usuários, que, muitas vezes, acabam passando por diferentes provedores de serviço.

Desta forma, é necessário projetar uma nova arquitetura de rede para lidar com esse problema. Tais mudanças iriam envolver o núcleo da rede, o que faz com que estas medidas não sejam muito bem vistas pelos administradores de rede. A falta de garantia de que tal processo poderia dar certo ocorre por conta da falta de testes com os novos protocolos e componentes em redes reais.

Assim, acaba-se tendo um problema sem solução, pois, novas tecnologias e protocolos não são testados em redes reais, devido à necessidade de mudanças em milhares de equipamentos ligados e, por esse motivo, provedores de serviço, não adquirem confiança necessária para a ampla implementação em ambientes reais [3].

Para solucionar tal problema, o IETF (*Internet Enfineering Task Force*) especifica duas propostas de arquitetura capazes de fornecer QoS na arquitetura de rede atual. A arquitetura de serviços integrados (Intserv) e a arquitetura de serviços diferenciados (Diffserv), como serão vistas nas seções 3.3 e 3.4, respectivamente.

3.2. IMPACTO DOS REQUISITOS DE QOS NA APLICAÇÃO

Como dito anteriormente, a avaliação de um serviço segundo um requisito de QoS é feita por usuários, tornando-a subjetiva. A exigência de qualidade de um serviço aumentou ao longo dos anos [29]. Um usuário que tivesse avaliado a imagem de uma TV nos anos 90 com nota oito em uma escala de zero a dez, provavelmente, se hoje lhe fosse apresentada a mesma TV, com a mesma imagem, ele a classificaria com um nota muito inferior.

A rápida evolução das tecnologias de redes de computadores tornou possível perceber as novas exigências, tanto de usuários, como de aplicações, no que diz respeito à eficiência e melhor utilização de recursos disponíveis [28]. Tais exigências estão ligadas à percepção do ser humano com alguns parâmetros. A seguir serão apresentadas a relação entre retardo, *jitter* e descarte de pacotes e o impacto que causam na percepção humana.

Impacto do Retardo

Latência de áudio fim-a-fim refere-se à latência através do enlace. Em outras palavras, o intervalo de tempo entre, quando o som é enviado em um extremo do enlace e quando é recebido no outro extremo. Esta relação entre a latência do áudio fim-a-fim e o ouvido Humano é mostrada na Tabela 3.1 [25].

Tabela 3.1: Relação de atraso do áudio fim-a-fim e o ouvido Humano [25]

Atraso de Áudio (ms)	Efeito do atraso na percepção da voz humana
> 600	Fala é ininteligível e incoerente
600	Fala é pouco coerente
250	Fala é irritante mas compreensível
100	Diferença entre áudio e fala real é imperceptível
50	Humanos não conseguem distinguir entre áudio e fala real

Em aplicações de interação com transmissão de som em tempo real, o atraso de ida (*one-way-delay*) global precisa ser menor, a fim de garantir aos usuários a impressão de tempo real de resposta [24]. Um valor máximo na ordem de 0.1 a 0.5 segundos é necessário para completar o objetivo [25].

Baseado nestes testes subjetivos, a especificação G.114 da União Internacional de Telecomunicações (*International Telecommunication Unit – ITU*) recomenda menos que 150 milissegundos de atraso de ida fim-a-fim para um tráfego de tempo real com alta qualidade

em telecomunicações. A Tabela 3.2 mostra o tempo limite de acordo com a norma G.114

Tabela 3.2: Limites de tempo de transmissão de ida pela G.114

Tempo de transmissão de ida	Efeito do atraso na percepção da voz humana
0 a 150 ms	Aceitável para a maioria dos usuários
150 a 400 ms	Aceitável, mas possui impacto
400 ms e maior	Inaceitável

Os humanos são muito mais sensíveis a alterações de áudio do que a sinais visuais [25]. Sincronização labial (*lip-synchronization*) é definido como a diferença entre a recepção visual do vídeo e a recepção audível do áudio. Para uma percepção adequada, a sincronização labial não deve exceder 100 milissegundos [25].

Em muitas aplicações, a transmissão de áudio e vídeo é feita ao mesmo tempo [24]. Sabendo-se que o atraso de um *stream* de tempo real deve ser menor que 150 milissegundos, o atraso de uma sequência de vídeo não deve exceder os 250 milissegundos para preservar os 100 milissegundos de limite da sincronização labial [24].

Impacto do Jitter

Jitter é um parâmetro essencial para performance na rede, pretendido para suporte em tempo real de som e imagem. De todos os tipos de informação, som em tempo real é o mais sensível ao *Jitter* [24].

A solução para superar o *jitter* em transmissões de som em tempo real é esperar por um período de tempo suficiente no sistema de recepção, chamado atraso *offset*, antes de enviar os sons recebidos [24].

As aplicações de vídeo têm diferentes necessidades de percepções humanas para a minimização do *jitter* [24]. Por exemplo, a variação de atraso do trânsito da rede não deve exceder 50 ms para HDTV, 100 ms para qualidade *broadcast*, e 400 ms para videoconferência [25].

Impacto do descarte de pacotes

Três considerações devem ser levadas em conta quando definimos os requerimentos aceitáveis pelos humanos com relação a erro. São elas: Técnicas de retransmissão (em serviços de áudio e vídeo em tempo real, técnicas de retransmissão não são apropriadas [24].); Normalmente humanos toleram uma alta taxa de transmissão com erros; Em caso de *bits stream* comprimido, pacotes errados ou perdidos podem invalidar maiores números ou pacotes subsequentes [24].

Além disso, os limites das taxas de erro variam de acordo com as aplicações. Em uma rede fim-a-fim, a taxa de erro de *bits* (*Bit error rate – BER*), antes da possível recuperação de erro entre os sistemas finais, não deve exceder 10^{-6} para HDTV, 10^{-5} para qualidade de TV *broadcast* e 10^{-4} para qualidade de videoconferência [25].

Nas Tabelas 3.3, 3.4, 3.5, são apresentadas, respectivamente, as métricas de QoS para Áudio Conferência, Vídeo sob Demanda Interativo (*Video on Demand – VoD*), e Videoconferência, sendo este último o foco principal deste trabalho.

Na Tabela 3.5 apresentam-se os padrões recomendados pela ITU-T para H.320 [37] (Sistemas de telefonia visuais de banda estreita e de equipamentos terminais), H.323 [38] (Sistemas de comunicação multimídia baseadas em pacotes) e H.324 [39] (Terminal de comunicação multimídia para baixa taxa de bits) sendo o codificador de vídeo H.261 possível de ser usado nestes três padrões ITU-T.

Tabela 3.3: Métricas de QoS para Áudio conferências [24]

Classe do Tráfego	Atributos Tecnológicos	Métricas QoS						
		Timeliness			*Preciseness*			Exatidão
		Tempo de resposta esperado pelos Usuários	Atraso (ms)	Jitter (ms)	Taxa de Dados (Kbps)	Largura de Banda Requerida (Kbps)	Taxa de Perda	Taxa de Erro
Áudio Conferência			< 150	< 400			< 1%	< 1%
Codificadores Padrões								
G.711					64	80		
G.726					40~16	50-22		
G.728					16	22		
G.729					8	11		
G.723.1					6.3/5.3	9 / 8		

GSM FR		13	18		
GSM EFR		12.2	17		

Tabela 3.4: Métricas de QoS para VoD interativo [24]

Classe de Tráfego	Atributos Tecnológicos	Métricas QoS						
		Timeliness			*Preciseness*			Exatidão
		Tempo de resposta esperado pelos usuários	Atraso (ms)	Jitter (ms)	N/a	Banda Requerida (bps)	Taxa de perda	Taxa de erro
Vídeo Interativo sob Demanda	Tempo real e altamente assimétrico	Tempo decorrido: alguns minutos Resposta a função interativa: 2-5 s Lip-synch: < 100 ms	< 150			Local: 1.5M (aplicação típica) Backbone: centenas de mega		
Aplicação Típica		coding						
Qualidade VCR		MPEG-1		< 100		1.2 – 1.5 M/ Um único Vídeo	<0.001%	<0.001%
Qualidade de Vídeo ligeiramente superior a TV broadcast (NTSC ou PAL) com uma taxa de bits de 4M		MPEG-2				4 – 60 M/ um único vídeo		
Taxa de bits requerida para HDTV de 15-34 M				< 50			<0.0001 %	<0.0001 %
Multimedia na web		MPEG-4		< 150		28.8 – 500K/ um único vídeo	<0.001%	<0.001%

Tabela 3.5: Métricas de QoS para Videoconferência [24]

Classe do Tráfego	Atributos Tecnológicos	Métricas QoS						
		Timeliness			*Preciseness*			Exatidão
		Tempo de resposta esperado pelos Usuários	Atraso (ms)	Jitter (ms)	Taxa de Dados (bps)	Largura de Banda Requerida (bps)	Taxa de Perda	Taxa de Erro
Áudio Conferência		Lip-synch: < 100 ms	< 150	< 400			<0.01 %	<0.01 %
Codificadores Padrões								
H.320					64 – 1920K	80K-2M		
H.323					64X K	80X K		
H.324					< 64 K	< 80 K		

Como os humanos são muito mais sensíveis às alterações de sinais de áudio do que de sinais visuais, a referência [24] recomenda para atraso e *jitter* de acordo com o padrão ITU G.114 um atraso de ida menor do que 150 milissegundos para um tráfego de tempo real de alta qualidade e um *jitter* menor que 400 milissegundos para uma conversa de telefone com qualidade [25].

Para videoconferência o [27] recomenda uma perda de pacotes menor que 1%, atraso de ida menor que 150ms e um *jitter* não maior que 30ms. Por outro lado, para *streaming* de vídeo, o mesmo autor sugere uma perda menor que 5% e um atraso não maior que 5 segundos. Para tal caso, não há valores significativos para a variação de atraso.

Tendo estes valores como base para videoconferência ([24] e [27]), entre os valores que diferem da tabela, o trabalho em questão, elegeu o menor, baseado no fato que se tais parâmetros forem alcançados, será possível estar de acordo com ambos os autores. Na Tabela 3.6 é possível ter os valores adotados como parâmetros de QoS de videoconferência para este trabalho.

Tabela 3.6: Parâmetros adotados neste trabalho para QoS de Videoconferência

Atraso (*milissegundos*)	*Jitter* (*milissegundos*)	Perda de pacotes (porcentagem)
< 150 ms	< 30 ms	< 0.01 %

3.3. INTSERV

O modelo de Serviços Integrados (*Integrated Services*) divide o tráfego da Internet em tráfego de melhor esforço e em tráfego de fluxo de dados de aplicativos que necessitam de reserva de banda com QoS garantida [5]. Para que seja possível aplicar o IntServ é necessário que os roteadores garantam um QoS apropriado para cada fluxo de dados, como pode ser observado na Figura 3.1

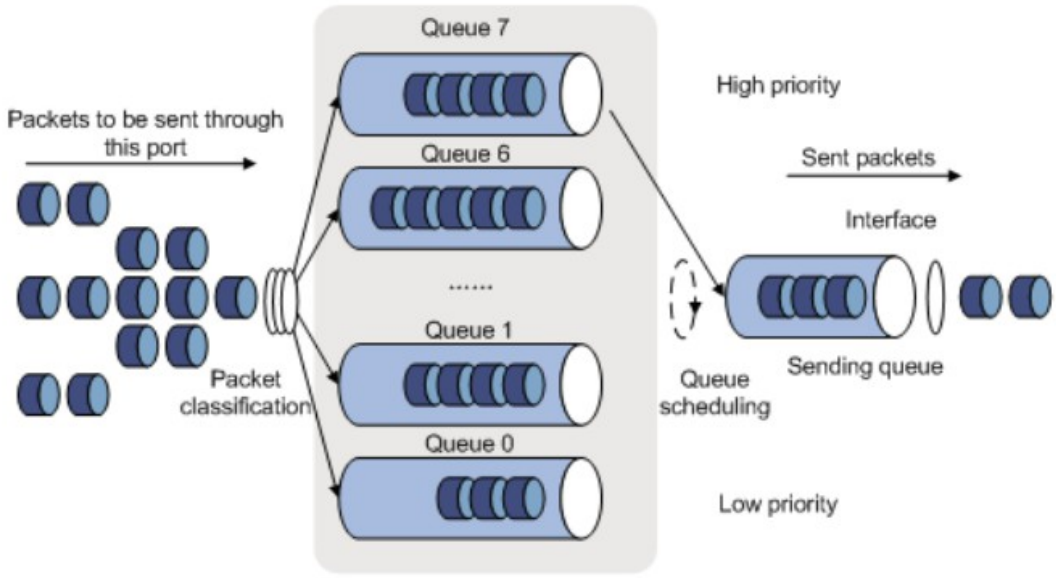

Figura 3.1: Funcionamento do IntServ [5].

A ideia básica do IntServ é garantir o QoS fim-a-fim. Ou seja, garantir que a qualidade necessária para o serviço seja oferecida como estabelecida na configuração original. O serviço é apenas estabelecido caso, seja possível alocar os recursos necessários entre os usuários.

Esta comunicação e preparação para estabelecer o serviço é feita através do Protocolo RSVP (*Resource Reservation Protocol*) e é por meio deste que será definida nesta "conversa" a alocação dos recursos da rede a fim de garantir o serviço. É por meio do protocolo RSVP que esta alocação é feita.

Este sistema de alocação de recursos fim-a-fim apresenta alguns impasses, principalmente, para redes de médio e grande porte, pois os roteadores precisam guardar as tabelas de fluxos completas para o serviço, o que exigiria um enorme espaço de

armazenamento, gerando sobrecarga de processamento para os roteadores [5]. Outro ponto fraco é a necessidade de que todos os equipamentos, alocados para o serviço, precisam suportar o mecanismo citado.

3.4. DIFFSERV

O modelo de Serviços Diferenciados (*Differentiated Service*) tem como objetivo, definir um pequeno conjunto de mecanismos que possam ser implementados nos nós da rede e que suportem uma grande variedade de serviços e aplicativos [5].

Para o DiffServ os tráfegos de fluxos individuais são agrupados em classes, baseados no comportamento do fluxo. Diferente do IntServ, nessa abordagem, os recursos da rede são alocados para a classe de fluxos, ao invés de para cada fluxo individual e não mais existe a necessidade de estados por fluxo e nem sinalização por salto [21]. O objetivo do DiffServ é prover a capacidade de manipular diferentes classes de tráfego de modos diferentes dentro da Internet [22].

A identificação dos pacotes, para diferentes prioridades, é feita pelo o campo ToS (*Type of Service*) ou DS (*Differentiated Services*) no cabeçalho IPv4. Este campo é composto por oito *bits,* sendo dois *bits* para IP ECN (*Explicit Congestion Notificaion*) e os outros seis *bits*, para campo denominado DS *field,* assumindo valores entre 0 e 63. As Tabela 3.7 e 3.8 representam a correlação "padrão" entre o ToS, CoS (*Class of Service*) e DSCP (*Differentiated Services Code Point*) e DS *field*, respectivamente.

Tabela 3.7: Correlação "padrão" entre ToS, CoS (Class fo Service) e DSCP (Differentiated Services Code Point).

ToS 0	CoS 0	DSCP 0-7
ToS 1	CoS 1	DSCP 8-15
ToS 2	CoS 2	DSCP 16-23
ToS 3	CoS 3	DSCP 24-31
ToS 4	CoS 4	DSCP 32-39
ToS 5	CoS 5	DSCP 40-47
ToS 6	CoS 6	DSCP 48-53
ToS 7	CoS 7	DSCP 54-63

Tabela 3.8: Campo ToS / DS byte. [5]

<---------- DS Field ---------->

DSCP – 6 bits	ECN – 2 bits

O serviço DiffServ é definido através de um Acordo de Nível de Serviço (*Service Level Agreement* - SLA) entre o cliente e o provedor de serviços. Nele são especificados o serviço de encaminhamento e a classificação de tráfego que o cliente deve receber, além de determinar as garantias mínimas de QoS à aplicação cliente [5].

A partir deste SLA é possível analisar se o serviço especificado está sendo prestado. Esta verificação é feita através da troca de pacotes entre os domínios de DiffServ, pois, durante esta troca, eles são examinados nos roteadores de borda e consequentemente se o SLA esta sendo mantido.

3.5. RESUMO CONCLUSIVO

A Qualidade de Serviço está presente em várias aplicações que dependem de um tratamento diferenciado na rede. Diversos serviços não sofrem grandes consequências com a perda de pacote, atraso (*delay*), variação de atraso (*jitter*) e funcionam perfeitamente na atual arquitetura da rede, como dito anteriormente, baseada no melhor esforço (*Best Effort*). Mas, esta mesma arquitetura gera um impasse para prover serviços que precisam de garantias, como áudio conferência, videoconferência e outros. Esta questão, como mencionada no começo do capítulo, existe pelo fato de que qualquer mecanismo de reserva de banda ou mudança de prioridade nos pacotes irá afetar o princípio do melhor esforço. Para solucionar tal impasse, o IETF especifica duas propostas de arquitetura capazes de fornecer QoS na arquitetura atual. São elas: IntServ e DiffServ. Baseando-se nestas propostas, os capítulos posteriores irão avaliar os resultados e verificar se estão dentro dos parâmetros de QoS exigidos para o serviço de videoconferência.

4. PROPOSTA DO TRABALHO

4.1. INTRODUÇÃO

Como mencionado no início deste trabalho, as redes de computadores se tornaram parte da infraestrutura crítica da nossa sociedade, participando do cotidiano de bilhões de pessoas. O sucesso das redes de computadores se deve, em grande parte, à simplicidade de seu núcleo.

Na arquitetura atual, a inteligência da rede está localizada nos sistemas de borda, enquanto o núcleo é simples e transparente. Embora essa abordagem tenha tido sucesso, viabilizando a Internet, também é a razão para sua inflexibilidade e incapacidade de atender às necessidades das novas aplicações que deverão surgir no futuro próximo [3].

A inflexibilidade da arquitetura das redes de computadores também traz um desafio para os pesquisadores da área, pois, seus experimentos dificilmente podem ser avaliados em redes reais, visto que, novas propostas, como apresentado no capítulo 2, em geral, requerem mudanças em todos os milhares de equipamentos de rede instalados [3], e geralmente os experimentos precisam ser feitos conjuntamente com tráfego real de produção [2]. Sendo assim, em geral, testes de novas tecnologias são realizados em simuladores de rede, o que implica em uma simplificação da realidade [3].

O paradigma de Redes Definidas por Software (Software Defined Networks – SDN), e o protocolo OpenFlow, oferecem um caminho para a implementação de uma arquitetura de rede programável, capaz de ser implementada de forma gradativa em redes de produção, que oferece a possibilidade de separação dos mecanismos de controle dos diversos fluxos de tráfego atendidos, de forma que, por exemplo, um experimento científico possa ser executado em uma rede real (adaptada para o SDN) sem interferir em seu funcionamento [3].

