Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Estudo Comparativo de Variantes TCP em Redes Sem Fio IEEE 802.11

Rafael Braga Ladeira Dutra

JUIZ DE FORA
FEVEREIRO, 2022

Estudo Comparativo de Variantes TCP em Redes Sem Fio IEEE 802.11

Rafael Braga Ladeira Dutra

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Edelberto Franco Silva

JUIZ DE FORA

FEVEREIRO, 2022

Estudo Comparativo de Variantes TCP em Redes Sem Fio IEEE 802.11

Rafael Braga Ladeira Dutra

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Edelberto Franco Silva
Doutor em Computação

Eduardo Pagani Julio
Doutor em Computação

Luciano Jerez Chaves
Doutor em Computação

JUIZ DE FORA
17 DE FEVEREIRO, 2022

Resumo

Ter uma conexão estável hoje em dia é de grande necessidade devido ao constante aumento do uso das tecnologias no dia-a-dia, seja para lazer, trabalho ou estudo. Junto a esse aumento no uso, o desafio de se ter uma conexão de qualidade se torna maior, pois com mais usuários conectados e uma maior demanda da rede, ocorrem lentidões. Uma possível forma de melhoramento da rede é com a modificação do algoritmo de implementação do Protocolo de Controle de Transmissão (TCP), pois esse protocolo é responsável por fazer um controle de congestionamento na rede. Ao utilizar a variante correta desse algoritmo para atender a necessidade do usuário, como por exemplo, o modo como o mesmo utiliza a Internet, e também o tipo de rede (sem fio, cabeada, fibra ótica), é possível obter uma melhora na conexão, seja na velocidade, na estabilidade, ou em ambos. Este trabalho busca estudar o protocolo TCP e suas variantes, para, através de testes por simulação, encontrar quais algoritmos apresentam melhores resultados em um ambiente de rede sem fio local.

Palavras-chave: Redes de Computadores, Protocolo TCP, Controle de Congestionamento.

Abstract

Nowadays, having a stable connection is of great necessity due to the constant increase in technologies in everyday life, whether for leisure, work, or study. Along with this increase in use, the challenge of having a quality connection becomes greater because slowdowns occur with more connected users and greater network demand. A possible way to improve the network is by modifying the Transmission Control Protocol (TCP) implementation algorithm, as this protocol is responsible for controlling congestion on the network. It is possible to improve the connection using this algorithm's correct variant to meet users' needs. Whether in speed, stability, or both. This work seeks to study the TCP protocol and its variants through simulation tests, which algorithms present better results in a local wireless network environment.

Keywords: Computer Network, TCP Protocol, Congestion Control.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ACK	*Acknowledgement* - Confirmação
BIC	*Binary Increase Congestion Control*
CWND	*Congestion Window* - Janela de Congestionamento
DCC	Departamento de Ciência da Computução
DHCP	Protocolo de Configuração Dinâmica de Endereços de Rede
DNS	*Domain Name System* - Sistema de Nome de Domínio
HTTP	*Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto
ICMP	*Internet Control Message Protocol* - Protocolo de Mensagens de Controle da Internet
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IGMP	*Internet Group Message Protocol* - Protocolo de Mensagens de Grupo da Internet
IP	*Internet Protocol* - Protocolo de Internet
MAC	*Media Access Control* - Controle de acesso ao meio
NS-3	*Network Simulator* 3 - Simulador de Rede
ONU	Organização das Nações Unidas
OSI/ISO	Interconexão de Sistemas Abertos/Organização Internacional para Padronização
PPP	*Point-to-Point Protocol* - Protocolo ponto-a-ponto
QoS	*Quality of Service* - Qualidade de Serviço
RFC	*Request for Comments* - Pedido para Comentários
RTT	*Round Trip Time* - Latência
SSH	*Secure Shell* - Shell Segura
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
UDP	*User Datagram Protocol* - Protocolo de Datagrama do Usuário
UFJF	Universidade Federal de Juiz de Fora

1 Introdução

Com o crescimento cada vez maior do número de usuários, e também do uso da tecnologia da área de redes, muitas vezes podem ocorrer lentidões ou problemas de congestionamento na rede. Isso se torna indesejável nos tempos atuais, já que dependemos cada vez mais da tecnologia de redes de comunicação em diversas situações, necessitando de conexões estáveis, rápidas e seguras em diversos casos.

Em Redes de Computadores, vários protocolos existem para permitir que dois ou mais computadores se comuniquem entre si, como, HTTP (Protocolo de Hipertexto), SSH (*Secure Shell*), DNS (*Domain Name Service*), TCP (*Transmission Control Protocol*), entre outros. Esses protocolos são organizados por camadas: camada de aplicação, camada de transporte, camada de rede, camada de enlace e camada física. O protocolo TCP (Protocolo de Controle de Transmissão), do Inglês *Transmission Control Protocol*, é um protocolo da camada de transporte responsável por verificar o tráfego da rede, controlando o envio e recebimento de pacotes. Além disso, esse protocolo certifica-se de que essa troca de pacotes esteja ocorrendo sem perdas, sem erros e em ordem para futura remotagem na camada superior (*i.e.*, Camada de Aplicação). O protocolo TCP junto com o protocolo IP (*Internet Protocol*) (um protocolo da camada de rede), formam o conjunto de protocolos de comunicação chamado de TCP/IP, um modelo indispensável para a Internet atualmente. Além disso, o TCP é um protocolo que visa auxiliar na comunicação da rede para tratar desafios como perdas, erros e atrasos intrínsecos de uma rede de melhor esforço, ou seja, sem garantias, como é a Internet.

Com o passar do tempo, o protocolo TCP foi evoluindo e foram surgindo novas implementações do mesmo, com diferentes objetivos. Por exemplo, diminuir a perda de pacotes, aumentar a estabilidade da rede, aumentar a velocidade de transmissão, entre outros. Temos diversas variantes do TCP, como Cubic, *New Reno*, *HighSpeed*, entre outras. Basicamente, esses protocolos mantém as funções e princípios do TCP, e modificam a forma como o controle de congestionamento é tratado pelo algoritmo.

Outro desafio é a gestão da conexão fim-a-fim do TCP sobre ambientes instáveis,

como é o caso das redes sem fio locais. As redes sem fio locais são apoiadas sobre o padrão IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) 802.11. Neste padrão há diversas emendas, seja para melhorarem a vazão, cobertura, ou segurança da rede, como são os casos das emendas *n, g, i.* Assim, para melhorar o desempenho das redes – principalmente as sem fio IEEE 802.11 – é necessário o estudo aprofundado das várias implementações do protocolo TCP. Isso é possível através de simulações ou em ambiente real. Neste trabalho avaliamos por simulação diversas variantes do TCP, a fim de conseguirmos definir qual implementação deve ser utilizada para redes sem fio IEEE 802.11.

1.1 Motivação

Os sistemas operacionais atualmente já vêm com uma variante do TCP predefinida habilitada em suas configurações, porém esse tema não é bem difundido para a grande maioria dos usuários, os quais não sabem da possibilidade da alteração dessa variante, e essa alteração se torna um desafio ao usuário quando não há avaliações e resultados sobre variantes TCP.

Assim, o estudo e os testes das diferentes variantes visam difundir essa informação para que usuários e empresas possam modificar essa configuração e então, otimizar sua conexão baseado em qual tipo de rede ele está utilizando.

Essa alteração da configuração é benéfica quando se utiliza a variante correta do protocolo para aquele tipo de rede, podendo trazer uma melhor conexão ao utilizador. Além disso, é possível verificar se há a possibilidade de propostas de novas variantes sobre o controle de congestionamento, ou métodos adaptativos de valores nas funções objetivo das variantes já existentes. Por esses motivos, se faz interessante o objeto de estudo deste trabalho.

1.2 Objetivos

Como objetivos centrais, temos a intenção de avaliar formas de reduzir os problemas atuais em conexões à Internet por redes sem fio, por exemplo, perda de pacotes, instabilidade, quedas de conexão e lentidão, através do estudo e da escolha de um melhor algoritmo

TCP, que deixe a rede mais rápida e, possivelmente mais estável.

Pode-se destacar os seguintes objetivos específicos:

- Estudar o protocolo TCP, assim como suas variantes, além de categorizá-las por objetivos.
- Analisar os algoritmos de implementação do TCP através de simulação, utilizando a ferramenta NS-3.
- Concluir quais variantes TCP apresentam melhores resultados em uma rede sem fio local.

2 Fundamentação Teórica

Em 2019, segundo estudo feito pela ONU, o número de usuários era de 4,1 bilhões (ONU, 2019). Esses números mostram uma grande quantidade de usuários atualmente, além do aumento significativo que vem constantemente ocorrendo nos últimos anos.

Em 2020, iniciou-se a pandemia do novo Coronavírus, e com isso o tráfego de Internet cresceu, tanto pela população estar em casa e, consequentemente utilizar mais a Internet para lazer, quanto pelo uso de plataformas online para reuniões, trabalho e estudos. No começo da pandemia, a infraestrutura brasileira de Internet registrou um tráfego de 11 TB/s em um dia, valor que é atípico se comparado à média de tráfego do ano de 2019, que foi de 4,69 Tb/s (GAGLIONI, 2019). De acordo com pesquisa publicada por Simon Kemp, no site *Data Reportal*[1], em Janeiro de 2021 haviam 4,66 bilhões de usuários de Internet no mundo todo, o que equivale a 59,5% da população mundial (KEMP, 2021).

Além do aumento do número de usuários, o número de serviços que funcionam através da Internet também têm aumentado. Isso faz com que o uso da rede pela população também cresça e aumente o tráfego, contribuindo para mais ocorrências de lentidão e instabilidade na conexão.

Como citado, o TCP/IP é um acrônimo para o termo *Transmission Control Protocol/Internet Protocol Suite*, dois dos mais importantes protocolos que conformam a pilha de protocolos usados na Internet. O protocolo IP, base da estrutura de comunicação da Internet é um protocolo baseado no paradigma de chaveamento de pacotes (*packet-switching*). Os protocolos TCP/IP podem ser utilizados sobre qualquer estrutura de rede, seja ela simples como uma ligação ponto-a-ponto ou uma rede de pacotes complexa. Como exemplo, pode-se empregar estruturas de rede como *Ethernet*, PPP (Protocolo Ponto-a-Ponto), X.25, *Frame-Relay*, enlaces de satélite, ligações telefônicas discadas, fibra óptica e várias outras (TANENBAUM, 2011). A arquitetura TCP/IP, assim como a arquitetura OSI/ISO (KUROSE; ROSS, 2000), realiza a divisão de funções do sistema de comunicação em estruturas chamadas de camadas, cada uma sendo responsável por uma função no

[1] ⟨https://datareportal.com/people/simon-kemp⟩

sistema, e permitindo que o desenvolvedor trate apenas da que lhe interessa, deixando a cargo da camada inferior ou superior um trabalho quase que transparente.

2.1 Arquitetura TCP/IP

Nesta seção veremos cada uma das camadas da pilha TCP/IP. A intenção desta seção é que auxilie o leitor durante o entendimento do foco deste trabalho, criando uma base teórica sólida em relação à Redes de Computadores.

2.1.1 Camada de Enlace

A camada de enlace é muitas vezes tratada como parte de uma única camada em conjunto com a camada física (Interface de Rede), conforme a Figura 2.1. Esta camada é responsável pelo envio de datagramas construídos pela camada superior, a camada de rede. Ela também realiza o mapeamento entre um endereço de identificação do nível de rede para um endereço físico ou lógico (*e.g.*, Endereço MAC).

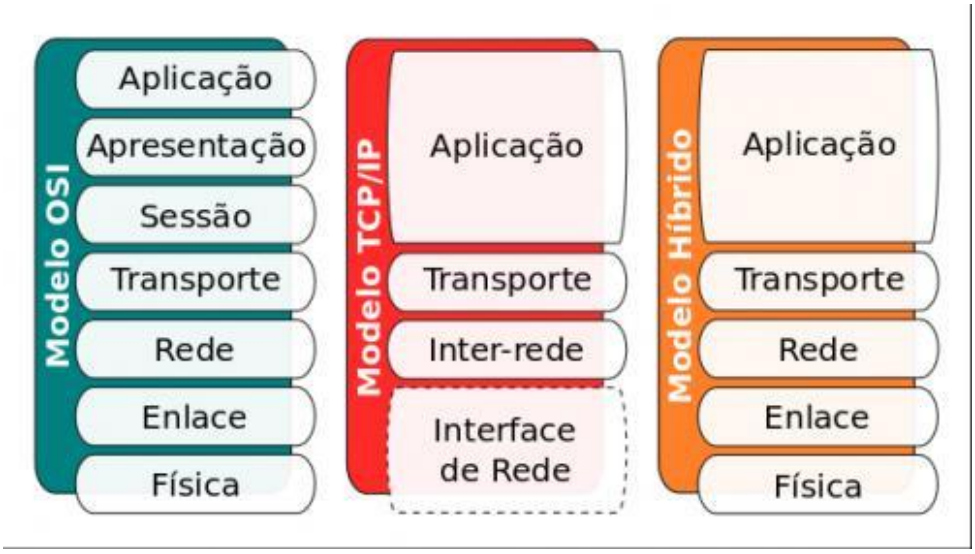

Figura 2.1: Comparação entre arquiteturas OSI, TCP/IP e o modelo híbrido. Fonte: Adaptado de (KUROSE; ROSS, 2000).

Os protocolos deste nível possuem um esquema de identificação das máquinas interligadas. Por exemplo, cada máquina situada em uma rede *Ethernet*, possui um identificador único chamado endereço MAC ou endereço físico que permite distinguir uma máquina de outra, possibilitando o envio de mensagens específicas para cada uma delas. Tais redes são chamadas redes locais de computadores. Um exemplo de endereço MAC é `00:1B:C9:4B:E3:57`. Os endereços MAC possuem uma padronização, que é administrada pela IEEE. Basicamente, ele é formado por um conjunto de seis *bytes* separados por dois pontos ou hífen, e cada *byte* é representado por dois algarismos na forma hexadecimal.

2.1.2 Camada de Rede (ou Inter-Rede)

Esta camada foi o que denominamos no início de responsável pela cintura fina, ou ampulheta. Nesta camada é onde se possibilita a comunicação entre máquinas vizinhas, e isso só é possível através do protocolo IP. Para cada dispositivo, e a própria rede onde essas estão situadas, é definido um endereço IP, que é o identificador independente de outras formas de endereçamento que possam existir nos níveis/camadas inferiores. No caso de existir endereçamento nos níveis inferiores é realizado um mapeamento para possibilitar a conversão de um endereço IP em um endereço deste nível. Por exemplo, um endereço MAC pode ser mapeado para o endereço IP em questão referente ao dispositivo destinatário ou fonte da comunicação.

O IP é tratado como o mais importante protocolo desta camada pois, apesar de existirem outros protocolos, como o ICMP (*Internet Control Message Protocol*) e o IGMP (*Internet Group Message Protocol*), o protocolo IP é aquele que implementa a função que permite a própria comunicação inter-redes. Isso é feito através da função de roteamento. O roteamento consiste no transporte de mensagens entre as diferentes redes, decidindo qual rota uma mensagem deve seguir através dos comutadores intermediários até chegar ao destino.

2.1.3 Camada de Transporte

Esta camada é a mais importante para o nosso estudo. É ela quem reúne os protocolos que realizam as funções de transporte na rede, e esse transporte, que conecta dois dispositivos, é chamado de conexões de dados fim-a-fim. Nessa comunicação não há preocupação com os elementos intermediários, apenas com a origem e o destino. Este transporte possui dois principais protocolos, que são o UDP (*User Datagram Protocol*) e TCP, foco do nosso estudo.

Pensando em ambientes cabeados, a perda, em geral, por interferência e outras disputas do meio, é muito menor que no meio sem fio. Assim, protocolos com confirmação, por exemplo, podem ser mais interessantes, apesar de introduzir mais complexidade na rede. Os dois protocolos que citamos são o UDP e o TCP, onde o protocolo UDP realiza apenas a multiplexação para que várias aplicações possam acessar o sistema de comunicação. Já o protocolo TCP realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. É por isso que o chamamos naturalmente de um protocolo confiável. Desta forma, são responsabilidades desse protocolo executar tarefas como: o controle de fluxo, o controle de erro, a sequenciação e a multiplexação de mensagens.

Sobre o Protocolo TCP temos a seguinte citação:

> *O TCP é um protocolo orientado a conexões confiável que permite a entrega sem erros de um fluxo de bytes originário de uma determinada máquina em qualquer computador da inter-rede. Esse protocolo fragmenta o fluxo de bytes de entrada em mensagens discretas e passa cada uma delas para a camada inter-redes. No destino, o processo TCP receptor volta a montar as mensagens recebidas no fluxo de saída. O TCP também cuida do controle de fluxo, impedindo que um transmissor rápido sobrecarregue um receptor lento com um volume de mensagens maior do que ele pode manipular* (TANENBAUM, 2011).

Estudando o protocolo TCP e suas diferentes implementações/variantes, é possível melhorar o controle de fluxo, diminuindo, por exemplo, problemas de congestionamento nas redes.

2.1.4 Camada de Aplicação

Por fim, na pilha de protocolos temos a camada de aplicação. Ela reúne os protocolos que fornecem serviços de comunicação a outros sistemas ou ao usuário final. Nesta camada há protocolos básicos ou de serviços para o usuário. Por exemplo, protocolos de serviços básicos, que fornecem serviços para atender as próprias necessidades do sistema de comunicação TCP/IP são: DNS, DHCP (*Dynamic Host Configuration Protocol*). Já protocolos de serviços para o usuário são os mais diversos, como: FTP (*File Transfer Protocolo*, HTTP, SSH e outros.

