UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
TECNÓLOGO EM REDES DE COMPUTADORES

FELIPE ALEX MARTINS DE SOUZA

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE CONTROLE DE CONGESTIONAMENTO TCP UTILIZANDO O SIMULADOR NS-2

QUIXADÁ
2014

FELIPE ALEX MARTINS DE SOUZA

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE CONTROLE DE CONGESTIONAMENTO TCP UTILIZANDO O SIMULADOR NS-2

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: computação

Orientador: Prof. MSc. Marcos Dantas

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

S715a Souza, Felipe Alex Martins de
Análise de desempenho de algoritmos de controle de congestionamento TCP utilizando o simulador NS-2 / Felipe Alex Martins de Souza. – 2014.
67 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Tecnologia em Redes de Computadores, Quixadá, 2014.
Orientação: Prof. Me. Marcos Dantas Ortiz
Área de concentração: Computação

1. Algoritmos computacionais 2. TCP/IP (Protocolo de rede de computador) 3. Redes de computadores I. Título.

CDD 005.1

FELIPE ALEX MARTINS DE SOUZA

ANÁLISE DE DESEMPENHO DE ALGORITMOS DE CONTROLE DE CONGESTIONAMENTO TCP UTILIZANDO O SIMULADOR NS-2

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: computação

Aprovado em: _____ / junho / 2014.

BANCA EXAMINADORA

_____
Prof. MSc. Marcos Dantas Ortiz (Orientador)
Universidade Federal do Ceará-UFC

_____
Prof. Dr. Arthur de Castro Callado
Universidade Federal do Ceará-UFC

_____
Prof. MSc. Paulo Antonio Leal Rego
Universidade Federal do Ceará-UFC

Dedico aos meus pais Antonio e Francisca, que sempre me apoiaram e me ensinaram a ter coragem e garra para conseguir alcançar os meus objetivos. Eu não conseguiria chegar até aqui sem o amor de vocês. Sou muito grato por tudo o que vocês fizeram e fazem por mim. Amo muito vocês e sempre vou amá-los.

AGRADECIMENTOS

Primeiramente a Deus por ter me proporcionado o dom da vida.

Ao prof. Marcos Dantas Ortiz, pelo ensino no decorrer da minha formação e pela orientação deste trabalho.

A todos os professores que me ensinaram e terem compartilhado seus conhecimentos no decorrer deste curso.

Aos meus amigos que não foram só de turma, Glailton Costa e Everton Monteiro juntamente com seus familiares, que me acolheram em suas residências por algum tempo, sempre serei grato por terem me suportado em seus domicílios, nunca me esquecerei, tenho vocês como meus irmãos.

Aos meus amigos de turma, Adonai Filho, Aline Oliveira, Carlos Bruno, David Santiago, Egberto Barreto, Evelyne Avelino, Francisco Nobre, Jammes Wilker, João Faustino, Joel Pereira, Júlio César, Maicon Camurça, Marcelo Miranda, Otacílio Aguiar, Paulo Victor Estevam, Rafael Pinheiro, Sebastião Nogueira, Tallis Maia e Thiago Torres, que me ensinaram muitas coisas, me proporcionaram alegrias e que são um exemplo de união entre colegas de turma.

Aos meus amigos de curso, em especial, Atricia Sabino, Clycia Najara, Eudes Sousa, Edigleison Barbosa, Janael Pinheiro, Luclécia Correia, Matheus Medeiros, Paulo Júnior, Rejane Lemos.

Aos meus irmãos Célis Henrique e Davi Eric, principalmente ao Henrique que conviveu mais tempo comigo e sempre me apoiou sabendo do que nós passamos no decorrer da vida.

A todos os meus familiares que acreditaram na concretização deste trabalho.

A todos aqueles que acreditaram em mim e também aqueles que não acreditaram (colegas, conhecidos, professores do curso). Vocês só me deram mais forças para conseguir e mostrá-los que nada é impossível quando se quer alcançar seus objetivos.

"Seja quem você for, seja qual for à posição social que você tenha na vida, de nível altíssimo ou o mais baixo, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá."

(Ayrton Senna)

RESUMO

O aumento significativo dos dispositivos conectados à rede mundial de computadores impactou também o aumento de demanda de tráfego. Quando a demanda de tráfego é maior que a capacidade de transmissão, ocorre congestionamento. Os algoritmos de controle de congestionamento foram criados para tentar solucionar esse problema, diminuindo o envio de tráfego de dados quando acontece congestionamento. O TCP Tahoe foi o primeiro algoritmo de controle de congestionamento criado para resolver esse problema. Com o passar dos anos, e com o aumento de demanda de tráfego, foram criados outros algoritmos de controle de congestionamento como o TCP Reno, TCP Vegas, TCP NewReno, TCP Sack e TCP Fack. O objetivo deste trabalho é analisar algoritmos de controle de congestionamento utilizando o simulador *Network Simulator*, simulador que é bastante utilizado para trabalhos acadêmicos. As métricas utilizadas para analise foram: descarte, taxa de pacotes recebidos e utilização dos *links*. Os cenários de testes simularam duas topologias da Rede Nacional de Pesquisa (topologia do ano de 2013 e a topologia atual), que englobam todos os estados brasileiros.

Palavras chave: Análise de desempenho, TCP, controle de congestionamento, simulação.

ABSTRACT

The significant increase of the devices connected to the world wide web also impacted the increase in traffic demand. Congestion occurs when the traffic demand is greater than the transmission capacity. The congestion control algorithms have been created to try to solve this problem by decreasing the transmission of data traffic when congestion occurs. The TCP Tahoe was the first congestion control algorithm designed to solve this problem. Over the years and with the increase of traffic demand, other congestion control algorithms such as TCP Reno, TCP Vegas, TCP NewReno, TCP SACK and TCP Fack were created. The objective of this work is to analyze algorithms of congestion control using simulator Network Simulator, simulator that is widely used for academic papers. The metrics used for analysis were: discharge, rate of incoming packets and use the links. The test scenarios simulated two topologies of the National Research Network (topology of the year 2013 and the current topology), which encompass all Brazilian states.

Keywords: Performance analysis, TCP, congestion control, simulation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O mundo se tornou cada vez mais dependente da tecnologia. Antigamente os dispositivos da rede mundial de computadores, chamada *Internet*, eram estações de trabalho, computadores de mesa e servidores que respondiam requisições e armazenavam dados de seus usuários finais. Mas a evolução tecnológica alcançou os sistemas finais, onde *smartphones*, TVs, *notebooks*, automóveis e sensores também entraram na rede mundial de computadores (KUROSE; ROSS, 2010).

De acordo com Kurose e Ross (2010, p. 02) "O termo *rede de computadores* está começando a soar desatualizado, dados aos equipamentos não tradicionais que estão sendo ligados à Internet".

Com o aumento do número de dispositivos conectados à Internet, também houve um aumento de demanda e de tráfego na rede. Protocolos mais sofisticados são criados para o processamento dessas informações. Um dos trabalhos mais importantes da rede está no roteamento de pacotes, que é o envio de dados de um sistema final a outro (KUROSE; ROSS, 2010).

As camadas da pilha TCP/IP que trabalham com a transmissão e encaminhamento dos pacotes são, respectivamente, as camadas de transporte e de rede. A camada de transporte fornece a comunicação e a transmissão dos dados entre sistemas finais, enquanto a camada de rede faz o roteamento e o encaminhamento dos dados.

Quando o número de pacotes enviados de um sistema final a outro é maior que a capacidade da rede utilizada, acontece o congestionamento. A responsabilidade de lidar com o congestionamento fica por conta das camadas de rede e transporte. A camada de rede tenta gerenciar o congestionamento, utilizando protocolos de roteamento para alternar rotas de dados, enquanto o serviço mais difícil fica por conta do TCP (*Transmission Control Protocol*). O TCP reduz a taxa de transmissão de dados quando ocorre congestionamento. Existem algoritmos de controle de congestionamento nas implementações do TCP que tentam minimizar e controlar esse estado da rede (KUROSE; ROSS, 2010; TANENBAUM, 2011).