Com o aumento de usuários também houve um aumento dos serviços prestados na Internet, como aumento de tráfego de vídeo e serviços que dependem da garantia de Qualidade de Serviço (QoS). Um exemplo disto é que em 2012, metade do tráfego da Internet era tráfego de vídeo [1]. Paralelo a este crescimento, prospera também a exigência em relação à Qualidade de Serviços [29] oferecidos, fomentando assim, a busca da comunidade científica por melhorias nas metodologias de provisão de Qualidade de Serviço.

Diante de tal cenário, as Redes Definidas por Software e o OpenFlow, que têm grande foco em redes corporativas, a fim de facilitar a administração e garantir mais serviços, como o QoS, para tais redes, motivaram a proposta deste trabalho a avaliar a Qualidade de Serviço de videoconferência com o Paradigma de Redes Definidas por Software, para uma rede

específica. Esta rede, específica, em questão é a rede metropolitana (MAN) da UnB, a REDUnB.

Objetivo

A atual arquitetura da REDUnb não dispõe de equipamentos capazes de prover o serviço SDN, os quais serão adquiridos. Por isso, este trabalho tem como objetivo simular um cenário real da REDUnB com parâmetros reais da mesma, a fim de garantir uma avaliação que possibilite, caso necessário, a implementação de QoS com o Paradigma de Redes Definidas por Software na rede real no futuro.

O trabalho foi realizado através do simulador de rede SDN *Mininet* (será discutido de forma mais aprofundada na seção 6.2). Com esta ferramenta é possível simular, um ramo da atual arquitetura da rede da Universidade de Brasília (UnB) com parâmetros do tráfego real da rede.

O simulador *Mininet* permite a migração da rede simulada para a rede física, ou seja, após ter a rede avaliada e estudada, é possível realizar a migração da configuração da rede virtualizada para a rede física.

O objetivo da avaliação de QoS para um ramo da REDUnB com o Paradigma de Redes Definidas por Software é fornecer uma simulação de videoconferência com dados reais, e por meio desta, verificar se existe a necessidade de provisão de Qualidade de Serviço de videoconferência para a rede.

Metodologia

Como a rede da UnB é muito ampla e dispõe de uma diversidade de equipamentos, torna-se muito complexa a simulação de toda a rede em apenas uma instância do simulador SDN *Mininet.* Pois, precisaríamos de um computador com um processamento muito potente. O Centro de Informática da UnB (CPD) possui um laboratório onde tivemos acesso a um PC *Blade* da Dell. A idéia inicial era fazer o *upload* da topologia completa da UnB, apresentada na Figura 4.1, e realizar toda a simulação nesta *Blade*. No entanto, a mesma não suportou, e o objetivo passou a ser a simulação de um ramo da topologia. O ramo escolhido foi o do *switch core* conectado aos outros 23 *switches* de distribuição (denominados "hosts") sendo um destes 23 *switches* o *switch* de distribuição da Faculdade de Tecnologia (FT) onde se encontra o Departamento de Engenharia Elétrica, como pode ser visto na Figura 4.1 em hachurado, e de forma ampliada na Figura 4.2

Para o cenário utilizado para simulação, não foi necessário o uso da *Blade,* pois a mesma foi suportada em uma máquina virtual em um Macbook. Todas as especificações dos equipamentos e máquinas utilizados para simulação podem ser vistas abaixo:

Configuração computador nativo:

º MacBook Pro (Retina) – OS X Yosemite versão 10.10.3

º Processador: 2.3 Ghz Inter Core i7

º Memória: 8GB 1600 Mhz DDR 3

º Gráficos: NVIDIA GeForce GT 650M 1024 MB

Configuração máquina virtual SDNHub:

º Ubuntu 14.04.1 LTS

º Processador: 4 CPUs

º Memória: 4000MB

A fim de buscar uma simulação mais realista possível, os cenários simulados neste trabalho foram montados baseados em dados reais, fornecidos pelo CPD, a respeito do monitoramento de tráfego na rede UnB. Tais informações sobre os cenários montados serão abordadas com mais atenção na seção 6.2.

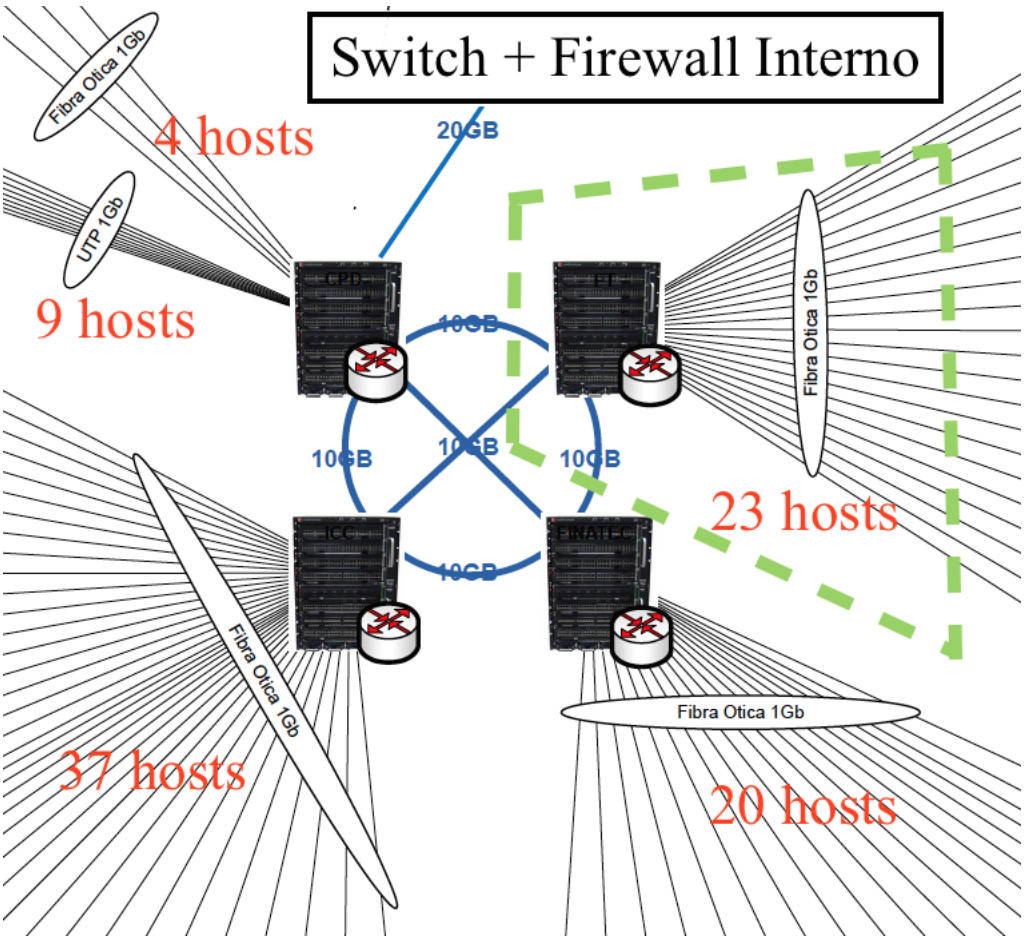

Figura 4.1: Topologia REDUnB com o ramo (hachurado) usado neste trabalho (Host = *switch* de distribuição).

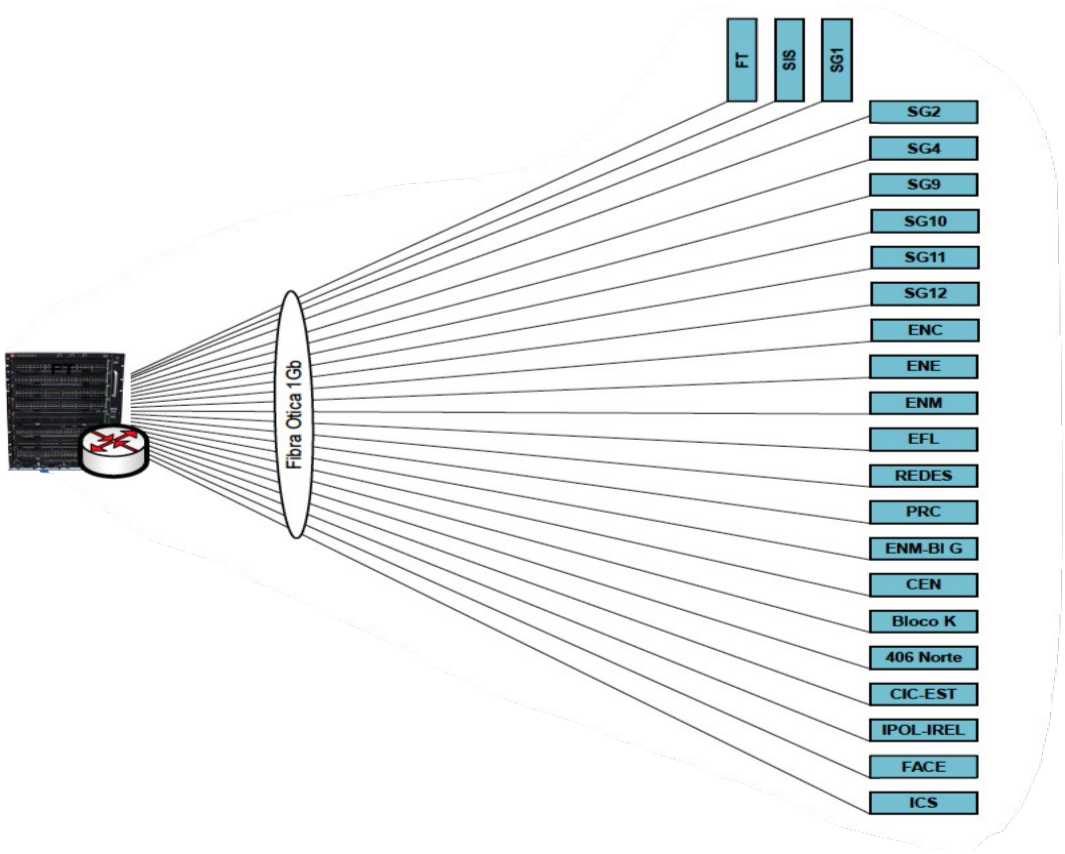

Figura 4.2: Imagem ampliada do ramo utilizado para o trabalho em questão.

Com tais cenários prontos, será simulada a transmissão de um tráfego com características de video do codificador H.261, um codificador com padrões utilizados para videoconferência [30]. A simulação será realizada no cenário proposto, com os dados de tráfego reais obtidos pelo *NAGIOS*. A primeira simulação da REDUnB foi realizada com uma alta carga de tráfego, denominado "Carga Alta" e outro quando a mesma se encontra com uma carga baixa, denominado "Carga Baixa". Mais à frente será explicado como foram obtidos os parâmetros reais da REDUnB e também a divisão da simulação com os devidos parâmetros.

O cenário real da rede utilizado para o trabalho é composto por 1 *switch core* (*Extreme Networks S-Series S8*) e 23 *switches* de distribuição (*C-Series C5*). Dos 23 *hosts* (*switches* de distribuição), 11 enviam o tráfego de vídeo para cada um dos outros 11 *hosts* receptores e um único *host* será o responsável por enviar tráfego *web* para os outros 11 *hosts* que estão recebendo o tráfego de vídeo. Na seção 6.3 será fornecida uma explicação melhor sobre a simulação em questão.

Após tais simulações, os resultados são comparados com os valores apresentados na Tabela 3.6 e verificado se os parâmetros de QoS para videoconferência foram alcançados.

4.2. RESUMO CONCLUSIVO

Neste capítulo foram apresentados o objetivo e a metodologia para o trabalho em questão, tendo como objetivo final a simulação de um ramo real da REDUnb e a avaliação de Qualidade de Serviço para videoconferência com os resultados obtidos.

São realizadas simulações para o cenário proposto com duas cargas diferentes, baseadas nas informações reais enviadas pelo CPD, obtidas pelo *NAGIOS* [31]. É então simulada uma transmissão de um tráfego com características de videoconferência onde 11 *hosts* irão enviar tráfego de vídeo para outros 11 *hosts* que irão recebê-los, um *host* irá enviar tráfego *web* ao mesmo tempo para os 11 *hosts* receptores a fim de simular atividade na rede.

Após a simulação, os resultados são comparados com os parâmetros de QoS estipulados na Tabela 3.6. Diante de tal comparação, serão analisados se os parâmetros QoS para videoconferência foram atingidos.

5. FERRAMENTAS UTILIZADAS NA SIMULAÇÃO

5.1. INTRODUÇÃO

Neste capítulo são abordadas as ferramentas e parâmetros utilizados no desenvolvimento deste trabalho.

Na seção 5.2, é apresentado o gerador de tráfego D-ITG (*Distributed Internet Traffic Generator*). Este foi o gerador utilizado para gerar tanto o tráfego de vídeo como o tráfego *web*. Para concluir todas as ferramentas utilizadas para este trabalho, a seção 5.3 explica a plataforma e o controlador utilizado para realizar a simulação.

5.2. GERADOR DE TRÁFEGO – D-ITG

O programa utilizado para gerar todo o tráfego na simulação é o *Distributed Internet Traffic Generator* (D-ITG). O gerador de tráfego D-ITG é uma plataforma capaz de produzir tráfego IPv4 e IPv6 reproduzindo com precisão a carga de trabalho de aplicativos atuais de Internet. Ao mesmo tempo, o D-ITG é também uma ferramenta de medição de rede capaz de medir as métricas mais comuns de desempenho (exemplo: Vazão, Atraso, J*itter*, Perda de Pacotes) em nível de pacote [33]. No Apêndice B.2. apresentam-se os passos para sua instalação.

D-ITG pode gerar tráfego que segue modelos estocásticos para o tamanho do pacote (*Packet size* – PS) e tempo de saída entre pacotes (*Inter Departure Time* - IDT) que imitam o comportamento do protocolo da camada de aplicação. Especificando a distribuição das variáveis aleatórias IDT e PS, é possível escolher diferentes processos de renovação para a geração de pacotes: usando caracterização e modelagem de resultados da literatura, o D-ITG é capaz de replicar propriedades estatísticas de tráfego de diferentes aplicações bem conhecidas (exemplo: *Telnet*, *VoIP* – G.711, G.723, G.729, Detecção de atividade de voz (*Voice Activity* Detection), *Compressed RTP* – DNS, *network games*) [33].

Na camada de transporte, D-ITG atualmente suporta TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*), SCTP1 (*Stream Transmission Control Protocol*) e DCCP1 (*Datagram Congestion Control Protocol*). Ele também suporta o ICMP (*Internet Control Message Protocol*). Entre os vários recursos descritos abaixo, o modo passivo FTP (*File Transfer Protocol*) também é suportado para realizar experimentos em presença de NAT (*Network Address Tranlation*), e é possível definir o ToS (DS) e campos de cabeçalho IP

(*Internet Protocol*) LLT (*Time to live*) [33].

Componentes

O gerador D-ITG é composto por cinco componentes, sendo eles:

º ITGSend – Responsável por enviar o tráfego. É possível enviar um fluxo comum como múltiplos fluxos através de um arquivo *script,* onde cada linha representa um fluxo.

º ITGRecv – Responsável por receber os múltiplos tráfegos paralelos. Ao executá-lo, o *host* fica em "escuta" esperando pelos tráfegos.

º ITGDec – Responsável por decodificar e analisar os arquivos *log.* ;

º ITGLog – Responsável por receber e coletar as informações *log* enviadas pelo ITGSend e ITGRecv. A Tabela 5.1 exemplifica todos os resultados gerados pelo ITGLog para cada fluxo e para o resultado de todos os fluxos ;

º ITGManager – Representa um exemplo de como usar o D-ITG API para controlar remotamente o ITGSend.

Na Figura 5.1 é mostrada a arquitetura do D-ITG assim como funções de suas componentes.

Tabela 5.1: Resultados armazenados pelo ITGLog para os fluxos [33].

º Relatórios Sintéticos
º Duração Experimento
º Pacotes transferidos
º Bytes de carga útil transferida
º Atraso de ida/ ida e volta (mínimo, máximo, média, desvio padrão)
º Taxa de bits média
º Taxa de pacotes média
º Pacotes descartados
º Pacotes duplicados
º Eventos perdidos
º Tamanho de rajada perdida média
º Primeiro/último número de sequência
º Métricas QoS amostradas em séries temporais
º Taxa de bits [Kbps]

º Atraso de ida / ida e volta [ms]

º Jitter [ms]

º Perda de pacotes [pps]

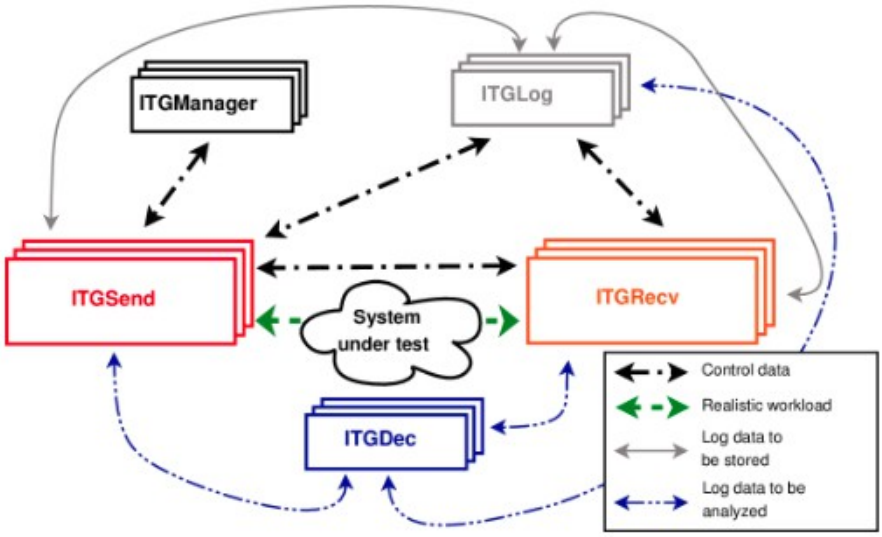

Figura 5.1: Arquitetura do D-ITG [33].

D-ITG *log* consiste em um conjunto de registros (um para cada pacote) incluindo os seguintes campos (Tabela 5.2) :

Tabela 5.2: Conjunto de registros para cada pacotes [33]

Flow	Número do fluxo
Seq	Número de sequência
SrcIP	Endereço IP da fonte
SrcPort	Número da porta da fonte
DestIP	Endereço IP destino
DestPort	Número da porta de destino
txTime	Transmissão carimbo de tempo (*timestamp*)
rxTime	Recepção carimbo de tempo
Size	Tamanho da carga útil do pacote, em bytes

Os valores apresentados no resumo padrão são calculados de acordo com os registros do *log* da seguinte forma:

º O tempo total é computado como a diferença entre *rxTime* do último e do primeiro pacote ;

º O atraso é computado como a diferença entre o *rxTime* e *txTime* para cada pacote ;

º O desvio padrão do atraso (σ) é computado de acordo com a Equação 5.1.

$$\sigma = \sqrt{(1/N)\sum_{i=1}^{N}(d_i - d')^2}$$

Equação 5.1 – Cálculo do desvio padrão do atraso

onde N é o número considerado de pacotes, d_i é o atraso do pacote *i,* e d' é a média de atraso dos pacotes.