2.2 Funcionamento do TCP

Nesta seção será explicado o protocolo TCP. Serão apresentadas cada uma das suas funções e fases envolvidas. Assim, acredita-se que a compreensão da avaliação proposta neste trabalho ficará mais clara para o leitor.

Em maio de 1974, o IEEE publicou um artigo intitulado de "*A Protocol for Packet Network Interconnection*" (CERF; KAHN, 1974). Os autores do artigo, Vint G. Cerf e Robert Kahn, descreveram um protocolo de interconexão para compartilhamento de recursos usando comutação de pacotes ao longo dos nós. Um componente central de controle deste modelo foi exatamente o TCP. Ele era visto, a princípio, não como parte de uma pilha, mas com diversas funções que possibilitariam a comunicação fim-a-fim. Esse programa de controle de transmissão monolítico foi dividido somente um tempo depois,

dentro de uma arquitetura modular formada de um protocolo de controle de transmissão na camada orientada a conexão e o protocolo de Internet na camada de interconexão, ou seja, o IP, de uma maneira mais arcaica. Esse modelo deu origem a pilha conhecida como TCP/IP.

Para entender o protocolo TCP, deve-se entender também das suas fases. São etapas/fases que o protocolo TCP utiliza durante uma conexão fim-a-fim: o estabelecimento da conexão, a transferência de pacotes e o término dessa conexão. O estabelecimento da ligação é feito em três passos, que chamamos de *3-Way Handshake*, ou acordo de 3 vias. Já o término da conexão é realizado em quatro passos. Apesar de apresentarmos somente as fases como o *overhead*/sobrecarga do TCP para a conexão fim-a-fim, e não entrarmos em detalhes desta troca de pacotes de controle, uma característica é interessante de ser destacada, a negociação do número de sequências de pacotes. Durante a fase de inicialização são inicializados diversos parâmetros, entre eles o *Sequence Number* (número de sequência), que é quem auxilia na garantia da entrega ordenada e na robustez da comunicação durante a transferência dos pacotes.

A Internet foi proposta sobre uma rede de melhor esforço, ou seja, não há garantia de entrega de pacotes e nem de que essa entrega esteja ordenada. Assim, é papel do TCP adicionar formas de realizar uma entrega mais confiável, porém, levando em consideração a redução do recursos de rede necessários. A principal etapa nesse cenário é a do controle de congestionamento. É ele que torna possível avaliar o tamanho da janela (W) de pacotes enviados com relação à latência (RTT – *Round-Trip Time* – tempo de ida e volta) de um pacote no caminho da rede e, assim, ajustar seus parâmetros para melhor atender às demandas do TCP de uma transmissão confiável e também justa.

O controle de congestionamento no TCP é feito através de quatro algoritmos: *Slow-Start*, *Congestion Avoidance*, *Fast Recovery* e *Fast Retransmit*.

- *Slow-Start*: o algoritmo *slow-start* é utilizado para tentar maximizar performance, pois o parâmetro inicial do tamanho da janela de congestionamento pode ter impacto sobre a performance da rede. O primeiro pacote enviado por um remetente tem o valor de uma pequena janela de congestionamento, que vai aumentando exponencialmente a cada RTT, ou seja, a cada *ACK* recebido pelo remetente. O

algoritmo finaliza seu trabalho ao atingir o limiar, também chamado de *threshold* ou *ssthresh*.

- *Congestion Avoidance*: o algoritmo *Congestion Avoidance*, ou, em português, prevenção de congestionamento, se inicia após o fim do *slow-start*, e começa a aumentar a janela de congestionamento linearmente (cwnd = cwnd + 1) a cada RTT, diferentemente do *slow-start* que aumentava-a exponencialmente. O algoritmo chega ao fim quando é detectado algum congestionamento.

- *Fast Retransmit*: a cada segmento perdido ou enviado fora de ordem, um *ACK* duplicado é gerado imediatamente, porém como não é possível saber se esse *ACK* duplicado foi enviado para indicar um segmento fora de ordem ou um segmento perdido, é necessário aguardar por mais *ACKs* duplicados, pois se o problema for de segmentos fora de ordem, apenas um ou dois *ACKs* duplicados serão enviados até que ocorra essa reordenação do segmento, porém se ocorre o recebimento de três ou mais *ACKs* duplicados, é provável que um segmento tenha sido perdido, e será então retransmitido.

- *Fast Recovery*: o algoritmo *Fast Recovery*, assim como o *Fast Retransmit*, tem por objetivo manter a rede sem congestionamentos, e ambos os algoritmos costumam funcionar juntos. Após três *ACKs* duplicados, entende-se que um segmento foi perdido, causando o reenvio do mesmo, além disso, o tamanho da janela de congestionamento é dividido por dois. Após essa retransmissão, o envio de outros pacotes continua normalmente. A cada novo *ACK* duplicado que chega, o tamanho da janela de congestionamento é aumentado de acordo com o tamanho do segmento, e o pacote é então transmitido.

Neste trabalho serão avaliadas as diversas propostas de modificação do TCP "padrão", proposto em 1974 por (CERF; KAHN, 1974). Basicamente, esse protocolo é também conhecido como Tahoe. Os protocolos selecionados, também chamados neste trabalho de variantes do TCP, serão apresentados logo a seguir.

2.3 Variantes TCP

Como uma forma de classificar os protocolos TCP estudados, apresentamos a Tabela 2.1. Nela é possível ver um total de 11 variantes do TCP, sendo 5 consideradas clássicas, se baseando no modo de operação pouco agressivo e mais voltado a baixas taxas de transferência, originais da Internet. Ainda vê-se 3 variações relativas à redes de mais alta velocidade, em geral providas sobre o ambiente de cabeamento de par trançado Cat5e, fibra óptica, ou em *backbones* da Internet. E, por fim, classificamos 3 variantes como adaptadas a ambientes com alto RTT.

Tabela 2.1: Classificação dos Protocolos TCP.

Algoritmo TCP	**Classificação**
BIC	Alto RTT
Cubic	Alto RTT
Hybla	Alto RTT
HighSpeed	Redes de Alta velocidade
H-TCP	Redes de Alta velocidade
Scalable	Redes de Alta velocidade
NewReno	Clássico
Westwood	Clássico
Veno	Clássico
Linux Reno	Clássico
Vegas	Clássico

Todos os protocolos TCP variantes em questão serão avaliados sobre uma rede sem fio local do tipo IEEE 802.11n, e serão analisadas sua possível aplicação a cada momento e densidade de uma rede local.

2.3.1 Reno

O TCP Reno (JACOBSON, 1990) é uma variante clássica e modifica uma das primeiras variantes do TCP, que podemos chamar como padrão, a implementação do TCP Tahoe. Ele utiliza, inicialmente, o princípio básico do *slow-start*, porém pacotes perdidos são detectados rapidamente. Essa detecção rápida de pacotes perdidos ocorre analisando ACKs duplicados recebidos, deduzindo-se que após três ACKs duplicados serem recebidos, o pacote provavelmente se perdeu. Ao realizar essa detecção de perda, é utilizado o *Fast Retransmit*, que reenvia o segmento perdido sem esperar por um *timeout*.

O Reno utiliza um tamanho de janela de congestionamento aleatório de começo, aumentando-o pouco a pouco caso nenhuma perda esteja ocorrendo, porém caso ocorram perdas, a janela é rapidamente reduzida.

O problema com o algoritmo é que ele não consegue detectar perdas múltiplas de pacotes em uma só vez, o que faz com o desempenho caia drasticamente nesses casos, porém em caso de pouca perda de pacotes, o desempenho é muito bom.

A figura 2.2 abaixo apresenta o gráfico do funcionamento do Reno comparado ao do algoritmo Tahoe. No eixo x temos a rodada de transmissão, no eixo y temos o tamanho da janela de congestionamento. Podemos notar que a diferença entre ambos ocorre após uma perda, pois enquanto o Reno volta a crescer a partir do *threshold*, o Tahoe reduz para 1 o valor do cwnd.

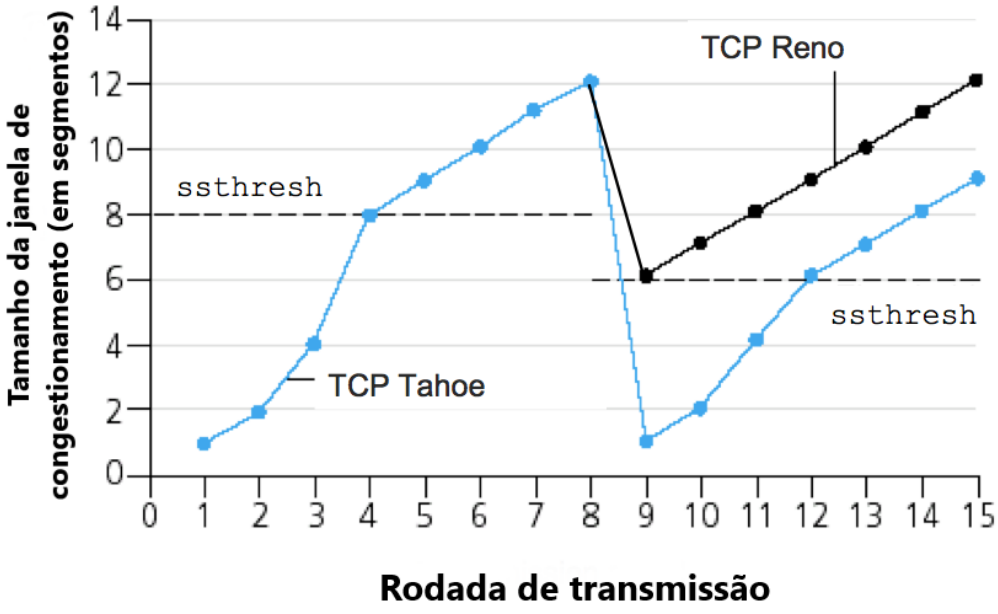

Figura 2.2: Visão sobre partida lenta e prevenção de congestionamento entre Tahoe e Reno (KUROSE; ROSS, 2000).

2.3.2 New Reno

O *New Reno* (FLOYD et al., 1999) é uma pequena modificação do TCP Reno, que tenta corrigir o problema do anterior em relação à detecção de perda de múltiplos pacotes simultâneos. Para corrigir isso, há uma modificação no algoritmo *Fast Recovery* para fazer com que outros pacotes perdidos também sejam retransmitidos, pois envia continuamente

ACKs duplicados mesmo após a retransmissão do primeiro perdido. Um problema dessa modificação é que outros pacotes perdidos só são detectados após a retransmissão do primeiro perdido.

2.3.3 Vegas

É um dos algoritmos clássicos do TCP. Diferente de outros algoritmos, como Reno e New Reno. Vegas foca em detectar o congestionamento antes de ocorrerem perdas de pacote. Consegue detectar congestionamento nos estágios iniciais através da medição do RTT. O TCP Vegas tem um bom controle de congestionamento e dá estabilidade para a rede, porém não consegue utilizar completamente a banda.

Funcionamento:

- O remetente mede o quociente de vazão esperado , que seria o o tamanho da janela de congestionamento dividido pelo RTT base, que seria o menor RTT medido até aquele determinado momento.
- O remetente encontra o quociente de vazão atual utilizando o RTT real do pacote. Quando a rede não está congestionada, o quociente de vazão real é próximo ao quociente de vazão esperado. Já quando a rede está congestionada, o quociente de vazão real é menor que o quociente de vazão esperado.
- Após isso, o remetente subtrai o quociente de vazão esperado pelo quociente de vazão real, e multiplica o resultado pelo RTT base.
- O resultado da operação do passe anterior é então comparado a duas variáveis limite, α (*alpha*) e β (*beta*). Caso o resultado da fórmula seja menor que α, o tamanho da janela de congestionamento aumentará. Caso seja maior que β, o tamanho diminui. Se estiver entre ambas, o tamanho da janela de congestionamento se mantém.

2.3.4 Veno

O TCP Veno (FU; LIEW, 2003) é uma mescla do Vegas e do Reno. Antes de atingir o *threshold*, o algoritmo age de forma mais agressiva, parecida com a do algoritmo Reno,

porém ao passar do limiar, age de forma mais conservadora, assim como o algoritmo Vegas, para evitar sobrecarga da rede. Quando o tamanho da janela de congestionamento ultrapassa o limiar do *slow-start*, ocorre uma redução na taxa de aumento desse tamanho para tentar evitar congestionamentos. Já quando o algoritmo assume que não está utilizando toda a banda, aumenta a janela de congestionamento de uma forma maior. O algoritmo evita retransmissões, perdas de pacotes e, também, controle de congestionamento desnecessário. Isso torna o Veno, em geral, um algoritmo mais flexível do que o Reno, e que consegue utilizar mais banda do que o Vegas.

2.3.5 Westwood

O *Westwood* (MASCOLO et al., 2001) calcula aproximadamente a banda da rede através do tempo entre o envio de pacote e o *ACK*. Com essa banda de rede aproximada, são calculados e atribuídos valores para a janela de congestionamento e para o *slow-start*. Com essa implementação do *Westwood*, as grandes reduções da janela de congestionamento são extintas, o que causa uma melhora na velocidade de recuperação e ajuda a evitar o congestionamento. O TCP *Westwood* calcula a janela de congestionamento e o *threshold* utilizando o mecanismo de banda estimada BWE após um congestionamento. Na etapa de *Congestion Avoidance* do algoritmo, ocorre a busca por banda extra até o momento que uma certa quantidade de *ACKs* duplicados é recebida, que significa que foi atingido o limite da rede. Após atingir esse limite, o *threshold* é recalculado com a fórmula $BWE \times RTTmin$, a janela de congestionamento é igualada ao *threshold* do *slow-start* e o algoritmo entra novamente em fase de *Congestion Avoidance* para buscar mais banda.

2.3.6 BIC

O TCP BIC (*Binary Increase Congestion Control*) é otimizado para redes de alta velocidade e latência. Utiliza um algoritmo de busca binária para encontrar o melhor tamanho para a janela de congestionamento. A cada intervalo de RTT, a janela é aumentada, e a cada perda de pacotes a janela é reduzida por um fator multiplicador.

O código do Algoritmo 1 representa o funcionamento do BIC para aumento da janela.

Algoritmo 1: Aumento de janela no TCP BIC

```
1  if cwnd < wmax then
2  |  bic_inc = (wmax−cwnd)/2
3  else
4  |  bic_inc = cwnd − wmax
5  end if
6  if (bic_inc > Smax) then
7  |  bic_inc = Smax
8  else
9  |  if (bic_inc < Smin) then
10 |  |  bic_inc = Smin
11 |  |  cwnd = cwnd + bic_inc/cwnd
12 |  end if
13 end if
```

Já o código do Algoritmo 2 representa o funcionamento do BIC para diminuição da janela.

Algoritmo 2: Diminuição da janela no TCP BIC

```
1  if cwnd < wmax then
2  |  wmax = cwnd * (2−β)/2
3  else
4  |  wmax = cwnd
5  |  cwnd = cwnd * (1 − β)
6  end if
```

2.3.7 Cubic

O *Cubic* é uma evolução do BIC e foi criado para melhorar a função de crescimento da janela de congestionamento, que estava muito agressivo, principalmente para redes de baixa velocidade ou baixo RTT. A função de crescimento do *Cubic* é calculada por uma função cúbica utilizando o tempo decorrido desde o último congestionamento, por isso tem esse nome. A função utilizada para esse cálculo é:

$$W(t) = C(t-K)^3 + Wmax \tag{2.1}$$

Onde C é um parâmetro do próprio algoritmo, t é o tempo que se passou desde a última redução da janela, e K é calculado pela seguinte fórmula:

$$\sqrt[3]{\frac{Wmax \times \beta}{C}} \tag{2.2}$$

2.3.8 Hybla

Em redes heterogêneas, as conexões TCP que incorporam um link de rádio terrestre ou por satélite possuem muita desvantagem em relação às conexões totalmente cabeadas, por causa de seus tempos de ida e volta (RTTs) mais longos. Para lidar com este problema, surge uma nova proposta TCP, o TCP Hybla, que é apresentada e discutida no artigo de (CAINI; FIRRINCIELI, 2004). Em particular, o desempenho do TCP Hybla é comparado com o alcançado pelo padrão TCP na presença de congestionamento e perdas de enlace, consideradas separadamente ou em conjunto. Em todos os casos examinados, a superioridade do TCP Hybla é evidente, pois reduz muito a severa penalização sofrida pelas *wireless*, especialmente por satélite.

O algoritmo mantém a filosofia de partida lenta e prevenção de congestionamento (*congestion avoidance*) do TCP tradicional, ajustando o tamanho de janela $W(t)$ independentemente do RTT em uma escala de tempo ajustada (para o ambiente), e relativizando ao dividir esse valor pelo RTT atual. Qualquer consideração ao RTT só é realizada após normalizá-lo. No trabalho que introduz o Hybla, é demonstrado que ele não depende mais do valor atual de RTT, gerando ganho de performance em meios com RTT variável.