Cavalcanti (2005) apresenta um trabalho no qual utiliza algoritmos de controle de congestionamento TCP, que são: TCP Tahoe, TCP Reno, TCP NewReno e TCP Vegas, em uma topologia com 6 nós (dois nós origem e dois nós destino) para verificar qual algoritmo teria um melhor desempenho com as métricas que foram estabelecidas. O resultado obtido neste trabalho foi o melhor desempenho do algoritmo TCP Vegas em relação às outras

implementações utilizadas. Em Prete e Shinoda (2009) foram utilizados os algoritmos TCP Tahoe, TCP Reno, TCP Vegas e o TCP NewReno, em uma topologia com 6 nós (4 fontes de tráfego e 1 nó receptor) sendo que cada fonte de trafego é para um algoritmo de controle de congestionamento que foi utilizado. Em Benitez (2010), ele compara os algoritmos de controle de congestionamento TCP Tahoe, TCP Reno, TCP Vegas, TCP NewReno juntamente com o Westwood, usando uma topologia com 8 nós (3 fontes de tráfego e 3 de destino). As métricas utilizadas foram: quantidade de retransmissões, vazão, desperdício e transferência efetiva. Esses trabalhos utilizaram o simulador Ns-2(*Network Simulator*, versão 2). Já em Abed, Ismail e Jumari (2011) utilizam todos os algoritmos de congestionamento propostos neste trabalho, mas não avaliaram e nem compararam os algoritmos em uma ferramenta de simulação ou emulação. Os autores apenas citam os mecanismos que cada algoritmo de congestionamento utiliza.

Este trabalho propõe uma análise de desempenho dos algoritmos de controle de congestionamento TCP, em ambiente simulado. Serão analisados os algoritmos de congestionamento implementados pelo TCP Tahoe, TCP Reno, TCP Vegas, TCP NewReno, TCP Sack, TCP Fack. O TCP Tahoe foi escolhido por ser a primeira implementação (CAVALCANTI, 2005; KUROSE; ROSS, 2010). O TCP Reno foi escolhido por ser um dos algoritmos que são mais utilizados nas implementações TCP (2001, apud PADHYE; KUROSE; ROSS, 2010, p.209). O TCP Vegas foi escolhido por ter um mecanismo pró-ativo para evitar o congestionamento (CAVALCANTI, 2005; CÁCERES, 2010). O TCP New Reno e foi escolhido por ser uma versão atualizada do TCP Reno (CAVALCANTI, 2005; PRETE, SHINODA, 2009; BENITEZ, 2010). O TCP Sack foi escolhido por tentar resolver os problemas do TCP Reno (RFC 2018, 1996; ABED; ISMAIL; JUMARI, 2011), e TCP Fack foi escolhido por trabalhar juntamente com o TCP Sack, mas com mecanismo diferente (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

Para construção e execução dos cenários de testes, foi utilizado o simulador de redes *Network Simulator* versão 2 (ns-2), que implementa todos os algoritmos de congestionamento validados neste trabalho (FALL; VARADHAN, 2013). Os cenários de testes implementaram situações de congestionamento como a capacidade de processamento do roteador menor do que o envio de dados e fluxos diferentes de envio de dados.

O estudo, análise e comparação desses algoritmos auxiliará um administrador de rede na escolha de qual algoritmo de congestionamento utilizar para obter melhor desempenho em cada tipo de situação de congestionamento identificada.

No capítulo 3 são abordados os aspectos importantes do protocolo TCP, apresentando suas principais características. Neste capítulo também é abordado o funcionamento e os algoritmos de controle de congestionamento TCP que foram implementados e utilizados no trabalho. É abordado também neste capítulo sobre os conceitos de simulação, mostrando as suas vantagens. Além disso, o capítulo 3 aborda as características e vantagens do simulador NS-2. No capítulo 4 são abordados os procedimentos utilizados para a execução do trabalho. No capítulo 5 são descritos os modelos e os parâmetros de rede que foram utilizados para a elaboração dos cenários de simulação. O capítulo 6 descreve os resultados das análises das simulações realizadas, mostrando o desempenho dos algoritmos de controle de congestionamento utilizados.

2 OBJETIVOS

2.1 Objetivo geral

Identificar entre os algoritmos de congestionamento TCP qual algoritmo apresenta melhor desempenho no controle de congestionamento em diferentes cenários.

2.2 Objetivos específicos

- Criar cenários de rede com aspectos que podem afetar ou não a avaliação de desempenho (fatores) e valores usados para cada um dos fatores (níveis) diferentes.

- Implantar, configurar e utilizar o ambiente do simulador NS2 para a execução dos cenários de rede.

- Avaliar os algoritmos de congestionamento TCP através dos resultados obtidos das simulações.

3 REVISÃO BIBLIOGRÁFICA

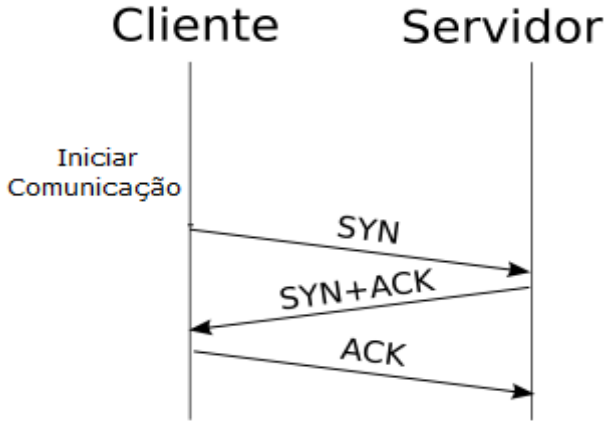
3.1 PROTOCOLO TCP

O TCP é um protocolo da camada de transporte da pilha TCP/IP e o seu principal objetivo é a entrega confiável dos fluxos de dados. Visto como o protocolo mais utilizado na *Internet*, o TCP também inclui mecanismos de controle de congestionamento que tentam limitar o envio excessivo de pacotes em uma determinada rede ou enlace quando o mesmo está congestionado (KUROSE; ROSS, 2010; TANENBAUM, 2011).

Como o TCP é um protocolo de transporte confiável, antes que aconteça uma comunicação TCP é necessário que as duas partes (nó origem e nó destino), se apresentem um ao outro, utilizando segmentos preliminares para estabelecerem conexão. Por conta disso, o TCP é orientado à conexão (KUROSE; ROSS, 2010; TANENBAUM, 2011; OLIFER, 2008).

O procedimento de estabelecimento de conexão é denominado apresentação de três vias (*three-way*). Como pode ser visto na Figura 1, o cliente primeiramente envia um segmento com a *flag* SYN ao servidor; o servidor responde com um segmento que tem uma *flag* SYN e uma confirmação ACK (*Acknowledge*) e tem como uma das finalidades fornecer um reconhecimento para os dados que o servidor recebeu; o terceiro segmento é um ACK que é enviado pelo cliente ao servidor e que tem o propósito de reconhecer que recebeu os dados do segmento 2, ou seja, os dados do servidor e podendo, consequentemente, estabelecer conexão com o servidor. Se houver perda de pacotes, ACKs serão retransmitidos. O protocolo TCP necessita de uma confirmação ACK para cada pacote ou conjunto de pacotes que chega ao receptor, por conta dele ser um protocolo de transporte confiável (KUROSE; ROSS, 2010; TANENBAUM, 2003).

Figura 1: Estabelecimento de conexão TCP (Handshake)



Fonte: elaborado pelo autor.

As mensagens ACK são importantes para determinar as condições atuais da rede. Essas mensagens utilizam temporizadores de resposta, quando esse temporizador se esgotar, fazer a retransmissão dos dados não enviados. Emissores e receptores podem fazer alteração na intensidade do fluxo de dados, fazendo assim a prevenção de um congestionamento na rede (KUROSE; ROSS, 2010; TANENBAUM, 2011).

O mecanismo de controle de fluxo do TCP compatibiliza as velocidades entre as taxas de envio e recepção, para evitar a saturação do *buffer* destinatário por conta do envio de mais *bytes* que o destino pode processar. A perda de pacotes pode ocorrer também por conta de gargalo na rede, quando a taxa de entrada de pacotes é maior que a taxa de saída. Portanto, era preciso adicionar ao TCP, além do controle de fluxo, mecanismos para controle de congestionamento (KUROSE; ROSS, 2010; TANENBAUM, 2011).