º O *jitter* é computado de acordo com a Figura 5.2 e a Equação 5.2, onde S_i e R_i correspondem a *txTime* e *rxTime,* respectivamente.

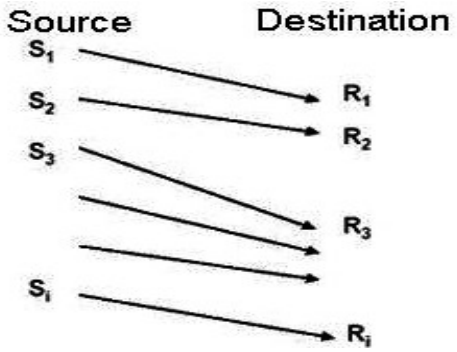

Figura 5.2: Cálculo do jitter
[33].

$$D_i = (R_i - S_i) - (R_{i-1} - S_{i-1}),$$
$$D_i = (R_i - R_{i-1}) - (S_i - S_{i-1}),$$
$$AvgJitter = (\sum_{i}^{n} |D_i|)/n$$

Equação 5.2 – Fórmula para cálculo do *jitter*.

Características do fluxo do D-ITG

D-ITG é capaz de gerar múltiplos fluxos unidirecionais de vários remetentes para vários receptores [33]. Cada um desses fluxos seguem características apresentadas na Tabela 5.3

Tabela 5.3: Características dos vários fluxos possíveis de serem gerados pelo D-ITG [33].

- º Propriedades do nível de fluxo customizáveis
  - º duração
  - º início com atraso
  - º número total de pacotes
  - º número total de KBytes
- º Ferramentas suportadas da Camada 3
  - º protocolos: Ipv4, IPv6
  - º campo de cabeçalhos customizáveis
    - º endereço IP de fonte e destino
    - º interface de origem obrigatória (para dispositivos *multi-homed*)
    - º tempo inicial do TTL
    - º DS byte
  - º NAT transversal: FTP – modo passivo
- º Ferramentas suportadas da Camada 4
  - º protocolos: TCP, UDP, ICMP, DCCP, SCTP
  - º campo de cabeçalhos customizáveis
    - º número das portas de fonte e destino
- º Ferramentas suportadas da Camada 7
  - º Perfis do tamanho do pacote (PS) estocástico pré-definido e tempo de saída entre pacotes (IDT)
    - º Telnet
    - º DNS
    - º Quake 3
    - º CounterStrike (ativo e inativo)
    - º VoIP (G.711, G.729, G.723)
  - º Conteúdo da carga útil: aleatório ou ler de um arquivo
  - º Processo estocástico suportado para PS e IDT
    - º Suporta as distribuições: Uniforme, Constante, Exponencial, Pareto, Cauchy, Poisson, Gamma, Normal, Weibull
    - º Seleção explícita aleatória de semente para replicação do mesmo processo estocástico
    - º Carregar o PS e séries IDT a partir de um arquivo
- º Métricas QoS em nível de pacotes
  - º Taxa de bits
  - º Taxa de pacotes
  - º Atraso de ida (requer sincronização com o relógio)

º Tempo de ida e volta

º Jitter

º Perda de pacotes

Características do tráfego de vídeo

Para o trabalho proposto, serão simulados, ao mesmo tempo, um tráfego de vídeo e um tráfego de fundo (será denominado tráfego de fundo, ou web, o tráfego da rede com características de tráfego de Internet) com características de um tráfego *web* (mais à frente serão exemplificadas as características do tráfego *web*). O tráfego de fundo tem como objetivo simular uma atividade normal da rede durante a simulação. O D-ITG não permite a leitura de arquivos com padrões de vídeo para gerar tal tráfego na rede, mas ele permite a criação de um tráfego com muitas características. Por meio desta possibilidade criou-se um tráfego de vídeo com características reais de um codificador de vídeo.

O tráfego de vídeo foi gerado baseado em um dos codificadores padrões de compressão de vídeo da ITU-T, voltado para videoconferência sendo ele o H.261 [30]. As informações referentes para a criação foram extraídas do artigo [34] em que o autor faz uma simulação de videoconferência comparando vários *Video Enconders*. Os dados foram extraídos da Tabela 5.4.

Tabela 5.4: Quantidades estatísticas das sequências de tamanho de quadros amostrados [34]

Codificador	NV	NVDCT	H.261	H.263	H263+	CellB	BVC
Tempo do experimento (segundos)				3600			
# De Quadros	50113	53336	53937	53453	17921	53749	53855
Taxa média de bit de vídeo (Kbits/s)	182	121	63	54	13	93	92
Taxa média de pacotes	14	15	15	15	5	15	15
Taxa média do tamanho do pacote	1638	1023	527	457	331	779	766
Variação	1589100	678870	174130	24588	401060	407130	467460

($bytes^2$)							
Tamanho mínimo do pacote (bytes)	24	24	78	196	80	77	40
Tamanho máximo do pacote (bytes)	10284	6468	2718	2122	5343	5959	4658

Com estas informações montou-se o tráfego de vídeo da seguinte forma:

º H.261 (*script* D-ITG) (Tabela 5.5)

Tabela 5.5: Script do tráfego de vídeo com características do codificador H.261.

-a 10.0.0.X -t 3600000 -rp 10001 -T UDP -E 15 -u 78 2718

Parâmetros utilizados para criação do tráfego:

-a (*Address*) : Endereço de destino ;

-t (*Time*) : Tempo da geração do fluxo em milissegundos ;

-rp (*Destination Port*) : Porta do destino ;

-T *(Protocol*) : Protocolo utilizado ;

-E (*Exponential Distribution*) : Taxa média de pacotes por segundos (como em sistemas reais, muitas vezes o tempo de chegada entre os pacotes possui distribuição exponencial a mesma foi utilizada para o tempo de envio entre os pacotes);

-u (*Uniform Distribution*) : Tamanho mínimo e máximo do pacotes em *bytes*.

Como não foi informada a métrica a ser utilizada no fluxo (-m {RTTM ou OWDM}), métrica utilizada é a padrão, que é o atraso de ida (OWDM - *One-way delay meter*).

Características do tráfego de Internet

Como mencionado na subseção anterior, o tráfego de fundo (*background*) tem como objetivo tornar a simulação ainda mais real. Ele será gerado paralelamente à transmissão de vídeos com o intuito de não permitir que o tráfego de vídeo seja transmitido de forma "dedicada", em outras palavras, caso seja realizada uma videoconferência entre alguns *hosts*, os outros usuários, não participantes do serviço de vídeo, não irão interromper suas atividades

na Internet, visando não atrapalhar os usuários que estão realizando a videoconferência. A videoconferência entre certos usuários pode ocorrer a qualquer momento do dia, sem a necessidade de saber como está a atividade na rede.

As informações necessárias para recriar um tráfego de fundo com características da Internet e construí-lo no D-ITG foram obtidas do artigo [35]. O artigo [35] traz estas informações sobre as características do tráfego da Internet após um monitoramento de 14 dias em uma estação na Alemanha. A Tabela 5.6 informa as estatísticas medidas durante este período.

Tabela 5.6: Estatísticas medidas durante o monitoramento de 14 dias [35].

Começo da medição	28/06/10
Fim da medição	12/07/10
Número de pacotes	4 946 071 829
Número de fluxos	202 429 622
Quantidade de dados observado	3 297 TB
Número de domicílios (ISP)	Cerca de 600
Velocidade de transporte do enlace	30 Mbps

Após o monitoramento eles constataram, dentro dos 15 protocolos mais comuns (Tabela 5.7), que o tráfego HTTP é o mais usado para transferência de dados com 18.4%. Eles também constataram que a vazão do HTTP é entre 5-6 Mbps [35].

Em um outro artigo, [36], foi realizado um estudo sobre a distribuição de tamanho dos pacotes no tráfego da Internet. Eles constataram que o tamanho dos pacotes atuais, mais comuns, seriam divididos em dois, sendo um com tamanho de 40 *bytes* e outro com 1500 *bytes* (com 40 % e 20% dos pacotes , respectivamente) [36]. No entanto, eles identificaram um forte modo por volta dos 1300 *bytes*.

Tabela 5.7: Protocolos avaliados durante o período de monitoramento [35].

Categoria	Protocolo	Categoria	Protocolo
	HTTP		Flash
HTTP	HTTPS		RTP

	DirectDownloadLink		SHOUTcast
	Jabber	MULTIMEDIA	PPStream
	Yahoo		MPEG
IM	ICQ		Windowsmedia
	MSN		IPSec
	IRC		DHCP
	IMAP		SNMP
MAIL	POP3		NETBIOS
	SMTP	OUTROS	NTP
	Bittorrent		SSH
P2P	Gnutella		DNS
	PANDO		ICMP
		Desconhecido	*Desconhecido*

Com os dados citados com relação a [35-36] tem-se informações suficientes para criar um fluxo no D-ITG com as características de um tráfego da Internet. Com estas informações, o tráfego de fundo foi criado da seguinte forma.

º Vazão do tráfego HTTP

→ 5-6 Mbps = 625K-750K *bytes/s*

º Distribuições mais comuns dos tamanhos de pacotes no tráfego da Internet

→ 40 – 1300 – 1500 *bytes*

A título de facilitar as contas, foi adotada a vazão de 750K *bytes* para o tráfego HTTP e distribuída igualmente entre os três tamanhos de pacotes. É possível verificar o tráfego de fundo criado para este trabalho na Tabela 5.8.

Tabela 5.8: Especificando as características do tráfego de Internet criado para este trabalho.

Tráfego web criado para este trabalho	250 K *bytes* /s
1 pacote = 40 *bytes*	6250 pacotes / s
1 pacote = 1300 *bytes*	192 pacotes / s
1 pacote = 1500 *bytes*	167 pacotes / s
Vazão total	750 K bytes / s

Com estes dados, é possível recriar o tráfego de fundo pelo D-ITG com os dados informados acima (Tabela 5.9).

º Tráfego de fundo (*script* D-ITG)

Tabela 5.9: Tráfego de Internet utilizado na simulação.

-a 10.0.0.X -t 3600000 -rp 10002 -T TCP -E 6250 -c 40
-a 10.0.0.X -t 3600000 -rp 10002 -T TCP -E 192 -c 1300
-a 10.0.0.X -t 3600000 -rp 10002 -T TCP -E 167 -c 1500

Desta forma, os três fluxos possuem características muito próximas, diferindo apenas na taxa média de pacotes enviados por segundo com distribuição exponencial e no tamanho de cada pacote. Cada linha, representa um fluxo que envia 250k *bytes*/s, que ao serem somadas, garantem a taxa representante de um tráfego HTTP (750 kB/s) [35].

Gráficos

O próprio D-ITG oferece uma ferramenta capaz de traçar os gráficos baseados em arquivos .dat. A ferramenta ITGDec é capaz de ler os arquivos *log* e gerar arquivos .dat e, através da ferramenta ITGPlot é possível ler estes arquivos .dat e gerar os gráficos. ITGPlot é um *script* Octave [23] que permite gerar gráficos a partir da informações contidas nos arquivos .dat. Os gráficos presentes neste trabalho foram obtidos por meio desta ferramenta ITGPlot.

5.3. SDNHUB + OPENDAYLIGHT

Como apresentado na seção 2.5 o SDNHub é uma plataforma totalmente voltada para SDN. A ideia principal, é fornecer um sistema operacional que tenha total suporte para as Redes Definidas por Software, trazendo as principais ferramentas em uma só máquina.

As principais ferramentas que a máquina virtual SDNHub traz instalada são as seguintes :

º Ubuntu 14.04.1 LTS

º Mininet 2.1.0

º Eclipse Kepler

º Controladores : OpenDaylight , POX, Ryu, ONOS, Floodlight, Floodlight-plus-of1.3 .

O código do controlador OpenDaylight permite que ele funcione de formas diferentes, como por exemplo, um *Hub*, estado padrão em que vem o controlador. Após seguir o tutorial oferecido pelo site oficial do SDNHub [16] é possível ativar o controlador para que ele funcione como um *switch* de aprendizagem. Este foi o estado utilizado para a realização das simulações.

Sem o controlador, o *switch* não possui informação sobre como tratar os fluxos e por isso não é possível realizar o comando *ping* entre os outros *hosts*. A Figura 5.3 exemplifica isto, mostrando a situação em que o cenário é inicializado pelo *Mininet* (terminal "Mininet" a esquerda) mas o controlador (terminal "OpenDaylight Controller" a direita) está desligado. Como o controlador está desligado o *Mininet* informa que não é possível conectar o *switch* ao controlador com IP 127.0.0.1 na porta 6633 (hachurado vermelho) e ainda nesta situação tenta-se executar o comando *ping* entre o host .1 e .2 e tem como retorno a mensagem de que host destino está inalcançável (hachurado vermelho).

Ao ligar o controlador no modo "*switch* de aprendizagem" ele repassa esta informação de como o *switch* deve tratar os fluxos da rede. É possível verificar, na Figura 5.4, pela interface *web* do controlador, que o *switch* não aprendeu nenhum *host*. Na Figura 5.5, é possível verificar que o controlador foi ligado e agora o *switch* recebe as informações de como tratar os fluxos, tornando assim os *hosts* alcançáveis.

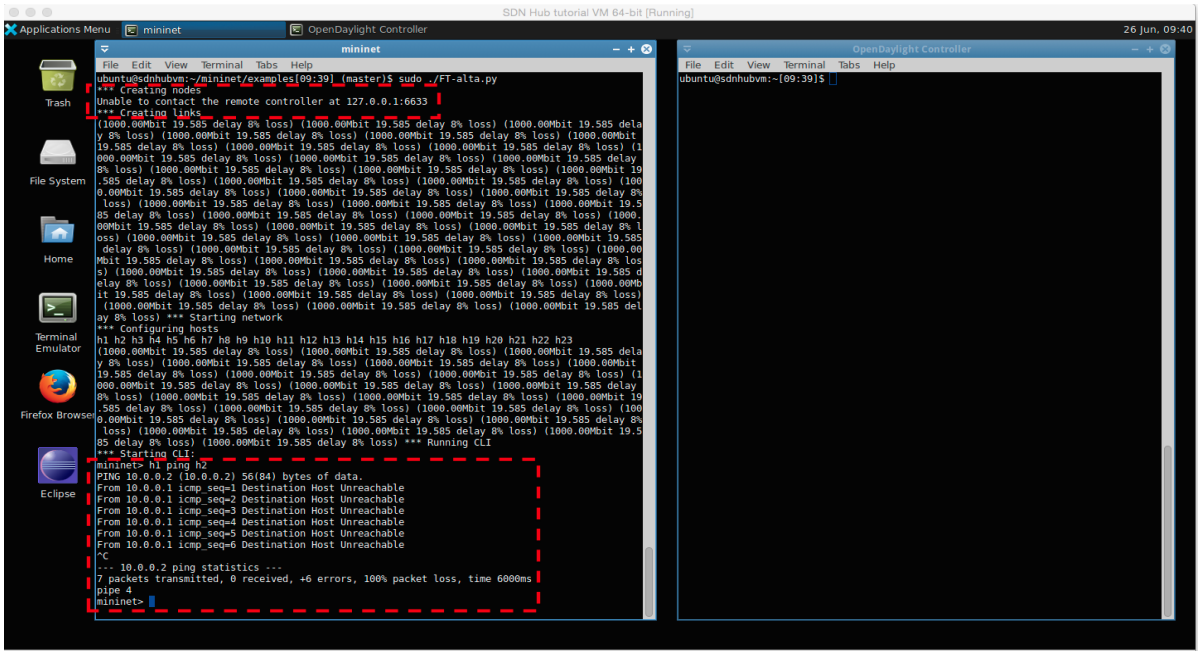

Figura 5.3: Controlador desligado. Switch não recebe informação e hosts ficam inalcançáveis.

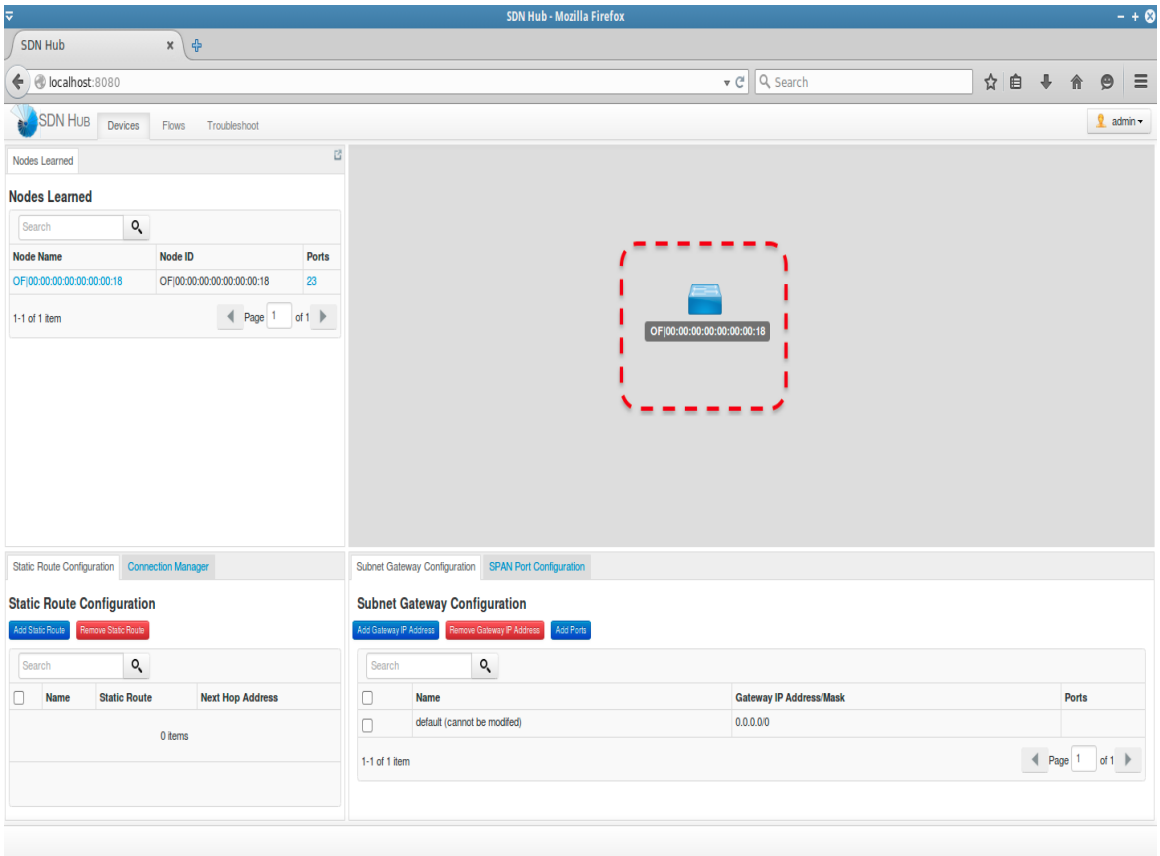

Figura 5.4: Interface web do controlador em estado inicial. Nenhum host aprendido.