2.3.9 HighSpeed

O *HighSpeedTCP*, ou simplesmente HSTCP, é uma variante do TCP para redes de alta velocidade. Ele altera o algoritmo de controle de congestionamento conforme a RFC 3649 (FLOYD, 2003). Sua intenção é alterar o TCP para funcionar com grandes janelas de congestionamento. Há um exemplo interessante na RFC do protocolo que diz: "para um TCP padrão que gerencia uma conexão com pacotes de tamanho 1500 *bytes* e um RTT de 100 ms, alcançar uma taxa de transferência estável de 10 Gbps exigiria uma janela de congestionamento média de 83.333 segmentos e uma taxa de perda de pacotes de, no máximo, um evento de congestionamento a cada 5.000.000.000 pacotes. Isto é amplamente reconhecido como uma restrição irreal. Para tratar esta limitação do TCP,

esta RFC propõe um TCP de alta velocidade.

Basicamente, para o HSTCP, quando um ACK é recebido (na prevenção de congestionamento – *congestion avoidance*), a janela é aumentada em $\frac{a(w)}{w}$, e quando uma perda é detectada por meio de ACKs triplos duplicados, a janela é igual $(1 - b(w))w$, onde w é o tamanho da janela atual. Quando a janela de congestionamento é pequena, o HSTCP se comporta exatamente como o TCP padrão, de modo que $a(w)$ é igual a 1 e $b(w)$ é 0,5. Quando a janela de congestionamento do TCP está além de um certo limite, $a(w)$ e $b(w)$ tornam-se funções do tamanho da janela atual. Nesta fase, à medida que a janela de congestionamento aumenta, o valor de $a(w)$ aumenta e o valor de $b(w)$ diminui. Isso significa que a janela do HSTCP crescerá mais rápido que o TCP padrão, e também é esperado que ele se recupere das perdas mais rapidamente. Para este último caso, é pensado o conceito de *bandwidth-delay product* (BDP), ou produto entre largura de banda (B em bps) e RTT (D em ms).

É interessante destacar que o HSTCP tem o mesmo comportamento de partida lenta e também de calculo do *timeout* do TCP padrão. Isso permite que a janela não inicie de maneira agressiva, o que poderia limitar sua aplicação no ambiente real (por ter que alocar *buffer* nos nós intermediários, por exemplo). Como apenas o mecanismo de controle de congestionamento foi modificado, o HSTCP é compatível com outras variantes de TCP.

2.3.10 H-TCP ou HTCP

O H-TCP do TCP foi apresentado no Internet-Draft *draft-leith-tcp-htcp-03*[2] e em seu artigo relacionado, "H-TCP: TCP for high-speed and long-distance networks" (LEITH; SHORTEN, 2004). O H-TCP modifica o controle de congestionamento do protocolo TCP para redes com altos produtos da relação atraso e largura de banda, como mostrado anteriormente para o HSTCP na explicação do *bandwidth-delay product - BDP*. Sua intenção é que que ele seja mais justo para fluxos semelhantes que trafegam na rede e também amigável/compatível com TCP padrão/convencional. Sua intenção também é que se faça uso da maior parte possível – ou até mesmo toda – a largura de banda livre disponível.

[2] ⟨https://datatracker.ietf.org/doc/html/draft-leith-tcp-htcp-03⟩

O algoritmo H-TCP faz uma alteração simples na função de crescimento do TCP padrão, relativo à variável *cwnd*. A sua principal característica é que a sua "agressividade" é uma função apenas do tempo decorrido desde o último recuo/*backoff*/perda de pacotes. Assim, podemos listar que: ele preserva muitas das principais propriedades do TCP padrão, como justiça, responsividade à perdas, e consideração ao *buffer*. É uma opção para melhorar a injustiça do RTT e dissociá-la ao aproveitamento do *buffer*. Porém, como demonstrado em (LEITH; SHORTEN, 2006), aumenta a sensibilidade à diferenças na taxa de sincronização, algo comum em algoritmos de "alta velocidade" baseados em perda, ou seja, uma vez que a sincronização, ou a percepção de perda é perdida, a agressividade no crescimento da janela de congestionamento impacta na sobrecarga da rede.

2.3.11 Scalable

Por fim, o *Scalable* (KELLY, 2003), modifica o algoritmo de controle de congestionamento TCP original de forma que, ao invés de diminuir a janela de congestionamento pela metade a cada perda de pacote ocorrida, diminui apenas $1/8$ $(0,125)$, além de aumentar em $0,01$ a cada *ACK* recebido quando não está ocorrendo perdas. Essa modificação é boa para redes de alta velocidade de transferência, melhorando muito o tempo de recuperação.

A motivação para sua proposta foi que, como sabemos, o controle de congestionamento TCP pode ter um desempenho ruim em redes de alta velocidade e de longa distância, principalmente por causa de sua resposta lenta quando aplicamos apenas um aumento no tamanho da janela de congestionamento. Assim, torna-se um desafio, para qualquer protocolo alternativo, utilizar melhor as redes com BDP de forma simples e robusta, sem gerar outros problemas com relação ao tráfego de rede já existente. O TCP *Scalable* é, portanto, uma alteração simples que, como o HSTCP e o HTCP, modifica o algoritmo de atualização da janela de congestionamento do TCP padrão, mas se mantendo compatível com o mesmo. Essa variante foi projetada para ser implementada de forma incremental, e deve se comportar de forma idêntica à implementação padrão do TCP quando a janela no TCP é pequena e suficiente.

No trabalho da proposta do algoritmo, em (KELLY, 2003), é ilustrada a questão do ajuste de janela de congestionamento. Os autores consideram na proposta o cenário

de estudo do TCP padrão onde, quando não há congestionamento a janela é definida por:

$$cwnd \leftarrow cwnd + \frac{1}{cwnd} \tag{2.3}$$

E quando há congestionamento:

$$cwnd \leftarrow \frac{cwnd}{2} \tag{2.4}$$

3 Trabalhos Relacionados

Em (NIGAR; AZIM, 2018) são avaliadas redes *ad hoc* móveis (MANETs), um ambiente sem infraestrutura de redes sem fio onde os dispositivos móveis atuam como roteadores e nós intermediários. O TCP nesse ambiente enfrenta sérios desafios quando usado em dispositivos móveis *ad hoc*. A funcionalidade do TCP se degrada devido à propriedades especiais da MANET, como falha de rota devido à mudança significativa de topologia de rede e erros de transmissão. O trabalho avalia as variantes do TCP, ou seja, os algoritmos de controle de congestionamento para o TCP, como Vegas, Reno, New Reno, SACK, FACK e Cubic. O Vegas é o que os autores afirmam ter melhor taxa de transferência em comparação a outras variantes TCP em uma rede com fio. Os autores realizaram simulação usando simulador de rede (NS-2) sobre protocolos de roteamento existente para redes *ad hoc*, a saber: AODV, AOMDV, DSDV e DSR. Os resultados da simulação mostram que o TCP Reno supera outras variantes de TCP no protocolo de roteamento DSDV.

Para (MOLIA; KOTHARI, 2018), são consideradas também as MANETs, e destacado que elas têm vários outros problemas, como erros de transmissão, topologias dinâmicas, contenções da camada de enlace. Erros de transmissão ou problemas de contenção são responsáveis por perdas de canal. As topologias dinâmicas são responsáveis pelas perdas por falha de rota. Os autores realizam uma revisão de variantes tradicionais de TCP em MANETs. O principal objetivo desta revisão é definir questões existentes e direções futuras para melhoria do TCP para MANETs, em um contexto mais teórico, sem foco na avaliação numérica ou por simulação.

Em (TARUK et al., 2017), destacam também redes sem fio, porém, com foco nas implementações da tecnologia móvel *Long Term Evolution* (LTE), de redes de telefonia móvel. Os autores avaliaram as variantes TCP Tahoe, Reno, SACK e TCP Vegas, e observaram, através da análise de *throughput*, que todas as variantes do TCP avaliadas possuem um QoS equivalente para o cenário avaliado, não havendo uma única variante TCP suficiente que atenda a todos os ambientes.

Já em (CHAUDHARY; KUMAR, 2017), demonstra-se uma estratégia básica de controle de congestionamento em redes sem fio locais. Destacam que em meados de 2016, houveram propostas de variações nos algoritmos TCP baseados exclusivamente nos mecanismos de *Slow-Start* (Partida Lenta) e *Congestion Avoidance* (Controle de Congestionamento). Os autores conduzem um estudo comparativo de cinco variantes de TCP, como TCP Tahoe, TCP Reno, TCP New Reno, TCP Vegas e TCP Dynamic Vegas, seus algoritmo de partida lenta, controle de congestionamento e detecção de congestionamento, e também analisa a melhor variante TCP no ambiente. Os autores concluem que o TCP Reno trabalha bem em ambientes quando o número de perda de pacotes é pequeno, embora em ambientes com muita perda ele não funcione bem e seu comportamento seja próximo ao TCP Tahoe. Em ambientes com perdas de pacotes, o comportamento do TCP New Reno foi destacado como o que melhor se comportou.

Em um trabalho anterior, de (WAGHMARE et al., 2011), foi analisado o desempenho de variantes de TCP também em redes sem fio. Foram realizadas simulações em NS-2 do TCP Tahoe, TCP Reno, TCP New Reno, TCP Sack, TCP Vegas e TCP New Jersey. Os autores focaram no impacto sobre as variantes utilizando os modelos de taxa de perda de pacote randômica *Random Packet Loss Rate* e variação de mobilidade. Nos estudos experimentais verificou-se que o TCP Vegas tem um desempenho melhor do que as outras variantes.

Assim, neste trabalho estendemos a avaliação das variantes do protocolo TCP utilizando a classificação apresentada na Tabela 2.1. Em um total de 11 variantes do TCP, avaliamos o impacto da utilização do padrão de rede sem fio IEEE 802.11n sobre os aspectos de *throughput*, RTT (*round-trip time*), perdas de pacotes e pacotes com erro, além da demonstração do *goodput* nos cenários. Este trabalho se diferencia do estado da arte, uma vez que estende o número das variantes e os avalia sobre um conjunto maior de nós sem fio. Porém, na rede sem fio avaliada, não analisamos o impacto da mobilidade, que é algo que poderá ser realizado no futuro.

4 Avaliações e Resultados

Neste trabalho foi apresentado um estudo referenciado da pilha TCP/IP. Após isso, foram dados mais detalhes do protocolo TCP para melhor compreensão de seu funcionamento e de como o mesmo é implementado, uma vez que esse protocolo é responsável por verificar o envio e recebimento de pacotes em uma rede, além de ser responsável pelo controle de congestionamento (KUROSE; ROSS, 2000). Ao fim desta etapa, foi possível apresentar a fundamentação teórica das variantes do TCP que são o objeto fim deste estudo. Além disso, foi criada uma classificação conforme suas características específicas. Feito isso, foi possível apresentar os trabalhos relacionados que tiveram objetivos semelhantes a este trabalho, na seção de trabalhos relacionados.

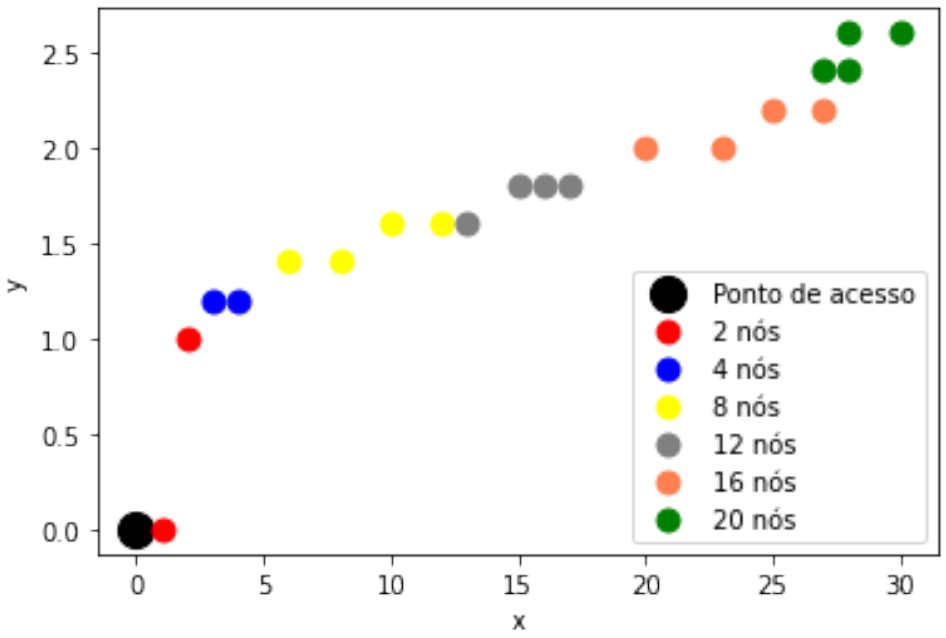

Figura 4.1: Topologia da rede usada para realização dos testes.

Para fins de interpretação do ambiente de simulação criado para a rede sem fio, segue a Figura 4.1, com a grade que representa a topologia. A cada iteração da simulação (2 nós, 4 nós, 8 nós, etc.) foram mantidas as posições dos nós anteriores e acrescentados os novos nós. No gráfico, cada acréscimo de nós é representado por uma nova cor. Foi criado um ponto de acesso na posição (0, 0), e os demais pontos são nós cliente da rede,

posicionados a x e y metros do ponto do acesso, por exemplo, o segundo nó cliente está na posição (2, 1), isso significa que ele está a 2 metros x e 1 metro y do ponto de acesso.

Agora, com a organização dos algoritmos, iniciaremos a avaliação das variações do TCP no ambiente NS-3, um *software* para simulação de redes que nos permite testar os diferentes algoritmos TCP, fazendo a manipulação das variáveis, como, tipo de rede e velocidade da conexão. Essas variáveis foram escolhidas previamente para representar um ambiente de uma rede local sem fio estruturado no padrão IEEE 802.11. Esse ambiente será utilizado em todos os algoritmos, para que os dados coletados sejam mais justos e precisos. Posteriormente, apresentaremos os resultados obtidos a partir de gráficos de comparação dos resultados para uma melhor compreensão e análise.

Foram avaliadas 11 variantes TCP utilizando o simulador de redes NS-3 (*Network Simulator* versão 3)[3]. O NS-3 é um simulador de rede de eventos discretos para sistemas de Internet, voltado principalmente para pesquisa e uso educacional. O NS-3 é um software livre, de código aberto, licenciado sob a licença GNU GPLv2 e mantido por uma comunidade mundial. Foram utilizadas as métricas de *Throughput*, *Goodput*, Retransmissões e RTT (*Round-Trip Time*) sobre um cenário de redes sem fio com IEEE 802.11n com 2, 4, 8, 12, 16 e 20 nós, sem mobilidade. A Tabela 4.1 sumariza os parâmetros utilizados na simulação. Esta seção, portanto, mostra os resultados e análises deste trabalho.

As métricas utilizadas estão explicadas a seguir:

- ***Throughput***: o *throughput* pode ser medido em *bits* por segundo (bps), *kilobits* por segundo (Kbps) *megabits* por segundo (Mbps), *gigabits* por segundo (Gbps) ou até mesmo em pacotes por segundo (pps), e é uma métrica que representa a quantidade de dados que foram transmitidos por uma rede.

- ***Goodput***: o *goodput*, diferentemente do *throughput*, mede apenas a quantidade de dados úteis que foram transmitidos com sucesso pela rede, ou seja, exclui dados como o cabeçalho de pacote do cálculo, além de não levar em conta pacotes que precisaram ser retransmitidos.

- **Retransmissões**: as retransmissões em TCP servem para evitar pacotes perdidos, ou seja, pacotes que foram transmitidos mas não chegaram ao seu destino. Após o

[3]⟨https://www.nsnam.org/⟩

envio de um pacote, o remetente começa uma contagem para aguardar seu retorno, que é reconhecido após o recebimento de um *ACK*, enviado pelo destinatário para avisar ao remetente que recebeu corretamente o pacote. Quando esse *ACK* não é recebido a tempo, a retransmissão ocorre.

- **RTT (*Round-Trip Time*)**: o RTT, também conhecido como latência, é medido em milissegundos e representa o tempo de ida e volta de uma requisição, ou seja, o tempo que um pacote demora para ir até seu destino e retornar ao ponto que partiu. O RTT é comumente utilizado para medir a confiabilidade da rede e quanto menor seu valor, melhor. Alguns fatores podem afetar negativamente o valor do RTT e devem ser evitados, como:
  - Distância: uma longa distância entre o local da requisição e o servidor aumenta o RTT.
  - Tempo de resposta do servidor: servidores dependem de sua capacidade de processamento e também da quantidade de requisições sendo respondidas por ele naquele momento para ter um bom tempo de resposta, o que influencia diretamente no RTT, pois um servidor que demora a responder consequentemente irá fazer o tempo de ida e volta da requisição aumentar.
  - Meio de transmissão: o meio de transmissão de um pacote afeta o RTT, pois diferentes tipos de rede possuem diferentes tempos de transmissão, por exemplo, redes de fibra óptica conseguem transmitir mais rápido do que uma rede sem fio.

Na Tabela 4.1 vemos algumas configurações que são interessantes de serem explicadas. Por exemplo, `YansErrorRateModel` modela a taxa de erro para diferentes modulações. É dito na documentação do NS-3 que um pacote de interesse (por exemplo, um pacote que pode ser recebido pela camada MAC) é dividido em pedaços (*chunks*). Cada pedaço está relacionado a um evento de recebimento de início/fim. Para cada pedaço, ele calcula a razão (SINR - Sinal Ruído) entre a potência recebida do pacote de interesse e a soma do ruído e da potência de interferência de todos os outros pacotes de entrada. Em seguida, ele calculará a taxa de sucesso do pedaço com base no BER da modulação.

