UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ALFREDO TOMAZ DE OLIVEIRA

GARANTIA DE QUALIDADE DE SERVIÇO USANDO OS ALGORITMOS SFQ E HTB

PONTA GROSSA

2022

ALFREDO TOMAZ DE OLIVEIRA

GARANTIA DE QUALIDADE DE SERVIÇO USANDO OS ALGORITMOS SFQ E HTB

Quality of Service Assurance Using SFQ and HTB Algorithm

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Departamento Acadêmico de Informatica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Augusto Foronda

PONTA GROSSA

2022

ALFREDO TOMAZ DE OLIVEIRA

GARANTIA DE QUALIDADE DE SERVIÇO USANDO OS ALGORITMOS SFQ E HTB

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Departamento Acadêmico de Informatica da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 08/novembro/2022

Augusto Foronda
Doutorado
Universidade Tecnológica Federal do Paraná

Lourival Aparecido de Gois
Doutorado
Universidade Tecnológica Federal do Paraná

Geraldo Ranthum
Mestrado
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2022

Dedico este trabalho ao meus avós Valdomiro
Tomaz e Adelaide (In Memoriam).

AGRADECIMENTOS

Gostaria de agradecer a Deus, a minha família e amigos por me auxiliarem nessa trajetória. Em especial aos meus pais por me ajudarem, a minha tia por me auxiliar nos momentos de angústia e as minhas avós.

Agradeço também ao meu orientador Prof. Dr. Augusto Foronda, pela grande ajuda e pela orientação durante a trajetória da escrita e desenvolvimento deste trabalho.

Qui-Gon Jinn: "Lembre-se sempre, o seu foco determina a sua realidade."(LUCAS, 1999)

RESUMO

Quando a internet surgiu, não havia necessidade em prover qualidade no fluxo de dados, porém com o avanço tecnológico e crescimento do número de usuários, constatou-se a necessidade de fornecer para as empresas e domicílios qualidade de serviço, para prevenção da perda de pacotes e diminuição da vazão na rede. Nesse contexto, o presente trabalho tem como objetivo analisar dois algoritmos de enfileiramento de pacotes, Stochastic Fair Queuing (SFQ) e o Hierarchical Token Bucket (HTB), configurando um ambiente de teste para medir a vazão de fluxos com cada algoritmo, e a partir dos resultados obtidos, realizar o processo de comparação entre eles. Os resultados obtidos com o SFQ mostram que é um algoritmo que fornece divisão justa de banda entre os fluxos. E o HTB é um algoritmo que provê largura de banda para cada fluxo conforme a sua necessidade.

Palavras-chave: QoS; Enfileiramento de Pacotes; SFQ; HTB.

ABSTRACT

When the internet emerged, there was no need to provide quality data flow, but with the technological advancement and growth in the number of users, there was a need to provide companies and households with quality of service, to prevent packet loss. and decreased throughput in the network. In this context, the present work aims to analyze two packet queuing algorithms, Stochastic Fair Queuing (SFQ) and the Hierarchical Token Bucket (HTB), configuring a test environment to measure the throughput of flows with each algorithm, and from of the results obtained, carry out the process of comparison between them. The results obtained with SFQ show that it is an algorithm that provides fair division of bandwidth between streams. And HTB is an algorithm that provides bandwidth for each stream as per your need.

Keywords: Quality of Service; Packet Queuing; SFQ; HTB.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

ARPANET	*Advanced Research Project Agency Network*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
FIFO	*First In, First Out*
FQ	*Fair Queueing*
HTB	*Hierarchical Token Bucket*
IP	*Internet Protocol*
ISO	*International Standards Organization*
OSI	*Open Systems Interconnection*
QDISC	*Queueing Discipline*
QOS	*Quality of Service*
SFQ	*Stochastic Fair Queuing*
TC	*Traffic Control*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

SUMÁRIO

1 INTRODUÇÃO

Com o aumento da tecnologia, a maioria das empresas tem buscado ferramentas cada vez mais modernas para melhorarem a produtividade, como servidores, computadores, roteadores, *switches* e pontos de acesso. Muitas dessas tecnologias são usadas para criar uma rede de comunicação. Segundo Tanenbaum, “Muitas empresas têm um número significativo de computadores. Por exemplo, uma empresa pode ter computadores separados para monitorar a produção, controlar os estoques e elaborar a folha de pagamento.” (TANENBAUM, 2003).

Primeiramente, é necessário entender o que são redes de computadores. Redes de computadores podem ser definidas como a conexão de vários equipamentos: computadores, servidores, *switches*, roteadores, roteadores sem fio através de meios de comunicação, como par metálico ou *wireless*. O uso de redes de computadores nos dias atuais é de suma importância, tanto para acessar a internet, onde diversas empresas necessitam para diversos fins, como por exemplo acesso ao sistema da empresa, comunicação entre funcionários, entre outros.

Em uma rede de computadores, um problema tem afetado os usuários devido ao grande número existente de computadores. Esse problema é conhecido como congestionamento de tráfego na rede. Tráfego de redes é a transmissão e recepção de dados entre dois equipamentos da rede. Quando se realiza um “*download*” de arquivo, acessa uma página web, realiza um serviço de *streaming* ou envia informações para outros computadores, pacotes com dados são enviados e recebidos a cada instante. E com o grande crescimento do número de computadores, o fluxo de dados que transitam nas redes aumentou na mesma proporção.

Em virtude do tráfego na rede, problemas podem ocorrer, entre eles a impossibilidade de comunicação entre computadores, problema de conexão com o servidor, perda de pacotes, diminuição da vazão (quantidade de bits que são transmitidos) entre outros. De modo que, faz-se necessária alguma solução a fim de tentar evitar problemas. Para isso é possível trabalhar com mecanismos de Qos (qualidade de serviço) em redes. Com certos algoritmos pode-se controlar o enfileiramento e envio de pacotes, esses algoritmos são chamados de *Queueing Discipline* (QDISC).

Um dos algoritmos é o *Stochastic Fair Queuing* (SFQ), baseado na ideia de enfileiramento justo dos pacotes. Utilizando a estratégia de *Round-Robin* (divisão de tempo de uso da CPU) o algoritmo cria múltiplas filas que nelas vão ser distribuídas o fluxo de pacotes de maneira que a quantidade de banda fica equilibrada entre as filas. Para seu funcionamento o algoritmo SFQ aloca inúmeras filas *First In, First Out* (FIFO) (outro algoritmo de enfileiramento de pacote que os pacotes são tratados na mesma ordem que eles chegam) e os pacotes de dados são distribuídos entre elas utilizando um algoritmo de *hash*.

Outro algoritmo é o *Hierarchical Token Bucket* (HTB), baseado na ideia de hierarquia. Bem parecido com a estrutura de dados Árvore onde tem nó pai e nó filha, nesse caso em vez de nó é chamado classe pai e classe filha. Nesse algoritmo a classe pai tem uma taxa média de bits que é garantida para ela e para suas classes filhas. Para seu funcionamento 2 parâmetros

são essenciais, *ceil* que é a taxa máxima de bits que ela pode emprestar da classe pai e *rate* que é a taxa máxima que ela garante para suas classes filhas. Outros parâmetros são utilizados para melhor funcionamento do algoritmo, como *priority*, *burst* e *cburst*.

Este trabalho tem como proposta criar uma topologia e implementar dois algoritmos: SFQ e HTB, e analisar a vazão e perda de pacotes entre o cliente e servidor enquanto se utiliza um gerador de tráfego.

1.1 Objetivos

Essa seção apresenta o objetivo geral e os objetivos específicos deste trabalho.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é analisar os algoritmos, SFQ e HTB para prover Qualidade de Serviço.

1.1.2 Objetivo Específicos

- Analisar os algoritmos SFQ e HTB;
- Configurar um ambiente de testes para simular os algoritmos;
- Medir a vazão na rede com os algoritmos;
- Comparar os resultados obtidos.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos básicos para a compreensão do trabalho que está dividido da seguinte forma: Na seção 2.1 é apresentada toda a estrutura do modelo OSI. Na seção 2.2 é apresentado toda a estrutura do modelo TCP/IP. Na seção 2.3 é apresentado sobre os roteadores na camada de rede. Na seção 2.4 é apresentado o conceito sobre Qos (Qualidade de Serviço) e formas de garantir a qualidade. Nas seções 2.5, 2.6 e 2.7 são apresentados os algoritmos FIFO, SFQ E HTB respectivamente. Na seção 2.8 é apresentado ao Mininet, um emulador de redes. Na seção 2.9 é apresentado o Traffic Control no Linux.

2.1 Redes com o modelo OSI

Em razão dos inúmeros protocolos de comunicação diferentes criados na década de 70/80, os computadores com marcas diferentes (cada empresa tinha seu protocolo de comunicação) não conseguiam se comunicar entre si. Para resolver esse problema foi desenvolvido pela *International Standards Organization* (ISO) em 1983 o modelo *Open Systems Interconnection* (OSI) com objetivo de ser a padronização dos protocolos empregados nas camadas de comunicação. Segundo Kurose “Em razão de seu impacto precoce na educação de redes, esse modelo continua presente em alguns livros sobre redes e em cursos de treinamento.” (KUROSE; ROSS, 2013). Visto isso, o modelo OSI é considerado um modelo conceitual voltado para a educação.

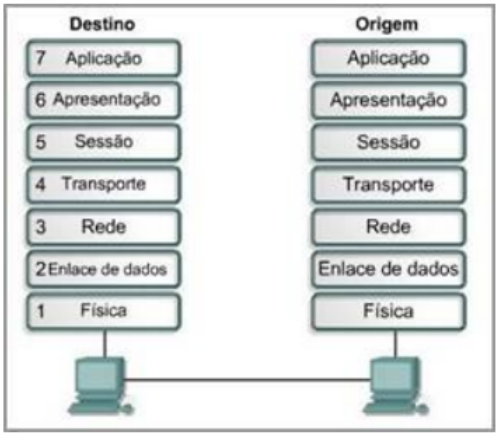
Pode-se definir o modelo OSI como um conjunto de regras e orientações padronizadas que possibilita a conectividade de dois dispositivos de rede para que ocorra a comunicação entre eles. O seu funcionamento é através de uma pilha de protocolos em grupos específicos, e para que ocorra a comunicação, o modelo OSI divide esses grupos em sete camadas diferentes, sendo elas: Física, Enlace, Rede, Transporte, Sessão, Apresentação e Aplicação.

A Figura 1 mostra uma forma resumida de como funciona a comunicação entre duas máquinas utilizando o modelo OSI e suas camadas.

2.1.1 Camada Física

Segundo Tanenbaum, "A camada física trata da transmissão de bits normais por um canal de comunicação.” (TANENBAUM, 2011). A camada física pode ser caracterizada pela transmissão dos bits, sem ligar pelo seu significado. Entre as funções mais comuns é garantir que um bit que sai do transmissor tenha o mesmo significado no momento que chega no receptor e que a transmissão dos bits pode ser feita simultaneamente ou não (*Simplex, Half-Duplex ou Full-Duplex*). Os principais dispositivos que pertencem a esse grupo são cabeamento, *patch panels*, repetidores, HUBs entre outros.