3.2 CONTROLE DE CONGESTIONAMENTO TCP

Quando a carga de pacotes oferecida é maior que a capacidade de processamento, acontece um congestionamento, o controle de congestionamento pode ser realizado no nível da camada de rede, mas mesmo que a camada de rede tente o controle de congestionamento, o trabalho mais importante é realizado pela camada de transporte, já que a verdadeira solução para o problema é diminuir a taxa de transmissão de dados (OLIFER, 2008; KUROSE; ROSS, 2010; TANENBAUM, 2011).

A identificação de congestionamento é feita pelo esgotamento do *timer* de envio de um pacote ou por 3 ACKs duplicados. O esgotamento do *timer* acontece quando um pacote não chega ao seu destino em um determinado tempo e os ACKs duplicados acontecem quando o emissor recebe 3 ACKs duplicados de um mesmo segmento, indicando que foi perdido o segmento que foi enviado depois do segmento que está sendo reconhecido. Sabendo disso, o receptor manda confirmações para o último pacote recebido, informando a perda (KUROSE; ROSS, 2010; TANENBAUM, 2011).

A criação de algoritmos de controle de congestionamento TCP foi uma das formas de superar os congestionamentos na rede, diminuindo a transmissão de dados quando ocorre congestionamento. (OLIFER, 2008; KUROSE; ROSS, 2010; TANENBAUM, 2011).

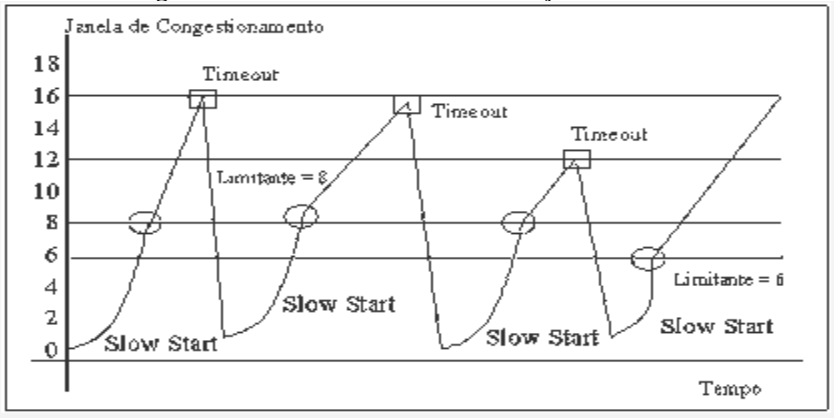
A primeira implementação de algoritmo de controle de congestionamento TCP foi o TCP Tahoe, e com o tempo foram desenvolvidos outros algoritmos com modificações e melhorias (RFC 2581, 1999; KUROSE; ROSS, 2010).

3.2.1 TCP Tahoe

Como a primeira implementação de algoritmo de controle de congestionamento TCP, o TCP Tahoe começa seu funcionamento com a utilização do *Slow Start* (Partida Lenta), como pode ser visto na Figura 2, o qual inicia o envio de dados com um segmento e de acordo com o envio e reconhecimento bem sucedido dos segmentos, a taxa de envio vai aumentando exponencialmente até ocorrer perda (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

A janela de congestionamento (*Congestion Window* – CWND) é a medida dinâmica da transmissão de dados da rede. Essa janela impõe limite ao envio de dados do transmissor, como pode ser visto no eixo vertical da Figura 2. A janela CWND começa com o valor de 1 segmento e vai crescendo a cada RTT (*Round Trip Time*) que representa o tempo decorrido a transmissão de um pacote e o recebimento do ACK. Se houver um evento de perda, ou seja, um congestionamento, o remetente TCP reestabelece o valor da janela CWND para 1, e o processo de partida lenta se inicia novamente (indicado na Figura 2 por um retângulo) (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

Figura 2: Funcionamento Slow Start e janela CWND



Fonte:< http://www.oficinadanet.com.br//imagens/conteudos/160/redes/image122.gif >

No momento em que o processo de Partida Lenta se inicia novamente, o remetente TCP também estabelece o valor de uma segunda variável chamada *ssthresh* (*slow start threshold* – Limiar de Partida Lenta) para CWND / 2 – metade do valor da janela de congestionamento, isso quando o congestionamento for detectado. Dessa forma, no segundo modo de uso, se a janela de congestionamento CWND for menor que a variável *ssthresh*, a janela continua a crescer exponencialmente, mas se a janela de congestionamento CWND se igualar ou ficar maior que o valor da variável *ssthresh*, a Partida Lenta termina e o TCP é

alterado para o modo de Prevenção de Congestionamento (*Congestion Avoidance*), no qual a CWND aumenta de forma linear (indicado na Figura 2 por uma elipse) (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

O TCP Tahoe também utiliza o *Fast Retransmit* (Retransmissão Rápida) para agilizar a recuperação rápida de dados. O seu funcionamento começa quando existem 3 ACKs duplicados para o mesmo segmento. O *Fast Retransmit* faz a retransmissão mesmo que o *timeout* não tenha se esgotado (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

Uma desvantagem do TCP Tahoe é que o algoritmo utiliza a todo momento de perda de dados ou ACKs duplicados, o *Slow Start*, fazendo com que a banda oferecida pela rede tenha uma baixa utilização (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

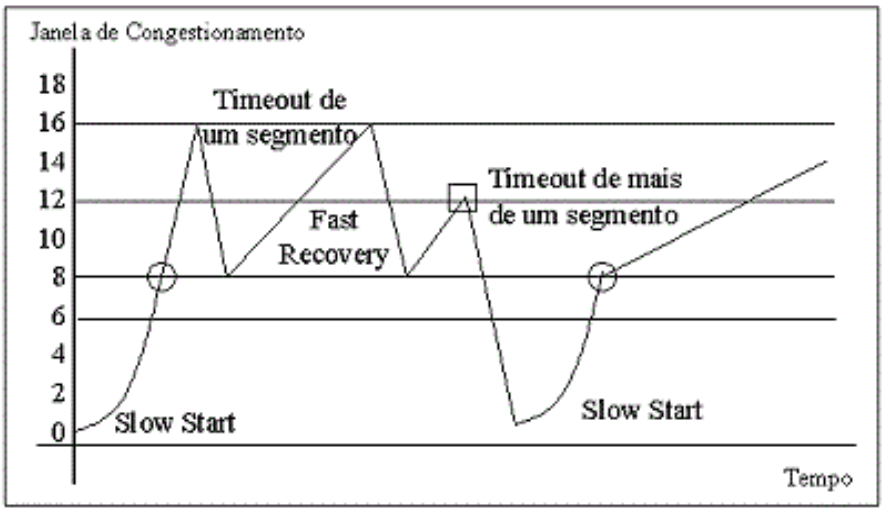
3.2.2 TCP Reno

O TCP Reno utiliza também *Slow Start, Congestion Avoidance* e *Fast Retransmit*. Ele foi criado para resolver o problema de desempenho do TCP Tahoe, o qual não trabalha bem com a perda de múltiplos dados, diminuindo a janela CWND a 1 segmento toda vez que ocorre uma perda (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

Junto com o *Fast Retransmit,* o TCP Reno utiliza o *Fast Recovery*. Como pode ser visto na Figura 3, o funcionamento do *Fast Recovery* é diferente do TCP Tahoe, o TCP Reno ele utiliza o *Congestion Avoidance*, ou seja, quando o TCP Reno observa que a rede esta congestionada, ele faz a retransmissão do pacote perdido e guarda a metade do valor da janela CWND na variável *ssthresh* (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

Sabendo que os segmentos reenviados estão sendo reconhecidos, o TCP Reno deduz que a rede não esta mais sofrendo congestionamento, e com isso, ele vai aumentando a janela CWND de forma linear utilizando o *Congestion Avoidance*, mas se houver *timeout* de mais de um segmento na fase de *Fast Recovery*, o TCP Reno utiliza o Partida Lenta (CAVALCANTI, 2005; KUROSE; ROSS, 2010).