Tabela 4.1: Configuração do ambiente de simulações e seus parâmetros.

Variável	**Valor**
Padrão de Redes Sem Fio	IEEE 802.11n
Modelo de Perda de Pacotes	3LogDistance
Modelo de Propagação	ConstantSpeedPropagationDelayModel
Payload do Pacote	1024 Bytes
Taxa física de transmissão de dados	HTMCS7
Modelo de Taxa de Erros	YansErrorRateModel
Mobilidade	Sem Mobilidade
Número de Nós	[2,4,8,12,16,20]
Número de APs	1
Tempo de Simulação	5 segundos

Assim, a taxa de recepção de sucesso do pacote é derivada da taxa de sucesso de todos os pedaços. Outro ponto interessante é que temos a taxa física de transmissão de dados com `HTMCS7` no NS-3. Neste caso é o equivalente ao valor HT MCS = 7 para 1 fluxo no padrão IEEE 802.11n. Isso corresponde a uma taxa de 65Mbps para um canal de 20MHz de largura de banda utilizam modulação 64-QAM e 5/6 para CR (*Coding Rate*). Também vemos na tabela o modelo de perda de pacotes como `3LogDistance`. Este modelo implementa um modelo de propagação de perda de caminho de distância logarítmica com três campos de distância. Este modelo é o mesmo que o `LogDistancePropagationLossModel`, exceto que possui três campos de distância: próximo, médio e distante com diferentes expoentes. Dentro de cada campo, a potência de recepção é calculada usando a equação de propagação log-distância:

$$L = L_0 + 10 \times n_0 log_{10}(\frac{d}{d_0}) \tag{4.1}$$

Cada campo começa onde o anterior termina e todos juntos formam uma função contínua. Ou seja, existem três campos de distância válidos: perto, meio, longe. Na verdade quatro: o primeiro de 0 até a distância de referência é inválido e retorna *txPowerDbm*, como segue:

$$\underbrace{0 \cdots\cdots}_{=0} \underbrace{d_0 \cdots\cdots}_{n_0} \underbrace{d_1 \cdots\cdots}_{n_1} \underbrace{d_2 \cdots\cdots}_{n_2} \infty$$

Onde a fórmula completa para a perda de caminho em dB está em:

Onde: d_0, d_1, d_2 são os 3 campos de distancia (em metros); n_1, n_2, n_3 expoente

$$L = \begin{cases} 0 & d < d_0 \\ L_0 + 10 \cdot n_0 \log_{10}(\frac{d}{d_0}) & d_0 \leq d < d_1 \\ L_0 + 10 \cdot n_0 \log_{10}(\frac{d_1}{d_0}) + 10 \cdot n_1 \log_{10}(\frac{d}{d_1}) & d_1 \leq d < d_2 \\ L_0 + 10 \cdot n_0 \log_{10}(\frac{d_1}{d_0}) + 10 \cdot n_1 \log_{10}(\frac{d_2}{d_1}) + 10 \cdot n_2 \log_{10}(\frac{d}{d_2}) & d_2 \leq d \end{cases}$$

de distância de perda de caminho para cada campo (sem unidade);L_0 perda no caminho com base na distância de referência (dB); d a distância (metros); e L a perda no caminho (dB).

4.1 Throughput

A seguir apresenta-se os resultados para as médias obtidas para o *throughput* ao fim de cada simulação para os ambientes descritos de 2 a 20 nós. Destaca-se que para esses resultados, a média é para todos os nós. Ou seja, não são observados apenas um ou outro nó em especial, mas sim, a média de todos eles. Sendo, portanto, diferente de quando observamos apenas um nó único, como veremos nos resultados das próximas seções.

4.1.1 Média

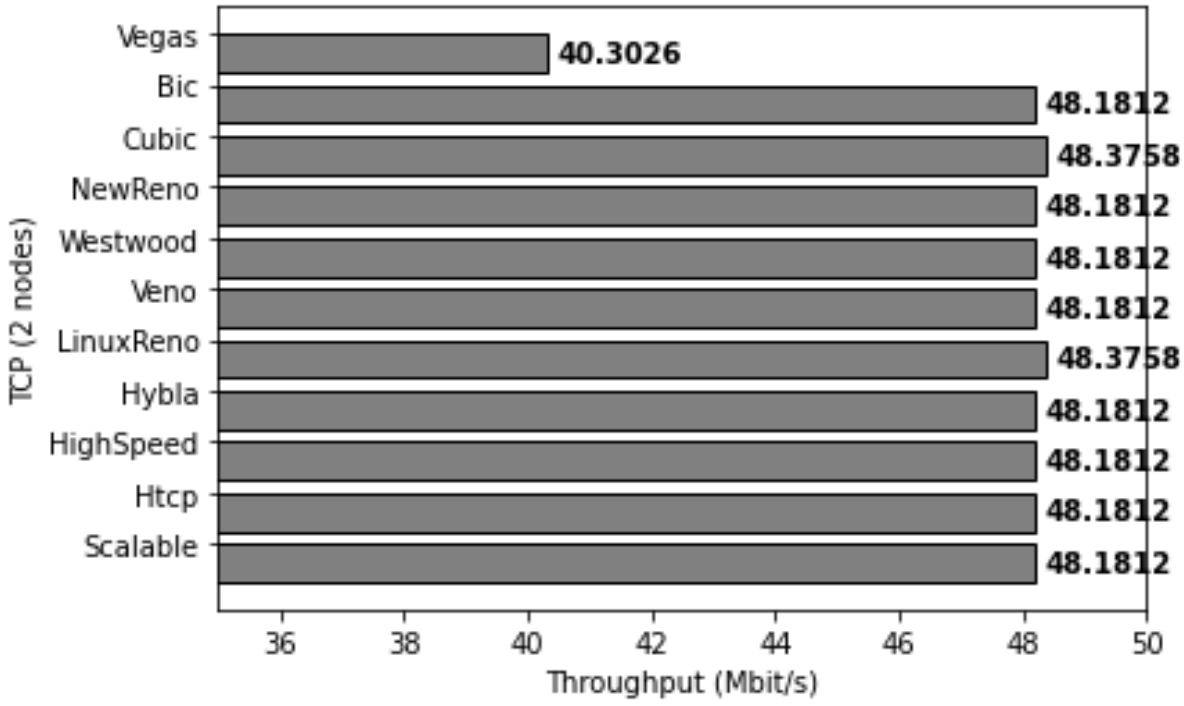

Figura 4.2: Gráfico de Throughput gerado por teste com 2 nós

Conforme é possível ver na Figura 4.2, todos as variantes apresentam resultados

próximos, ou até mesmo idênticos. A única que se destaca com resultado inferior é a variante Vegas. O cenário em questão tem apenas 2 nós, o que acredita-se ser um fator de grande influência. Já com relação ao resultado do Vegas, identifica-se que esse protocolo funciona melhor em ambientes com maior restrição e perdas, como veremos nas figuras mais à frente. Assim, em redes muito pequenas, com apenas 2 nós, qualquer variante TCP tem desempenho satisfatório, à exceção do TCP Vegas.

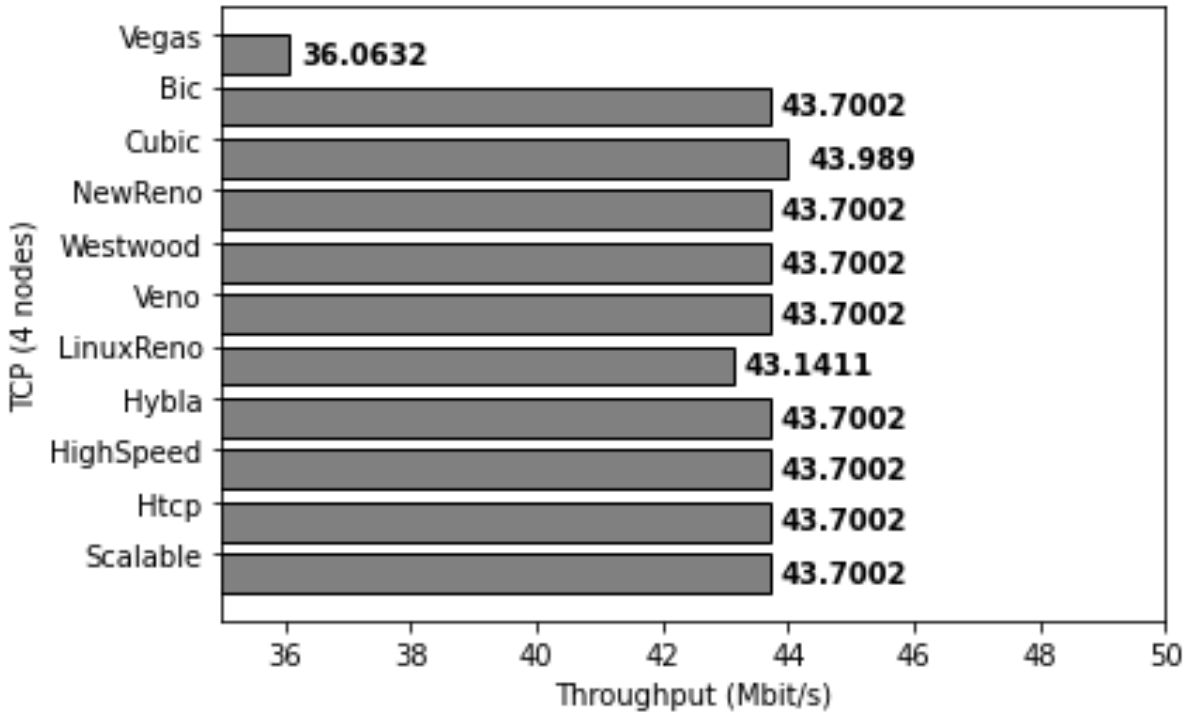

Figura 4.3: Gráfico de Throughput gerado por teste com 4 nós

Para o cenário com 4 nós, conforme vê-se na Figura 4.3, o comportamento se assemelha muito à simulação com 2 nós. Porém, agora verifica-se o decaimento do *throughput* para outro protocolo clássico além do Vegas, o Linux Reno, ou Reno como foi chamado anteriormente.

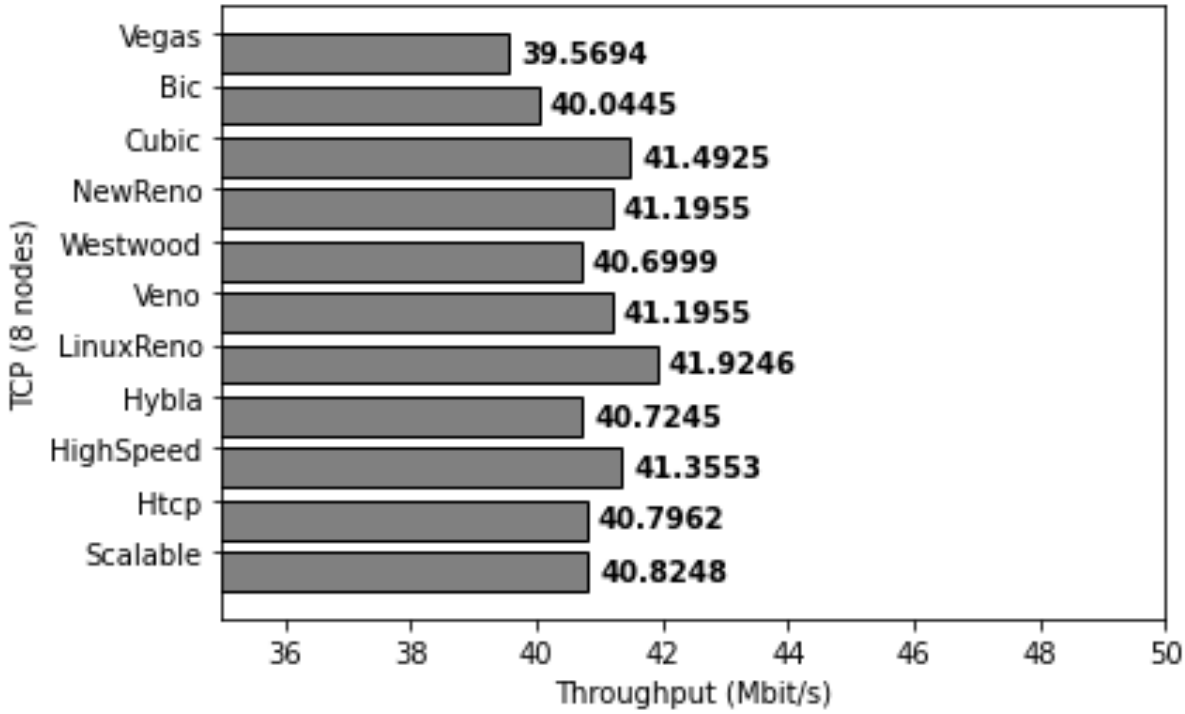

Figura 4.4: Gráfico de Throughput gerado por teste com 8 nós

No gráfico gerado para 8 nós (Figura 4.4), pode-se observar uma recuperação do algoritmo Vegas em relação ao cenário de 4 nós pois foi o único a ter um aumento no valor de *Throughput* enquanto os outros tiveram uma queda, embora não tão expressiva como a ocorrida entre os cenários de 2 e 4 nós. Nota-se, também, que para o algoritmo LinuxReno, a queda em relação ao teste anterior foi menor do que a de outros algoritmos, sendo esse o algoritmo que teve um maior valor de *Throughput* no cenário de 8 nós.

É possível observar também que os valores de *Throughput* dos algoritmos começaram a se diferenciar, pois em ambos os testes anteriores houveram muitos valores semelhantes, provavelmente devido ao pequeno número de clientes (nós) da rede.

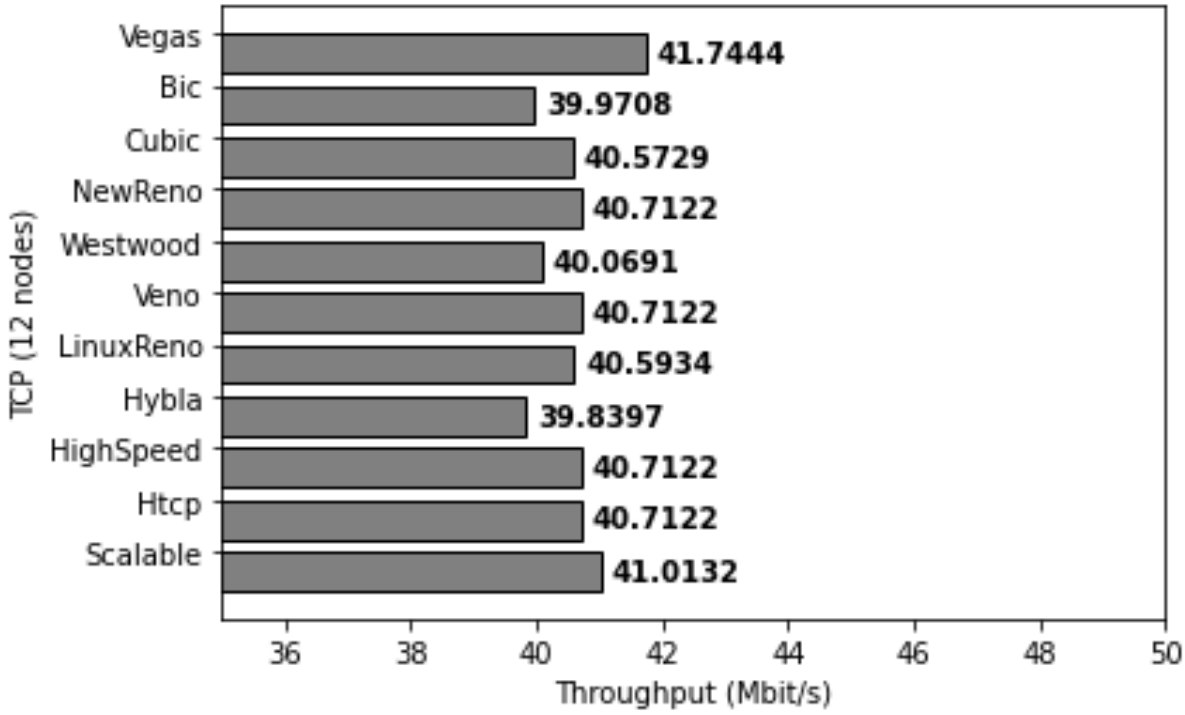

Figura 4.5: Gráfico de Throughput gerado por teste com 12 nós

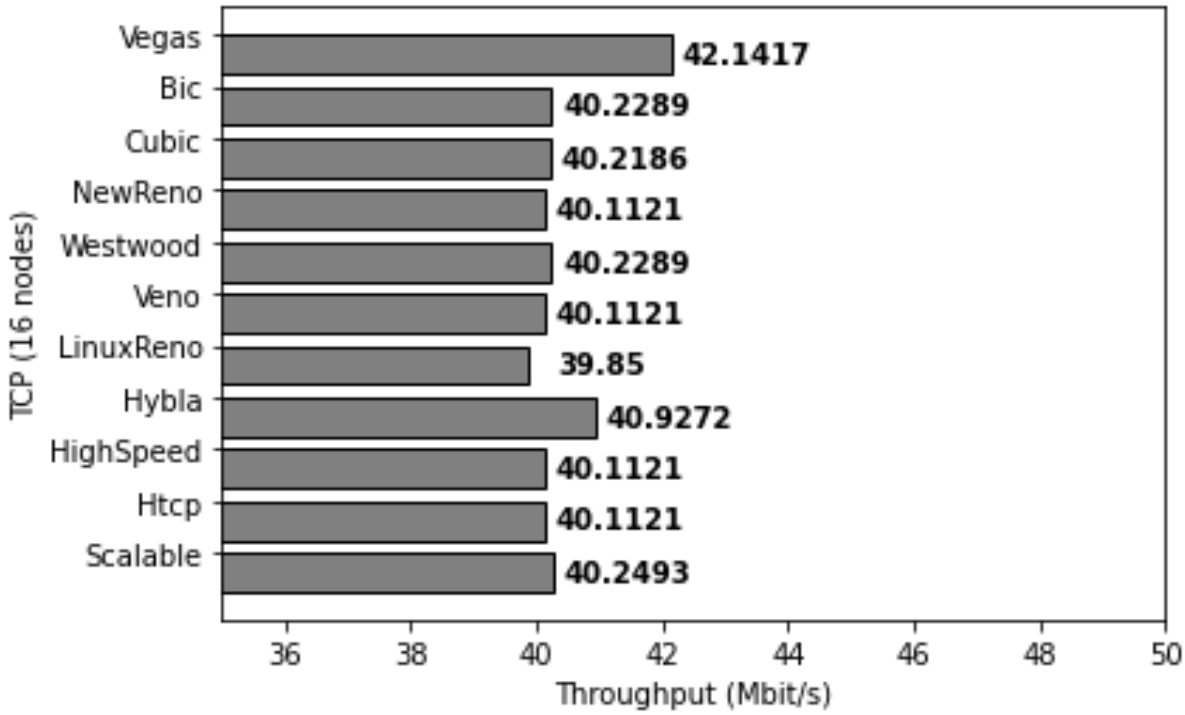

Figura 4.6: Gráfico de Throughput gerado por teste com 16 nós

É possível observar nas Figuras 4.5 e 4.6, houve uma estabilização dos valores de *throughput* dos algoritmos, onde alguns tiveram uma queda bem pequena e outros se mantiveram estáveis. A exceção foi o algoritmo Vegas, que teve um aumento em ambos

os gráficos observados, tornando-o assim o algoritmo com maior valor de *throughput* nos gráficos gerados para 12 e 16 nós.