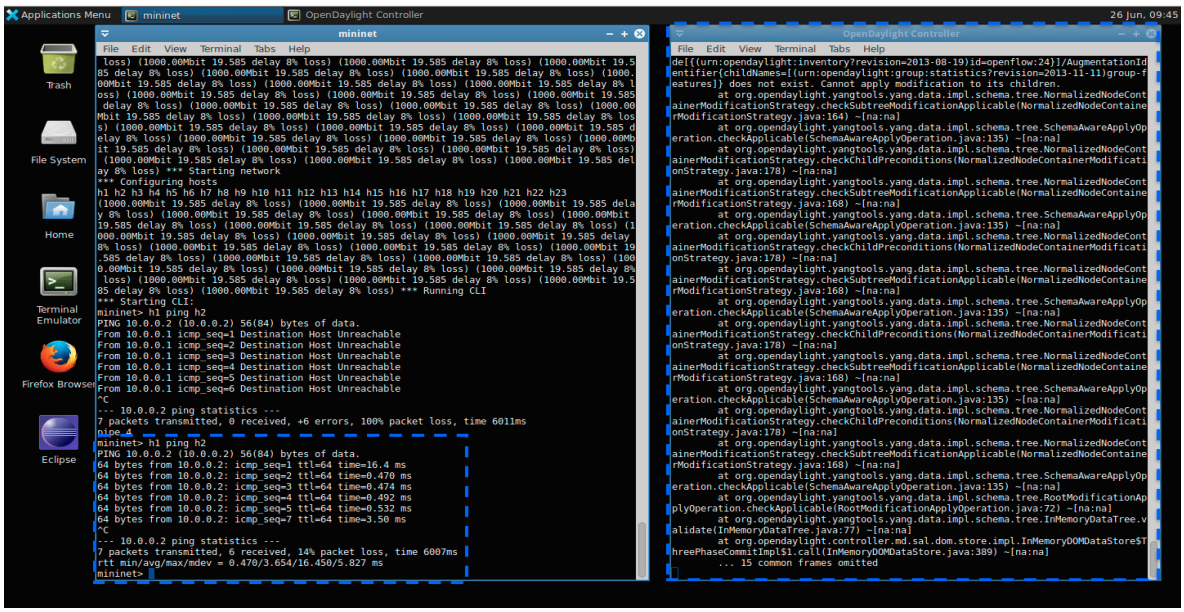

Figura 5.5: Controlador inicializado, switch recebe informação e hosts ficam alcançáveis.

As Figuras 5.6 e 5.7, mostram , respectivamente, a relação entre o resultado da execução do comando *ping* realizado e os *hosts* que foram aprendidos.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=8.60 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.01 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=8.29 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 3 received, 57% packet loss, time 6036ms
rtt min/avg/max/mdev = 6.018/7.642/8.609/1.155 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=10.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.942 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.514 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 3010ms
rtt min/avg/max/mdev = 0.514/3.873/10.164/4.451 ms
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=3.36 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.471 ms
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 2 received, 50% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.471/1.917/3.364/1.447 ms
mininet> h1 ping h20
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_seq=1 ttl=64 time=5.04 ms
64 bytes from 10.0.0.20: icmp_seq=3 ttl=64 time=0.539 ms
64 bytes from 10.0.0.20: icmp_seq=5 ttl=64 time=0.501 ms
^C
--- 10.0.0.20 ping statistics ---
5 packets transmitted, 3 received, 40% packet loss, time 4010ms
rtt min/avg/max/mdev = 0.501/2.026/5.040/2.131 ms
mininet> h10 ping h18
PING 10.0.0.18 (10.0.0.18) 56(84) bytes of data.
64 bytes from 10.0.0.18: icmp_seq=1 ttl=64 time=14.3 ms
64 bytes from 10.0.0.18: icmp_seq=2 ttl=64 time=5.56 ms
64 bytes from 10.0.0.18: icmp_seq=3 ttl=64 time=5.67 ms
^C
--- 10.0.0.18 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 5.565/8.527/14.347/4.116 ms
mininet>
```

Figura 5.6: Execução do comando ping nos hosts h1, h3, h4, h10, h18 e h20.

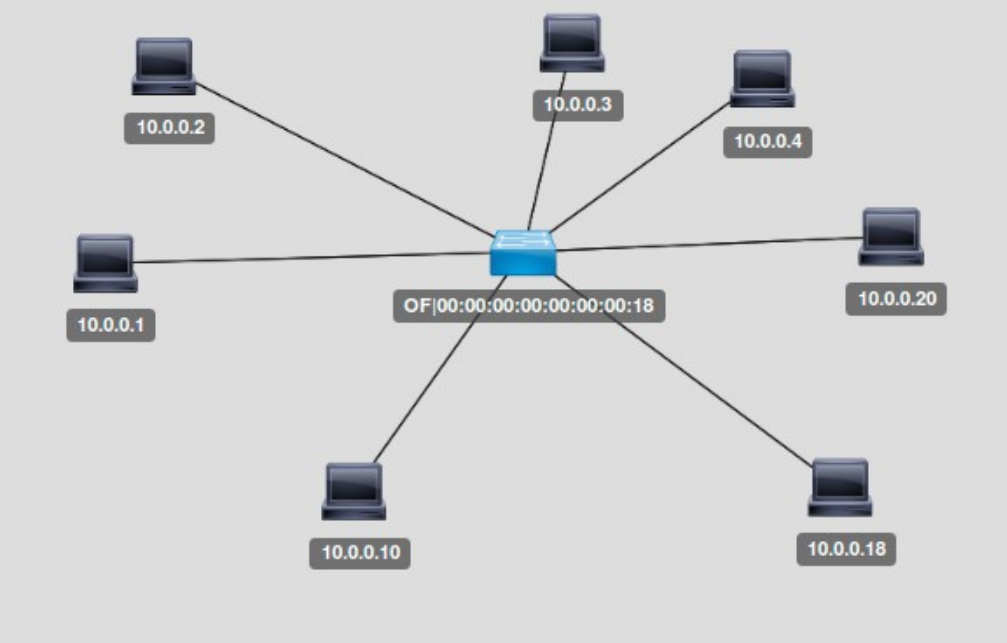

Figura 5.7: Interface web do controlador OpenDaylight com os hosts h1, h2, h3, h4, h10, h18 e h20 aprendidos.

As Figuras 5.8 e 5.9 mostram a execução do comando "*pingall"* no *Mininet* e a interface *web* com todos os *hosts* da rede aprendidos, respectivamente.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X h5 X X h8 h9 h10 h11 X h13 X h15 X h17 X X h20 h21 h22 X
h2 -> X X h4 h5 h6 h7 h8 h9 h10 h11 h12 X h14 h15 h16 X h18 X h20 h21 h22 X
h3 -> h1 h2 h4 h5 h6 h7 h8 X X h11 h12 h13 h14 X X X X h19 h20 h21 h22 h23
h4 -> h1 h2 h3 h5 X h7 X X h10 h11 h12 h13 h14 h15 X h17 h18 h19 h20 h21 X h23
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 X X X h14 ^C
Interrupt
mininet>
```
Figura 5.8: Comando pingall executado.

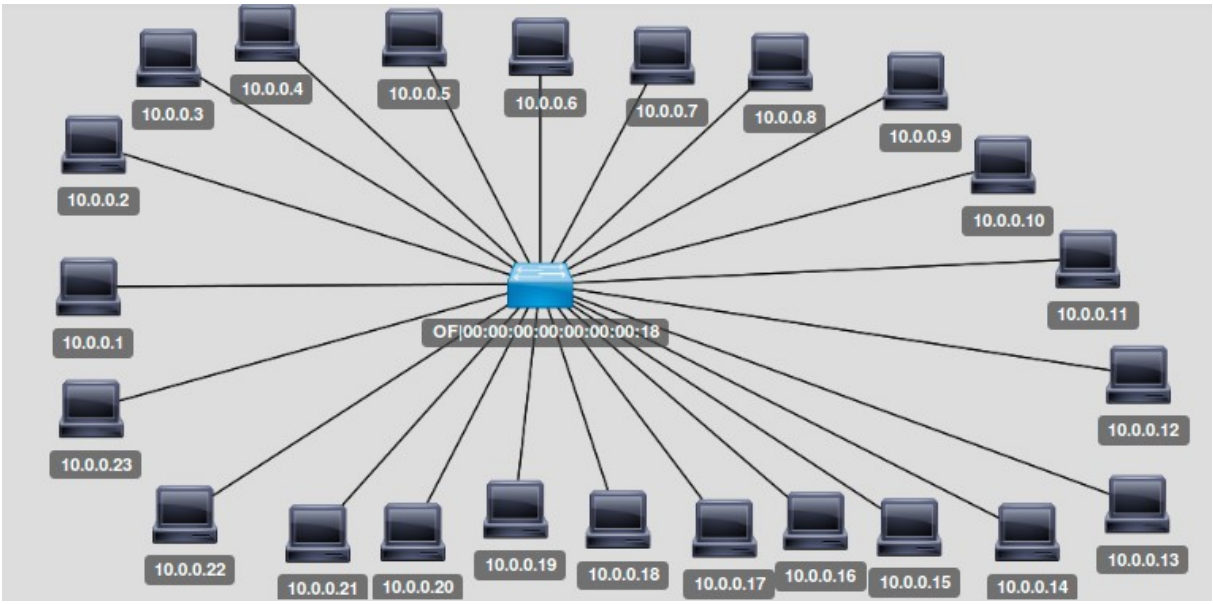

Figura 5.9: Interface web do controlador OpenDaylight com todos os hosts da rede aprendidos.

5.4. RESUMO CONCLUSIVO

Foram apresentadas as ferramentas utilizadas neste trabalho. Primeiramente, sobre o gerador de tráfego, D-ITG, responsável por gerar todos os tráfegos da simulação (tanto o de vídeo como o de fundo) com os parâmetros propostos. Foi explicado também como foram definidos e montados os parâmetros dos tráfegos.

Posteriormente abordou-se a plataforma SDNHub, uma máquina virtual totalmente voltada para SDN a qual foi utilizada neste trabalho. Em seguida, o controlador OpenDaylight, que para a simulação funcionou no estado de *Switch* de Aprendizagem (*Learning Switch*).

6. AVALIAÇÃO DE DESEMPENHO

6.1. INTRODUÇÃO

Neste capítulo são abordadas todas as ferramentas e parâmetros utilizados no desenvolvimento deste trabalho, e também as informações sobre as simulações e análise dos resultados obtidos.

Na seção 6.2, é apresentada a ferramenta *Mininet*, responsável pela simulação do cenário. Nela serão explicados como foram criados os cenários "Carga Alta" e "Carga Baixa" além de apresentar a plataforma "Ramon Fontes"[18] que possibilita a criação de cenários *Mininet* com uma interface gráfica.

Na seção 6.3, serão apresentadas as simulações e seus resultados.

6.2. MININET

Os parâmetros utilizados na geração do tráfego foram baseados em medições realizadas pelo *NAGIOS* por uma semana na REDUnB do *switch core S8* ligado aos outros 23 *switches* de distribuição (o ramo utilizado no projeto) e fornecidas pelo CPD. Os dados colhidos foram:

º Tempo de atraso médio de ida e volta – RTA (*Round-trip Average Delay Time*), calculada sobre dados colhidos de uma semana de monitoramento da REDUnB.

º Perda de pacotes – PL (*Packet Loss*), calculada sobre dados colhidos de uma semana de monitoramento da REDUnB.

Baseado em tais dados, decidiu-se simular os dois cenários em questão. Um quando a REDUnB está com baixa atividade na rede, denominada "Carga Baixa" e outra quando a REDUnB está com alta atividade, nomeada de "Carga Alta". Os parâmetros utilizados neste trabalho, para a simulação dos dois cenários, podem ser vistos na Tabela 6.1.

Tabela 6.1: Parâmetros utilizados para simular os cenários com dados reais da REDUnB.

	Atraso – ms (ida)	Perda de Pacotes – %	Largura de banda do enlace (*Switch-Host*)
CARGA BAIXA	2.10285 ms	0.00%	1 000 MB

CARGA ALTA	19.19585 ms	8.00%	1 000 MB

Os dados da Tabela 6.1 foram obtidos do *NAGIOS*, da seguinte forma:

Para o atraso, foram utilizados os dados referentes ao "RTA" sendo o "RTA *Last*" (*Round-trip* 4.2057 ms) para a "Carga Baixa" e "RTA *Max"* (*Round-trip* 38.3917 ms) para a "Carga Alta". Já para a perda de pacotes utilizou-se o "PL *Last*" (0%) para a "Carga Baixa" e o "PL *Max*" (8.0%) para a "Carga Alta".

Como os dados obtidos pelo *NAGIOS* estão com atraso de ida e volta (*Round-Trip*) eles foram divididos por dois a fim de obter o atraso de ida.

Com esses dados, iniciou-se a criação dos cenários com tais características. Os cenários do *Mininet* são feitos na linguagem de programação Python. Os códigos do cenário da "Carga Alta" e "Carga Baixa" podem ser vistos, respectivamente, nos Apêndices A.1 e A.2.

A classe *mininet.link.TCIntf* é a responsável por adicionar os parâmetros aos enlaces do cenário. Na Tabela 6.2 é possível ver os parâmetros que ela permite.

No site oficial do *Mininet* seção *downloads* (http://mininet.org/download/), é possível encontrar um tutorial bem detalhado. A plataforma SDNHub já traz o *Mininet* instalado.

Tabela 6.2: Parâmetros da classe mininet.link.TCIntf [32].

Def mininet.link.TCIntf.config (	Self,	
	Bw =	None,
	Delay =	None,
	Jitter =	None,
	Loss =	None,
	disable_gro =	True,
	Speedup =	0,
	use_hfsc =	False,
	use_tbf =	False,
	latency_ms =	None,
	enable_ecn =	False,
	enable_red =	False,

```
                                    max_queue_size =                None,
                                    params
                                )
```

Ramon Fontes

Esta plataforma, atualmente em desenvolvimento, permite, entre outras funções, a criação de cenários para *Mininet* com a utilização de interface gráfica e a opção para exportar o cenário criado já na linguagem Python, para caso seja selecionado o *Mininet*. Nas Figuras 6.1 e 6.2 é possível observar a interface, assim como as opções de exportação, e o cenário utilizado para este trabalho, respectivamente.

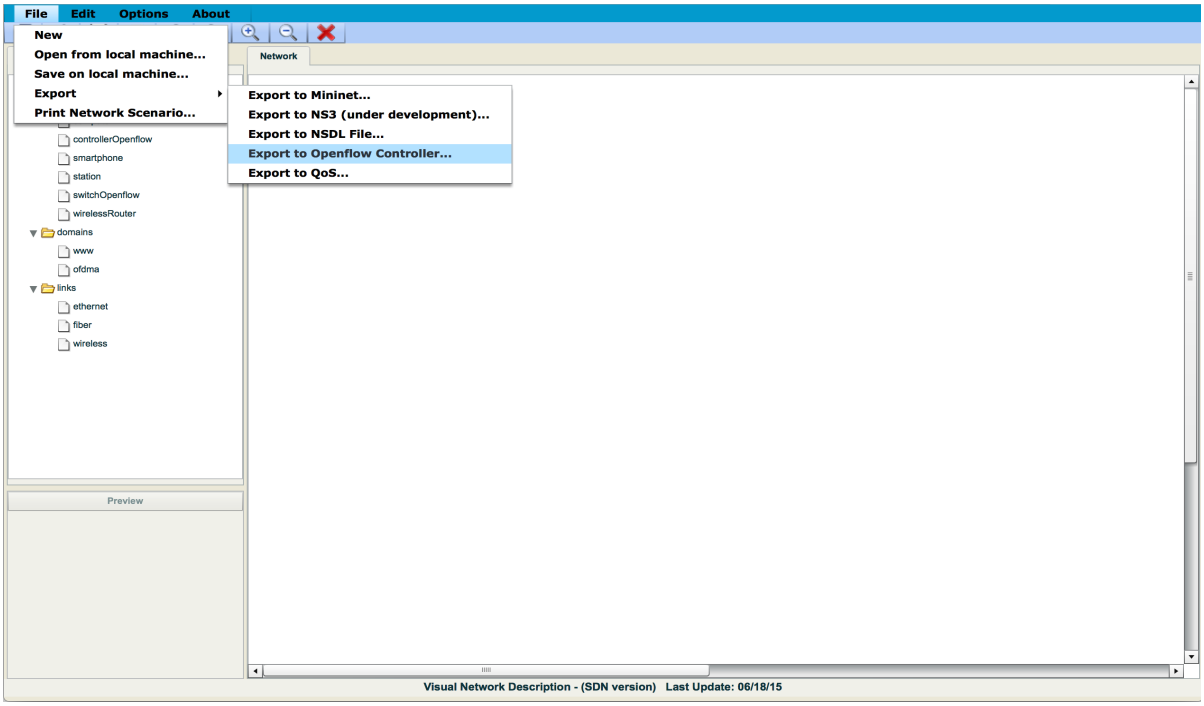

Figura 6.1: Interface gráfica do Ramon Fontes[18].

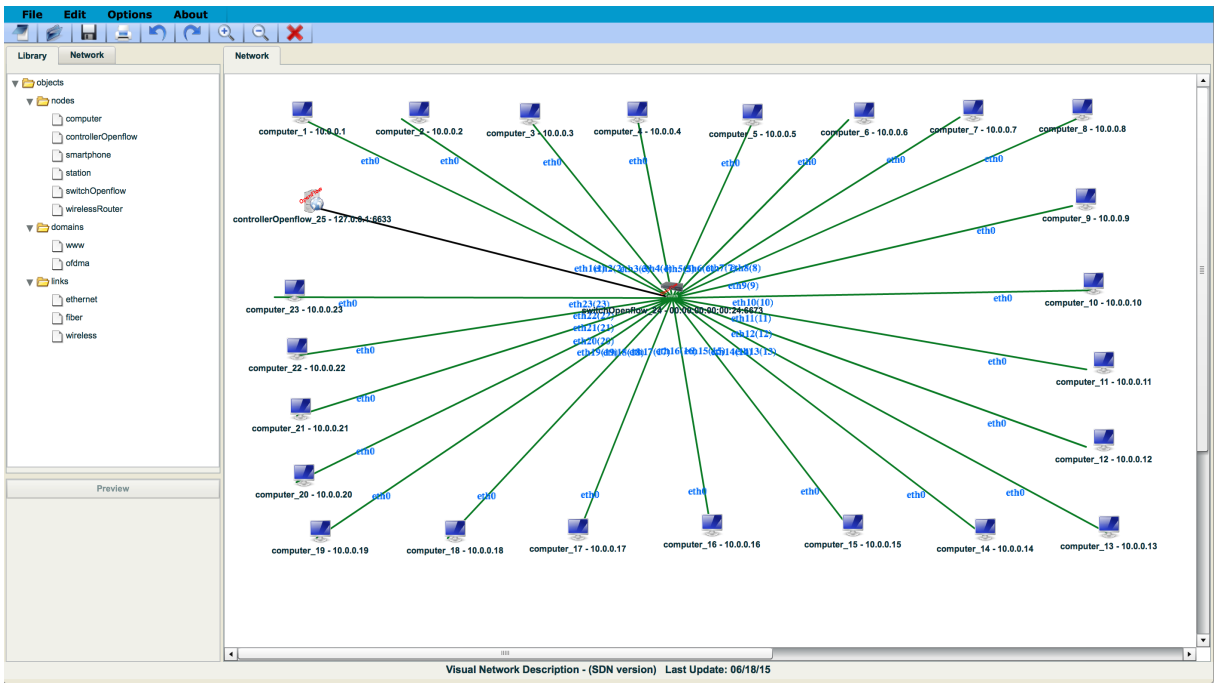

Figura 6.2: Interface gráfica do Ramon Fontes com o cenário utilizado neste trabalho.