Figura 1 – Modelo OSI



Fonte: Academy (2022)

2.1.2 Camada de Enlace

A camada de enlace é caracterizada pelo controle dos dados entre o transmissor e receptor. Entre as principais tarefas está o controle do fluxo (como os quadros vão ser recepcionados e transmitidos), detecção e correção de erros. Nessa camada, também são acrescentados aos dados de entrada os endereços físicos de origem e destino, chamando assim de quadros. É possível identificar a partir dessa camada todos os *hosts* disponíveis sem passar pelo roteador, além de descrever a topologia da rede atual. O *switch* é um dispositivo que atua na camada de enlace.

2.1.3 Camada de Rede

A camada de rede determina como os pacotes são roteados da origem até o destino considerando a passagem por redes intermediárias (MACEDO *et al.*, 2018). Para conseguir realizar o roteamento dos pacotes, a camada de rede tem como função controlar as operações de sub-rede, como o controle do congestionamento, controle de roteamento (caminho que os pacotes vão ser roteados até o seu destino, pode ser estático ou dinâmico), detecção de erros. Nela os dados de entrada serão chamados de pacotes, pois é acrescentado o endereço lógico de origem e destino. Protocolos como *Internet Protocol* (IP), IPv4 e IPv6 e dispositivos como o roteador são atuantes na camada 3.

2.1.4 Camada de Transporte

A camada de transporte tem como função ser um link entre a camada de aplicação e a camada de rede. Entre as funções estão dividir os dados recebidos de camadas superiores e dividi-los em unidades menores, e principalmente garantir comunicação fim-a-fim. A garantia da

confiabilidade da comunicação fim-a-fim faz com que uma máquina consiga mandar mensagem para outra máquina além de garantir o controle do fluxo de dados. Outra função importante é a Multiplexação, juntar conexões de transporte para criar uma conexão de rede com o intuito de reduzir custos, e splitting que é dividir a conexão de rede para criar várias conexões de transporte.

Um dos principais protocolos dessa camada é o *Transmission Control Protocol* (TCP) que fornece um fluxo de bytes confiável (dados sem erros, sem duplicação, na ordem certa, controle de congestionamento).

2.1.5 Camada de Sessão

A camada de sessão fornece a possibilidade para máquinas diferentes estabelecer sessões entre si. Segundo Kurose “A camada de sessão provê a delimitação e sincronização da troca de dados, incluindo os meios de construir um esquema de pontos de verificação e de recuperação” (KUROSE; ROSS, 2013). Sendo assim as sessões são caracterizadas pelo controle de diálogo, gerenciamento de símbolos e a sincronização. Por conta dos protocolos dessa camada, em caso de perda de conexão entre as máquinas pode haver a recuperação a partir do ponto que ocorreu a falha.

2.1.6 Camada de Apresentação

Para que ocorra a comunicação e o entendimento dos dados transmitidos de uma máquina para outra, tendo em vista que essas máquinas podem ter sistemas diferentes, a camada de apresentação serve como um tradutor dos dados. Essa camada tem como função traduzir os dados de forma que outro sistema possa entender o conteúdo do dado transmitido (com sistemas diferentes, a sintaxe e a semântica podem ser representadas diferentes).

2.1.7 Camada de Aplicação

A camada de aplicação é um conjunto de protocolos que realizam a comunicação com o usuário final. Quando se deseja acessar um email, transmitir arquivos, correio eletrônico, acessar uma página WEB, login entre outros são os protocolos da camada de aplicação que são encarregados de realizar essas atividades. Um protocolo altamente utilizado é o HTTP, que é a base para a comunicação de dados da *World Wide Web*. *Domain Name System* (DNS), *Dynamic Host Configuration Protocol* (DHCP), Telnet outros são outros protocolos usados nessa camada.

2.2 Redes com o modelo TCP/IP

Em 1969 a *Advanced Research Project Agency Network* (ARPANET) foi criada pelos militares norte-americanos para a transmissão de dados entre seus computadores (utilizada posteriormente também por universidades). Para garantir que aconteça a comunicação entre duas máquinas com sistemas diferentes, o modelo TCP/IP foi criado. E em 1982 todos os computadores militares usariam esse modelo.

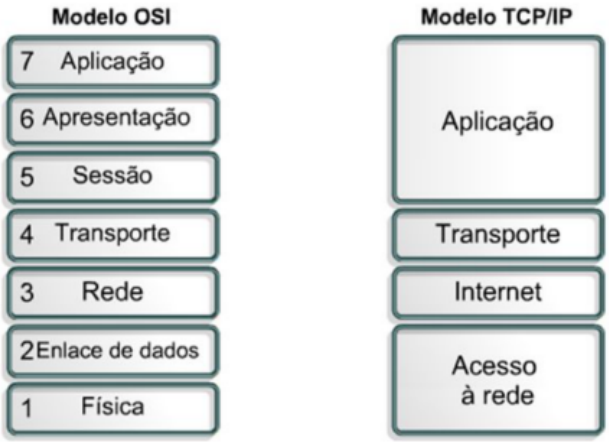
Em comparação com o modelo OSI, o modelo TCP/IP também utiliza grupos de regras e protocolos chamados de camadas, porém com a diferença de utilizar somente 4 camadas, sendo elas: Acesso à Rede, Internet, Transporte e Aplicação. A Figura 2 mostra uma representação desse modelo.

Figura 2 – Modelo TCP/IP

Fonte: Academy (2022)

A Figura 3 mostra uma comparação entre as camadas do modelo OSI e TCP/IP.

Figura 3 – Modelo TCP/IP



Fonte: Academy (2022)

A Figura 3 mostra que a camada Física e a camada de enlace representam a camada de Acesso à rede e a camada de aplicação, apresentação e sessão “representam” a camada de aplicação no modelo TCP/IP. As camadas de apresentação e de sessão na verdade não estão presentes no modelo TCP/IP pois quando estavam modelando o modelo perceberam que essas camadas eram muito pouco utilizadas, porém ainda que não tenham essas camadas no modelo TCP/IP elas são mostradas como equivalentes como forma de ensino.

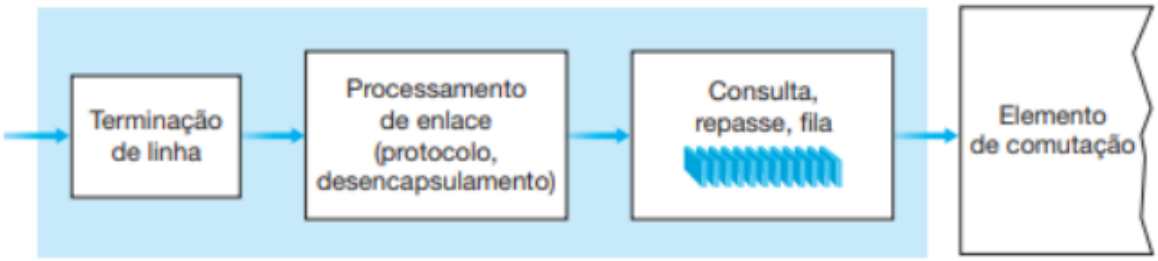
2.3 Roteadores na camada de Rede

Em redes de computadores, a finalidade é a comunicação entre dois ou mais dispositivos, e como visto nas seções anteriores, isso ocorre através de muitos protocolos e regras. Essa comunicação pode desempenhar funções diferentes, entre elas a de controle (realizados por protocolos) e de dados (e-mail, vídeo, imagem, arquivo, entre outros),e para isso ocorra, esses dados são divididos em partes menores, chamados de pacotes (um vídeo é dividido em vários pacotes quando vai ser transmitido). Para que tudo ocorra na sua devida forma, uma grande peça fundamental é o roteador.

Um roteador em geral terá como função transferir um pacote de um enlace para outro. Levando como base essa afirmação, o roteador tem como função encaminhar os pacotes entre a origem e o destino, fazendo funções de repasse (encaminhar o pacote para o próximo enlace apropriado) e roteamento (determinar a rota que o pacote irá assumir), além de tratar erros que podem vir aparecer e estabelecer conexão (comunicação entre roteadores para começarem a encaminhar os pacotes).

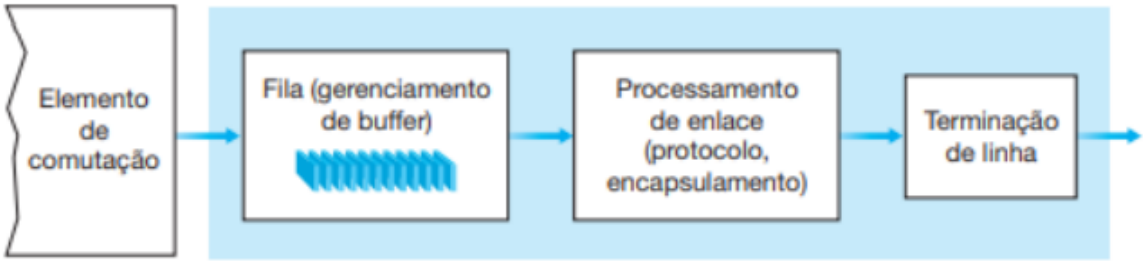
Dentro do roteador existem mecanismos que garantem que os elementos discutidos anteriormente ocorram de forma correta, para isso divide-se o roteador em duas partes, os da porta de entrada e de saída. Nas portas de entrada ocorre o processo de desencapsulamento do pacote, consulta/roteamento, repasse. Já nas portas de saída ocorre o processo de encapsulamento de pacotes. Entre essas portas é realizado o processo de comutação, que é a ligação das portas de entrada com as portas de saída, possibilitando que pacotes possam percorrer diferentes rotas até o seu destino final.

Figura 4 – Porta de Entrada



Fonte: kurose (2013)

Figura 5 – Porta de Saída



Fonte: Kurose (2013)

A Figura 4 e a Figura 5 mostram os mecanismos das portas de entrada e saída. Dentre esses mecanismos, aquele que vai ser tratado neste trabalho e comentado nas próximas seções é o da Fila e do escalonamento de pacotes (modo como os pacotes são enfileirados e tratados). Segundo Kurose “O escalonamento de pacotes desempenha um papel crucial no fornecimento de garantia de qualidade de serviço.” (KUROSE; ROSS, 2013). Visto isso, o roteador tem um papel fundamental para a garantia da qualidade de serviço, tópico que vai ser abordado na próxima seção.

2.4 Qualidade de Serviço

Quando a Internet surgiu, pouco se importava em prover qualidade no fluxo dos dados, pois era usado praticamente para acessar email e arquivos, tudo tinha a mesma prioridade, porém com os novos serviços de *streaming*, jogos online, transferência bancária online entre outros se fez necessária uma atenção maior. O modelo de serviço de melhor esforço do IP não tem garantias de desempenho para atraso, confiabilidade, *jitter*, entre outros (SHAH.; PARVEZ, 2014).