Na Figura 4.6, observa-se também um aumento no valor do algoritmo Hybla em relação ao gráfico da Figura 4.5, tornando-o o algoritmo com segundo maior valor de *throughput* para o cenário de 16 nós.

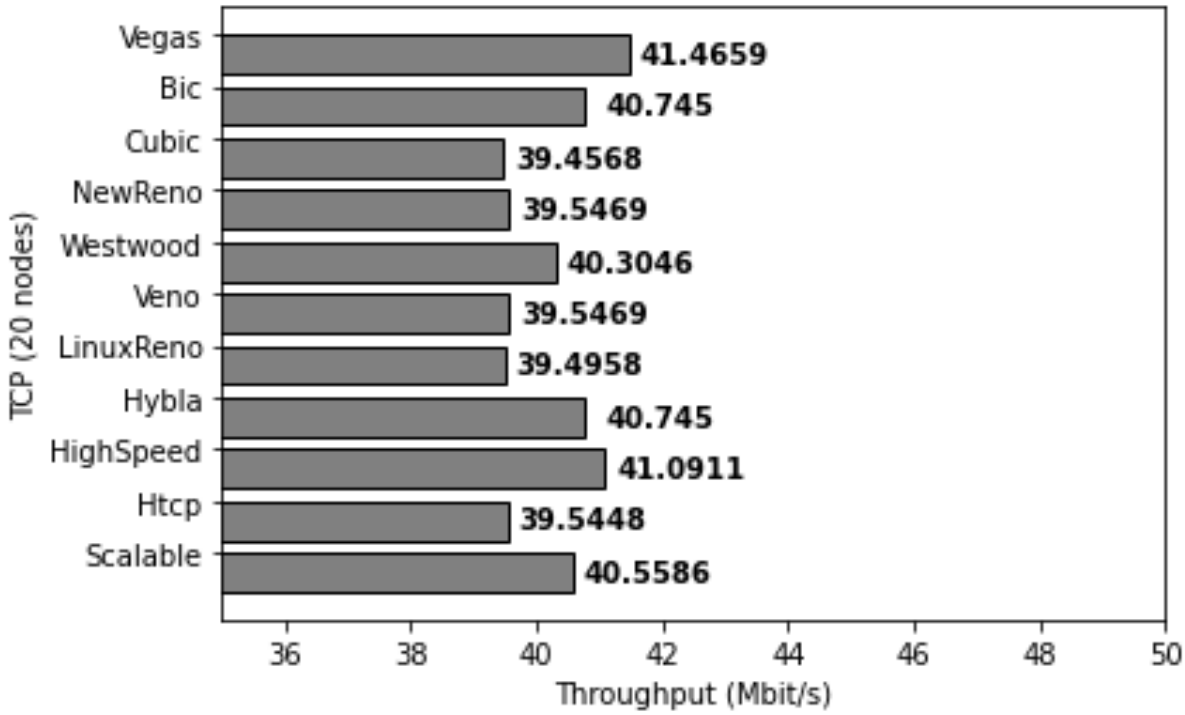

Figura 4.7: Gráfico de Throughput gerado por teste com 20 nós

Na Figura 4.7, pode-se observar uma pequena queda no valor de *throughput* do algoritmo Vegas, que nos três cenários anteriores só havia aumentado. Apesar dessa queda, o Vegas ainda continuou com o maior valor do teste com a rede de 20 nós. Quase todos os demais algoritmos também tiveram uma pequena queda, porém, é possível observar um pequeno aumento no valor do algoritmo Bic, e um aumento considerável no algoritmo *HighSpeed*.

4.1.2 Throughput e Goodput ao Longo do Tempo

Os gráficos apresentados a partir dessa seção foram gerados utilizando o *software Wireshark* a partir de arquivos com extensão pcap gerados na simulação com o NS-3. Para cada teste realizado, foram gerados arquivos pcap para cada nó da rede. O arquivo utilizado para a geração dos gráficos foi sempre o arquivo do segundo nó *station* de cada

algoritmo, ou seja, o segundo nó considerado cliente da rede, não contando o nó do *access point*, para assim ser possível analisar o impacto que esse segundo nó teve a cada cenário.

2 nós

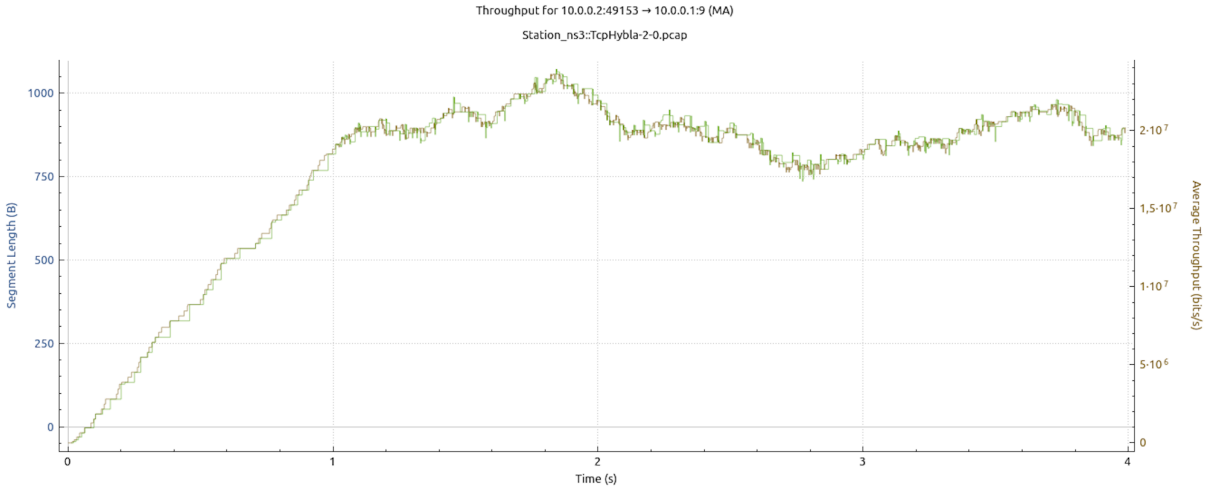

Figura 4.8: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 2 nós

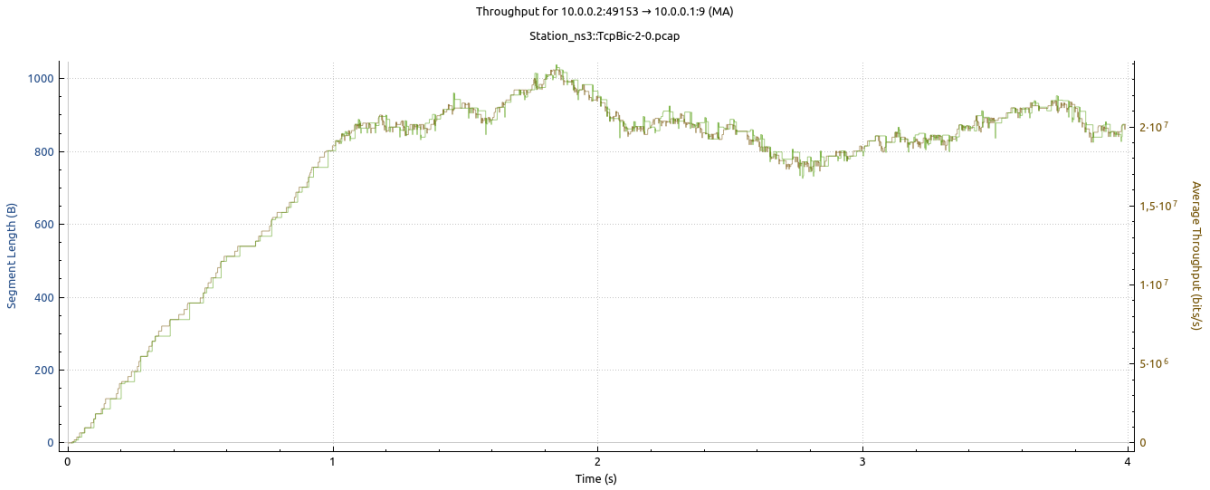

Figura 4.9: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 2 nós

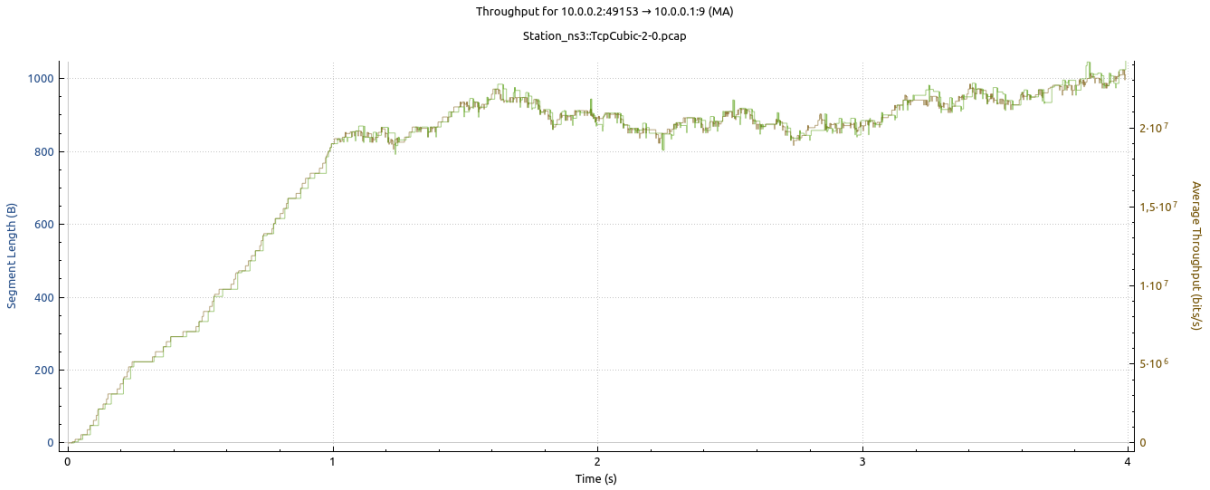

Figura 4.10: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 2 nós

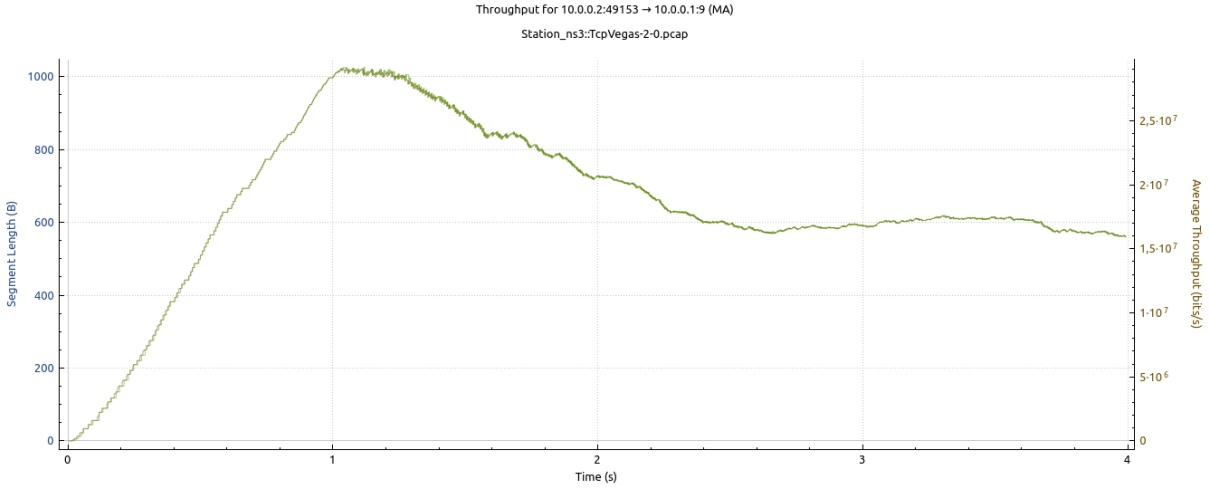

Figura 4.11: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 2 nós

É possível observar que os gráficos gerados por testes na rede com 2 nós e representados pelas Figuras 4.8, 4.9, 4.10 e 4.11, possibilitam ver que, no geral, o *throughput* e o *goodput* são bem próximos um do outro, principalmente no caso do algoritmo Vegas, onde praticamente não notamos as duas linhas do gráfico e sim apenas uma. Observa-se também que no Vegas, o *throughput* atinge um pico por volta de um segundo de teste e depois cai um pouco e se estabiliza, já os demais algoritmos tendem a crescer até certo ponto e variar pouco a partir dali.