6.3. SIMULAÇÃO PROPOSTA E RESULTADOS

Ao longo dos capítulos anteriores explicaram-se as ferramentas que seriam usadas para a simulação, como o cenário utilizado, o tipo de tráfego e os programas.

A simulação foi dividida em duas partes, a primeira com o tráfego de vídeo com características do codificador H.261 para a "Carga Alta" e a segunda para a "Carga Baixa".

Os resultados foram analisados dentro das exigências dos parâmetros QoS para uma videoconferência e avaliados para concluir se existiu, ou não, a necessidade de prover um serviço QoS a fim de garantir que tais parâmetros para videoconferência sejam atingidos.

Características da simulação:

º Tempo de simulação: Uma hora.

º Tráfego de Internet: Discutido na seção 5.3, Tabela 5.5 e 5.9.

º Fluxo dos dados: A Figura 6.3 exemplifica o fluxo dos tráfegos durante a simulação. O cenário é composto por 23 hosts (*switches* de distribuição). Durante a simulação, cada um dos 11 hosts (*h1*, *h2, ...*, *h11*) irá enviar tráfego de vídeo (em azul) para um dos outros 11 hosts (*h12*, *h13*, …, *h22*) que irá receber o tráfego de vídeo. O host h23 será responsável por enviar para os 11 hosts receptores (*h13*, *h14*, …, *h22*) o tráfego de fundo (internet – em vermelho)

simultaneamente aos hosts que enviam o tráfego de vídeo.

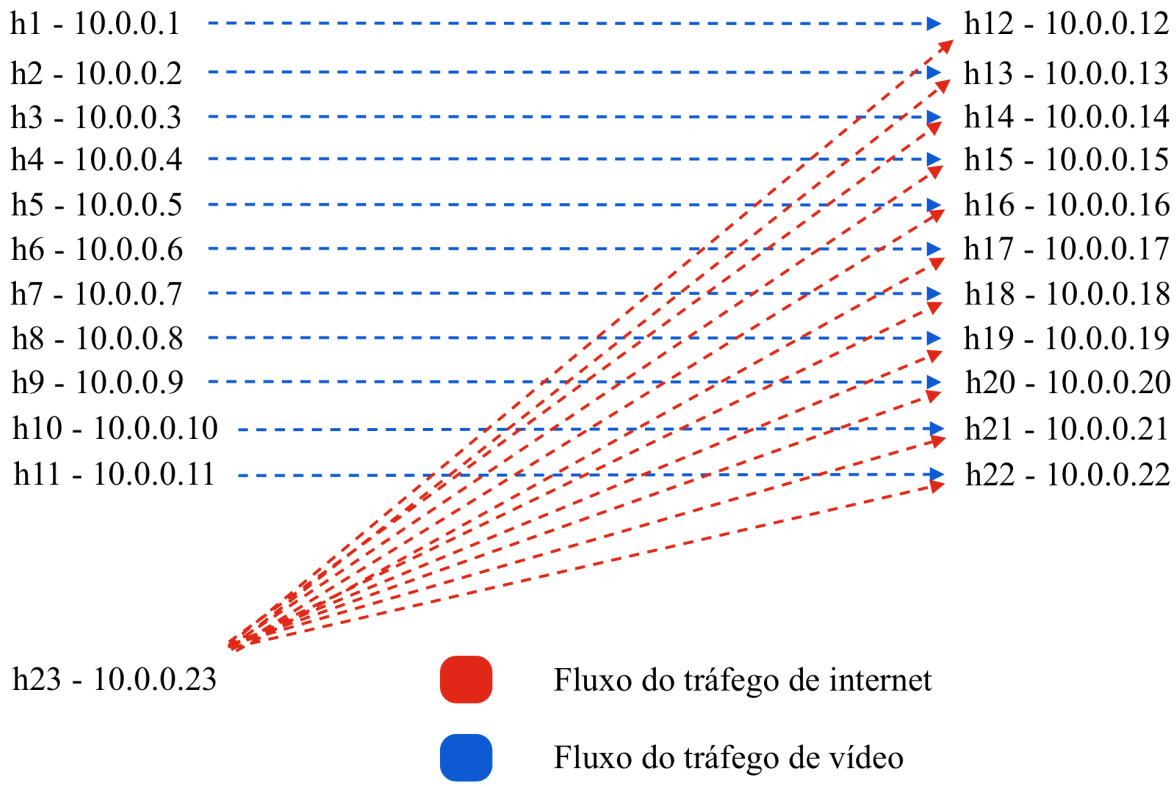

Figura 6.3: Origem e destino dos fluxos de tráfego de internet e vídeo.

H.261 – Carga Alta

O tráfego utilizado nesta simulação foi o da Tabela 5.9 para o tráfego de fundo e Tabela 5.5 para o tráfego de vídeo. Os parâmetros do cenário foram retirados da Tabela 6.1 Os gráficos foram amostrados a cada 36000 ms totalizando 100 amostras por gráfico (tanto para a "Carga Alta", como para "Carga Baixa").

A Figura 6.4 mostra o gráfico do "*h23* – 10.0.0.23" responsável por gerar o tráfego de fundo para todos os receptores. É possível observar na mesma que os fluxos do tráfego de fundo para todos os hosts receptores se encontram na parte debaixo do gráfico e em amarelo (acima) é o resultado da soma de todos os fluxos enviados pelo host *h23*.

Observa-se na Figura 6.4 uma diminuição na taxa do tráfego *web* (soma de todos os fluxos), visto que o mesmo utiliza o protocolo TCP, até que se "estabilize" (por volta dos 700s). Devido à configuração de alta taxa de perda nos enlaces, o tráfego *web* varia muito,

sempre procurando se “encaixar” no perfil da rede através do controle do tamanho da janela, que pode ser dividido, basicamente, em duas fases distintas, chamadas de “partida lenta” e prevenção de congestionamento”.

Por outro lado, o protocolo UDP não possui tal mecanismo, como é possível observar na Figura 6.9, onde os gráficos referentes ao tráfego *web* variam bastante, tentando se adaptar à rede e já o tráfego de vídeo (UDP – em azul), permanece numa faixa constante. Esta faixa constante de *bits* recebidos é um pouco menor que a enviada (Figura 6.5).

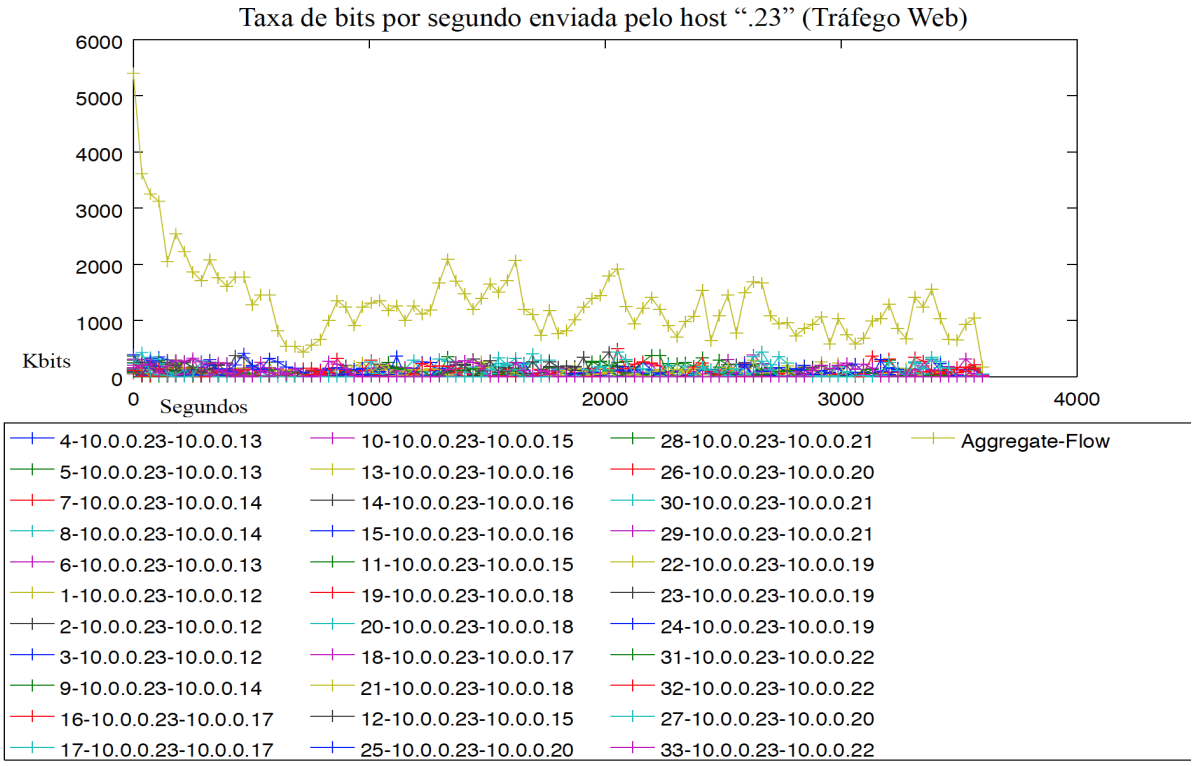

Figura 6.4: Taxa de bits por segundo enviado pelo host .23 de tráfego web (Carga Alta).

A Figura 6.5 mostra a taxa de *bits* enviados pelo *h1*. Por tratar de apenas um fluxo, a taxa de envio do vídeo se sobrepõe ao gráfico da soma dos fluxos. É possível observar sua variação dentro dos parâmetros configurados para o fluxo de vídeo (distribuição exponencial 15 pacotes/s e uma distribuição uniforme do tamanho dos pacotes entre 78 e 2718 bytes.

Observa-se também uma taxa de envio de b/s mínima, um pouco abaixo de 150kb/s e uma máxima, um pouco acima de 180 kb/s. O *host* 1 tem uma taxa de envio média de 167,2 kb/s, enquanto a taxa de *bits*, referente ao tráfego de vídeo recebida do *host* 1 pelo *host* 12, teve uma média de 141,6 Kbps (Figura 6.9)

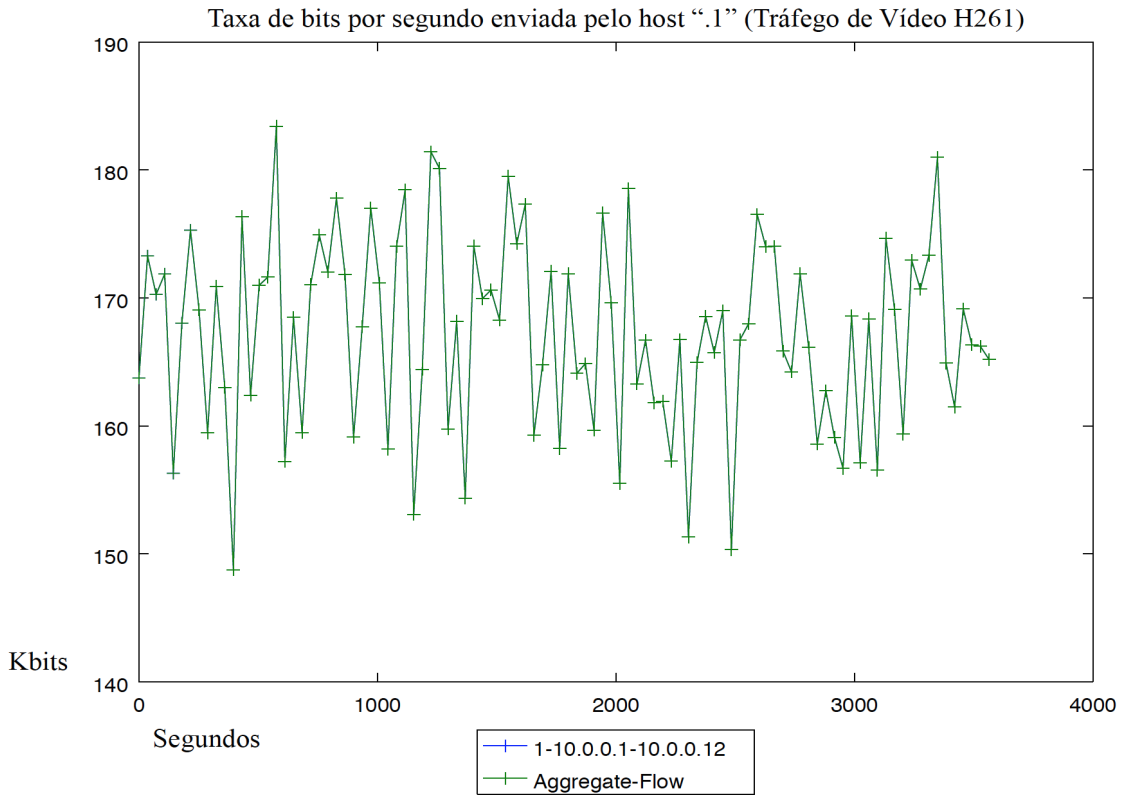

Figura 6.5: Taxa de bits por segundo enviada pelo host .1 de tráfego de vídeo H.261 (Carga Alta).

As Figuras 6.6, 6.7 são referentes aos gráficos de atraso (valor máximo de 32ms para o tráfego de vídeo) e *jitter* (valor médio de 0,13ms para o tráfego de vídeo), respectivamente. Analisando ambas as figuras é possível observar a relação entre o atraso e o *jitter*. Como o *jitter* representa a variação do atraso, onde acontece um maior atraso (por volta dos 2000s – Figura 6.6) também será o ponto em que ocorrerá um maior valor do *jitter*.

Como estes gráficos são referentes a simulação em carga alta, onde existe uma alta taxa de perda de pacotes, observa-se um atraso e *jitter* bem maiores para os fluxos com protocolo TCP pois o mesmo realiza uma troca, diminuindo a sua taxa de envio e aumentando o seu atraso mas garantindo que não exista perda de pacotes.

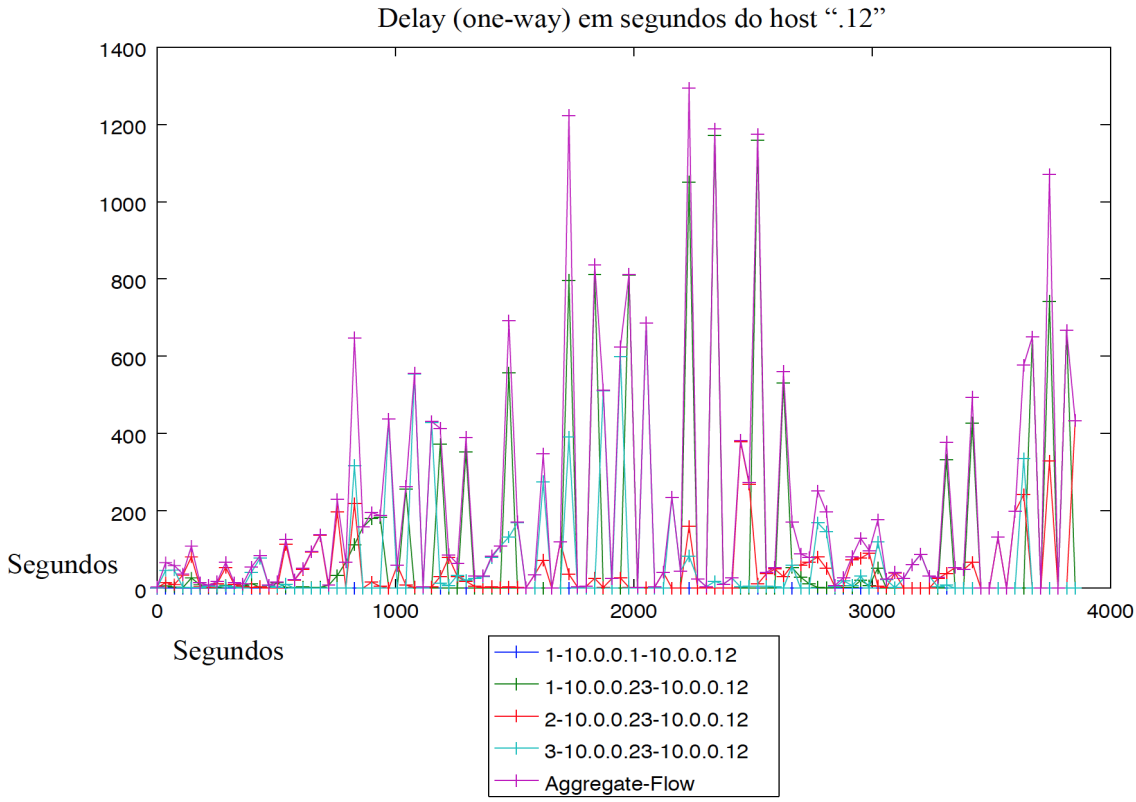

Figura 6.6: Atraso (ida) em segundos no receptor .12 (Carga Alta).

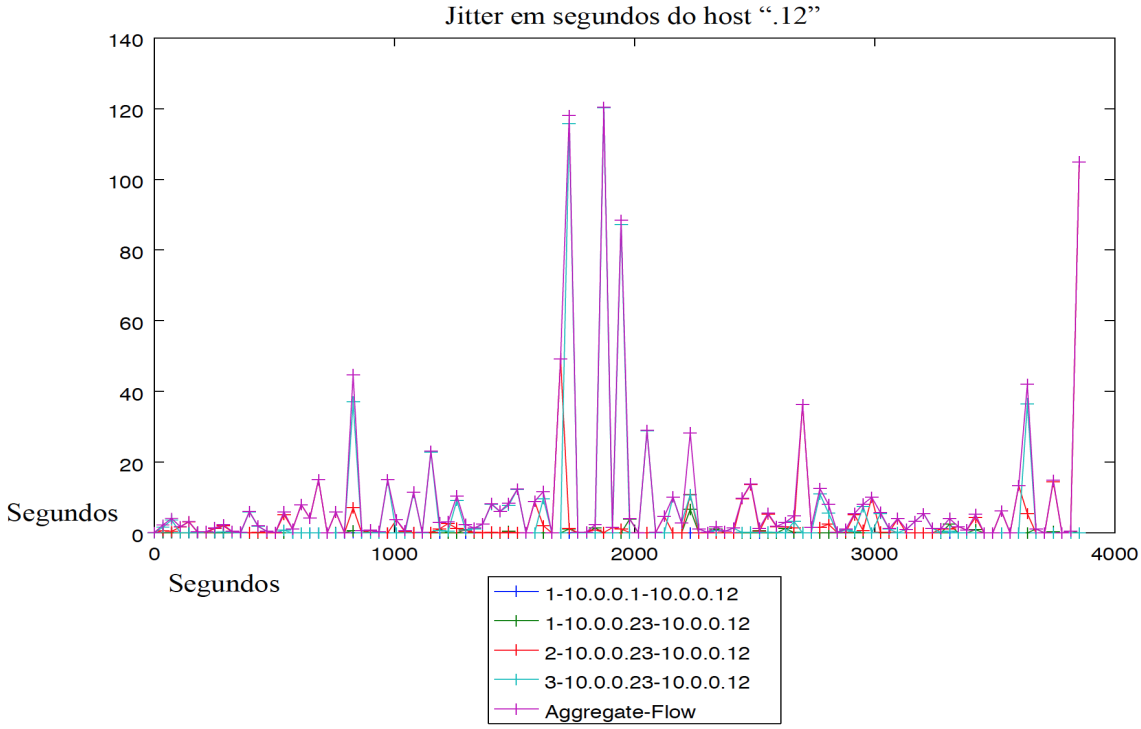

Figura 6.7: Jitter em segundos no receptor .12 (Carga Alta).