Dessa forma, se faz necessário ter algum meio que resolva os problemas de desempenho da rede, para isso pode-se encontrar soluções a partir do *Quality of Service* (QOS), que é um conjunto de técnicas e protocolos que organizam como os dados vão ser priorizados e segmentados(prioridade e classificação dos fluxos de dados). Alguns requisitos devem ser atendidos para se realizar o Qos, que são:

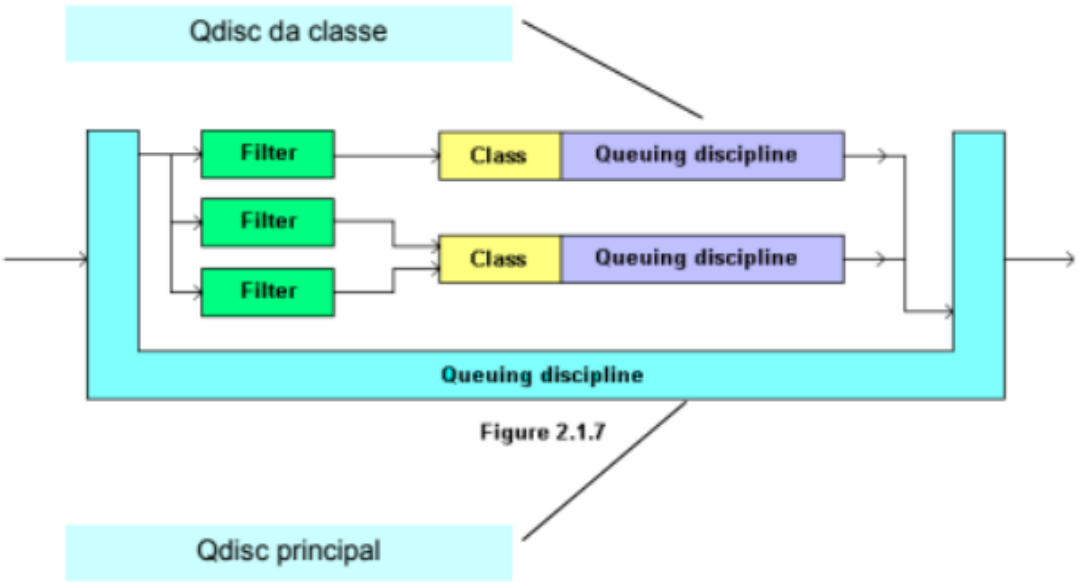
- **Confiabilidade:** garantia de entrega dos dados ao destinatário sem modificação ou erros;
- **Perda de pacotes:** quando um pacote não chega no seu destino final. Pode ter como causa o congestionamento da rede ou erro na transmissão. Pacotes corrompidos também são considerados perda de pacotes;
- **Vazão:** quantidade de bits/seg que pode ser transmitido num canal;

- **Atraso:** quantidade de tempo que um pacote leva para chegar no seu destino final. Os atrasos podem ser causados na hora da transmissão, no processamento no cabeçalho do pacote, nas disciplinas de enfileiramento (dependendo do tráfego na rede) e pelo meio que o dado está sendo propagado (par trançado, fibra óptica, entre outros).

Portanto, é importante que existam medidas para garantir a Qualidade de Serviço para que seja possível ter o melhor aproveitamento da rede e o usuário final fique satisfeito. Para garantir a Qualidade de Serviço e seus requisitos, existem algoritmos de enfileiramento de pacotes ou qdisc (*Queueing Disciplines*). Esses algoritmos servem para controlar o enfileiramento dos pacotes e como eles serão classificados, tratados ou descartados. Existem vários tipos de algoritmos e cada um deles trata os pacotes de maneiras diferentes, entre eles FIFO (*First In First Out*), HTB (*Hierarchical Token Bucket*), SFQ (*Stochastic Fair Queuing*) entre outros.

Os algoritmos de enfileiramento não funcionam sozinhos, existem outras políticas de Qos, como as Classes (classificação dos pacotes), *Filters* e *Policers*. Além disso tem o qdisc principal e de classes, a principal serve como uma base da política de Qos (estratégia a ser utilizada), e a de classe serve para classificar os pacotes. Na Figura 6 mostra o funcionamento dessa política de Qos utilizando os algoritmos de enfileiramento.

Figura 6 – Mecanismo de Qos

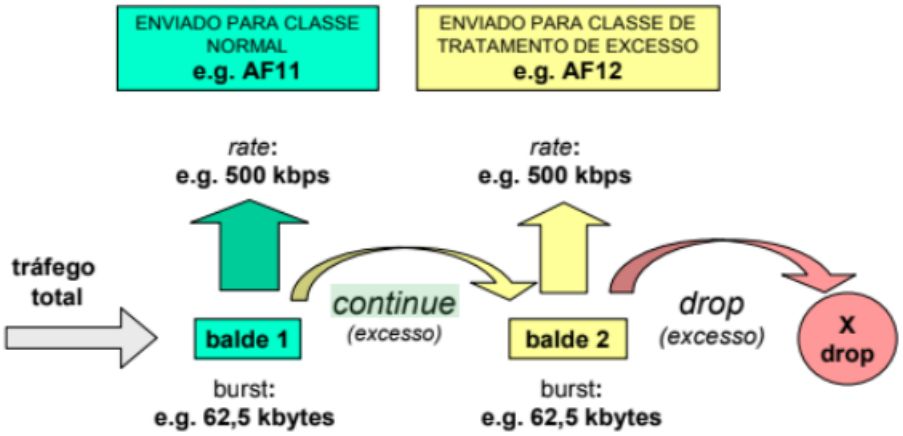


Fonte: Edgard (2015)

Quando os pacotes chegam, eles passam pelo *Policers* ou políticas de controle, garantindo o controle do tráfego. Essas políticas servem para prevenir que os pacotes não cheguem a uma velocidade muito rápida ou que superem um limite pré-definido. Para sua realização é necessário que se classifique o tráfego em três, o garantido (pacotes enviados por uma taxa de banda aceitável), o excedente (pacotes enviados por uma taxa de banda superior ao aceitável) e o violado (pacotes enviados por uma taxa de banda muito superior ao aceitável). O

tráfego excedente realiza um tratamento de excesso para esses pacotes e os pacotes violado são descartados. A Figura 7 mostra uma representação da política.

Figura 7 – Política de Qos



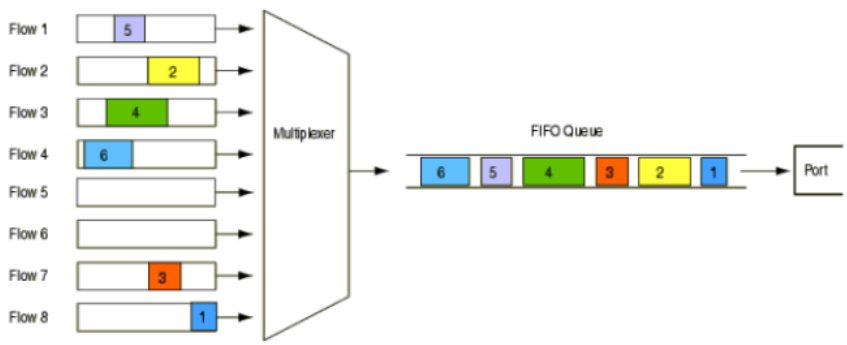
Fonte: Edgard (2015)

Após os pacotes passarem pelas políticas de controle, eles vão para os filtros que servem para classificar os pacotes e são encaminhados para diferentes tipos de classes. Quando são classificados, passam pelos algoritmos de enfileiramento de classes, que servem para determinar como e quando os pacotes (prioridade) vão ser transmitidos.

2.5 *First In First Out*

Muito utilizado em estrutura de dados, circuitos digitais, entre outros, o algoritmo de escalonamento/enfileiramento de pacotes FIFO tem como principal característica tratar os pacotes à medida que eles chegam, o primeiro a chegar é o primeiro a sair.

Figura 8 – Mecanismo FIFO



Fonte: Edgard (2015)

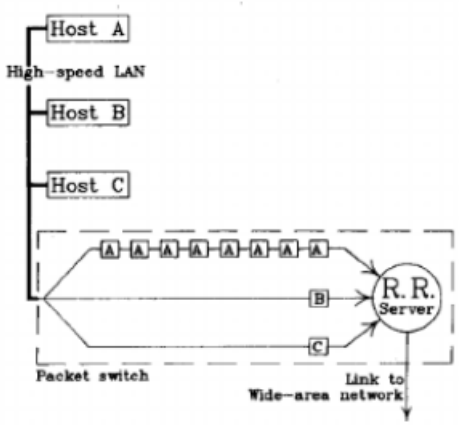
Na Figura 8, é apresentado o funcionamento deste algoritmo, os inúmeros fluxos de pacotes chegam e o multiplexador coloca esses fluxos em uma única fila na ordem que chegam. Após chegarem na fila, eles são transmitidos na ordem em que eles foram colocados na fila. Se a fila estiver lotada quando um novo fluxo de pacote chegar, esse pacote é descartado.

Esse algoritmo é considerado muito simples e eficaz até certo ponto, pois quanto mais o congestionamento do tráfego aumentar, mais o desempenho da rede será afetada por não ter uma forma de equilibrar a quantidade de banda. Se pacotes diferentes como TCP ou *User Datagram Protocol* (UDP) estiverem na mesma fila, tem grande chance de os pacotes UDP dominarem o uso da banda por ser um protocolo bem mais rápido que o TCP, além de que os mecanismos do FIFO não terem condições de lidar com isso.

2.6 *Stochastic Fair Queuing*

O algoritmo *Fair Queueing* (FQ) tem como base impedir que um único fluxo consuma mais recursos da rede que os demais fluxos. Esse algoritmo tem um mecanismo justo já que a largura de banda é dividida igualmente entre as filas, utilizando mecanismos de *round-robin* (NAGLE, 1987). A Figura 9 mostra como funciona esse algoritmo, cada host envia seus pacotes para o switch, os pacotes são enfileirados e cada vez que o algoritmo de round-robin é executado os pacotes são enviados para o *link* de saída.