4 nós

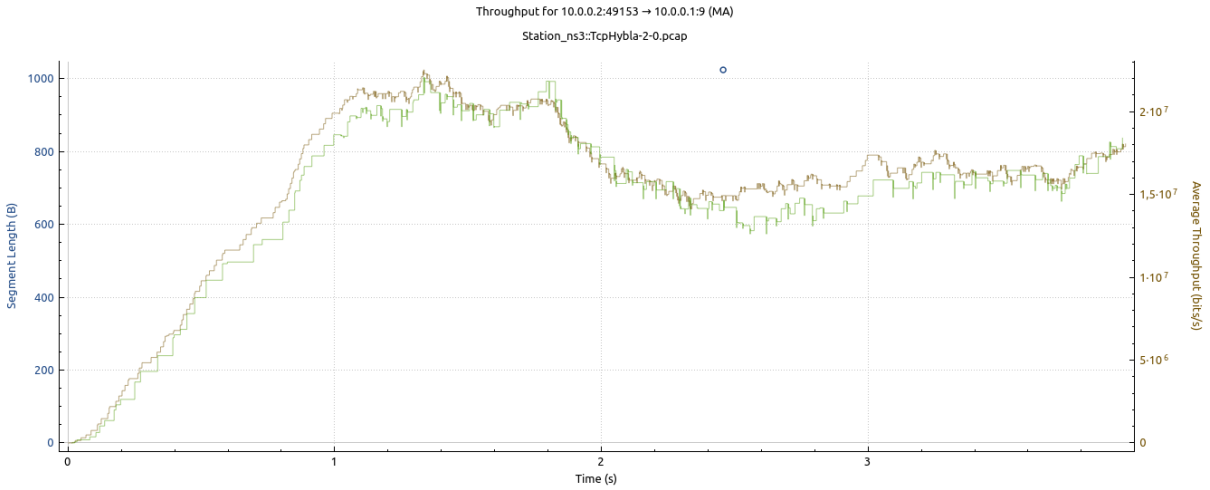

Figura 4.12: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 4 nós

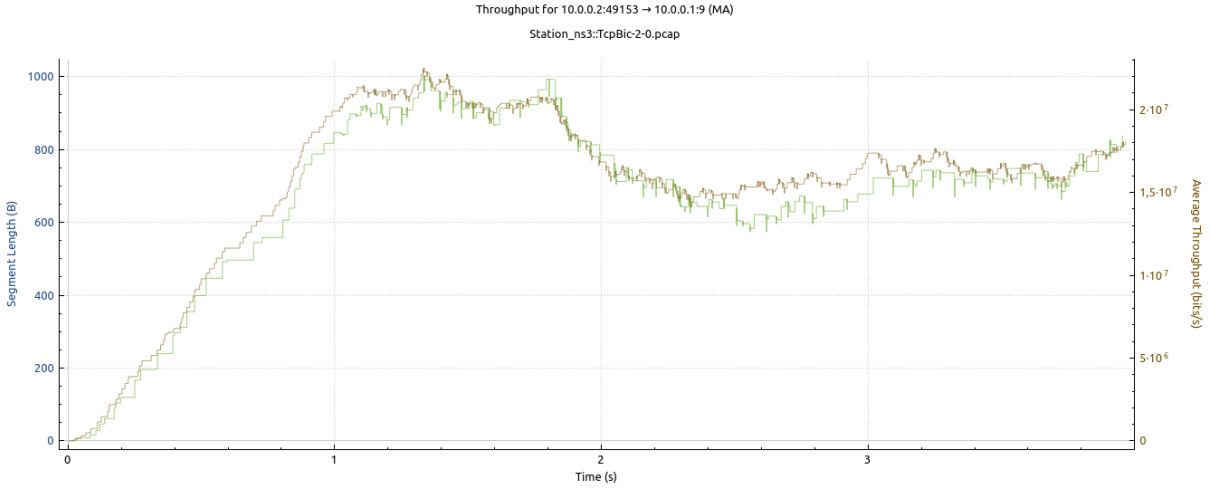

Figura 4.13: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 4 nós

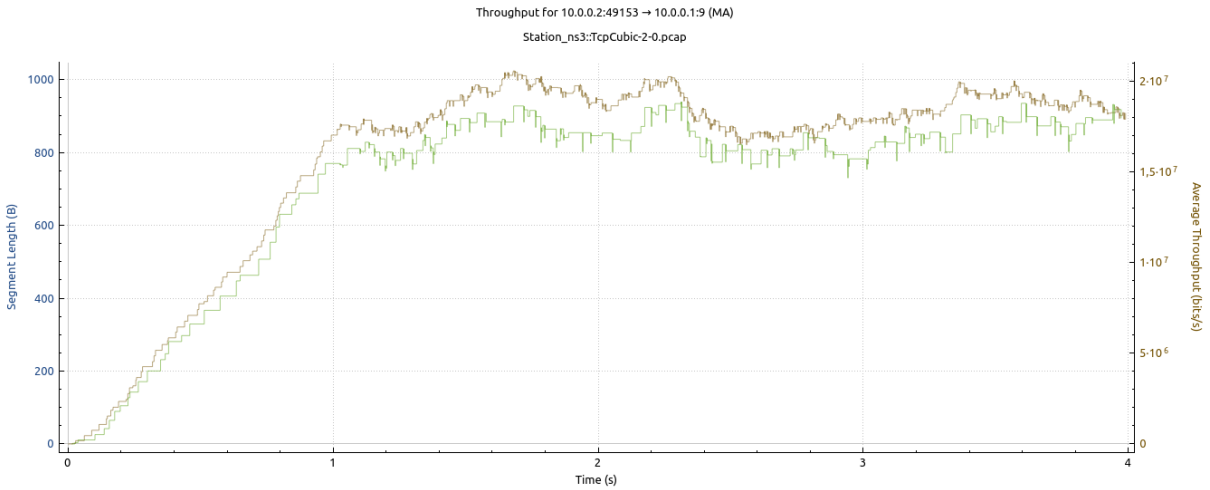

Figura 4.14: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 4 nós

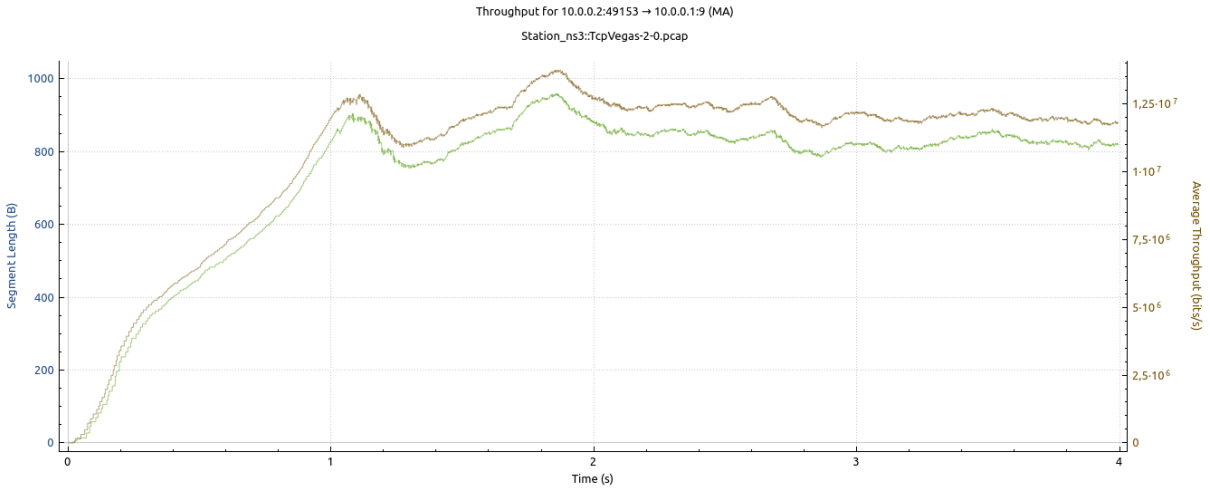

Figura 4.15: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 4 nós

Pode-se observar pelas figuras 4.12, 4.13, 4.14, 4.15 que, para o teste com 4 nós, Bic e Hybla apresentam *throughput* e *goodput* bem próximos um do outro e que o Vegas apresenta um gráfico mais estável, ou seja, com menos variações em relação aos outros três. Nota-se também que o *throughput* máximo atingido pelo Vegas ainda é menor que o atingido pelos demais algoritmos, assim como seu *throughput* médio, que havia sido observado na Figura 4.3