Figura 3: Funcionamento do TCP Reno



Fonte: < http://www.oficinadanet.com.br//imagens/conteudos/160/redes/image123.gif >

O motivo da não utilização do *slow start* nesse caso é que o recebimento de ACKs duplicados indica mais do que simplesmente que um segmento foi perdido. Sabe-se que o destino só pode gerar ACKs duplicados quando outro segmento for recebido, isto é, o segmento deixou a camada física e está no *buffer* do destino. Como ainda temos dados trafegando entre os dois nós, não é aconselhável reduzir o fluxo brutalmente usando o *slow start* (CAVALCANTI, 2005).

3.2.3 TCP Vegas

O TCP Vegas utiliza os mecanismos *Slow Start* e *Congestion Avoidance*, mas não utiliza a perda de dados para detectar um congestionamento, e sim tenta evita-lo, de forma pró-ativa, detectar o congestionamento. Ele utiliza um mecanismo diferente do TCP Tahoe e TCP Reno, o qual faz o reenvio logo após o primeiro ACK duplicado recebido, e não após 3 ACKs duplicados (CAVALCANTI, 2005; BENITEZ, 2010).

O funcionamento do *Fast Retransmit* no TCP Vegas começa também com um segmento, mas sempre que o segmento é enviado, o TCP Vegas tem seu aumento exponencial após dois segmentos e não após um segmento como o TCP Tahoe e o TCP Reno. Ele utiliza dois RTTs, porque ele usa o primeiro RTT para fazer o cálculo do fluxo da rede para saber a capacidade disponível (CAVALCANTI, 2005; BENITEZ, 2010).

Para controlar o congestionamento na rede, o TCP Vegas se baseia no resultado obtido pela diferença entre a taxa de dados esperado (variável vazão esperada) e a taxa

efetivamente medida (variável vazão real) a cada RTT, comparando as duas variáveis de forma pró-ativa, para só depois alterar o valor da janela CWND. O controle da janela CWND é observado pela mudança do RTT. Se o RTT estiver pequeno, o TCP Vegas aumenta a janela CWND, por outro lado, se o RTT estiver grande, o algoritmo reconhece que a rede está congestionada e diminui a janela CWND (CAVALCANTI, 2005; BENITEZ, 2010).

Em uma conexão TCP, o algoritmo limita o valor máximo e mínimo para a vazão, mantendo a eficiência dentro desse limite, baseando-se na comparação entre o valor da vazão real e o valor da vazão esperada (CAVALCANTI, 2005), conforme a equação abaixo:

$$Diff = (VazãoReal - VazãoEsperada) \quad (3.1)$$

A vazão esperada é calculada pela divisão do valor da janela CWND e o valor do menor RTT encontrado. O calculo da vazão real é feito pela divisão do valor da janela CWND e o valor do RTT atual (CAVALCANTI, 2005).

Segundo Cavalcanti (2005) e Benítez (2010) sempre que um segmento é enviado, o TCP Vegas lê o seu relógio interno e armazena o instante de tempo. Quando houver o recebimento do ACK referente ao segmento, ele faz a leitura novamente do relógio e calcula o RTT com base no calculo das variáveis vazão real e a vazão esperada, anteriormente armazenadas. Com o cálculo obtido, o valor é registrado na variável *timestamp*. O valor estimado pelo RTT é utilizado para decidir a retransmissão de segmentos em duas situações:

(1) Quando um ACK duplicado chegar, o TCP Vegas checa se a diferença entre o tempo atual e o tempo armazenado é maior que o valor do tempo de *timeout*: se for maior, o segmento é retransmitido sem ter que esperar pelo terceiro ACK duplicado ou pelo estouro de tempo de retransmissão.

(2) Se um ACK que não é duplicado é reconhecido e se ele for o primeiro ou o segundo após uma retransmissão, o TCP Vegas checa se o tempo desde que ele foi enviado é maior que o tempo do *timeout*. Se for maior, o segmento é retransmitido.

3.2.4 TCP New Reno

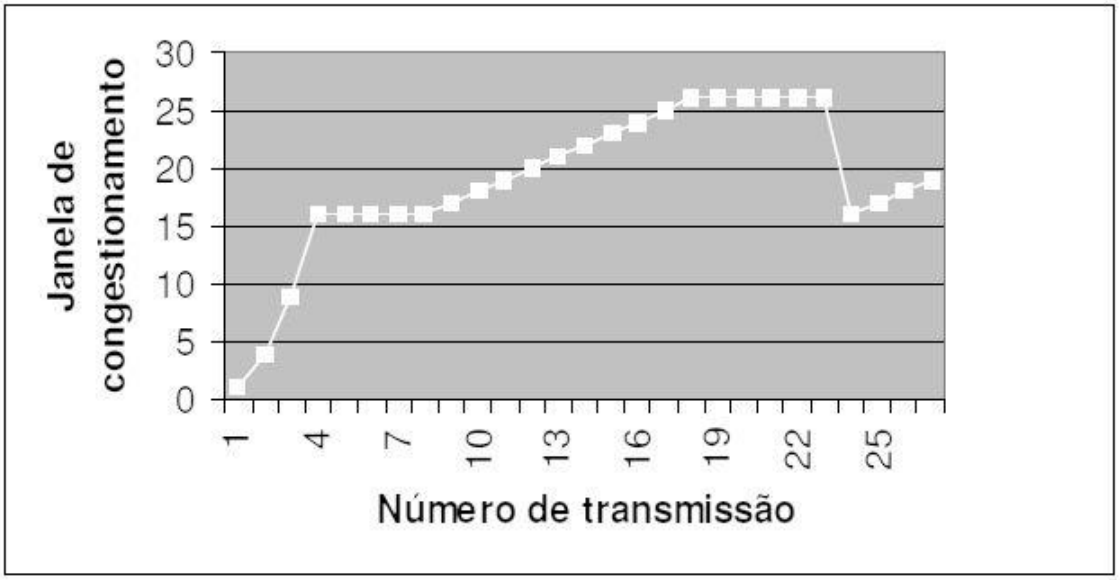
O TCP New Reno foi criado com o objetivo de otimizar o TCP Reno no caso de múltiplas perdas de pacotes em uma única janela de congestionamento, utilizando o algoritmo *Fast Recovery* de forma diferenciada, eliminando a necessidade de se ter que esperar um

estouro de *timeout* no caso de múltiplos descartes (CAVALCANTI, 2005; PRETE, SHINODA, 2009; BENITEZ, 2010).

No TCP Reno, mostrado na Figura 3, o valor da janela de transmissão é reduzido à *ssthresh* na chegada de reconhecimentos parciais. Além disso, a execução do algoritmo de *Fast Recovery* é interrompida, iniciando-se a fase de *Congestion Avoidance*. Esse processo é repetido para cada novo conjunto de reconhecimentos parciais, fazendo com que o TCP reduza a janela de transmissão pela metade seguidas vezes (CAVALCANTI, 2005; PRETE, SHINODA, 2009; BENITEZ, 2010).

Como pode ser visto na Figura 4, o TCP New Reno, ao receber reconhecimentos parciais, se mantém no algoritmo de *Fast Retransmit* evitando as múltiplas reduções no valor da janela de congestionamento. Cada reconhecimento parcial é tratado com uma indicação de que mais um pacote foi perdido e deve ser retransmitido. Desta forma, quando vários pacotes são perdidos em uma mesma janela de dados, o TCP New Reno é capaz de evitar o *timeout*. Para isso, ele retransmite um pacote perdido por RTT até que todos os pacotes perdidos desta janela tenham sido retransmitidos. Para sair do algoritmo de *Fast Recovery*, o TCP New Reno espera pelo recebimento de um reconhecimento que confirme todos os pacotes pendentes quando este algoritmo foi iniciado (CAVALCANTI, 2005; PRETE; SHINODA, 2009; BENITEZ, 2010).