Figura 9 – Fair Queueing

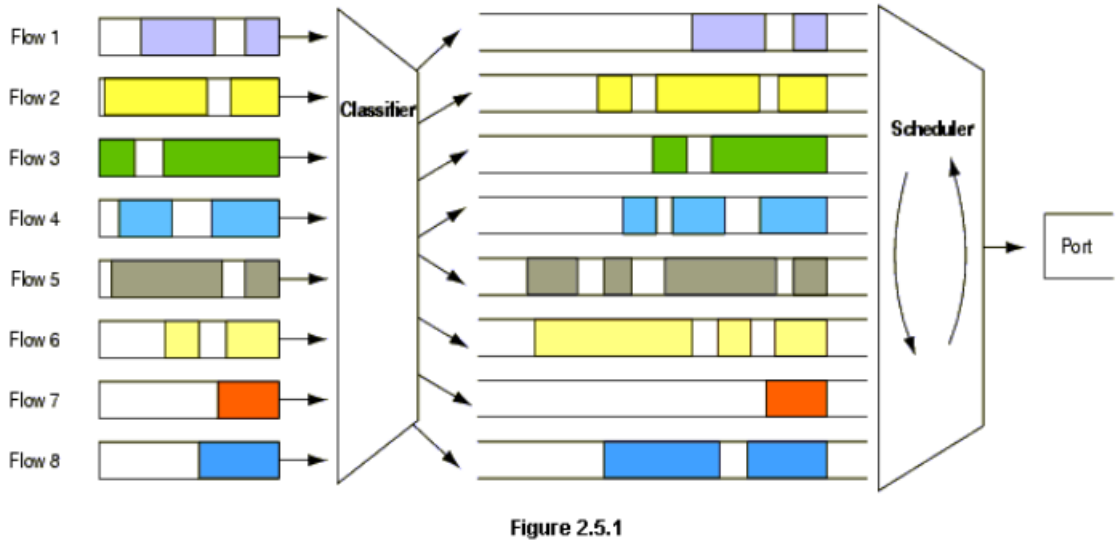


Fonte: Nagle (1987)

O algoritmo SFQ (*Stochastic Fair Queuing*) é considerado uma implementação simples da família de algoritmos FQ, que consiste em dividir o tráfego em inúmeras filas FIFO. Cada fila vai receber uma fatia de tempo para utilizar os recursos da rede (essa estratégia é chamada de Round-Robin e é muito utilizada em diversos mecanismos de escalonamento). O algoritmo SFQ é um escalonador de uma única classe, portanto a divisão do tráfego em fila corresponde às sessões TCP ou ao fluxo de segmentos UDP.

O termo “Stochastic” aparece no algoritmo pois não aloca uma única fila para cada sessão, utiliza um algoritmo hash para realizar a divisão do tráfego em um número limitado de filas (BALLIACHE, 2012). Para impedir que as sessões entrem em colisão e a eficiência do algoritmo caia, o algoritmo *hash* é periodicamente reconfigurado. A Figura 10 mostra o funcionamento do algoritmo, onde o fluxo é dividido em várias filas.

Figura 10 – Mecanismo SFQ



Fonte: Edgard (2015)

Esse algoritmo é controlado principalmente por dois parâmetros:

- ***Perturb*:** intervalo de tempo que o algoritmo hash é reconfigurado. É recomendado que seja reconfigurado a cada 10 segundos;
- ***Quantum*:** quantidade de pacotes que são removidos da fila por interação.

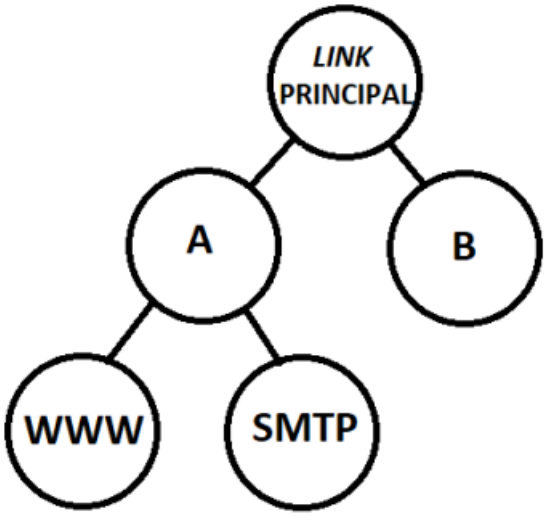
2.7 *Hierarchical Token Bucket*

O algoritmo HTB (*Hierarchical Token Bucket*) foi desenvolvido para ser um substituto do CBQ que apresentava algumas limitações e complexidades elevadas de compreensão para a sua implementação. O HTB propõe-se ser um substituto mais compreensível, intuitivo e rápido para o CBQ qdisc no Linux (LESSA, 2003).

Em comparação com o algoritmo SFQ, o HTB é uma qdisc de escalonamento de múltiplas classes filhas, como pode ser visto na Figura 11, essa figura representa o seguinte cenário: em uma rede existem dois clientes A e B ambos conectados à internet. No cliente B e A é alocado 60 kbps e 40 kbps respectivamente. Em seguida é dividido a largura de banda alocada em A em duas, 30 kbps para WWW e 10 kbps para qualquer outro serviço. Quando a largura de

banda alocada para alguma classe não estiver sendo utilizada por ela, vai ter que ficar disponível para outra classe utilizar.

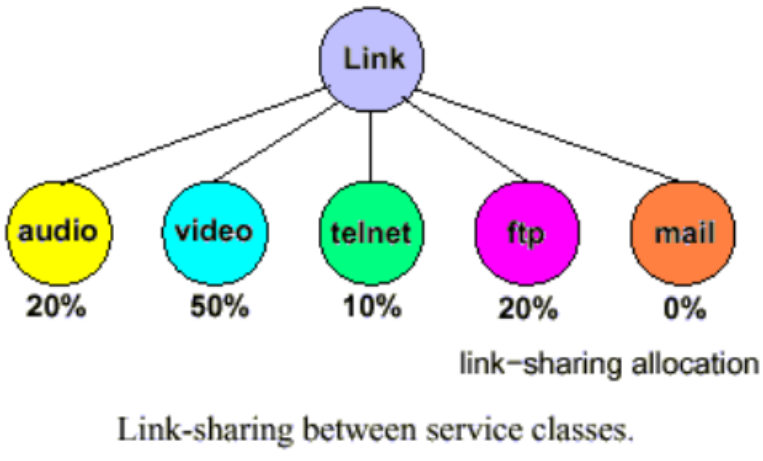
Figura 11 – Representação do Cenário



Fonte: Adaptado de Lessa (2003)

Considerando a Figura 11 e o exemplo descrito anteriormente, o HTB serve perfeitamente para essa situação pois para seu funcionamento é realizada uma divisão de banda hierárquica, onde tem o link principal com 100 kbps para dividir com dois clientes, e dependendo da classe, poderá reservar parte da banda só para ela. Portanto o HTB garante uma divisão de banda em um link entre várias classes, e quando alguma classe não está usando a sua banda garantida, deve estar disponível para outra usar. Na Figura 12 é pode-se ver a distribuição de banda entre as classes.

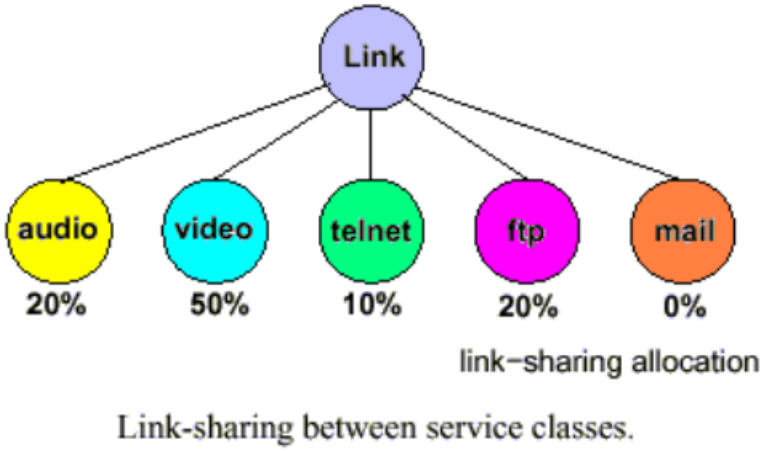
Figura 12 – Divisão de banda HTB



Fonte: Balliache (2012)

Esse algoritmo é chamado de hierárquico pois o compartilhamento da banda funciona como uma estrutura hierárquica, onde a classe “Pai” pode distribuir sua banda para as suas classes “filhas”. Na Figura 13 é possível ver como funciona a hierarquia, onde o main link distribui a banda entre 3 agências e essas agências distribuem sua banda recebida entre protocolos, classes de serviço e conexões individuais.

Figura 13 – Hierarquia HTB



Fonte: Adaptdado de Balliache (2012)

O algoritmo HTB é controlado por alguns parâmetros:

- ***Rate*:** banda garantida para uso próprio e para suas filhas;
- ***Ceil*:** banda máxima garantida que a classe pai pode emprestar;
- ***Burst*:** banda máxima que pode ser enviada pelo parâmetro ceil;
- ***Cburst*:** banda máxima que a classe pai pode enviar (quando não houver um limite estabelecido);
- ***Priority*:** prioridade de banda, a classe com maior prioridade recebe o excesso de banda primeiro (quando alguma classe não estiver utilizando sua banda garantida).

2.8 Mininet

O Mininet é um emulador de rede que possibilita a criação de *hosts*, *switches*, controladores e *links*. O intuito desse emulador é realizar testes, desenvolvimento de sistemas de redes, prototipagem, entre outras funções que possam vir a ser usadas em uma rede. Por causa do intuito de emular um sistema real, códigos desenvolvidos e testados no mininet podem ser facilmente implementados em um sistema real (MININET, 2022).

Vantagens de utilizar o Mininet:

- Possibilidade de testes na rede;
- Criação de topologias complexas;
- Inicialização rápida;
- Possibilidade de testar rede em escalas grandes (centenas de hosts);
- Fácil instalação;
- Fornece mais largura de banda dependendo do hardware;
- Conecta a redes reais;
- Desempenho interativo.

Desvantagens de utilizar o Mininet:

- Redes baseados em Mininet não podem exceder a CPU ou a largura de banda do servidor;
- Não executar certas aplicações ou equipamentos não compatíveis com Linux.

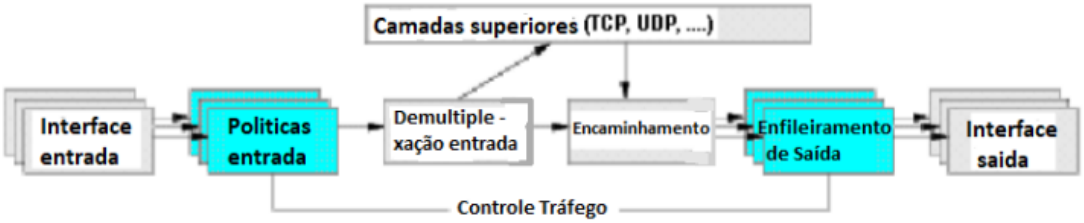
2.9 TC - *Traffic Control*

Como descrito anteriormente, a qualidade de serviço é a garantia de certos requisitos como confiabilidade, perda de pacotes, vazão e atraso. No kernel do Linux os mecanismos de QoS estão disponíveis a partir de linha de comando denominado como *Traffic Control* (TC). No controle de tráfego existem 4 componentes principais que são os algoritmos de escalonamento, as classes, filtros e políticas. Suas principais funções são:

- **Algoritmo de escalonamento:** algoritmos que decidem como os pacotes vão ser enfileirados e enviados;
- **Classes:** classificação dos pacotes;
- **Filtros:** classificar os pacotes;
- **Políticas:** controle do congestionamento, colocando limites na chegada dos pacotes;

O controle de tráfego influencia duas partes no processamento de pacotes em uma rede, no policiamento de pacotes, quando é decidido quais pacotes serão descartados (Políticas), e na fila de saída onde os algoritmos de escalonamento vão tratar como os pacotes serão enviados. Na Figura 14 pode-se ver um diagrama de como funciona o processamento de pacotes, os quadrados em azul é onde o controle de tráfego atua, entre elas estão o que determina se o pacote atual é para o nó local ou para outro e a consulta da tabela de roteamento para determinar o próximo salta do pacote.