8 nós

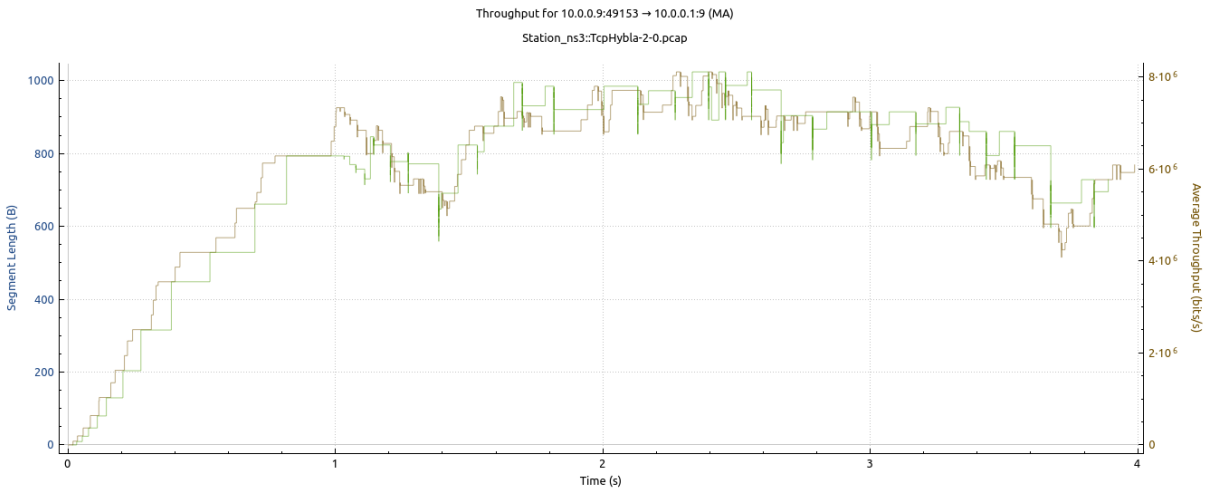

Figura 4.16: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 8 nós

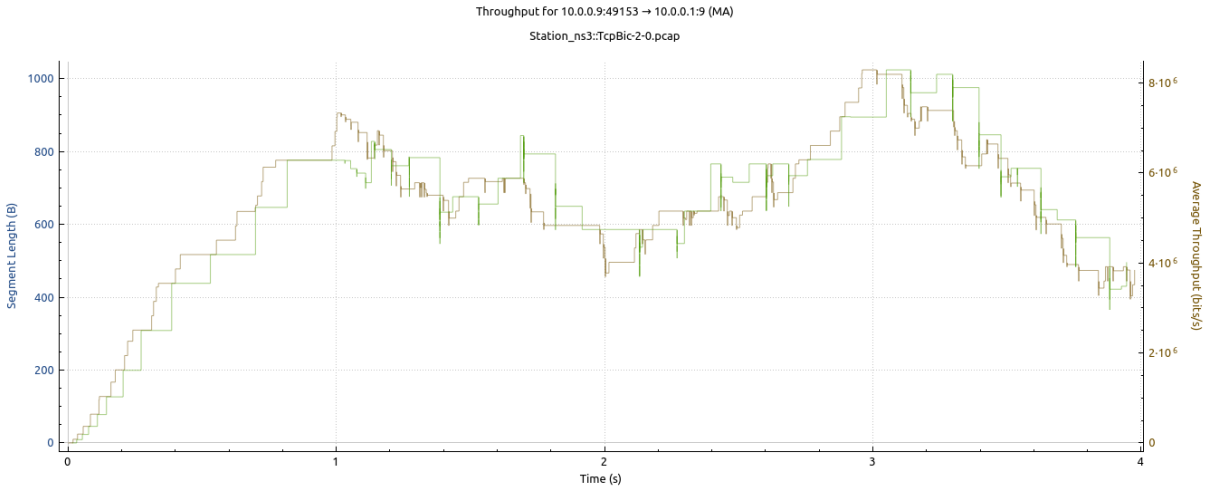

Figura 4.17: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 8 nós

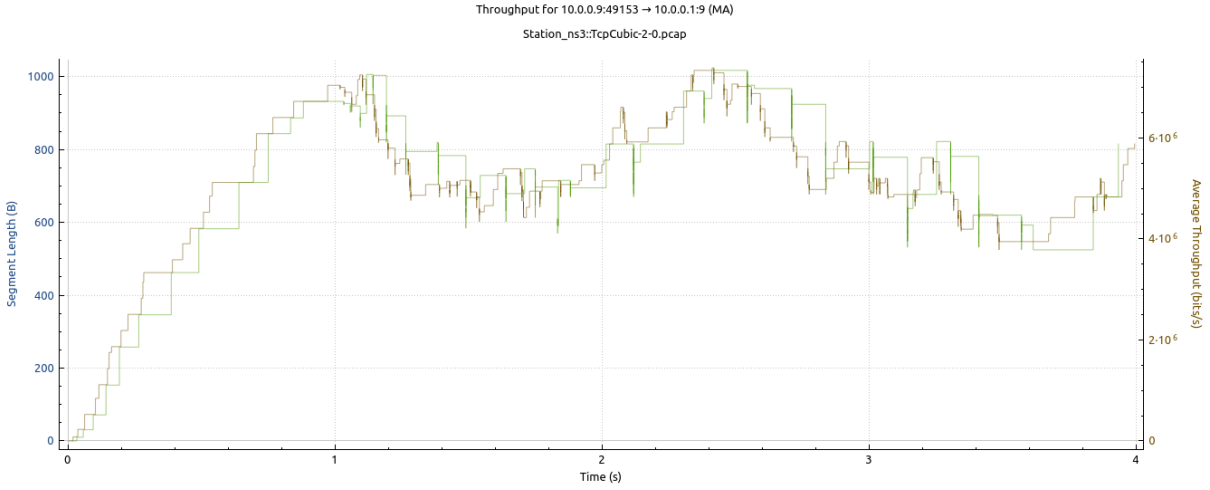

Figura 4.18: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 8 nós

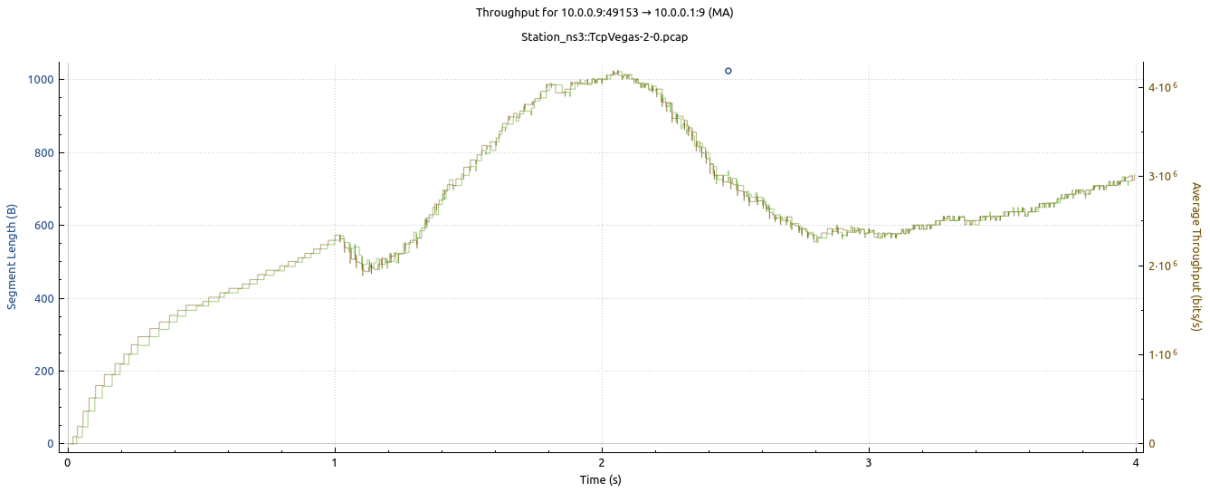

Figura 4.19: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 8 nós

12 nós

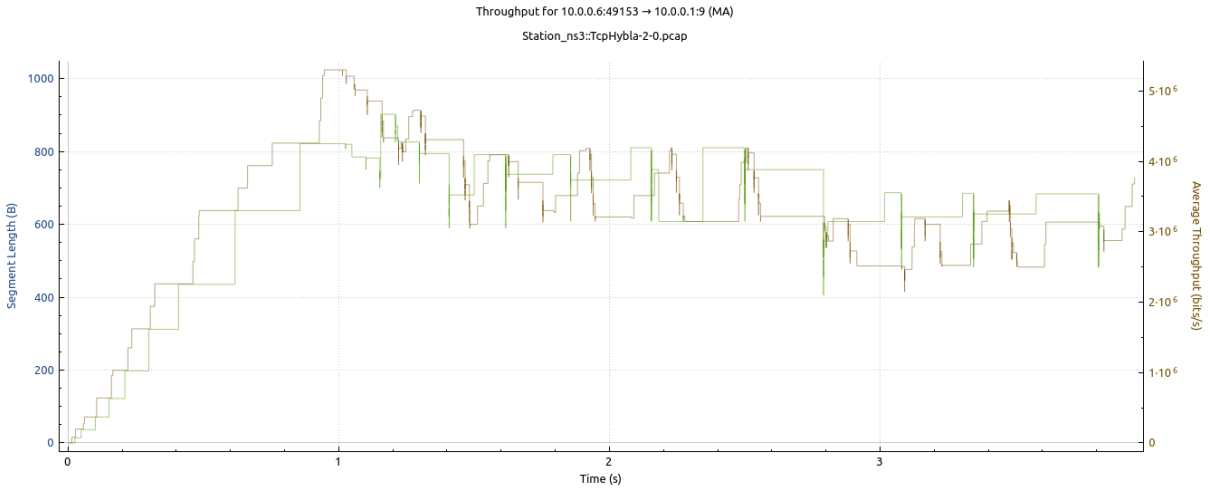

Figura 4.20: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 12 nós

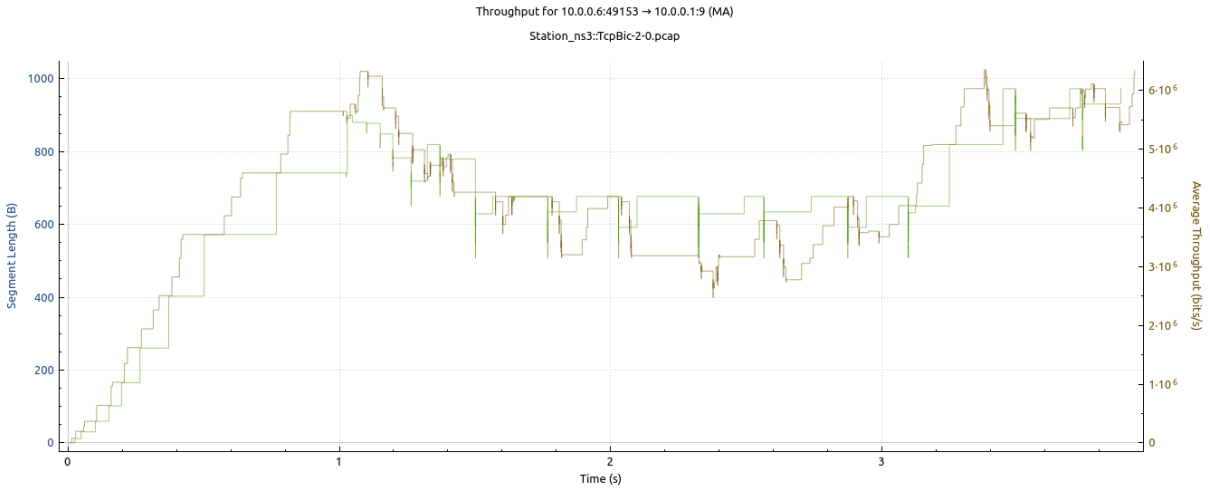

Figura 4.21: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 12 nós

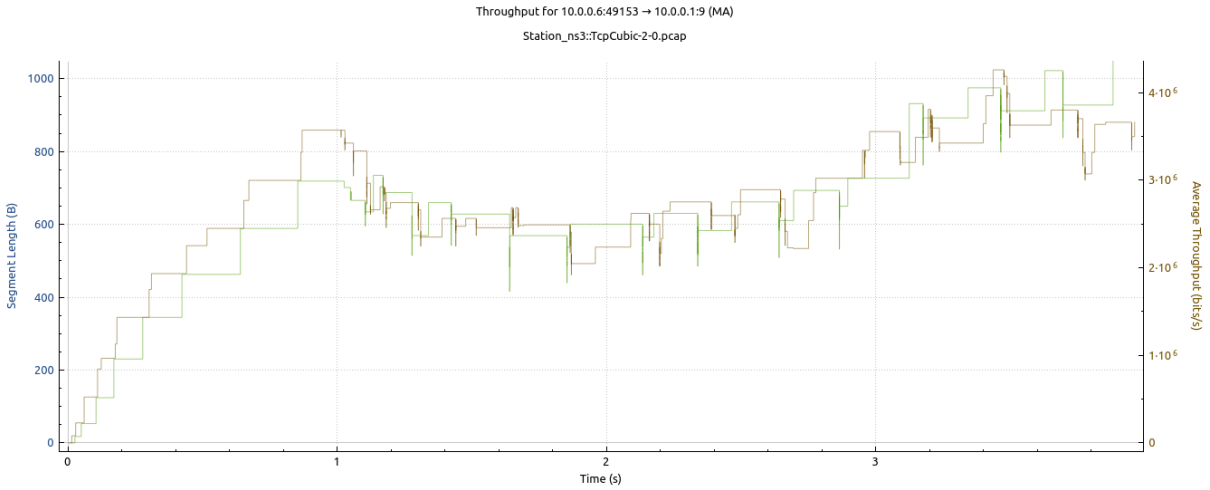

Figura 4.22: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 12 nós

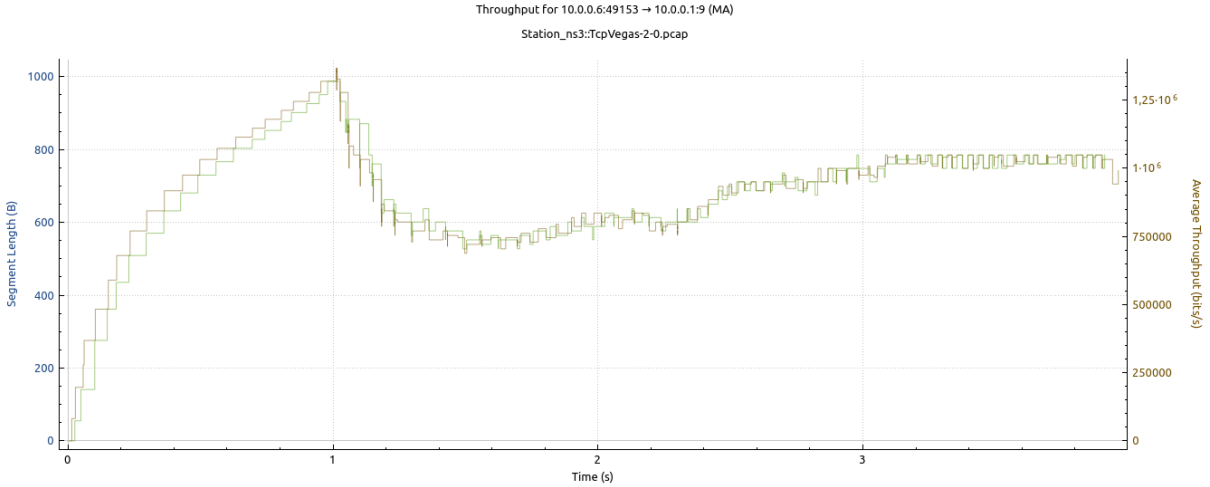

Figura 4.23: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 12 nós

Nos gráficos obtidos através dos testes para redes com 8 e 12 nós, pode-se notar uma grande variação entre o *throughput* e o *goodput* para os algoritmos Hybla, Bic e Cubic, já no Vegas as linhas de *throughput* e *goodput* estão mais próximas uma da outra, principalmente no gráfico para o teste com 8 nós.

16 nós

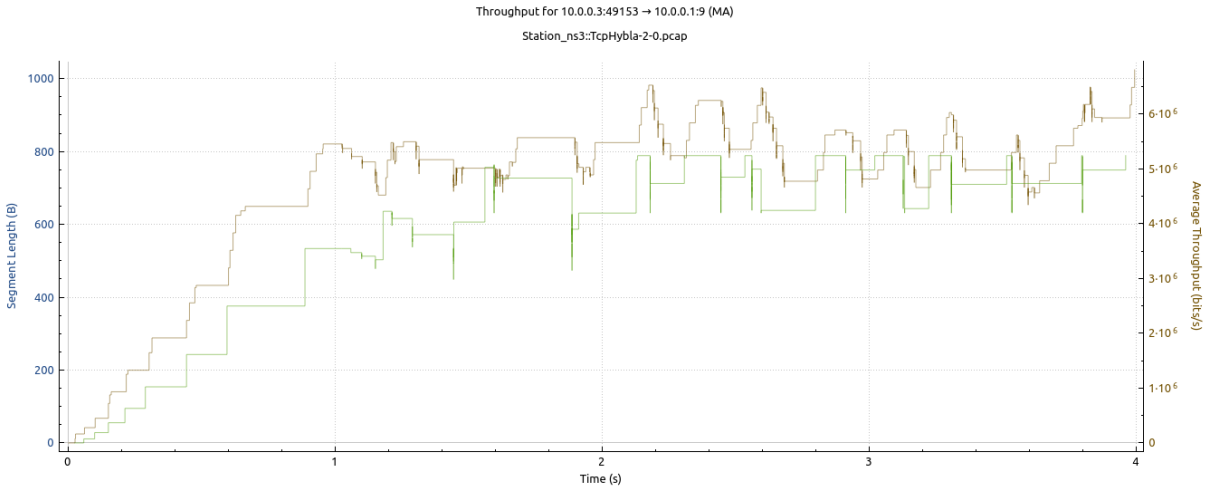

Figura 4.24: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 16 nós

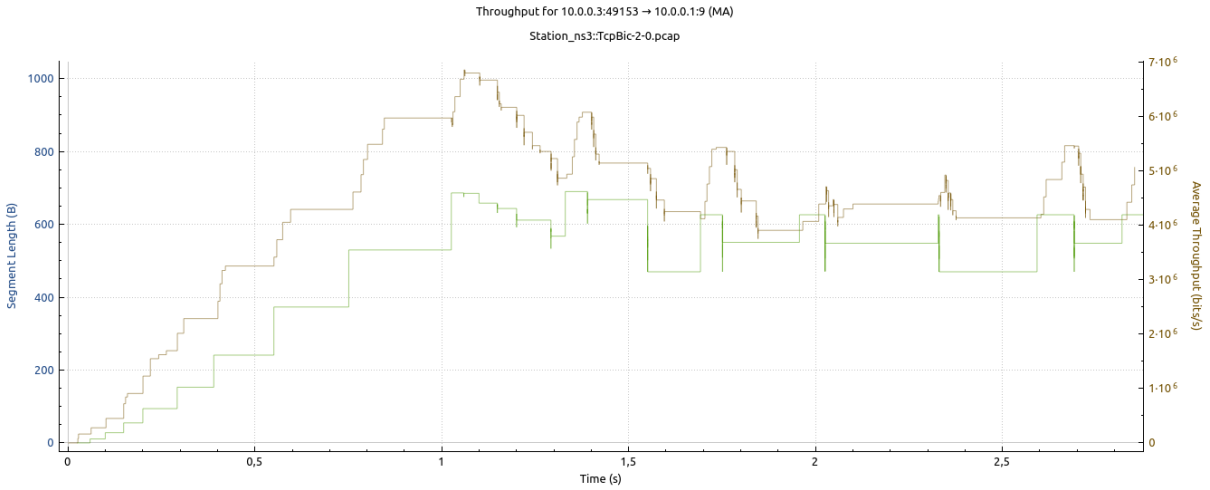

Figura 4.25: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 16 nós

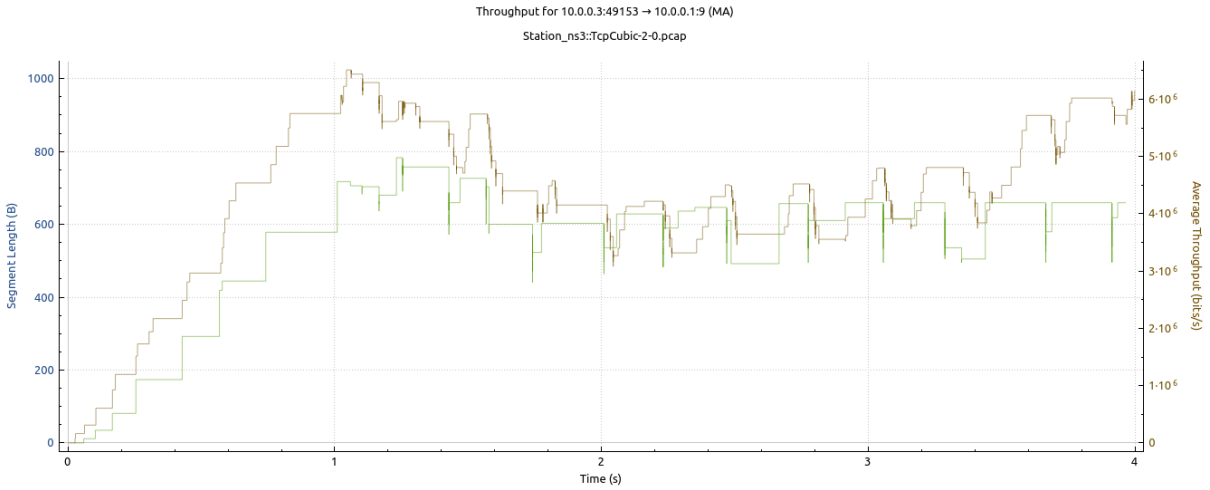

Figura 4.26: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 16 nós

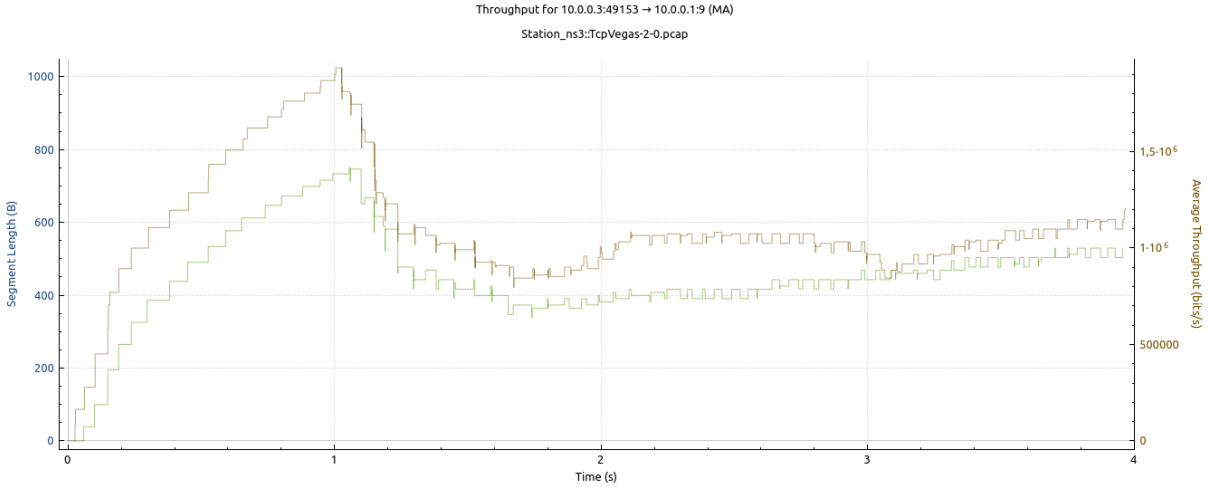

Figura 4.27: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 16 nós

20 nós

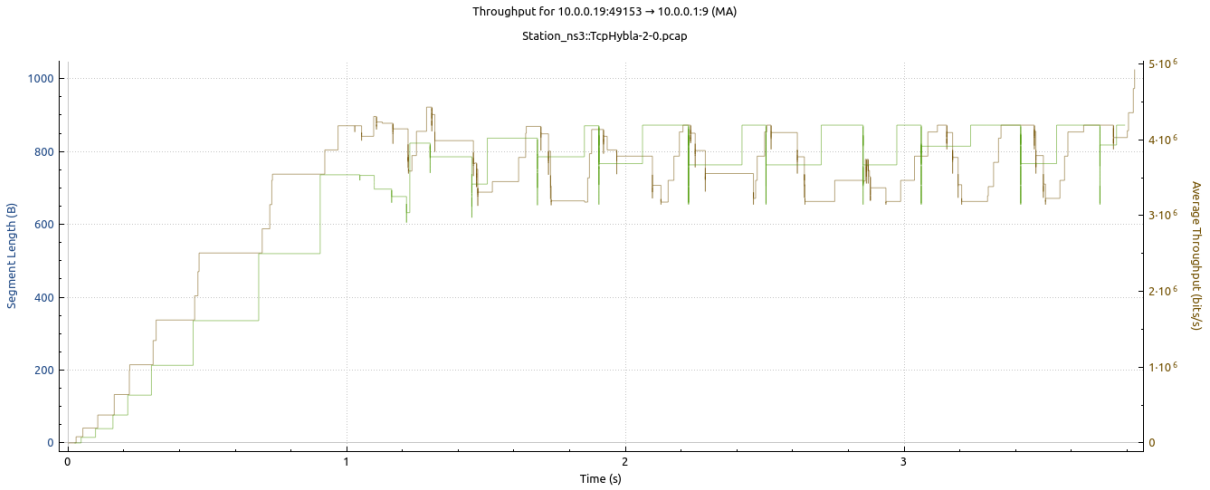

Figura 4.28: Gráfico de Throughput e Goodput do algoritmo Hybla gerado a partir do teste com 20 nós

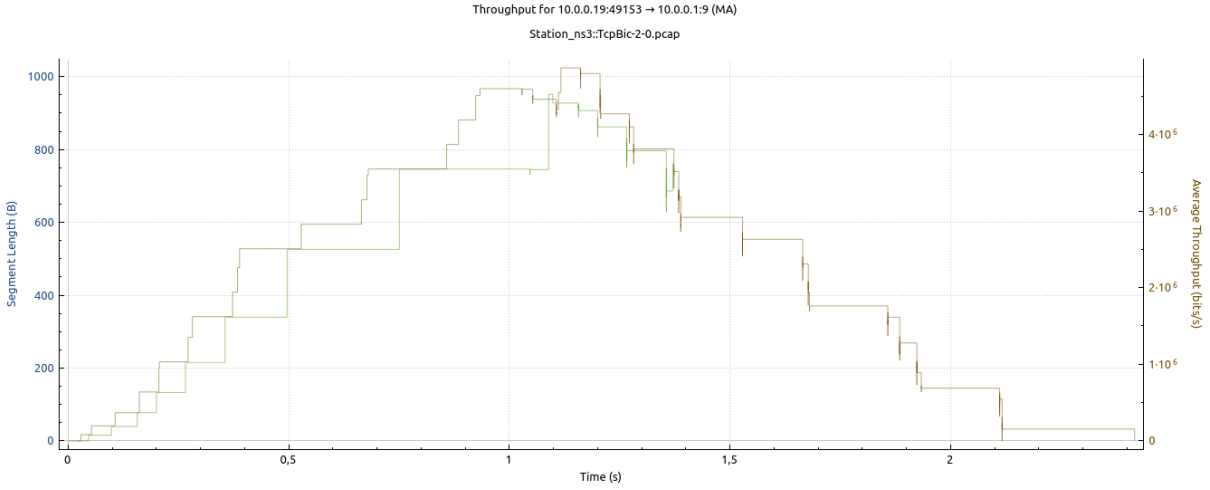

Figura 4.29: Gráfico de Throughput e Goodput do algoritmo Bic gerado a partir do teste com 20 nós