As Figuras 6.8 e 6.9 mostram os gráficos, respectivamente, referentes ao receptor *h12* para perda de pacotes e taxa de *bits*.

Na Figura 6.8 observa-se a taxa de perda de pacotes variando entre, aproximadamente, 60 e 100 pacotes por segundo. Para este exemplo, do *host* 1 para o *host* 12, foram enviados, durante a simulação, aproximadamente, 53708 pacotes de vídeo e foram perdidos um total de 8279 pacotes, dando um total de 15.41% de pacotes de vídeo perdido. O gráfico referente a soma total dos fluxos se sobrepõe com a de vídeo, visto que não existiram perdas de pacotes referentes aos fluxos que compõem o tráfego de fundo.

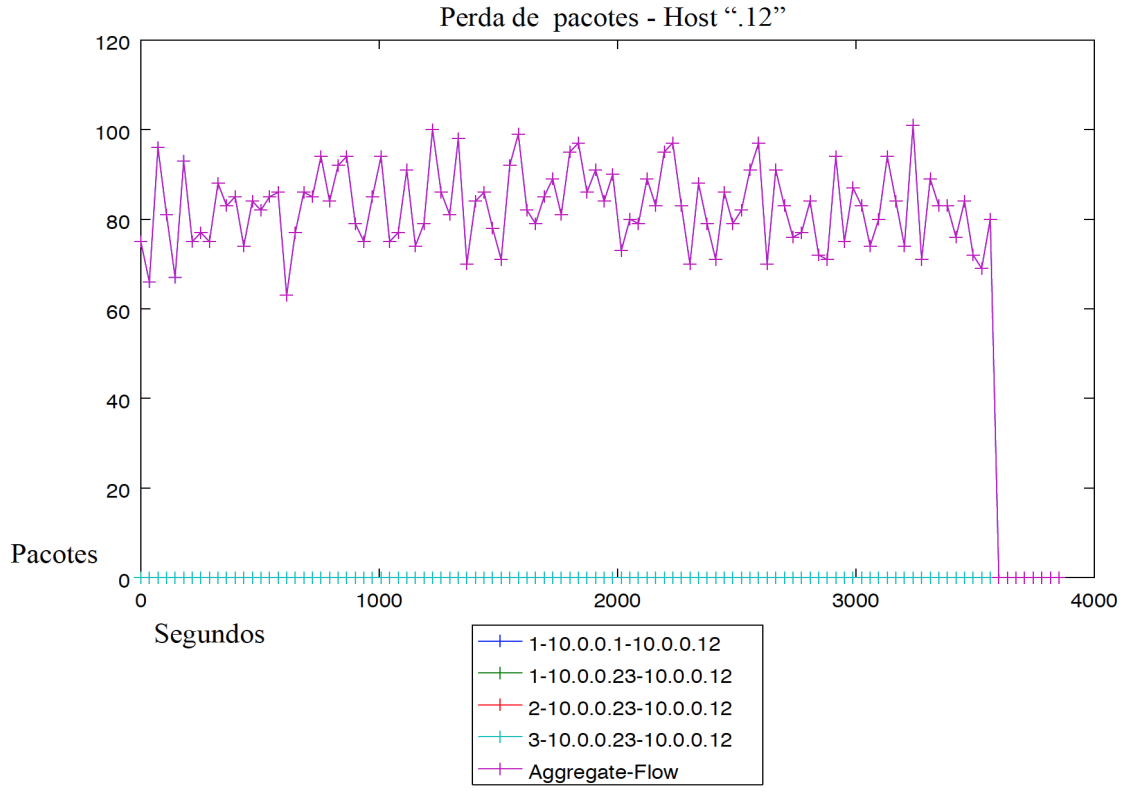

Figura 6.8: Perda de pacotes para o receptor .12 (Carga Alta).

A Figura 6.9 mostra todos os fluxos recebidos pelo host *h12*. São 4 fluxos sendo o fluxo de vídeo representado pela cor azul e os outros 3 fluxos representam o tráfego de fundo (verde, azul claro e vermelho). O gráfico rosa é referente à soma de todos os fluxos. Como mencionado anteriormente, o fluxo de vídeo (protocolo UDP) é recebido de forma "constante" como ele foi enviado, já os fluxos com o protocolo TCP variam bastante a sua taxa e atraso a fim de compensar essa mudança com uma não perda de pacotes.

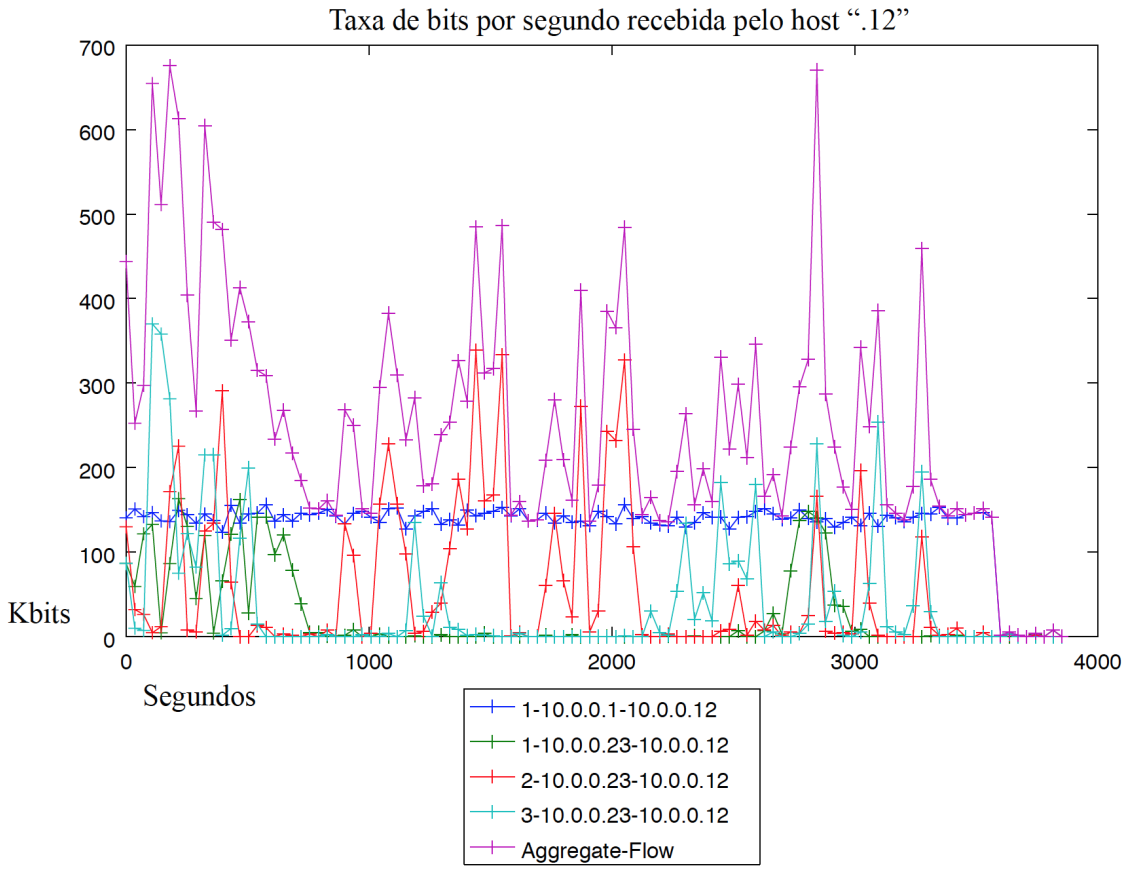

Figura 6.9: Taxa de bits por segundo recebida pelo host .12 (Carga Alta).

A tabela Tabela 6.3 apresenta os resultados analisados baseados nos arquivos *log* armazenados nos receptores.

Tabela 6.3: Resultados da simulação de uma hora para H.261 com "Carga Alta"

Dado de vídeo enviado	823.135.832 bytes
Dado de internet enviado	597.485.460 bytes
Atraso Médio (Vídeo)	0,00 189 072 segundos – (1,89 ms)
Média do Atraso Máximo (Vídeo)	0,09 909 054 segundos – (99,09 ms)
Valor máximo do Atraso (Vídeo)	0,156153 segundos – (156,15 ms)
Jitter Médio (Vídeo)	0,0 005 064 segundos – (0,5 ms)
Média da Perda de Pacotes (Vídeo)	17,77 %
Valor máximo da Perda de Pacotes (Vídeo)	18,81%

Analisando os resultados com os parâmetros necessários de acordo com a Tabela 3.6,

observou-se que algumas medidas não estão dentro dos parâmetros QoS exigidos para a realização de videoconferência. A Tabela 6.4 mostra em vermelho os parâmetros que não foram alcançados e em azul os que foram.

Tabela 6.4: Em vermelho, parâmetros que não foram alcançados e em azul os que foram, após a simulação de uma hora para H.261 com "Carga Alta".

Atraso Médio (Vídeo)	0,00 189 072	segundos – (1,89 ms)
Média do Atraso Máximo (Vídeo)	0,09 909 054	segundos – (99,09 ms)
Valor máximo do Atraso (Vídeo)	0,156153	segundos – (156,15 ms)
Jitter Médio (Vídeo)	0,0 005 064	segundos – (0,5 ms)
Média da Perda de Pacotes (Vídeo)	17,77 %	
Valor máximo da Perda de Pacotes (Vídeo)	18,81%	

Ao comparar os resultados com os parâmetros para QoS definidos na Tabela 3.6, constatou-se uma alta perda de pacotes, assim como um "rompimento" no parâmetro exigido para o atraso, sendo o atraso máximo medido em 156ms, um pouco superior aos 150ms exigidos. Neste cenário de simulação, o *jitter*, atraso médio e média dos valores máximos de atraso, estão dentro das exigências.

H.261 – Carga Baixa

Os parâmetros utilizados nesta simulação foram os da Tabela 5.9 para o tráfego de fundo e a Tabela 5.5 para o tráfego de vídeo. Os parâmetros do cenário foram retirados da Tabela 5.1.

A Figura 6.10 mostra o gráfico do "*h23* – 10.0.0.23" responsável por gerar o tráfego de fundo para todos os receptores. É possível observar na mesma que os fluxos do tráfego de fundo para dos os hosts receptores se encontram na parte debaixo do gráfico e em amarelo (acima) é o resultado da soma de todos os fluxos enviados pelo host *h23*.

Como dito antes para "Carga Alta", o protocolo TCP possui algumas ferramentas de controle de tráfego e como a rede estava sobrecarregada, houve uma perda grande na taxa de envio do tráfego *web*. Já nesta simulação, os enlaces estão configurados com uma taxa de perda de pacotes de 0%, fazendo com que a taxa de envio de tráfego *web* seja muito superior e varie menos.

É possível observar uma variação no começo da transmissão, momento em que o TCP

tenta se adequar à rede, mas por volta dos 600s, o protocolo consegue estabilizar o tamanho da janela de transmissão, se mantendo constante durante o resto da simulação.

Porém, da mesma forma que analisado em para "Carga Alta", o tráfego UDP (de vídeo) não possui tais ferramentas e injeta tráfego na rede com a taxa que foi configurado, como pode ser visto na Figura 6.13 (em azul claro), na parte inferior do gráfico.

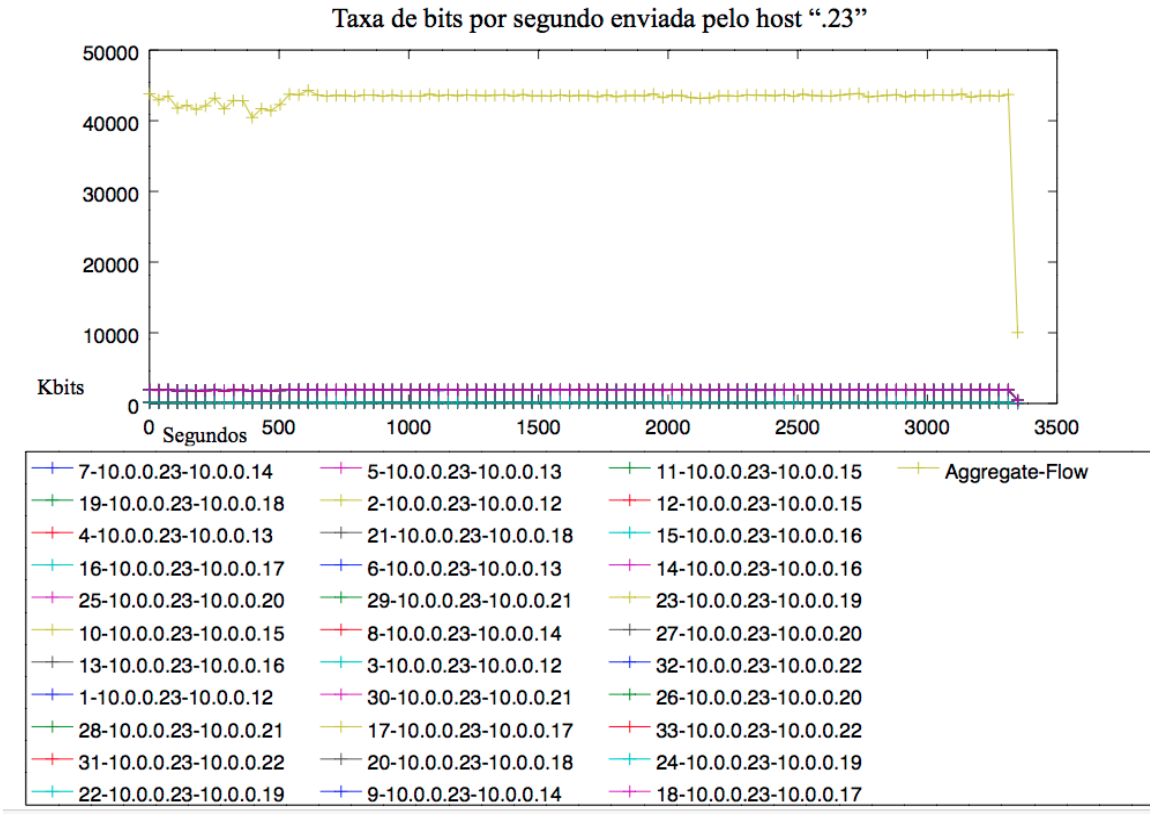

Figura 6.10: Taxa de bits por segundo enviado pelo host .23 de tráfego web (Carga Baixa).

A Figura 6.11 mostra a taxa de *bits* enviados pelo *h1*. Como mencionado na seção da "Carga Alta", por representar apenas um fluxo no gráfico, a taxa de envio do vídeo se sobrepõe ao gráfico da soma dos fluxos. Sua variação está de acordo com os parâmetros configurados para o fluxo de vídeo, no entanto ele não é idêntico a Figura 6.5 pois os parâmetros seguem distribuição exponencial e uniforme, fazendo com que sigam uma média na geração de pacotes (em uma janela de 1s, em média, são gerados 15 pacotes de acordo com a distribuição exponencial) e já na uniforme, recebam valores iniciais diferentes, dentro do parâmetro escolhido.

Na Figura 6.11 observa-se uma taxa mínima um pouco abaixo de 150kb/s e uma máxima em torno de 190 kb/s. O *host* 1 tem uma taxa de envio média de vídeo de 167,3 kb/s enquanto a taxa de *bits*, referente ao tráfego de vídeo, recebida do *host* 1 pelo *host* 12 é uma média de

167,3 kb/s (Figura 6.15),

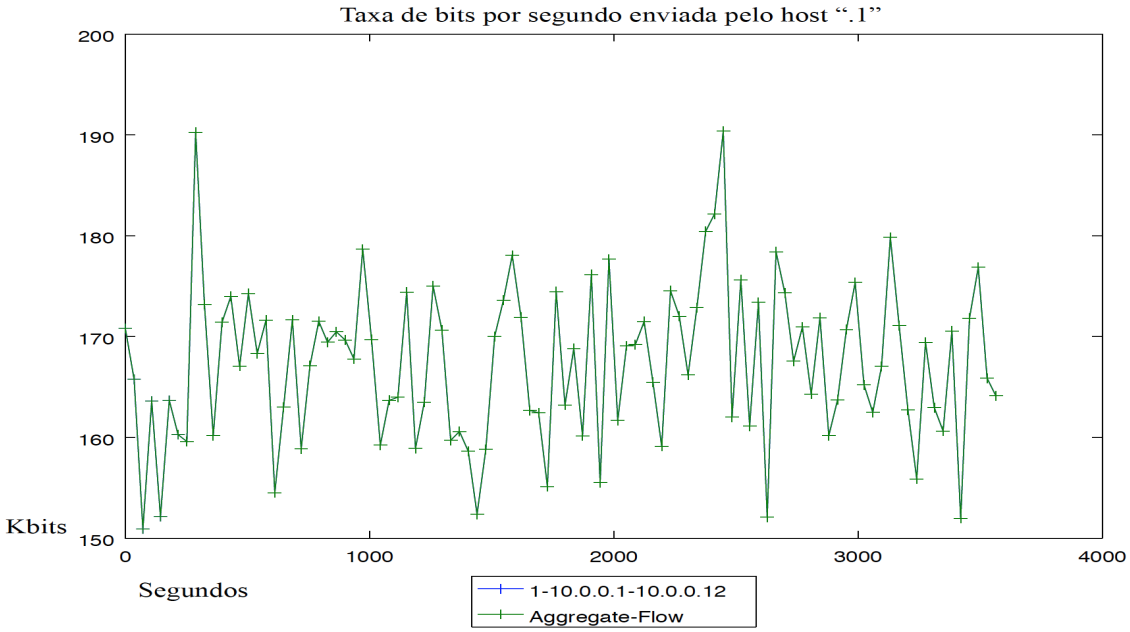

Figura 6.11: Taxa de bits por segundo enviada pelo host .1 de tráfego de vídeo H.261 (Carga baixa).

As Figuras 6.12, 6.13, são referentes aos gráficos de atraso e *jitter*, respectivamente. Como mencionado na seção de "Carga Alta", o *jitter* esta relacionado ao atraso, portanto os pontos onde existe um maior atraso serão também os pontos onde existirão um maior valor do *jitter*.