Figura 14 – Processamento de pacotes



Fonte: Adaptdado de Balliache (2012)

Para a configuração das políticas, filtros, classes e qdisc de QoS, sempre começa com tc e em seguida o comando de identificação para adicioná-las (HUBERT, 2001). A Figura 15 mostra um exemplo de como é a sintaxe, os comandos servem para criar a qdisc principal que vai ser associada a interface eth0, após a criação a qdisc vai ter um identificador denominado *handle* e o tipo de algoritmo vai ser selecionado, que no exemplo da Figura foi o htb.

Figura 15 – Sintaxe TC

```
> tc qdisc add dev eth0
  root handle 1:0
  htb
```

Fonte: Edgard (2015)

3 DESENVOLVIMENTO

Neste capítulo é mostrado o funcionamento dos protocolos SQF e HTB, utilizando uma ferramenta presente dentro do simulador de redes Mininet, o Miniedit. Essa ferramenta é a versão gráfica simples do Mininet, podendo visualizar o ambiente de testes e configurar as máquinas, e seu funcionamento é relativamente simples para quem já tem um conhecimento prévio do mininet. Este capítulo será dividido da seguinte forma: Na seção 3.1 será abordado sobre a preparação do ambiente de testes, na seção 3.2 e 3.3 são apresentados os testes dos algoritmos SFQ e HTB respectivamente.

3.1 Preparação do ambiente de simulação

A preparação do ambiente de simulação para ambos os algoritmos são muito parecidos, diferenciando-se apenas em alguns aspectos que vão ser abordados ao decorrer deste tópico. Para ambos os testes é utilizado a mesma topologia, a mesma modificação do buffer dos hosts e a mesma limitação na taxa de transmissão no *switch*.

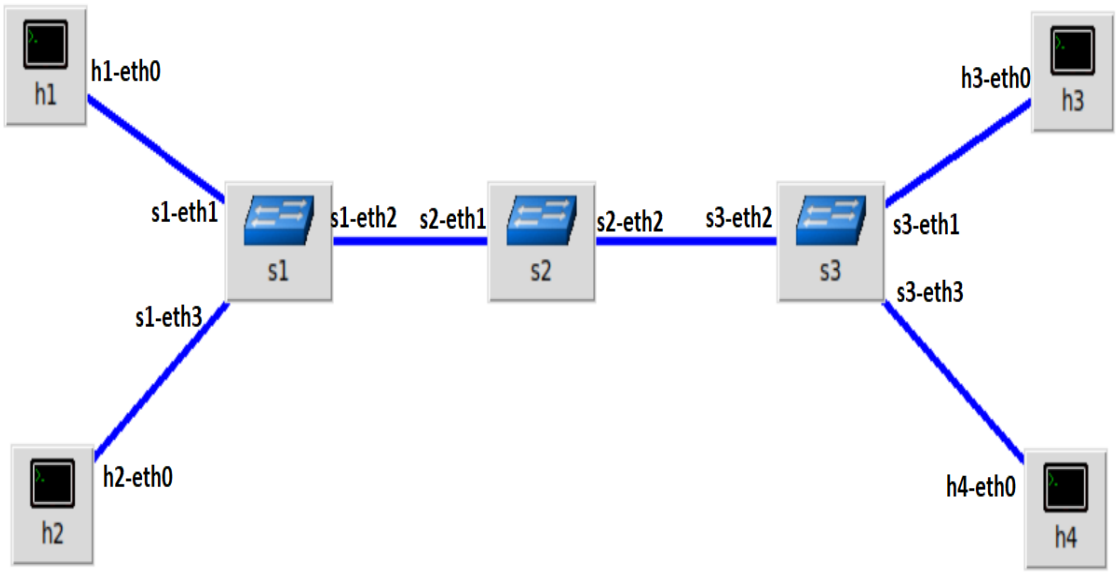
3.1.1 Topologia

Para a realização dos testes, fez-se necessário criar uma topologia cliente-servidor. Essa escolha veio pelo fato que para os testes serem mais precisos com a realidade, dois clientes vão fazer requisições para dois servidores diferentes, porém vão competir o mesmo *link*, sendo assim competindo a banda entre si. Os *hosts* 1 e 2 vão representar os clientes e os *hosts* 3 e 4 vão representar os servidor.

Para diferenciar qual a função de cada *host*, vai ser utilizado Iperf (IPERF, 2022), que é um software de teste de largura de banda, que pode servir como um gerador de tráfego para medir o desempenho da rede ou fazer com que um *host* atue em modo servidor (o iperf tem outras funções porém são essas que vão ser utilizadas no teste).

Na Figura 16, pode-se observar a representação da topologia que foi criada utilizando o miniedit, sendo mostrado qual identificador do dispositivo e qual interface está sendo utilizada. Pode-se observar que estão sendo utilizados 3 *switchs*, isso vem pelo fato que os mecanismos de Qualidade de serviço acontecem na interface de saída (esse tópico foi abordado na seção de referencial teórico, roteadores na camada de rede).

Figura 16 – Topologia



Fonte: Autoria própria (2022)

Por padrão, o Miniedit deixa o IP base da rede como 10.0.0.0/8, não vai ser alterado e o endereço da porta IP do Switch são gerenciados pelo protocolo OpenFlow. A Tabela 1 mostra o endereçamento IP da topologia criada.

Tabela 1 – Tabela Endereçamento IP

Equipamento	Endereço IP	Máscara
Host 1	10.0.0.1	255.0.0.0/8
Host 2	10.0.0.2	255.0.0.0/8
Host 3	10.0.0.3	255.0.0.0/8
Host 4	10.0.0.4	255.0.0.0/8

Fonte: Autoria própria (2022)

3.1.2 Alteração do Buffer

O conceito e a utilização dos buffer é de extrema importância quando se relaciona à área de comunicação de dados. Buffer é um espaço de memória que armazena os dados temporariamente até repassar para o sistema, isso é necessário pois muitas vezes a velocidade que os dados vão chegando é mais rápida do que são processados. No desenvolvimento deste trabalho foi modificado o buffer TCP.

O buffer TCP é dividido em dois, o buffer de envio e o de recebimento. Os pacotes presentes no buffer ainda não foram processados. O tamanho do buffer é um fator importante, pois quando ocorre uma comunicação prévia entre o cliente e o servidor, o tamanho do buffer é informado para ambos os lados terem ciência da quantidade de dados que podem enviar

para o outro sem sobrecarregar. Durante o envio e recebimento de dados o buffer pode ser ocasionalmente alterado se for necessário (caso o buffer esteja cheio e não estája dando conta de processar todos os dados).

Para a realização dos testes, o buffer precisará ser alterado para que não ocorra uma sobrecarga na rede, onde o volume do fluxo de dados é maior do que a rede atual permite, fazendo assim que o fluxo seja prejudicado ou até interrompido, esse conceito é definido como *bottlenecks*. No linux, por padrão o tamanho do buffer é 8 Mbytes podendo se estender para no máximo de 16 Mbytes.

A modificação do buffer ocorrerá da seguinte forma: a taxa de transferência do switch vai ser de 1 GB e o *delay* vai ser de 20ms (nas próximas sessões, será abordado a taxa de transferência e o *delay*), sendo assim, a quantidade de dados que será transmitidos considerando o atraso será de 1GB * 0.02, que equivale a 20 milhões de bits ou 2.5 Mbytes. Para garantir que não ocorra *bottleneck* nos *Hosts* finais, o tamanho do buffer tcp vai ser 8*2.5 Mbytes, sendo assim 20.971.520 bytes.

Na Figura 17, é mostrado a alteração do buffer de envio e recebimento tcp. Para alterar foi utilizado o sysctl (STAIKOS, 2020), que é um software presente no linux que permite modificar alguns atributos no kernel. Ambos os comandos são parecidos, modificando apenas o buffer que vai ser modificado, o *rmem* é o buffer de recebimento e o *wmem* é o buffer de envio. Os valores 10240, 87380 e 20971520 representam o mínimo, padrão e máximo que o tamanho do buffer pode ter.

Figura 17 – Alterando o buffer

Fonte: Autoria própria (2022)

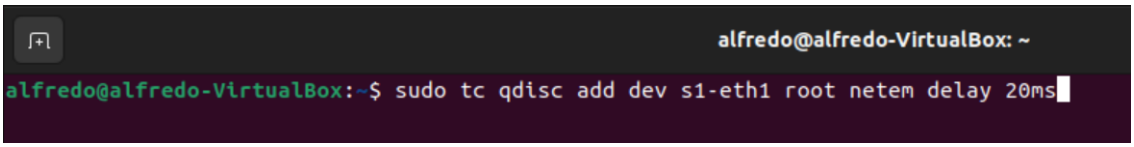
Na figura mostra alterando somente o *host 1*, porém todos os *hosts* foram alterados.

3.1.3 Adicionando *delay*

A latência tem grande significado quando se quer verificar a qualidade na velocidade da internet, pois esse termo pode ser definido como uma variação de tempo entre o começo de uma trasmissão, e a resposta para essa transmissão. Um comando altamente conhecido para verificar a latência é o ping. Esse comando utiliza o protocolo ICMP que é integrante do Protocolo IP (qualquer computador que utiliza o protocolo Ip, também utiliza o ICMP), com isso verifica-se o tempo para os dados enviados chegarem ao seu destino e voltarem.

Para tentar simular uma situação real, terá um aumento de latência na interface 1 do switch 1 (s1-eth1). Agora os dois *hosts* estarão competindo entre si estando em situações diferentes. Na Figura 18, mostra o comando, ele foi usado no terminal linux enquanto o miniedit estava rodando, fazendo assim que o dispositivo s1-eth1 fosse detectado e pudesse ser alterado.

Figura 18 – Adicionando Delay



Fonte: Autoria própria (2022)

O comando utiliza o *traffic control*, nele é adicionado um *delay* de 20 ms no dispositivo s1-eth1.