Figura 4: Funcionamento TCP New Reno



Fonte: Cavalcanti, 2005, p.25

3.2.5 TCP Sack (*Selective Acknowledgment*)

Como o algoritmo TCP New Reno, O TCP Sack tem como objetivo evitar os problemas de desempenho do Reno quando múltiplos pacotes são descartados. Quando existem perdas de mais de um segmento, a retransmissão fica lenta e ocorrem muitos reenvios, o TCP Sack foi criado com o objetivo de recuperar múltiplos segmentos perdidos no intervalo de um RTT. A proposta foi que o receptor enviasse no ACK a informação dos segmentos já recebidos, dando uma maior chance ao TCP origem de inferir nos envios dos segmentos perdidos. Com isso o transmissor sabe exatamente que segmentos foram perdidos, podendo retransmiti-los, melhorando dessa forma largura de banda disponível (RFC 2018, 1996; ABED; ISMAIL; JUMARI, 2011).

Enquanto o receptor estiver recebendo segmentos na ordem certa, ele não utiliza o TCP Sack. O TCP Sack utiliza o campo *Options* do cabeçalho TCP para transportar as informações sobre os segmentos recebidos. Caso receba um segmento fora de ordem, armazenará os segmentos em um bloco contíguo e informará ao emissor o sequencial desses dados pelo campo *Options* do TCP. No campo *Acknowledgement Number* envia o número de sequência recebido + 1, ou seja, o TCP Sack não altera o valor desse campo no TCP. Complementando o SACK estará em funcionamento apenas em ACKs Duplicados ou reconhecimentos parciais (RFC 2018, 1996; ABED; ISMAIL; JUMARI, 2011).

3.2.6 TCP Fack (*Forward Acknowledgement*)

O TCP Fack usa as informações adicionais fornecidas pelo TCP Sack para manter uma medida explícita do número total de dados em circulação na rede. O objetivo do algoritmo TCP Fack é utilizar essas informações para adicionar um controle mais preciso à injeção de dados na rede (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

Os algoritmos TCP Reno e TCP Sack estimam que cada ACK duplicado representa um segmento perdido na rede, já o TCP FACK é capaz de estimar com a utilização de duas variáveis: *snd.fack* e *retran_data*. Além disso, o remetente deve reter informações em blocos de dados mantidos pelo receptor, a fim de utilizar as informações fornecidas pelo TCP Sack para retransmitir corretamente os dados. Para além do que é necessário para controlar a retransmissão de dados, informações sobre os segmentos retransmitidos devem ser mantidas, a fim de determinar de forma precisa quando estes dados tiverem deixado à rede (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

A variável *snd.fack* é atualizada para refletir os dados mantidos pelo receptor, sua atualização é feita pelo número de confirmações no cabeçalho TCP e é o mesmo que a variável *snd.una*. Durante a recuperação (receptor armazenando dados não contínuos) o remetente continua atualizando a variável *snd.una* do número de confirmação do cabeçalho TCP, no mesmo tempo, utiliza-se as informações contidas no TCP Sack para atualizar a variável *snd.fack*. Quando um bloco do TCP Sack é recebido, e os reconhecimentos de dados com um número de sequência mais alto do que o valor atual de *snd.fack*, *snd.fack* é atualizado para refletir o número de sequência mais alto conhecido (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

Remetentes que tratam de transporte confiável, continuam a usar a variável *snd.una* de estado existente. Remetentes que abordam a gestão dos congestionamentos são alteradas para usar *snd.fack*, que fornece uma visão mais precisa para o estado da rede (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

A variável *awnd* é definida para ser a estimativa do remetente da quantidade real de dados em circulação na rede. Supondo-se que todos os segmentos não confirmados não deixaram a rede:

$$awnd = snd.nxt - snd.fack \tag{3.2}$$

Durante a recuperação, dados que são retransmitidos também devem ser incluídos no cálculo de *awnd*. O remetente calcula uma nova variável, *retran_data*, que reflete a quantidade de circulação de dados retransmitidos na rede. Cada vez que um segmento é retransmitido, *retran_data* é aumentado pelo tamanho do segmento, quando um segmento retransmitido está determinado a ter deixado a rede, *retran_data* é diminuída pelo tamanho do segmento. Portanto, a estimativa do TCP da quantidade de dados pendentes na rede durante a recuperação é dada por:

$$awnd = snd.nxt - snd.fack + retran\_data \tag{3.3}$$

O TCP Reno invoca o *Fast Recovery* quando existem três reconhecimentos duplicados, fazendo com que ocorra um atraso desnecessário quando diversos segmentos são perdidos antes de receber três reconhecimentos duplicados. No TCP Fack, o ajuste *cwnd* e retransmissão também são acionados quando o receptor informa que a fila de remontagem é maior do que 3 segmentos:

```
if ((snd.fack - snd.una) > (3 * MSS) ||(dupacks == 3)) {
    ...
    }
```

Se exatamente um segmento é perdido, os dois algoritmos desencadeiam recuperação exatamente no mesmo reconhecimento duplicado (MATHIS; MAHDAVI, 1996; ABED; ISMAIL; JUMARI, 2011).

3.3 SIMULAÇÃO

A simulação é um importante elemento para experimentos de rede, sendo um método de implementação barata, que tenta imitar características de sistemas reais, prever algum desempenho e comparar arquiteturas e algoritmos de forma a obter resultados que são mais próximos da realidade, controlando e podendo repetir seus experimentos em tempo menor que o tempo real. A utilização da simulação pode trazer outras vantagens como a identificação de acontecimentos de uma determinada rede, mesmo quando não tenha sido implementada na vida real, ou obter os conhecimentos para ter a clareza de qual equipamento ou algoritmo sua rede precisa para manter um bom desempenho (HASSAN; JAIN, 2004). Existem alguns simuladores para redes TCP/IP como o OPNET, Comnet III, Network Simulator 2 (NS-2), e Network Simulator 3 (NS-3) (HASSAN; JAIN, 2004).

O NS-2 vai ser utilizado no trabalho por ser de código aberto e os algoritmos e protocolos utilizados no trabalho já estarem implementados no simulador. A versão atual é o NS-3, mas ela não implementa a maioria dos algoritmos de congestionamento que estão sendo analisados no trabalho. A seguir será explicado o simulador utilizado.

3.3.1 Simulador NS-2 (*Network Simulator*)

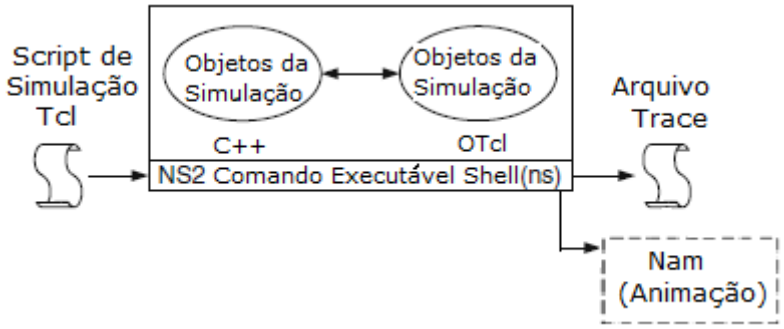
O NS-2 é um simulador de rede de código aberto e de distribuição gratuita, podendo ser modificado ou personalizado. *Software* ideal para simulação de protocolos e algoritmos que constituem a *Internet* para topologias com ou sem fio, como TCP, UDP, protocolos de roteamento, algoritmos de controle de congestionamento TCP, entre outros (POTNOI; ARAÚJO, 2003; HASSAN; JAIN, 2004; FALL; VARADHAN, 2013).

Como pode ser visto na Figura 5, a programação do NS é feita em duas linguagens: C++ para a estrutura básica (protocolos, agentes, etc) e OTcl (Object-oriented

Tool Command Language) para uso como *fronten*d. A linguagem é OTcl porque utiliza a linguagem de programação tcl estendido com a Orientação a Objetos. A utilização de duas linguagens atende à necessidade de uma linguagem eficiente e precisa para implementação dos protocolos, requisitos suportados pela linguagem C++, e outra que suporta alterações constantes, pois, como as simulações mudam frequentemente, ficaria muito dispendioso só utilizar a linguagem C++, em que cada alteração na simulação provocaria a compilação de todo o código fonte. Esse problema foi resolvido utilizando a linguagem OTcl, interpretada, que permite uma rápida alteração do cenário de simulação sem ter que compilar todo o código fonte (POTNOI; ARAÚJO, 2003; HASSAN; JAIN, 2004; FALL; VARADHAN, 2013).