Assim, analisando as figuras 6.12, 6.13, respectivamente, é possível observar a taxa de atraso (valor máximo de 0,2 ms para o vídeo), *jitter* (valor médio de 0,12 ms para o vídeo) de vídeo dentro de uma variação constante por boa parte da simulação, visto que a carga da rede está baixa (sem perdas de pacotes) e não há necessidade do fluxo TCP aumentar o seu atraso e diminuir a sua taxa de envio para não perder pacotes.

Mesmo quando a rede está em "Carga Baixa" é possível observar que o atraso dos pacotes referentes ao fluxo de fundo (verde, azul e vermelho), possuem um atraso e *jitter* superior ao tráfego de vídeo (azul claro).

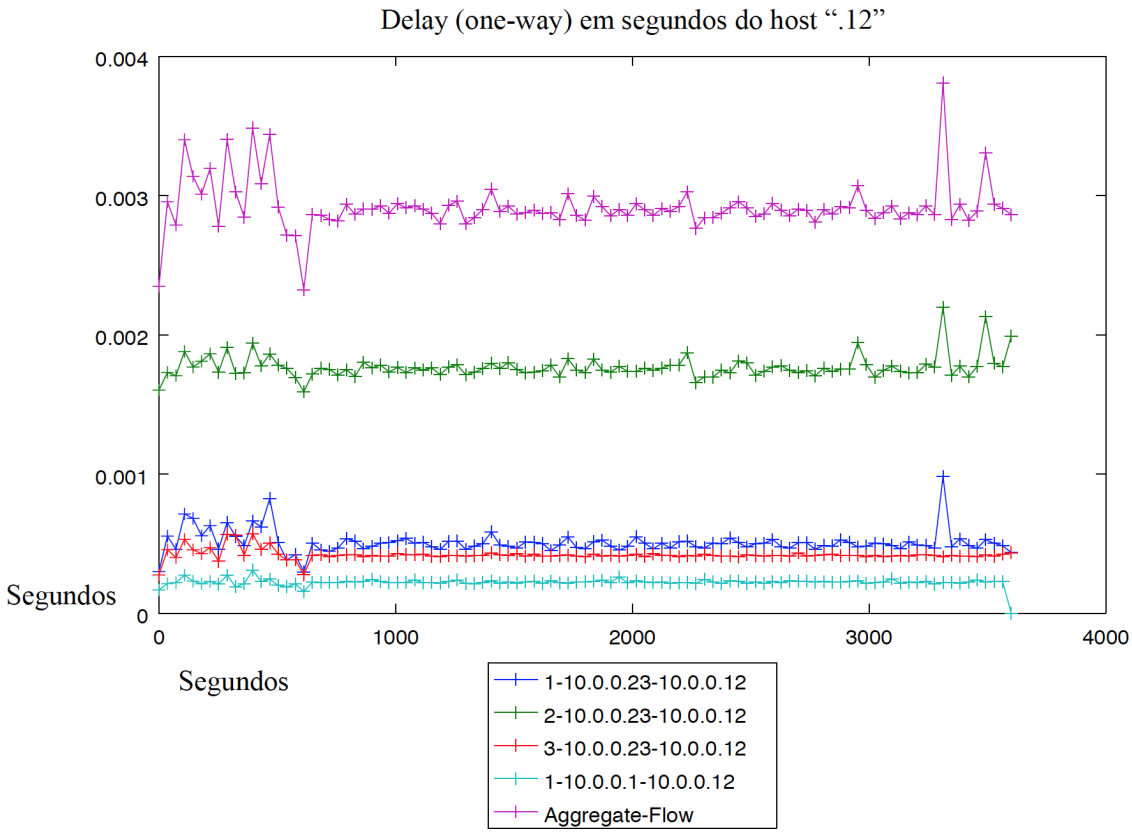

Figura 6.12: Atraso (ida) em segundos no receptor .12 (Carga Baixa).

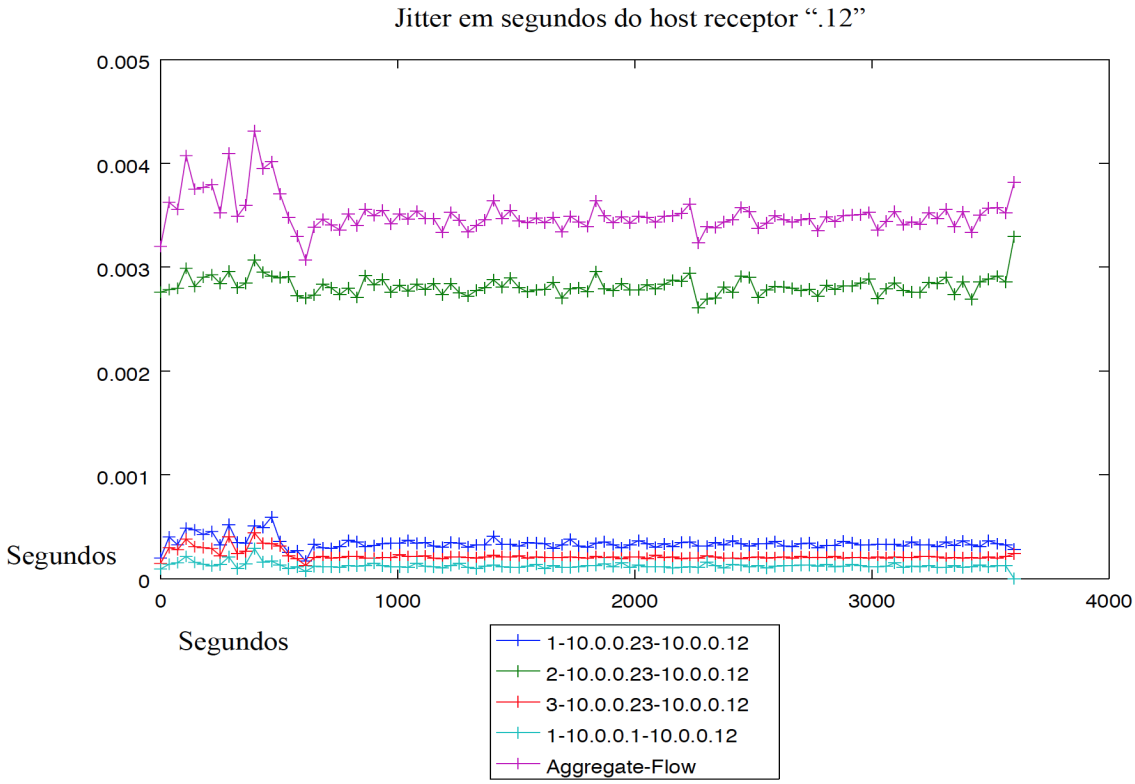

Figura 6.13: Jitter em segundos no receptor .12 (carga baixa).

As Figuras 6.14 e 6.15 mostram os gráficos referentes ao receptor *h12* para perda de pacotes e taxa de *bits,* respectivamente. .

Pela Figura 5.23 observa-se a taxa de perda de pacotes constante em cima da linha do zero, existindo apenas uma alteração perto dos 3600 segundos da simulação. Para este exemplo, do *host* 1 para o *host* 12, foram enviados, durante a simulação, aproximadamente 53592 pacotes de vídeo e foram perdidos um total 0,0% de pacotes. Com isso, o fluxo de vídeo se sobrepõe com o gráfico da soma dos resultados e o referente a perda de pacotes do tráfego de fundo permanece zero durante toda a simulação.

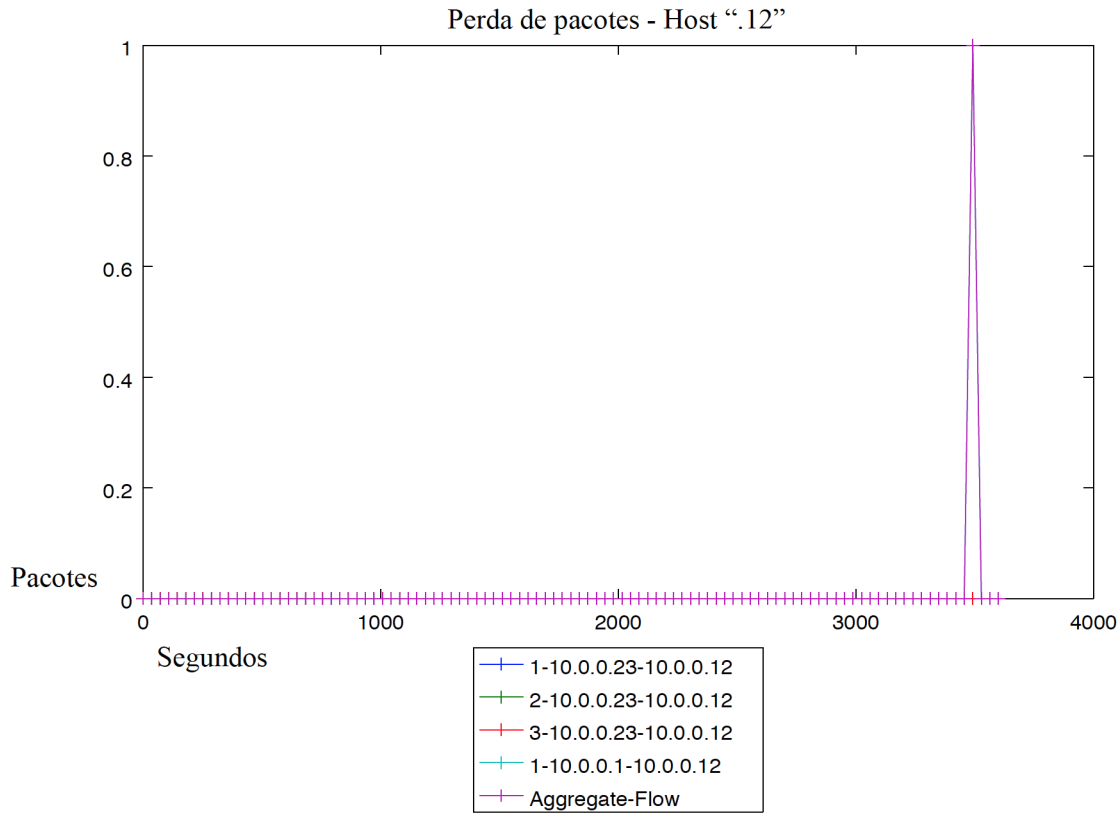

Figura 6.14: Perda de pacotes para o receptor .12 (carga baixa).

A Figura 6.15 mostra todos os fluxos recebidos pelo host *h12*. São 4 fluxos sendo o fluxo de vídeo representado pela cor azul claro e os outros 3 fluxos representam o tráfego de fundo (verde, azul e vermelho). O gráfico rosa é referente à soma de todos os fluxos. Como mencionado anteriormente, o fluxo de vídeo (protocolo UDP) é recebido de forma "constante" como ele foi enviado, já os fluxos com o protocolo TCP variam bastante a sua taxa e atraso a fim de compensar essa mudança com uma não perda de pacotes.

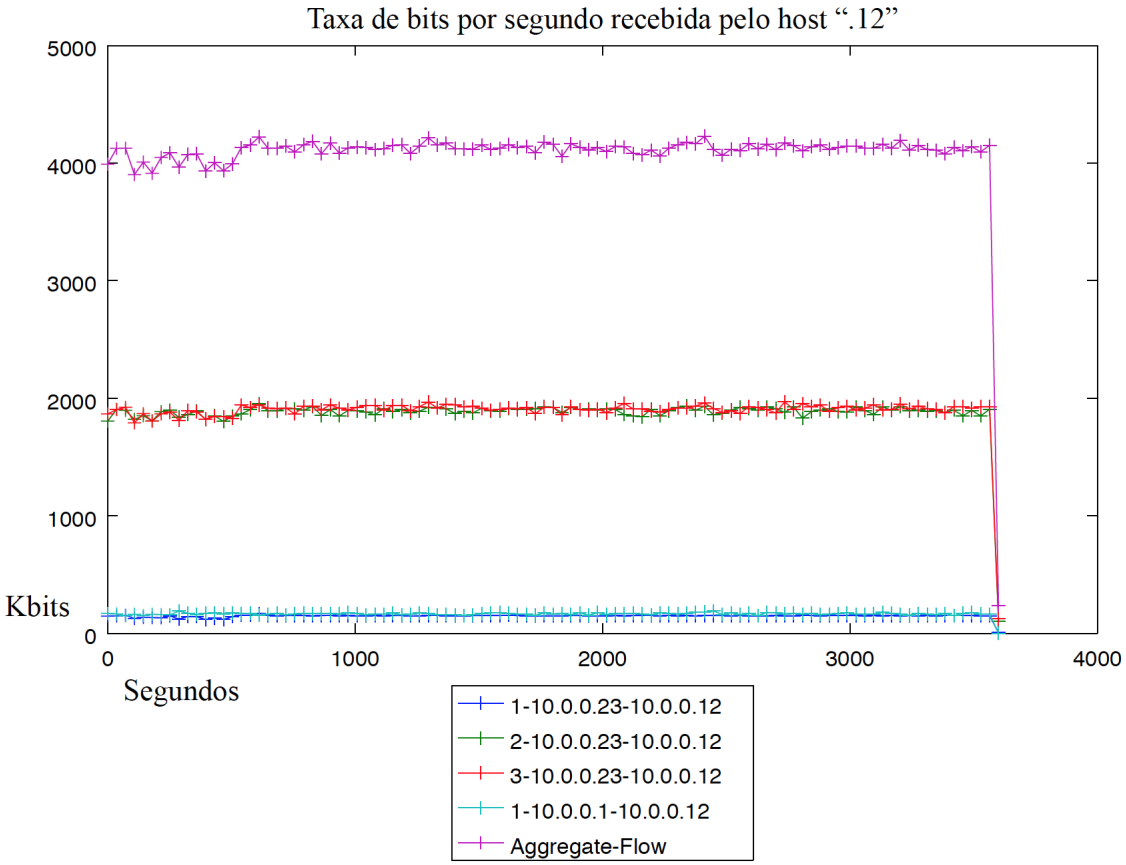

Figura 6.15: Taxa de bits por segundo recebido pelo host .12 (carga baixa).

Ao contrário do cenário com a "Carga Alta", para a simulação com "Carga Baixa" esperava-se um resultado bem mais satisfatório, e de fato foi. Ao simular a rede, os resultados foram bem mais promissores para um serviço de videoconferência com a rede configurada com parâmetros de "Carga Baixa".

A tabela Tabela 6.5 apresenta os resultados analisados baseados nos arquivos *log* armazenados nos receptores.

Tabela 6.5: Resultados da simulação de uma hora para H.261 com "Carga Baixa".

Dado de vídeo enviado	824.839.281 bytes
Dado de internet enviado	18.191.965.300 bytes
Atraso Médio (Vídeo)	0,00 022 555 segundos – (0,22 ms)
Média do Atraso Máximo (Vídeo)	0,040 937 segundos – (40,93 ms)
Valor máximo do Atraso (Vídeo)	0,280 487 segundos – (280 ms)
Jitter Médio (Vídeo)	0,00 012 472 segundos – (0,12 ms)

Média da Perda de Pacotes (Vídeo)	0.00%
Valor máximo da Perda de Pacotes (Vídeo)	0.00%

Analisando os resultados com os parâmetros necessários de acordo com a Tabela 3.6, observa-se que uma medida não está dentro dos parâmetros QoS exigidos para a realização de videoconferência. A Tabela 6.6 mostra em vermelho o parâmetro que não foi alcançado e em azul os que foram.

Tabela 6.6: Em vermelho, parâmetros que não foram alcançados, e em azul os que foram, após a simulação de uma hora para H.261 com "Carga Baixa".

Atraso Médio (Vídeo)	0,00 022 555	segundos – (0,22 ms)
Média do Atraso Máximo (Vídeo)	0,040 937	segundos – (40,93 ms)
Valor máximo do Atraso (Vídeo)	0,280 487	segundos – (280 ms)
Jitter Médio (Vídeo)	0,00 012 472	segundos – (0,12 ms)
Média da Perda de Pacotes (Vídeo)	0.00%	
Valor máximo da Perda de Pacotes (Vídeo)	0.00%	

Ao realizar as simulações com a rede em um estado com menos carga, observa-se um cenário mais promissor. É possível notar que praticamente todos os parâmetros foram alcançados. Nesta simulação, os parâmetros de Atraso, *Jitter* e Perda de Pacotes estão dentro das características exigidas. Apesar do valor máximo de atraso ter alcançado um valor igual, acima do exigido, a média dos valores máximos (40,93ms) ficou bem abaixo do valor exigido (150ms).

6.4. RESUMO CONCLUSIVO

Neste capítulo foram explicadas as simulações e seus resultados. Como mencionado nas seções anteriores, nenhum dos resultados obtidos está 100% dentro dos parâmetros estipulados neste trabalho, com foco em videoconferência.

A simulação com a "Carga Alta" teve um resultado, como era de se esperar, bem inferior

ao resultado obtido com a “Carga Baixa”, tendo uma taxa de perda de pacotes de vídeo, muito superior àquela estabelecida na Tabela 3.6, além também de ter apresentado um atraso máximo fora dos padrões.

Já a simulação em “Carga Baixa” ficou perto de apresentar um resultado 100% satisfatório, pois, apenas o seu valor máximo de Atraso não cumpriu com os parâmetros exigidos. Apesar do atraso máximo ter alcançado um valor fora dos parâmetros exigidos para uma videoconferência, todos os outros valores estão dentro do padrão, inclusive a média com os valores máximos de atraso, que por sua vez, está bem abaixo do exigido.

Com estes resultados, ficou concluído que em “Carga Baixa” é possível realizar uma videoconferência no ramo real da REDUnB com as reais características de tráfego da mesma, sem necessidade de provisão de QoS. No entanto, já para a “Carga Alta”, apesar do valor da média dos valores máximos de atraso ter ficado muito próximo e a sua média do atraso médio ter ficado dentro dos padrões, a sua porcentagem de perda de pacotes foi extremamente alta para os padrões exigidos para uma videoconferência. Desta forma, a provisão de QoS para REDUnB com a “Carga Alta” é necessária, a fim garantir que seja possível realizar a videoconferência mesmo que a rede esteja sobrecarregada.

7. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foram apresentadas conceitos de Redes Definidas por Software juntamente com a Qualidade de Serviço voltada para videoconferência. A partir destes conceitos, foi possível realizar uma simulação de um cenário da REDUnB com dados reais a fim de avaliar a Qualidade de Serviço de videoconferência na rede, com o Paradigma de Redes Definidas por Software.

Compreender as necessidades e vantagens da utilização de Redes Definidas por Software é um passo importante para proporcionar soluções na área. Como visto ao longo do trabalho, as redes SDN aparecem como uma solução promissora para a internet do futuro, permitindo com um baixo custo e alto desempenho, que as redes de computadores se tornem mais flexíveis e podendo assim, se adaptar s novas tecnologias que estão por vir.

Como abordado ao longo do trabalho, o número de usuários da internet aumentou e junto a isto, houve um crescimento nos serviços prestados à internet, em especial a serviços que dependem da Qualidade de Serviço. Estes, estão ligados ao grau de satisfação de um usuário e para seu correto funcionamento, dependem de um serviço diferenciado na rede, a fim de garantir que seus pacotes sejam entregues dentro dos parâmetros exigidos.