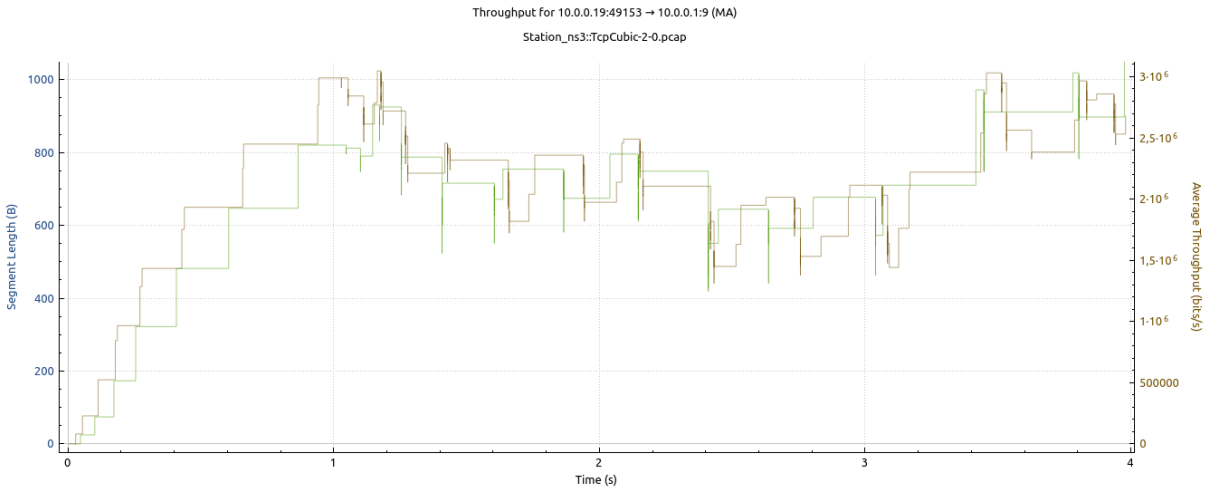

Figura 4.30: Gráfico de Throughput e Goodput do algoritmo Cubic gerado a partir do teste com 20 nós

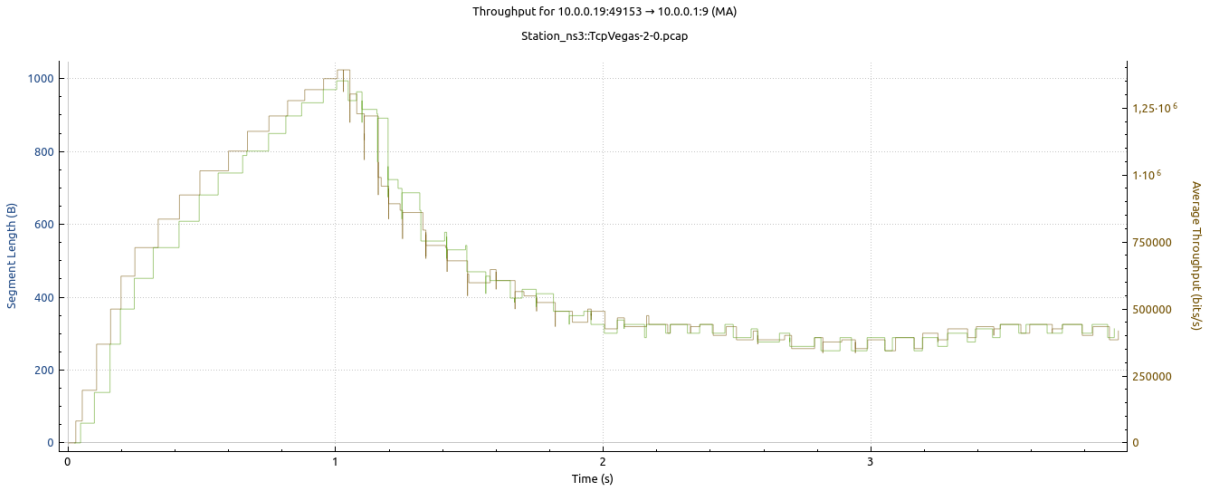

Figura 4.31: Gráfico de Throughput e Goodput do algoritmo Vegas gerado a partir do teste com 20 nós

Pode-se observar através dos gráficos dos testes com 16 e 20 nós, representados pelas Figuras 4.24, 4.25, 4.26, 4.27, 4.28, 4.29, 4.30 e 4.31 que para os algoritmos Hybla, Bic e Cubic, a diferença entre o *throughput* e o *goodput* é bem grande. Acredita-se que isso se deve pela grande quantidade de nós na rede. Já no caso do algoritmo Vegas, ocorre uma grande diferença entre *throughput* e *goodput* no primeiro segundo do teste com 16 nós, porém após esse começo a diferença diminui. Para o algoritmo Vegas no teste com 20 nós, a diferença se mantém bem baixa, principalmente se comparada à diferença dos demais algoritmos.

4.2 RTT

Para observar a métrica RTT, foram gerados gráficos no *software Wireshark* de RTT dos algoritmos Hybla e Vegas no cenário de 20 nós representados abaixo pelas Figuras 4.32 e 4.33.

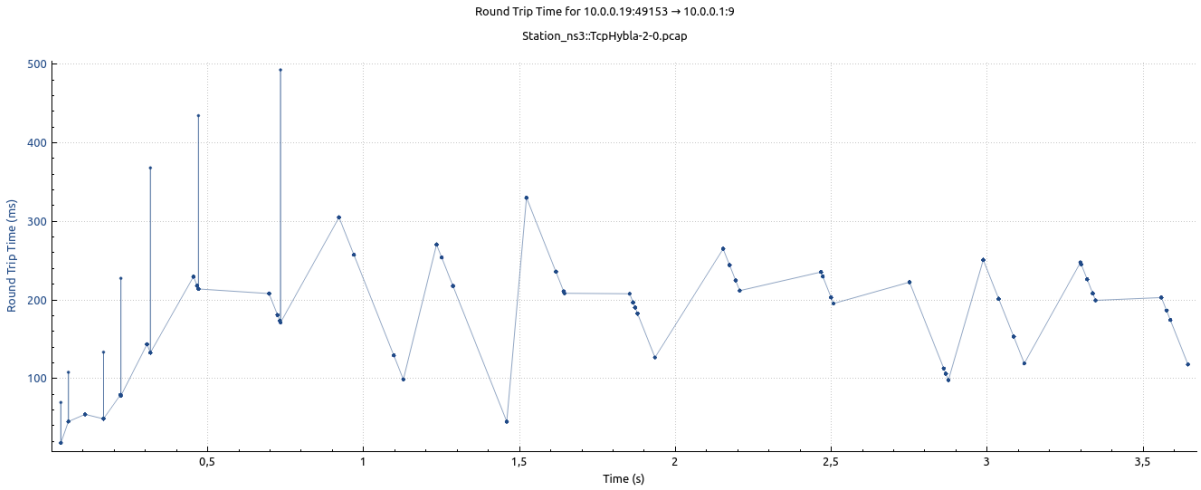

Figura 4.32: Gráfico de RTT do algoritmo Hybla gerado a partir do teste com 20 nós

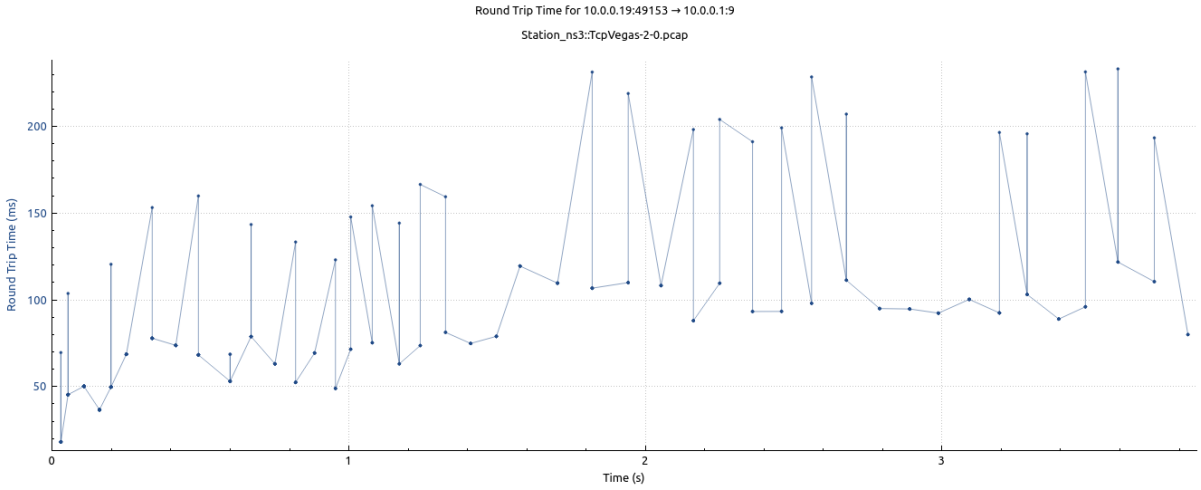

Figura 4.33: Gráfico de RTT do algoritmo Vegas gerado a partir do teste com 20 nós

Analisando os gráficos de RTT, é possível notar que o algoritmo Hybla tem uma menor variação do que o Vegas, além de ter atingido um pico de quase 500 ms. Já o Vegas varia mais, tendo aumentos e quedas constantes no RTT, porém o valor máximo de RTT atingido foi de aproximadamente 280 ms, quase metade do pico máximo observado no Hybla. Em termos de RTT, podemos concluir que, devido a essa grande diferença entre os dois algoritmos, o algoritmo Vegas se sobressai, pois uma alta latência (RTT) na rede

não é um cenário adequado.

4.3 Erros e Retransmissões

Por fim, temos a Figura 4.34, onde é possível separar as linhas de pacotes enviados, com erro e retransmissões. Observamos que o número de pacotes com erro ou retransmissões é pequeno. Com isso, é possível concluir que o ambiente em questão pode ter sido avaliado com pouca interferência, ou o modelo de perda utilizado na simulação criou um ambiente favorável às variantes TCPs clássicas ou que se utilizam bem de redes com banda livre e BDP, como é o caso do Vegas e das variantes *Hybla*, *HighSpeed* ou *Scalable*. A Tabela 4.2 sumariza os valores mostrados no gráfico, deixando clara a baixa taxa de pacotes retransmitidos para o Vegas e o Hybla.

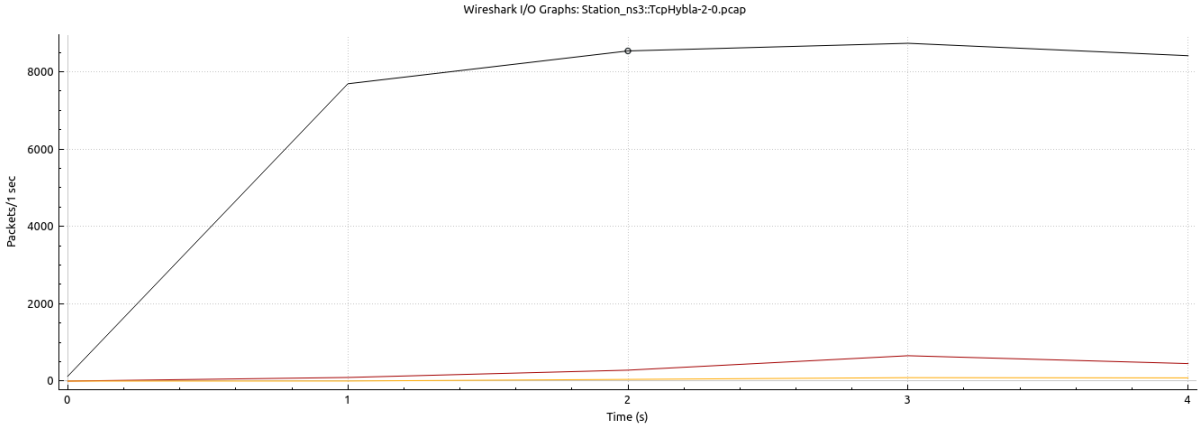

Figura 4.34: Gráfico pacotes/segundo para o Hybla com 20 nós. Linhas para total de pacotes (preto), erro (vermelho) e retransmitidos (laranja)

Tabela 4.2: Número de pacotes transmitidos e retransmitidos para TCP Vegas e Hybla.

Variante TCP	Total de Pacotes Transmitidos	Total de Pacotes Retransmitidos
Vegas	33.947	46 (0.1%)
Hybla	33.509	218 (0.7%)

5 Conclusões

Neste trabalho foram avaliadas diversas variações do protocolo TCP (*i.e.*, um total de 11 variantes) sobre um ambiente de rede sem fio com combinações diferentes de nós sem fio. A rede utilizada foi do tipo infraestruturada sem fio com 1 ponto de acesso e [2,20] nós clientes. Todas as avaliações levaram em consideração o protocolo IEEE 802.11n como padrão de comunicação.

No intuito de entender o funcionamento do protocolo TCP e suas diversas variantes, esses foram avaliados em relação ao *Throughput*, perda e RTT. Mesmo em uma rede sem fio simples, como a apresentada, é possível verificar a redução do desempenho de algumas variantes TCP em relação a outras. Sabemos que é um desafio definir um protocolo TCP para um ambiente específico considerando o atraso entre perda (BDP), ou em redes de RTT muito alto, como é o caso de redes sem fio densas e com nós distantes.

Assim, é possível concluir que, pela avaliação, algoritmos como Cubic e Vegas se sobressaíram para esses cenários de rede sem fio. O Cubic por manter uma boa vazão para qualquer quantidade de nós. Já o Vegas, que, apesar de ter uma baixa vazão nos cenários iniciais, tem a melhor vazão a partir de uma certa quantidade de nós. Além disso, o Vegas tem, em geral, valores de *goodput* próximos aos de *throughput* e apresenta menor perda de pacotes entre os algoritmos em todos os cenários. Além dos fatores citados, como perda e atraso, a tecnologia de comunicação adotada (*e.g.* IEEE 802.11n com MIMO) influencia diretamente na performance do protocolo, assim como outras técnicas de encaminhamento de pacotes (*e.g.*, enfileiramento). Assim, achamos interessante um futuro estudo e avaliação dos valores de parâmetros para alguns dos protocolos TCP variantes analisados. Por exemplo, o protocolo *Cubic* se mostrou bem comportado em termos de vazão para quaisquer combinações de nós. Portanto, pode ser interessante avaliar seu ajuste de valores de parâmetros em um próximo trabalho. Além disso, gostaríamos de avaliar no futuro o impacto da mobilidade, e da tecnologia de rede sem fio utilizada, como o padrão IEEE 802.11ax (Wi-Fi 6).

Bibliografia

CAINI, C.; FIRRINCIELI, R. Tcp hybla: a tcp enhancement for heterogeneous networks. *International journal of satellite communications and networking*, Wiley Online Library, v. 22, n. 5, p. 547–566, 2004.

CERF, V.; KAHN, R. A protocol for packet network intercommunication. *IEEE Transactions on communications*, IEEE, v. 22, n. 5, p. 637–648, 1974.

CHAUDHARY, P.; KUMAR, S. Comparative study of tcp variants for congestion control in wireless network. In: IEEE. *2017 International conference on computing, communication and automation (ICCCA)*. [S.l.], 2017. p. 641–646.

FLOYD, S. *RFC3649: HighSpeed TCP for large congestion windows*. [S.l.]: RFC editor, 2003.

FLOYD, S. et al. The newreno modification to tcp's fast recovery algorithm. RFc 2582, April, 1999.

FU, C. P.; LIEW, S. C. Tcp veno: Tcp enhancement for transmission over wireless access networks. *IEEE Journal on selected areas in communications*, IEEE, v. 21, n. 2, p. 216–228, 2003.

GAGLIONI, C. *Como a pandemia afeta a infraestrutura da internet*. 2019. Disponível em: ⟨"https://nic.br/noticia/na-midia/como-a-pandemia-afeta-a-infraestrutura-da-internet/"⟩.

JACOBSON, V. Modified tcp congestion avoidance algorithm. *Email to the end2end-interest mailing list*, 1990.

KELLY, T. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM computer communication Review*, ACM New York, NY, USA, v. 33, n. 2, p. 83–91, 2003.

KEMP, S. *Digital 2021: Global Overview Report*. 2021. Disponível em: ⟨"https://datareportal.com/reports/digital-2021-global-overview-report"⟩.

KUROSE, J.; ROSS, K. *Redes de computadores e a Internet: uma nova abordagem*. [S.l.: s.n.], 2000.

LEITH, D.; SHORTEN, R. H-tcp: Tcp for high-speed and long-distance networks. In: *Proceedings of PFLDnet*. [S.l.: s.n.], 2004. v. 2004.

LEITH, D. J.; SHORTEN, R. N. Impact of drop synchronisation on tcp fairness in high bandwidth-delay product networks. 2006.

MASCOLO, S. et al. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In: *Proceedings of the 7th annual international conference on Mobile computing and networking*. [S.l.: s.n.], 2001. p. 287–297.

MOLIA, H. K.; KOTHARI, A. D. Tcp variants for mobile adhoc networks: Challenges and solutions. *Wireless Personal Communications*, Springer, v. 100, n. 4, p. 1791–1836, 2018.

NIGAR, N.; AZIM, M. A. Fairness comparison of tcp variants over proactive and reactive routing protocol in manet. *International Journal of Electrical and Computer Engineering*, IAES Institute of Advanced Engineering and Science, v. 8, n. 4, p. 2199, 2018.

ONU. *Estudo da ONU revela que mundo tem abismo digital de gênero*. 2019. Disponível em: ⟨"https://news.un.org/pt/story/2019/11/1693711"⟩.

TANENBAUM, A. S. *Redes de Computadores*. 5ª. ed. [S.l.: s.n.], 2011.

TARUK, M. et al. Comparison of tcp variants in long term evolution (lte). In: IEEE. *2017 5th international conference on electrical, electronics and information engineering (ICEEIE)*. [S.l.], 2017. p. 131–134.

WAGHMARE, S. et al. Comparative analysis of different tcp variants in a wireless environment. In: IEEE. *2011 3rd international conference on electronics computer technology*. [S.l.], 2011. v. 4, p. 158–162.