O NS-2 pode ser utilizado em diferentes sistemas operacionais, como plataformas Windows e Unix (Linux, Solaris, FreeBSD, entre outras). A interação entre usuário e NS-2 fica por conta da linguagem OTcl, na qual o usuário criará seu *script* especificando os parâmetros da rede que quer simular. O *script* será lido por um interpretador podendo ser gerados dois outros arquivos: um arquivo com a extensão .nam e outro arquivo com a extensão .tr (PORTNOI; ARAUJO, 2003; CAVALCANTI, 2005; FALL; VARADHAN, 2013).

Figura 5: Esquema de utilização do NS-2.



Fonte: Elaborado pelo autor

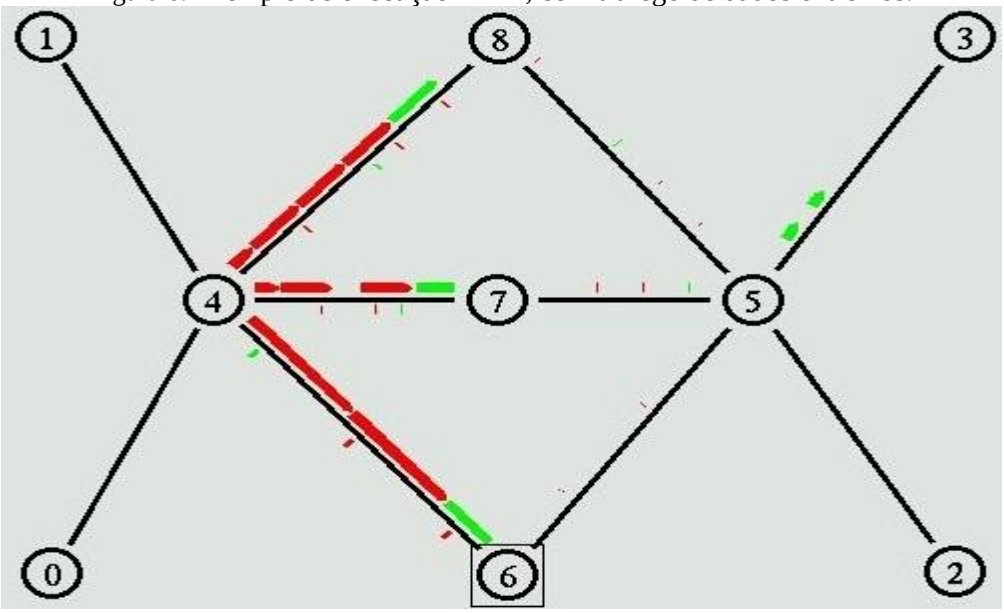
Para facilitar a criação desse *script*, Hassan e Jain (2004) e Fall e Varadhan (2013) recomendam seguir o roteiro abaixo:

1. Criação do objeto simulador;
2. Abertura de arquivos para posteriormente ser feita a análise (*trace*);
3. Criação da topologia da rede (nós e enlaces);
4. Criação dos agentes da camada de transporte e conexão com os nós;
5. Criação dos geradores de tráfego e conexão com os agentes da camada de transporte;

6. Programação dos eventos da simulação;
7. Fechamento da simulação, animação (NAM) e geração de estatísticas.

O arquivo com a extensão .nam é utilizado pelo NAM (*Network Animator*) para visualização gráfica do *script* criado pelo usuário, dessa forma, ele pode interagir com a simulação, podendo parar, voltar ou avançar o tempo da simulação. Pode também visualizar o tráfego dos pacotes, quando eles estão no *buffer* de espera, pacotes sendo descartados, enlaces ficando indisponíveis, dentre outros tipos de eventos. Um exemplo de cenário de rede no NAM pode ser visto na Figura 6, com tráfego de dados com cores diferentes e as mensagens ACK trafegando na rede. (CAVALCANTI, 2005, FALL, VARADHAN, 2013).

Figura 6: Exemplo de execução NAM, com tráfego de dados entre nós.



Fonte: Elaborado pelo autor.

O arquivo *trace* com a extensão .tr é utilizado para análise do cenário executado, no qual estão armazenados todos os eventos que foram registrados na simulação. Como pode ser visto na Figura 7, existem alguns tipos de operações que podem ser vistas no arquivo *trace*, como: pacotes descartados (*drop* - *d*), pacotes entrando na fila de espera (+), pacotes saindo da fila de espera (-), pacotes que vão para o nó seguinte (*receive* - *r*). Também pode ser visto o tipo de pacote, tamanho e o número de sequência dos pacotes de dados, o tempo do evento, endereços de origem e destino e identificador de pacotes (HASSAN, JAIN, 2004; CAVALCANTI, 2005; FALL, VARADHAN, 2013).

Figura 7: Exemplo de arquivo trace (extensão .tr) com todos os seus campos.

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Fonte: Elaborado pelo autor

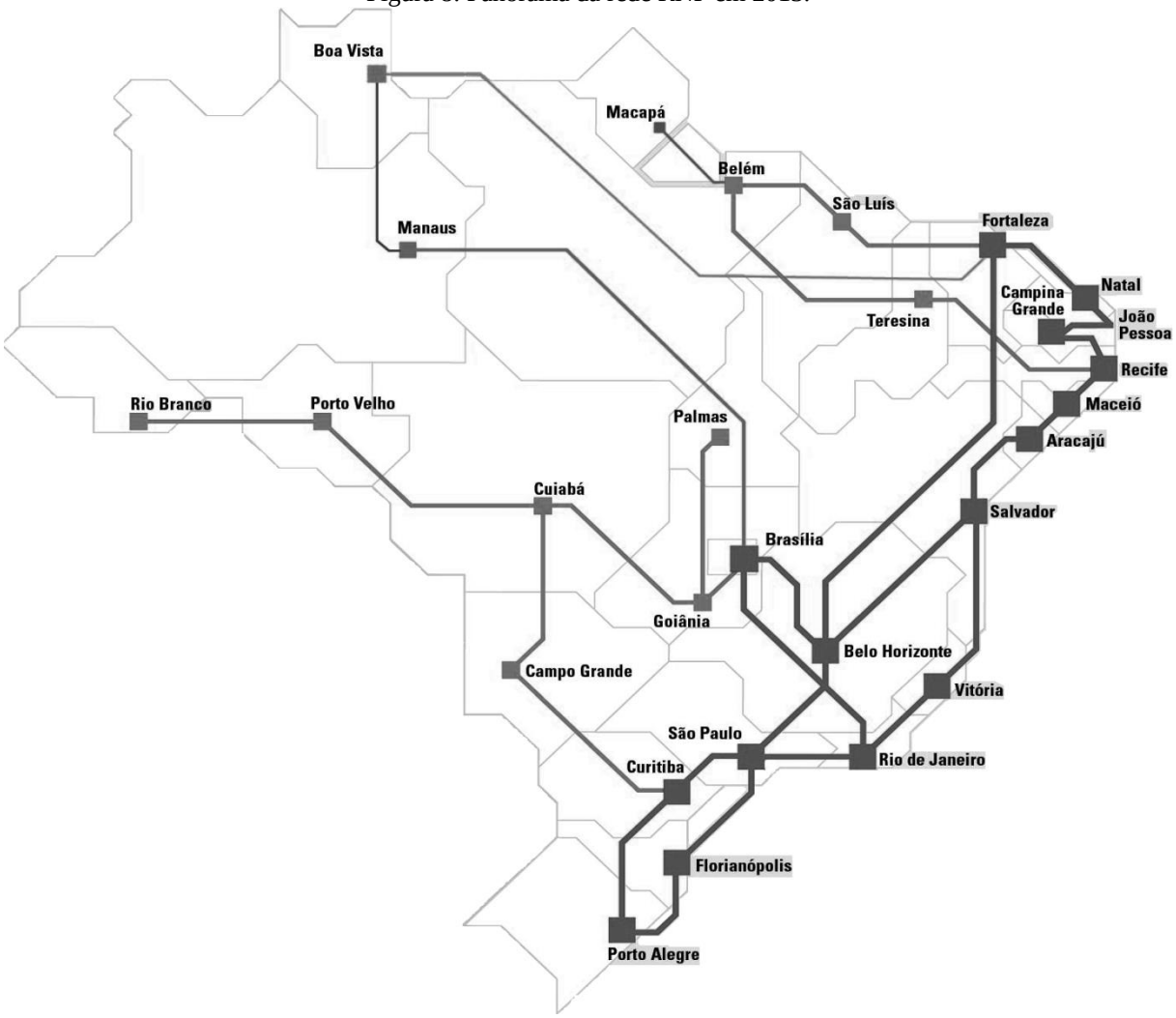
4 PROCEDIMENTOS METODOLÓGICOS

Os procedimentos metodológicos desse trabalho seguiram os 8 passos que foram sugeridos por Hassan e Jain (2004) para estruturar e executar uma análise de desempenho:

1. Definição do objetivo de estudo: a simulação e análise de desempenho dos algoritmos de roteamento TCP.

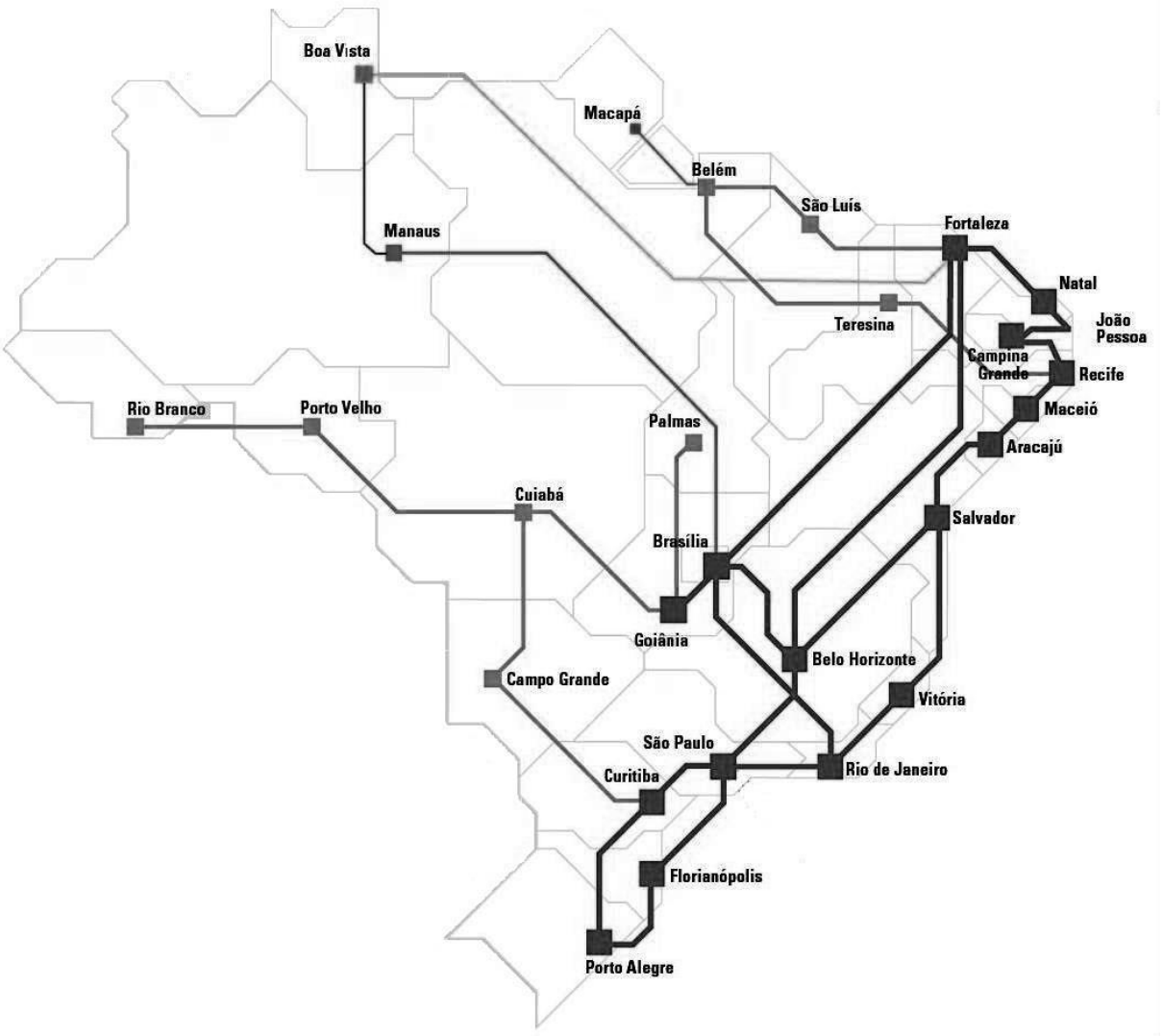
2. Modelo de rede e seleção de parâmetros fixos: serão utilizadas duas topologias para execução dos diferentes cenários de rede. Os cenários têm o formato da topologia da RNP (Rede Nacional de Pesquisa) que interliga todos os estados brasileiros. Essa topologia teve aumento de duas ligações de enlaces entre os anos de 2013 e 2014 (Ceará com Rio de Janeiro e Ceará com Brasília), podendo ser vistas na Figura 8 e na Figura 9. Os parâmetros fixos são as capacidades dos enlaces e seus retardos.

Figura 8: Panorama da rede RNP em 2013.



Fonte: < www.rnp.br/backbone/index.html >

Figura 9: Panorama da rede RNP em maio de 2014



Fonte: < www.rnp.br/backbone/index.html >

3. Seleção das métricas de desempenho: As métricas são: taxa de descartes, taxa de pacotes recebidos e utilização dos *links*.

A taxa de descarte é um fator muito importante para estudo sobre controle de congestionamento, sabendo que a quantidade de pacotes descartados aumenta com o crescimento da transmissão de dados, significando o primeiro indício de congestionamento na rede. A taxa de pacotes recebidos também é relevante para mensurar a quantidade de dados que o algoritmo de congestionamento consegue entregar aos destinos, mesmo acontecendo congestionamento e consequentemente a utilização de seus mecanismos que reduzem o envio de dados. A utilização do *link* também foi escolhida como métrica para analisar se os algoritmos de congestionamento transmitem dados sempre próximos à capacidade do enlace.

4. O parâmetro variável é o percentual de geradores de tráfego. Nesta etapa também serão criados diferentes valores para cada parâmetro variável, chamado de níveis. Por exemplo, sendo um fator o número de geradores de tráfego, os níveis que podem ser utilizados são: 10 aplicações e 20 aplicações. Em cada cenário, serão utilizados níveis diferentes, com um algoritmo de controle de congestionamento TCP.

Os geradores de tráfego serão tanto de aplicações TCP (transferência de arquivo – FTP – *File Transfer Protocol*) quanto de aplicações UDP (tráfego multimídia constante - CBR - *Constant Bit Rate*), para tentar imitar o comportamento de uma rede real onde existe a concorrência desses tráfegos.

5. Escolha da técnica de avaliação: Os cenários de rede serão implementados no NS-2 em linguagem OTcl. O NS-2 foi instalado e configurado em uma máquina com a distribuição Linux (Ubuntu).

6. Configurar o *software* de simulação: Implementação dos cenários no ns-2.

7. Executar o programa e coletar os dados de desempenho: A execução de cada simulação gera dois arquivos: um arquivo *.nam* e um arquivo *.tr* . Como o arquivo *.nam* só apresenta os passos das simulações com visualizações gráficas, não seria viável fazer análise com esse arquivo. Dessa forma foram criados *scripts* para executar a análise nos arquivos *.tr* onde fica armazenado todos os eventos da execução da simulação. Os *scripts* capturam os dados necessários do arquivo *.tr* para análise e os armazena em outro arquivo que agrega os resultados para serem plotados em gráficos. Dessa forma, é possível fazer uma comparação dos diversos resultados.

8. Apresentar e interpretar os resultados: Como término, será interpretado e apresentado o melhor algoritmo de controle de congestionamento para cada métrica utilizada.