Diante destas novas características de exigências para serviços da Internet, o trabalho, teve como objetivo, a realização de uma simulação para avaliação de Qualidade de Serviço de videoconferência para a REDUnB com o Paradigma de Redes Definidas por Software.

Através da avaliação de QoS realizada, este trabalho ofereceu um estudo, sobre a possibilidade de realização de um serviço de videoconferência em um ramo atual da REDUnB com suas características de tráfego reais.

Após a simulação do ramo da REDUnB com os parâmetros reais, obtidos pelo *NAGIOS*, quando a rede está em "Carga Baixa" concluiu-se que é possível sim realizar uma videoconferência no cenário em questão, sem a necessidade de provisão de QoS, visto que os parâmetros exigidos para o serviço de videoconferência foram alcançados.

Por outro lado, a simulação do mesmo ramo da REDUnB quando ela está com uma sobrecarga, denominada "Carga Alta", mostrou que não é possível realizar uma videoconferência, sem provisão de QoS na rede, visto que apesar de a média dos atrasos médios ter ficado próxima dos parâmetros exigidos a sua porcentagem de perda de pacotes ficou muito acima dos padrões exigidos para um serviço de videoconferência.

Desta forma, neste trabalho, foi concluído que o serviço de videoconferência com um

tráfego de vídeo com características de um codificador H.261, para a REDUnB, é possível para a “Carga Baixa” sem a provisão de QoS na rede. No entanto, quando a rede está com uma sobrecarga (“Carga Alta”) é necessária a provisão de Qualidade de Serviço, a fim de garantir que os parâmetros exigidos para um serviço de videoconferência sejam alcançados, garantindo assim, que o serviço seja possível, mesmo no pior cenário de sobrecarga da rede.

7.1. SUGESTÕES DE TRABALHOS FUTUROS

Como visto nos resultados apresentados, a REDUnB não é capaz de garantir o serviço de videoconferência com um tráfego de vídeo com características de um codificador H.261 quando a rede está com sobrecarga, sem a provisão de Qualidade de Serviço.

Com estes resultados, uma das sugestões de trabalhos futuros é a provisão de QoS de videoconferência para a REDUnB com o Paradigma de Redes Definidas por Software por meio de uma priorização do tráfego de vídeo, com o propósito de garantir que o serviço de videoconferência seja realizável em qualquer estado da rede, com ou sem sobrecarga.

Ainda como uma sugestão a mais, seria a avaliação e provisão de QoS para toda a REDUnB e não só apenas para o codificador H.261 mas para outros, também utilizados para videoconferência, como por exemplo, o H.263+, H.264.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] KENDE, M. (2014). *Internet Society Global Internet Report 2014*, 146. Retirado de: http://www.internetsociety.org/doc/global-internet-report

[2] ARAÚJO, M. (2013). *Uma Abordagem para Aprovisionamento de QoS em Redes Definidas por Software baseadas em OpenFlow*. 51 f. Monografia (Graduação) – Centro de Informática, Universidade Federal de Pernambuco, Recife.2013

[3] COSTA, L. (2013). *OpenFlow e o Paradigma de Redes Definidas por Software* (pp. 1–155). Monografia (Graduação) – Departamento de Ciência da Computação, Universidade de Brasília, Brasília. 2013

[4] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G. M., PETERSON, L. L., REXFORD, J., LOUIS, S. (2008). *OpenFlow: enabling innovation in campus networks*. Computer Communication Review, 38(2), 69–74. http://doi.org/10.1145/1355734.1355746

[5] NETO, T. (2014). *Avaliação de QoS em redes Dumbbell através do simulador Network Simulator v2*. 72 f. Monografia (Graduação) – Departamento de Ciência da Computação, Universidade de Brasília. 2014

[6] Site oficial do OpenFlow, https://www.openflow.org, Último acesso: 22/06/15

[7] MARTINS, V., FERNANDEZ, S., CARVALHO, C., SILVA, L. PONTES, O. JÚNIOR, R. (2014). *Estudo Comparativo de Controladores de Software Defined Networks com Métricas de Qualidade de Serviço.* 17 f.

[8] MATTOS, D., FERNANDES, N., DUARTE, O. (2011). *Xenflow: Um sistema de processamento de fluxos robusto e eficiente para migração em redes virtuais*. P 309–322. Retirado de: http://sbrc2011.facom.ufms.br/files/anais/files/main/ST07_2.pdf

[9] ROTHERNBERG, C. E., NASCIMENTO, M. R., SALVADOR M. R., MAGALHÃES M. F. *Open-Flow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes*. Cad. CPqD Tecnologia, 7(1):1–6, July 2010.

[10] METZLER, J., METZLER, A. (2013). Ten Things to Look for in an SDN Controller, (Maio), 1-11

[11] CohesiveFT. White Paper: OpenFlow is Software Defined Networking; *Software Defined Networking is Not Only OpenFlow*, September 24th, 2012.

[12] Site oficial do Open Networking Foundation, https://www.opennetworking.org/, Último acesso: 20/06/15

[13] Stanford University. OpenFlow Switch Specification, 1.1.0 edition, February 2011. viii, 19, 20, 23, 27, 42, 60

[14] MUNTANER, Guillermo Romero de Tejada, *Evaluation of OpenFlowControllers*, October 15, 2012.

[15] OPEN NETWORKING FOUNDATION, OpenFlow Switch Specification Version 1.5.0, 2014

[16] Site oficial do SDNHub, http://sdnhub.org, Último acesso: 15/06/15

[17] Site oficial do SDNHub OpenDaylight, http://sdnhub.org/tutorials/opendaylight-helium/, Último acesso: 15/06/15

[18] Site oficial do Ramon Fontes, http://ramonfontes.com

[19] Plataforma do Ramon Fontes para criação de cenários, http://ramonfontes.com/vnd

[20] GÓMEZ, H. *Dynamical Quality of Service Over Software Defined Networking.* 2015

[21] BRADEN, R., CLARK, D., and SHENKER, S. *Integrated services in the internet architecture: an overview*, june 1994. Status: Informational.

[22] RODRIGUEZ, A. e International Business Machines Corporation. Tcp2002 *tutorial and technical overview*. Prentice Hall PTR, 2002.

[23] Site oficial do Octave, http://www.gnu.org/software/octave/, Último acesso: 19/06/15

[24] CHEN, Y., FARLEY, T., & YE, N. (2004). *QoS Requirements of Network Applications on the Internet*. Information-Knowledge-Systems Management, 4(1), P 55–76.

[25] SZUPROWICZ, B.O. , *Multimedia Networking*. McGraw-Hill, Inc., 1995, Pg 161-162

[26] Silveira, R.M., Margi, C.B., Gonzalez, L.G., Favero, E., Vilcachagua, O.D., Bressan, G. & Ruggiero, W.V.A. (1996). *Multimedia on Demand System for Distance Education. LARC-Laboratory of Computer Architecture and Networks*, PCS-Department of Computer and Digital System Engineering EPUSP-Polytechnic School, University of San Paulo, Standards: ITU-T G.114.

[27] HATTINGH, C., SZIGETI, T. (2004). *End-to-end QoS network design: quality of service in LANs, WANs, and VPNs.*

[28] BINI, T. (2007).*Negociador de QoS utilizando RMI para um domínio DiffServ*

[29] SOARES, C. (2006). *Avaliação de Desempenho da Engenharia de Tráfego com MPLS Através de Simulações*. 83 f. Monografia (Graduação) – Departamento de Ciência da Computação, Universidade de Goiás, Catalão. 2006

[30] ITU-T rec H.261 – https://www.itu.int/rec/T-REC-H.261/en, Último acesso: 21/06/15

[31] Site oficial do Nagios, https://www.nagios.org, Último acesso: 23/06/15

[32] Site oficial do Mininet – http://mininet.org, Último acesso: 15/06/15

[33] Site oficial do D-ITG – http://traffic.comics.unina.it/software/ITG/, Manual oficial – http://traffic.comics.unina.it/software/ITG/manual/, Último acesso: 22/06/15

[34] DOMOXOUDIS, S., KOUREMENOS, S., *Measurement, modelling and simulation of videoconference traffic from VBR video encoders*, 2004

[35] GEBERT, S., PRIES, R., SCHLOSSER, D., KLAUS, H. *Internet access traffic measurement and analysis*, 2012, pg 29-42

[36] SINHA, R., PAPADOPOULOS, C., HEIDEMANN, J. *Internet packet size distributions: Some observations*, 2007, pg 1-7

[37] ITU-T rec H.320 – https://www.itu.int/rec/T-REC-H.320/en, Último acesso: 21/06/15

[38] ITU-T rec H.323 – https://www.itu.int/rec/T-REC-H.323/en, Último acesso: 21/06/15

[39] ITU-T rec H.324 – https://www.itu.int/rec/T-REC-H.324/en, Último acesso: 21/06/15

[40] Site Oficial OpenDaylight – http://www.opendaylight.org, Último acesso: 15/06/15

APÊNDICE A - CÓDIGO DOS CENÁRIOS UTILIZADOS NO TRABALHO

A.1. Código cenário Mininet para “Carga Alta”

```
#!/usr/bin/python

"""
Script created by VND - Visual Network Description (SDN version)
"""
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import Link, TCLink

def topology():
    "Create a network."
    net = Mininet( controller=RemoteController, link=TCLink,
switch=OVSKernelSwitch )

    print "*** Creating nodes"
    h1 = net.addHost( 'h1', mac='00:00:00:00:00:01',
ip='10.0.0.1/8' )
    h2 = net.addHost( 'h2', mac='00:00:00:00:00:02',
ip='10.0.0.2/8' )
    h3 = net.addHost( 'h3', mac='00:00:00:00:00:03',
ip='10.0.0.3/8' )
    h4 = net.addHost( 'h4', mac='00:00:00:00:00:04',
ip='10.0.0.4/8' )
    h5 = net.addHost( 'h5', mac='00:00:00:00:00:05',
ip='10.0.0.5/8' )
    h6 = net.addHost( 'h6', mac='00:00:00:00:00:06',
ip='10.0.0.6/8' )
    h7 = net.addHost( 'h7', mac='00:00:00:00:00:07',
ip='10.0.0.7/8' )
    h8 = net.addHost( 'h8', mac='00:00:00:00:00:08',
ip='10.0.0.8/8' )
    h9 = net.addHost( 'h9', mac='00:00:00:00:00:09',
ip='10.0.0.9/8' )
    h10 = net.addHost( 'h10', mac='00:00:00:00:00:10',
ip='10.0.0.10/8' )
    h11 = net.addHost( 'h11', mac='00:00:00:00:00:11',
ip='10.0.0.11/8' )
    h12 = net.addHost( 'h12', mac='00:00:00:00:00:12',
ip='10.0.0.12/8' )
    h13 = net.addHost( 'h13', mac='00:00:00:00:00:13',
ip='10.0.0.13/8' )
    h14 = net.addHost( 'h14', mac='00:00:00:00:00:14',
ip='10.0.0.14/8' )
    h15 = net.addHost( 'h15', mac='00:00:00:00:00:15',
ip='10.0.0.15/8' )
    h16 = net.addHost( 'h16', mac='00:00:00:00:00:16',
ip='10.0.0.16/8' )
    h17 = net.addHost( 'h17', mac='00:00:00:00:00:17',
ip='10.0.0.17/8' )
```

```
    h18 = net.addHost( 'h18', mac='00:00:00:00:00:18',
ip='10.0.0.18/8' )
    h19 = net.addHost( 'h19', mac='00:00:00:00:00:19',
ip='10.0.0.19/8' )
    h20 = net.addHost( 'h20', mac='00:00:00:00:00:20',
ip='10.0.0.20/8' )
    h21 = net.addHost( 'h21', mac='00:00:00:00:00:21',
ip='10.0.0.21/8' )
    h22 = net.addHost( 'h22', mac='00:00:00:00:00:22',
ip='10.0.0.22/8' )
    h23 = net.addHost( 'h23', mac='00:00:00:00:00:23',
ip='10.0.0.23/8' )
    s24 = net.addSwitch( 's24', protocols='OpenFlow13' ,
listenPort=6673, mac='00:00:00:00:00:24' )
    c25 = net.addController( 'c25', controller=RemoteController,
ip='127.0.0.1', port=6633 )

    print "*** Creating links"
    net.addLink(h23, s24, 0, 23, bw=1000 , delay=19.585 , loss=8 )
    net.addLink(h22, s24, 0, 22, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h21, s24, 0, 21, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h20, s24, 0, 20, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h19, s24, 0, 19, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h18, s24, 0, 18, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h17, s24, 0, 17, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h16, s24, 0, 16, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h15, s24, 0, 15, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h14, s24, 0, 14, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h13, s24, 0, 13, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h12, s24, 0, 12, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h11, s24, 0, 11, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h10, s24, 0, 10, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h9, s24, 0, 9, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h8, s24, 0, 8, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h7, s24, 0, 7, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h6, s24, 0, 6, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h5, s24, 0, 5, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h4, s24, 0, 4, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h3, s24, 0, 3, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h2, s24, 0, 2, bw=1000 , delay=19.585 , loss=8)
    net.addLink(h1, s24, 0, 1, bw=1000 , delay=19.585 , loss=8)

    print "*** Starting network"
    net.build()
    c25.start()
    s24.start( [c25] )

    print "*** Running CLI"
    CLI( net )

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    topology()
```

A.2. Código cenário Mininet para “Carga Baixa”

```
#!/usr/bin/python

"""
Script created by VND - Visual Network Description (SDN version)
"""
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import Link, TCLink

def topology():
    "Create a network."
    net = Mininet( controller=RemoteController, link=TCLink,
switch=OVSKernelSwitch )

    print "*** Creating nodes"
    h1 = net.addHost( 'h1', mac='00:00:00:00:00:01',
ip='10.0.0.1/8' )
    h2 = net.addHost( 'h2', mac='00:00:00:00:00:02',
ip='10.0.0.2/8' )
    h3 = net.addHost( 'h3', mac='00:00:00:00:00:03',
ip='10.0.0.3/8' )
    h4 = net.addHost( 'h4', mac='00:00:00:00:00:04',
ip='10.0.0.4/8' )
    h5 = net.addHost( 'h5', mac='00:00:00:00:00:05',
ip='10.0.0.5/8' )
    h6 = net.addHost( 'h6', mac='00:00:00:00:00:06',
ip='10.0.0.6/8' )
    h7 = net.addHost( 'h7', mac='00:00:00:00:00:07',
ip='10.0.0.7/8' )
    h8 = net.addHost( 'h8', mac='00:00:00:00:00:08',
ip='10.0.0.8/8' )
    h9 = net.addHost( 'h9', mac='00:00:00:00:00:09',
ip='10.0.0.9/8' )
    h10 = net.addHost( 'h10', mac='00:00:00:00:00:10',
ip='10.0.0.10/8' )
    h11 = net.addHost( 'h11', mac='00:00:00:00:00:11',
ip='10.0.0.11/8' )
    h12 = net.addHost( 'h12', mac='00:00:00:00:00:12',
ip='10.0.0.12/8' )
    h13 = net.addHost( 'h13', mac='00:00:00:00:00:13',
ip='10.0.0.13/8' )
    h14 = net.addHost( 'h14', mac='00:00:00:00:00:14',
ip='10.0.0.14/8' )
    h15 = net.addHost( 'h15', mac='00:00:00:00:00:15',
ip='10.0.0.15/8' )
    h16 = net.addHost( 'h16', mac='00:00:00:00:00:16',
ip='10.0.0.16/8' )
    h17 = net.addHost( 'h17', mac='00:00:00:00:00:17',
ip='10.0.0.17/8' )
    h18 = net.addHost( 'h18', mac='00:00:00:00:00:18',
ip='10.0.0.18/8' )
    h19 = net.addHost( 'h19', mac='00:00:00:00:00:19',
ip='10.0.0.19/8' )
    h20 = net.addHost( 'h20', mac='00:00:00:00:00:20',
ip='10.0.0.20/8' )
    h21 = net.addHost( 'h21', mac='00:00:00:00:00:21',
ip='10.0.0.21/8' )
    h22 = net.addHost( 'h22', mac='00:00:00:00:00:22',
ip='10.0.0.22/8' )
    h23 = net.addHost( 'h23', mac='00:00:00:00:00:23',
ip='10.0.0.23/8' )
    s24 = net.addSwitch( 's24', protocols='OpenFlow13' ,
```

```
listenPort=6673, mac='00:00:00:00:00:24' )
    c25 = net.addController( 'c25', controller=RemoteController,
ip='127.0.0.1', port=6633 )

    print "*** Creating links"
    net.addLink(h23, s24, 0, 23, bw=1000 , delay=2.10285 , loss=0 )
    net.addLink(h22, s24, 0, 22, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h21, s24, 0, 21, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h20, s24, 0, 20, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h19, s24, 0, 19, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h18, s24, 0, 18, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h17, s24, 0, 17, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h16, s24, 0, 16, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h15, s24, 0, 15, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h14, s24, 0, 14, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h13, s24, 0, 13, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h12, s24, 0, 12, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h11, s24, 0, 11, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h10, s24, 0, 10, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h9, s24, 0, 9, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h8, s24, 0, 8, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h7, s24, 0, 7, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h6, s24, 0, 6, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h5, s24, 0, 5, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h4, s24, 0, 4, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h3, s24, 0, 3, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h2, s24, 0, 2, bw=1000 , delay=2.10285 , loss=0)
    net.addLink(h1, s24, 0, 1, bw=1000 , delay=2.10285 , loss=0)

    print "*** Starting network"
    net.build()
    c25.start()
    s24.start( [c25] )

    print "*** Running CLI"
    CLI( net )

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    topology()
```

APÊNDICE B - TUTORIAL DE INSTALAÇÃO DAS FERRAMENTAS

B.1. Instalação SDNHub (O controlador OpenDaylight já vem instalado)

O SDNHub pode ser obtido no site oficial do SDNHub [16] onde é possível encontrar o download da máquina virtual pronta para uso. O programa vem em ".ova" e para instala-lo

bastar realizar os seguintes passos (caso esteja utilizando o programa VirtualBox):

File > *Import Appliance* > Selecionar o programa baixado "SDN Hub tutorial VM 64-bit.ova" > *Import* ;

Pronto, a máquina SDNHub já está pronta para uso. Para se familiarizar com tal, recomenda-se os tutoriais disponíveis em seu site oficial.

Como dito anteriormente, a plataforma SDNHub já vem instalada com o *Mininet* e os principais controladores SDN.

B.2. Instalação D-ITG

O D-ITG suporta vários sistemas operacionais, como Windows e todos os Unix [33].

Como o sistema operacional utilizado neste trabalho foi o Ubuntu (14.04 e 12.04), sua instalação será apresentada para tal sistema. São apenas quatro comandos.

1 – $ **wget** http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip

2 – $ **unzip** D-ITG-2.8.1-r1023-src.zip

3 – $ **cd** D-ITG-2.8.1-r1023-src.zip

4 – $ **make**

Pronto, o D-ITG já está instalado.