3.1.4 Limitando a taxa de transferência

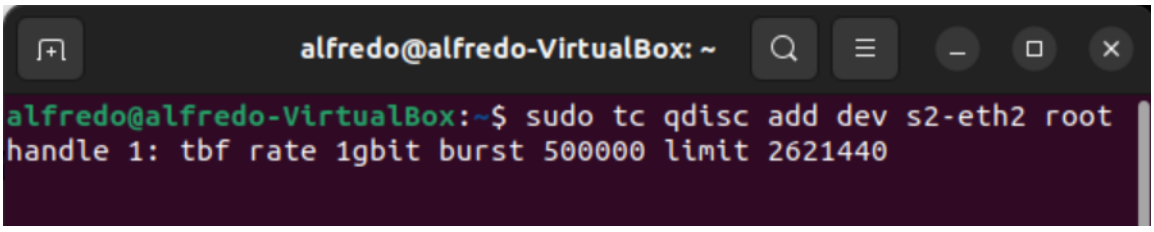
Com o intuito de realizar os testes, a taxa de velocidade do switch 2 será limitada em 1 Gbps. Para isso o *Traffic Control* será usado junto com o algoritmo TBF. Essa limitação será implementada na interface 2 do switch 2 (s2-eth2).

Os parâmetros a serem utilizados:

- ***Rate***: velocidade de transmissão;
- ***Burst***: quantidade de bytes que a fila de pacotes possui;
- ***Limit***: tamanho máximo de bytes que serão enviados para a rede por vez.

A Figura 19 mostra utilizando o comando no terminal do linux.

Figura 19 – Limitando a velocidade



Fonte: Autoria própria (2022)

3.1.5 Algoritmo de controle de congestionamento - SFQ

O protocolo TCP utiliza uns algoritmos de controle de congestionamento focado nos *hosts* finais. Esses algoritmos são considerados a prevenção primária no controle do congestio-

namento, adicionado com as inúmeras outras técnicas de prevenção, isso auxilia para que não ocorra *bottleneck* em algum dos links.

No teste do SFQ, quando se utiliza o algoritmo é esperado que o consumo de banda nos clientes se equilibre, e quando não se utiliza, mantém-se desequilibrada. Para garantir que ocorra esse desequilíbrio de consumo de banda, um dos dois *hosts* que representam os clientes vão ter seu algoritmo de controle de congestionamento alterado.

Por padrão, o algoritmo de controle de congestionamento no linux é o Cubic. Um dos *hosts* vai usar esse algoritmo padrão e o outro o BBR. Na Figura 20, verifica-se que o algoritmo *Cubic* está sendo utilizado e na Figura 21 é modificado o algoritmo para o BBR.

Figura 20 – H1 - Cubic

Fonte: Autoria própria (2022)

Figura 21 – H2 - BBR

Fonte: Autoria própria (2022)

3.2 Teste com algorItmo SFQ

Para obter maior visibilidade do algoritmo funcionando, dois tipos de testes vão ser realizados, o teste sem e com o algoritmo. Como primeiro passo, serão definidos os dois *hosts* clientes e os dois *hosts* servidores, após isso os clientes vão ser preparados para utilizar o iperf3 e poder analisar o trafego na rede.

Os *Hosts* 3 e 4 vão ser os servidores, para isso vai ser utilizado o Iperf3 para transformá-los utilizando o comando iperf3 -s. A Figura 22 mostra a utilização do comando no *host* 3, porem foi feito também no *host* 4.

Figura 22 – Iperf3 Server

Fonte: Autoria própria (2022)

Com os servidores definidos, vai ser criado um diretório para armazenar os resultados da análise do tráfego. O cliente 1 vai enviar pacotes para o servidor 3, e o cliente 2 vai para o servidor 4. O analisar de tráfego Iperf3 permite exportar os resultados em um arquivo JSON, com isso é utilizado o comando iperf3 -c 10.0.0.3 -t 60 -J > out.json no cliente 1 e 2 simultaneamente, só modificando o IP. A Figura 23 e a Figura 24 é demonstrado como foi realizado e utilizado os comandos para realizar os testes.

Figura 23 – Comando Teste sem SFQ H1

Fonte: Autoria própria (2022)

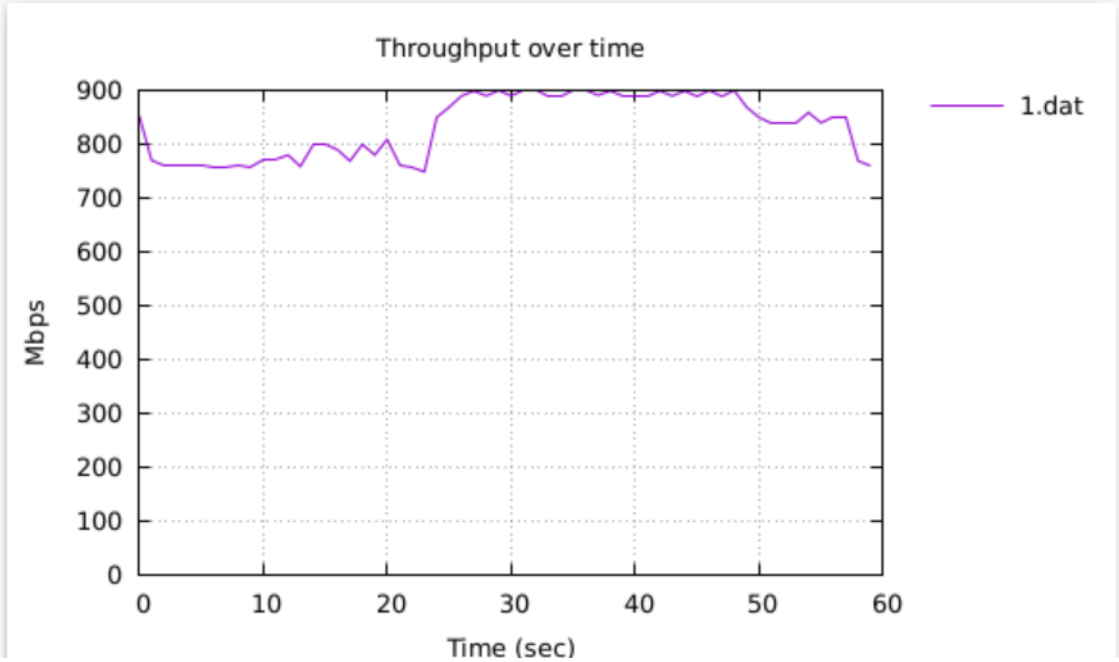
Figura 24 – Comando Teste sem SFQ H2

Fonte: Autoria própria (2022)

Para plotar os gráficos, é utilizado o plot iperf, que é um script em shell. Ele recebe como entrada o arquivo JSON e deixa como saída um arquivo pdf. Após plotar os arquivos para ambos resultados dos clientes 1 e 2, é possível observar os resultados. Nos gráficos de resultados, o eixo y representa a banda consumida e o eixo x representa o tempo.

O Gráfico 1 representa o resultado da análise do *Host* 1 sem o algoritmo implementado, como resultado percebe-se que quase a banda total de 1 Gbps é consumida no processo durante o teste.

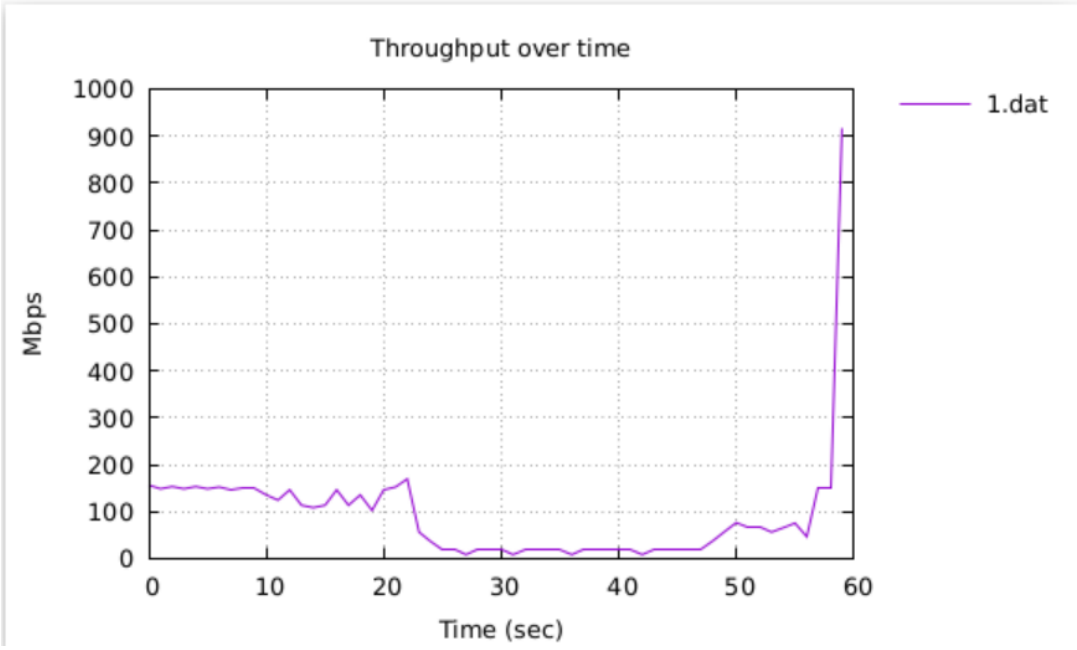
Gráfico 1 – Teste H1 sem SFQ



Fonte: Autoria própria (2022)

O Gráfico 2 representa o resultado da análise do *Host* 2 sem o algoritmo implementado, como resultado percebe-se que não consumiu quase nada de banda, pois o *Host* 1 já estava consumindo quase tudo.

Gráfico 2 – Teste H2 sem SFQ



Fonte: Autoria própria (2022)

Como ambos os testes foram executados quase no mesmo tempo, tendo uma diferença de iniciação de poucos segundos, quando o cliente 1 terminou a análise, o consumo de banda

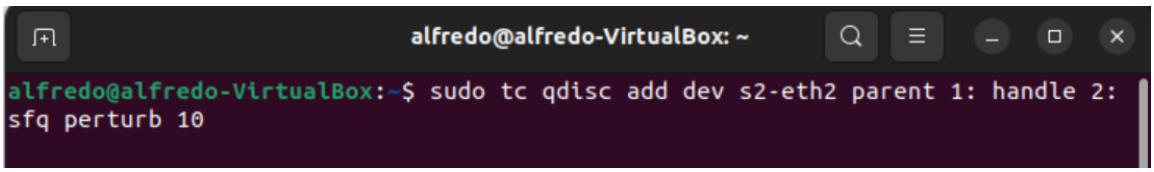
aumentou para o cliente 2, pois o *link* de 1 Gbps não estava tendo competição de banda, podendo ser exclusivo para o cliente 1. Percebe-se nos testes apresentados que o consumo de banda não é justo, sendo o *host* 1 consumindo 90% da banda e deixando apenas 10% para o *host* 2.

A ideia de utilizar o algoritmo SFQ é para resolver o problema de consumo de banda desigual, os próximos passos mostram a implementação desse algoritmo e consecutivamente os resultados do teste. A ideia de 2 clientes e 2 servidores permanece nesse teste.

Para implementar o algoritmo, é utilizado como descrito anteriormente no trabalho o comando de *Traffic Control* no linux. O comando em questão é sudo tc qdisc add dev s2-eth2 parent 1: handle 2: sfq perturb 10. Nesse comando é invocado uma qdisc na interface do Switch 2 (s2-eth2). Foi adicionado anteriormente um limite de banda de 1 Gbps na interface 2 do Switch 2, tendo em vista que o comando tc funciona hierarquicamente, começando pela raiz root (no caso foi o limite de bada), essa qdisc do SFQ é conectada com a raiz (parent 1) e agora representa o 2 elemento dessa hierarquia (handle 2). Após isso é explicitado que vai ser usado o algoritmo SFQ com um perturb de 10 (seção 2.6).

A Figura 25 mostra a utilização do comando no terminal do Linux.

Figura 25 – Comando SFQ

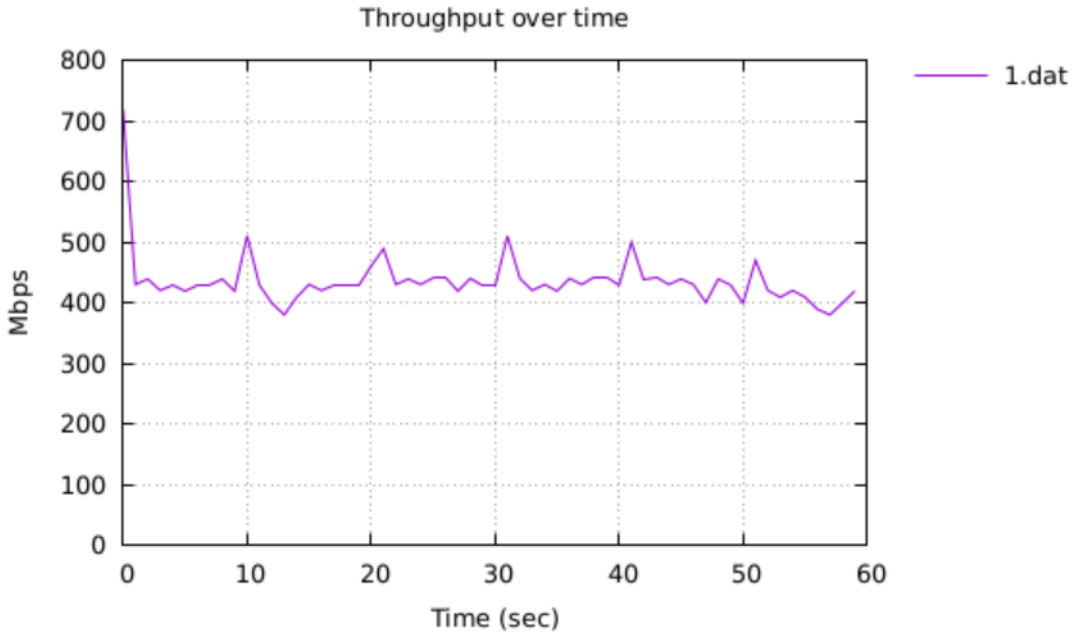


Fonte: Autoria própria (2022)

Dando sequência aos testes com o controle de tráfego implementado, as mesmas configurações do teste anterior vão ser usadas. Os clientes vão ser os *hosts* 1 e 2 e os servidores os *hosts* 3 e 4. Executando o iperf3 nos dois clientes ao mesmo tempo e plotando os gráficos são obtido os resultados.

O Gráfico 3 representa o resultado da análise do *Host* 1 com o algoritmo implementado, como resultado percebe-se que o consumo de banda ficou em torno dos 450 Mbps.

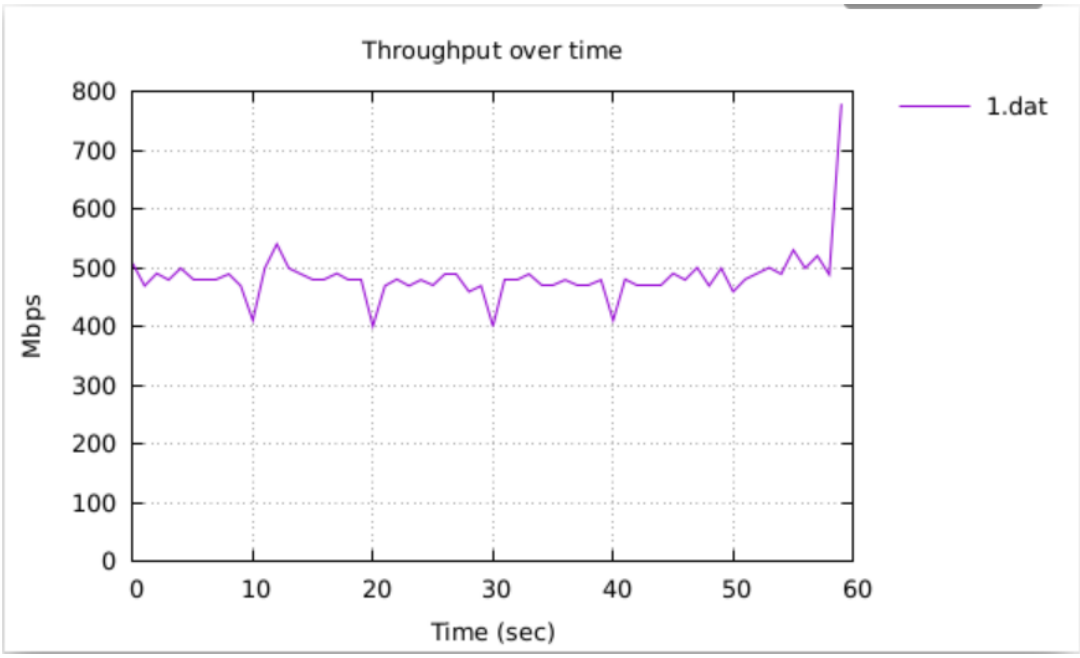
Gráfico 3 – Teste H1 com SFQ



Fonte: Autoria própria (2022)

O Gráfico 4 representa o resultado da análise do *Host* 2 com o algoritmo implementado, como resultado percebe-se que o consumo de banda ficou por volta dos 500 Mbps.

Gráfico 4 – Teste H2 com SFQ



Fonte: Autoria própria (2022)

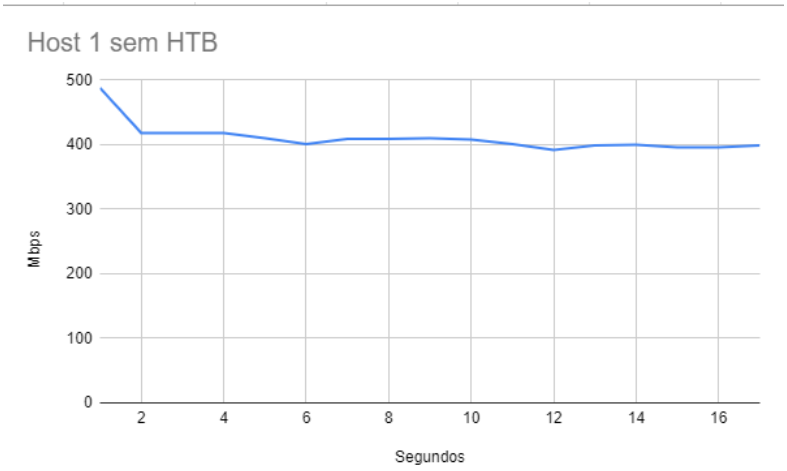
Por fim, percebe-se por meio dos testes com o algoritmo implementado que o consumo de banda se igualou e ambos os clientes tiveram uma distribuição justa de banda, sem ter o problema que se verifica no primeiro teste, em que quase toda a banda era consumido por um só cliente.

3.3 Teste com algoritmo HTB

De forma semelhante como foram realizados os testes na seção 3.2, serão realizados dois testes, um fazendo o uso do algoritmo e outro não. Utilizando a ideia de cliente-servidor com os *hosts* 1 e 2 sendo os clientes e os *hosts* 3 e 4 sendo os servidores. A Figura 22, Figura 23 e a Figura 24 representam procedimentos semelhantes ao usados nesta seção.

Como primeiro teste sem o algoritmo implementado, vai ser executado o iperf3 nos clientes 1 e 2 ao mesmo tempo, e os resultados serão obtidos. No Gráfico 5 representa o resultado da análise do *Host* 1 sem o algoritmo implementado, e como resultado percebe-se que fica em torno dos 400 Mbps.

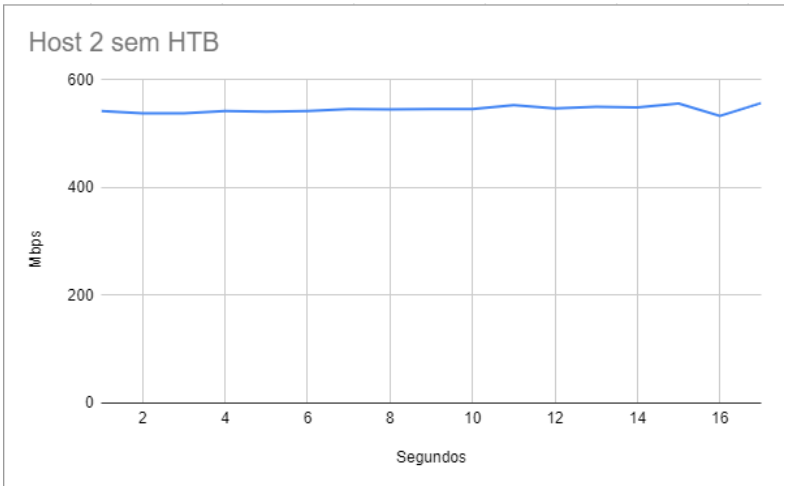
Gráfico 5 – Teste H1 sem HTB



Fonte: Autoria Própria (2022)

O Gráfico 6 representa o resultado da análise do *Host* 2 sem o algoritmo implementado, e como resultado percebe-se que o consumo de banda ficou por volta dos 550 Mbps.

Gráfico 6 – Teste H2 sem HTB



Fonte: Autoria própria (2022)

Com o teste feito, o consumo de banda acontece normalmente sem nenhum controle, utilizando o algoritmo de controle de tráfego HTB, a ideia é ter um controle da quantidade de banda que cada cliente pode consumir.

Sendo assim, com o intuito de implementar o algoritmo HTB e ter maior controle do consumo de banda dos clientes, é utilizado o comando de *Traffic Control* no terminal do Linux. Para isso é utilizado uma série de comandos para criar as classes e os filtros do algoritmo. A Figura 26 representa os comandos utilizados para os testes.

Figura 26 – Comando HTB

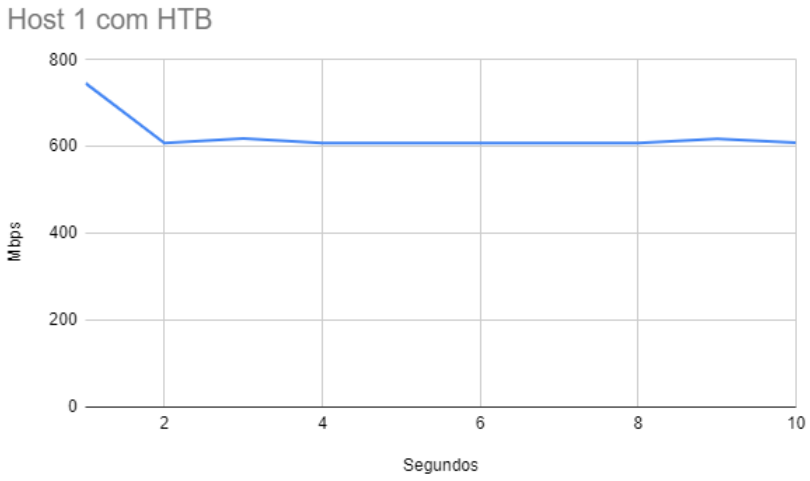
```
sudo tc class add dev s2-eth1 parent 1:0 classid 1:1 htb rate 1gbit ceil 1gbit
sudo tc class add dev s2-eth1 parent 1:1 classid 1:10 htb rate 700mbit ceil 1gbit
sudo tc class add dev s2-eth1 parent 1:1 classid 1:20 htb rate 300mbit ceil 1gbit
sudo tc filter add dev s2-eth1 protocol ip parent 1:0 prio 1 u32 match ip src 10.0.0.1 flowid 1:10
sudo tc filter add dev s2-eth1 protocol ip parent 1:0 prio 1 u32 match ip src 10.0.0.2 flowid 1:20
```

Fonte: Autoria própria (2022)

Todos os parâmetros foram explicados na seção 2.7. Na Figura acima, o primeiro comando representa a criação da classe raiz, onde ela terá direito de 1 Gbps de banda para uso próprio e para passar para suas classes filhas. O segundo e terceiro comando é a criação das classes filhas, onde elas herdarão 700 Mbps e 300 Mbps da classe pai respectivamente. E por último, o quarto e quinta comando representam o filtro, onde a filtragem vai ser feita por IP. Sendo assim, quando um cliente com esse IP estiver transmitindo, eles irão para a classe que este filtro está indicando. Para os testes, o cliente 1 ele representa a classe dos 700 Mbps e o cliente 2 para a classe dos 300 Mbps.

No Gráfico 7 representa o resultado do *Host* 1 com o algoritmo implementado, e como resultado percebe-se que o consumo de banda fica em torno dos 600 Mbps.

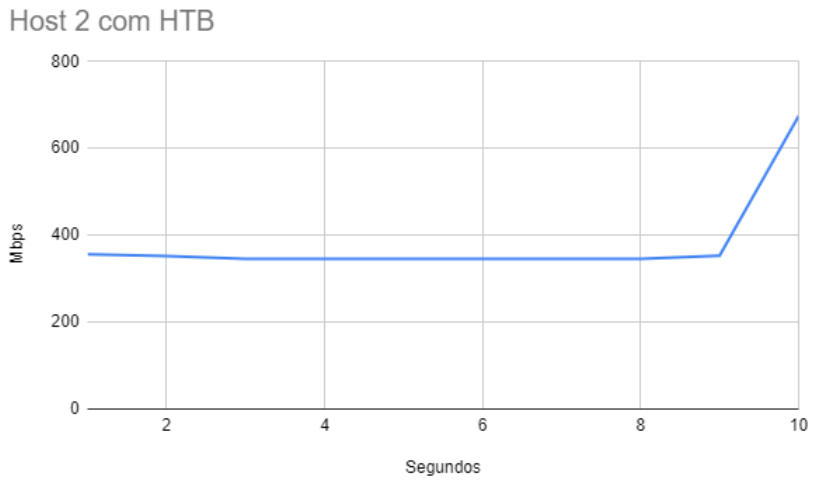
Gráfico 7 – Teste H1 com HTB



Fonte: Autoria própria (2022)

No Gráfico 8 representa o resultado do *Host* 2 com o algoritmo implementado, e como resultado percebe-se que o consumo de banda fica em torno dos 350 Mbps.

Gráfico 8 – Teste H2 com HTB



Fonte: Autoria própria (2022)

Ainda que os clientes ficaram um pouco abaixo ou acima dos parâmetros definidos, é possivel ter uma percepção do funcionamento do algoritmo, no primeiro teste sem o HTB o *Host* 2 teve mais consumo de banda que o *Host* 1. Em contrapartida, no segundo teste ele ficou nas proximidades do consumo de banda que foi determinado para ele poder usar, e o *Host* 1 teve seu consumo aumentado. Sendo assim, é possível ter controle de quanto cada cliente pode consumir de banda.

4 CONCLUSÃO

No decorrer do desenvolvimento deste trabalho, pode-se concluir que a utilização de mecanismos de controle de tráfego pode auxiliar o modo que os recursos da rede operam, podendo ser justa distribuição de banda, ou podendo selecionar o ambiente que necessita de mais recurso para o funcionamento.

Isso se confirma no trabalho de MACHADO e MOREIRA (2012), em que o uso de mecanismo de QoS, conseguiu resolver um problema de banda numa pequena empresa, e para eles, "O uso adequado do controle de tráfego permite uma maior utilização dos recursos da rede previsível e uma disputa menos volátil por esses recursos.".

Durante este trabalho foi apresentada a ideia de Qualidade de Serviço e como ela é importante atualmente. Foi explicado seu funcionamento na camada de rede e foram apresentados dois mecanismos, SFQ e HTB. Em seguida, foi desenvolvida uma topologia e configurado um ambiente de teste por meio simulador, promovendo a possíbilidade de realizar alguns testes com e sem os mecanismos de controle de tráfego implementados para medir a vazão na rede.

Além dos dois mecanismos de controle de tráfego apresentados, o SFQ e o HTB, existem muitos outros; porém com a finalização do desenvolvimento deste trabalho percebe-se que cada mecanismo pode ser considerado situacional, podendo vir de acordo com a necessidade do cliente. Ainda que somente dois algoritmos de Qos foram apresentados e estudados, é possível ter um entendimento e noção do seu funcionamento e dos seus benefícios para a rede.

As dificuldades que pode-se observar foram principalmente encontrar referenciais teóricos para explicar o funcionamento dos mecanismos, pois a maioria do conteudo era relaciona a implementação utilizando os mecanismos de *Traffic Control* no Linux.

Como limitação encontrada neste trabalho foi a falta de computadores físicos, pois seria interessante além de mostrar os resultados num simulador, mostrar o funcionamento dos mecanismos de QoS numa rede Física, criando uma mini Topologia com computadores reais.

REFERÊNCIAS

ACADEMY, C. N. 2022. Disponível em: https://slideplayer.com.br/amp/1862502/. Acesso em: 22 jun. 2022.

BALLIACHE, L. **Practical IP Network Qos.** 2012. Disponível em: http://softwareopal.com/qos/default.php?p=linux101-ds. Acesso em: 26 maio 2022.

EDGARD, J. **Mecanismos de QoS em Linux tc - Traffic Control**. 2015. Disponível em: https://www.ppgia.pucpr.br/~jamhour/Pessoal/Mestrado/TARC/QoSLinux.pdf. Acesso em: 07 set. 2021.

HUBERT, B. **tc(8) — Linux manual page**. 2001. Disponível em: https://man7.org/linux/man-pages/man8/tc.8.html. Acesso em: 14 out. 2022.

IPERF. 2022. Disponível em: https://iperf.fr/iperf-doc.php#3doc. Acesso em: 14 out. 2022.

KUROSE, J. F.; ROSS, K. **Redes de computadores e a Internet**. 6. ed. São Paulo: Campus, 2013.

LESSA, C. V. B. **Manual Traduzido do Controle de Banda com HTB**. 2003. Disponível em: https://br-linux.org/tutoriais/001794.html. Acesso em: 30 maio 2022.

LUCAS, G. **Star Wars: Episódio I – A Ameaça Fantasma**. [*S.l.*]: Lucasfilm, 1999.

MACEDO, R. *et al.* **REDES DE COMPUTADORES**. 1. ed. Santa Maria: UFSM, 2018.

MACHADO, J. C.; MOREIRA, J. Controle de tráfego no linux - aplicação a uma empresa de médio porte. **Revista T.I.s**, São Carlos, p. 121–129, set 2012.

MININET. 2022. Disponível em: http://mininet.org/l. Acesso em: 1 jun. 2022.

NAGLE, J. On packet switches with infinite storage. **Communications, IEEE Transactions on**, p. 435–438, 1987.

SHAH., J. L.; PARVEZ, J. Evaluation of queuing algorithms on qos sensitive application in ipv6 network. **International Conference on Advances in Computing, Communications and Informatics (ICACCI)**, p. 106–111, 2014.

STAIKOS, G. **sysctl(8) — Linux manual page**. 2020. Disponível em: https://man7.org/linux/man-pages/man8/sysctl.8.html. Acesso em: 14 out. 2022.

TANENBAUM, A. S. **Redes de Computadores**. 4. ed. Rio de Janeiro: Campus, 2003. 19 p.

TANENBAUM, A. S. **Redes de Computadores**. 5. ed. São Paulo: Pearson, 2011. 26 p.

APÊNDICE A – Scripts dos algoritmos de controle de tráfego

A Listagem 1 e a Listagem 2 representam um script Shell dos comandos de *Traffic Control* para configurar os mecanismos de SFQ e HTB respectivamente.

Listagem 1 – Script SFQ

```
1  #!/bin/bash
2  sudo tc qdisc add dev s1-eth1 root netem delay 20ms
3  sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate 1gbit burst 500000
4  limit 2621440
5  sudo tc qdisc add dev s2-eth2 parent 1: handle 2: sfq perturb 10
```

Fonte: Autoria própria (2022)

Listagem 2 – Script HTB

```
1  #!/bin/bash
2  sudo tc qdisc add dev s1-eth1 root netem delay 20ms
3  sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate 1gbit burst 500000
4  limit 2621440
5  sudo tc qdisc add dev s2-eth1 root handle 1: htb
6  sudo tc class add dev s2-eth1 parent 1:0 classid 1:1 htb rate 1gbit
7  ceil 1gbit
8  sudo tc class add dev s2-eth1 parent 1:1 classid 1:10 htb rate 700mbit
9  ceil 1gbit
10 sudo tc class add dev s2-eth1 parent 1:1 classid 1:10 htb rate 300mbit
11 ceil 1gbit
12 sudo tc filter add dev s2-eth1 protocol ip parent 1:0 prio 1 u32 match ip src
13 10.0.0.1 flowid 1:10
14 sudo tc filter add dev s2-eth1 protocol ip parent 1:0 prio 1 u32 match ip src
15 10.0.0.2 flowid 1:20
```

Fonte: Autoria própria (2022)