5 CONFIGURAÇÃO DOS CENÁRIOS

Esta seção apresenta os modelos de rede, os parâmetros fixos e as variáveis utilizadas para a elaboração dos cenários de rede.

5.1 MODELOS

Para comparar os algoritmos de controle de congestionamento TCP, foram utilizados cenários de rede que simulam as conexões das topologias da RNP citadas acima. Todos os enlaces das topologias possuem um *buffer* finito, com pequena capacidade. Os enlaces são *Full-Duplex* e utilizam a disciplina de fila FIFO (*First-In First-Out*). Todos os cenários foram executados em simulações de 10 segundos cada, tempo que foi suficiente para gerar uma quantidade de dados adequada à fase de análise dos resultados. Os cenários implementados estão apresentados no Apêndice A (*cenario01.tcl* e *cenario04.tcl*).

5.2 PARÂMETROS DE REDE

Os parâmetros fixos das topologias são: as capacidades dos enlaces e os seus retardos, enquanto os parâmetros variáveis são: percentual de geradores de tráfego.

As capacidades dos enlaces são de 5Mb e 2Mb com exceção das ligações entre (1) Ceará e Roraima e (2) Pará e Amapá, que tem capacidade inferior. A capacidade dos enlaces não está igual ao da topologia real pelo fato que para gerar situações de congestionamento em uma rede com a capacidade de enlaces iguais ao da RNP, seria necessário criar centenas ou milhares de geradores de tráfego. Nesse caso o simulador não suportaria essa quantidade imensa de tráfego de dados além das simulações se tornarem excessivamente lentas.

Os retardos foram implementados de acordo com as distâncias físicas das ligações dos enlaces, por exemplo, à ligação entre Ceará e Minas Gerais foi atribuído o retardo de 20ms por ser distante, já a ligação entre Ceará e Rio Grande do Norte, foi atribuído o retardo de 10ms, por serem estados mais próximos.

Os geradores de tráfego foram atribuídos pelo número de enlaces das topologias, foram escolhidos 50%, 70% e 80% dos enlaces gerando trafego pela rede.

Nos cenários que imitam características da rede antiga da RNP, existem 34 enlaces, nos quais foram criados 17 geradores de tráfego (nove geradores FTP e oito geradores CBR) para cenários com 50% de quantidade de geradores de tráfego, 25 geradores

(13 geradores FTP e 12 geradores CBR) para cenários com 70% e 27 geradores (14 geradores FTP e 13 geradores CBR) para cenários com 80% de geradores de tráfego.

Nos cenários que imitam características da rede utilizada atualmente pela RNP, existem 36 enlaces, pois foram adicionadas mais duas ligações (Ceará com Rio de Janeiro e Ceará com Brasília). Foram criados 18 geradores de tráfego (nove geradores FTP e nove geradores CBR) para cenários com 50% de quantidade de geradores de tráfego, 25 geradores (13 geradores FTP e 12 geradores CBR) para cenários com 70% e 29 geradores (15 geradores FTP e 14 geradores CBR) para os cenários com 80% de geradores de tráfego.

Os geradores de tráfego dos cenários foram configurados para inicializarem com tempos diferentes. Os tráfegos FTP começam a gerar tráfego com 0,1 segundo de simulação, enquanto os geradores de tráfegos CBR começam com 0,5 segundo de simulação e terminam com 9,5 segundos.

De acordo com o número de algoritmos de congestionamento utilizados e o número de parâmetros variáveis, foram criados 36 cenários de rede (18 cenários na topologia antiga da RNP e 18 cenários na topologia atual). Os parâmetros fixos dos cenários podem ser vistos na tabela 1:

Tabela 1: Parâmetros fixos das simulações

PARÂMETRO	**VALOR**
Capacidade dos enlaces (Mb)	1, 2, 5
Retardos (ms)	5, 10, 15, 20, 25
Aplicações	FTP e CBR
Transporte	TCP e UDP
Tempo de Simulação	10 s

6 ANÁLISE DOS CENÁRIOS

Para analisar os descartes nos cenários, foi criado um *script* em *shell* que pode ser encontrado no Apêndice B (*perda_pacotes.sh*). Esse *script* verifica a quantidade de pacotes que foram perdidos durante os 10 segundos de simulação, em períodos de 0,5 segundos, verificando os pacotes FTP perdidos. A análise foi feita somente nos pacotes que continham dados (FTP), desconsiderando as perdas de pacotes ACK, estabelecimento de conexão, sincronização e trafego de dados CBR, esses pacotes foram desconsiderados por serem enviáveis para utilizar na análise.

Para analisar os pacotes recebidos, foi criado um *script* em *shell* com características parecidas com o *script perda_pacotes.sh*. O *script* verifica a quantidade de pacotes FTP recebidos pelos destinos, durante os 10 segundos de simulação, em períodos de 0,5 segundo.

Para a utilização do canal, foi criado um *script* em *shell* para analisar o *link* entre os nós Goiás e Distrito Federal, esse link foi escolhido por ter um enlace com 5Mb de capacidade, e que trafegam dados FTP e CBR simultaneamente ocorrendo congestionamento, mas com poucos descartes.

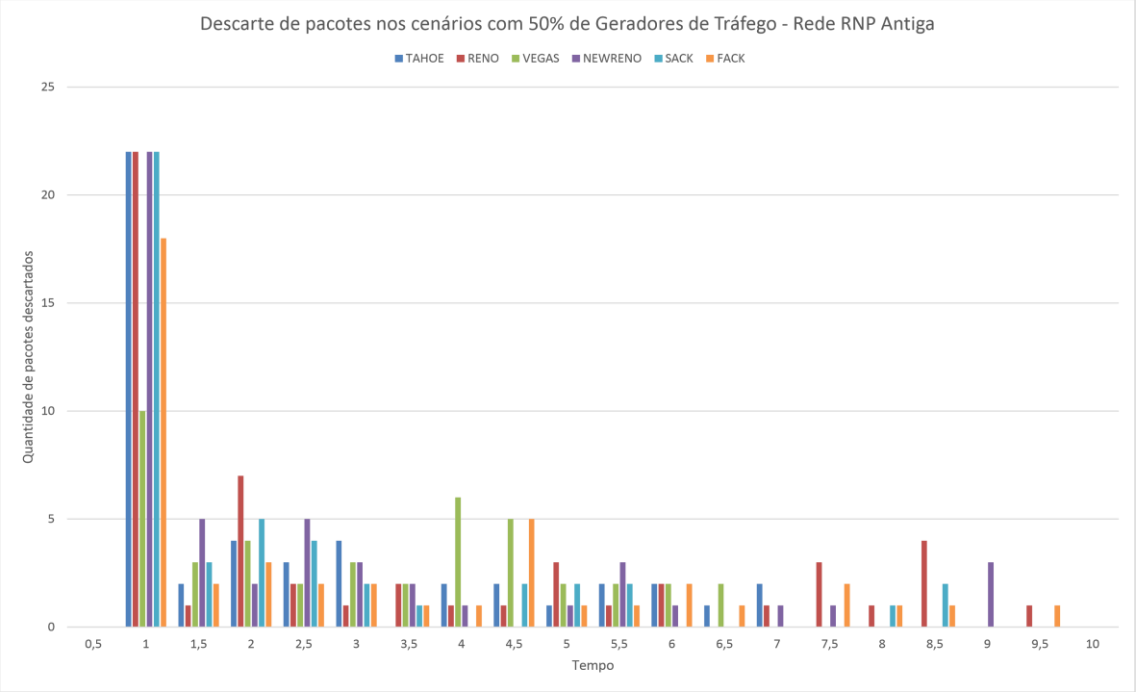
A seguir serão mostrados os resultados das análises feitas, de acordo com as métricas estabelecidas neste trabalho. A análise foi dividida em duas partes, a primeira parte com as análises feitas nos cenários que imitam as características da Rede RNP Antiga e a segunda sobre a topologia atual da Rede RNP.

6.1 REDE RNP ANTIGA

6.1.1 Descarte

As Figuras 10, 11 e 12 mostram como os algoritmos de controle de congestionamento se comportaram no decorrer da execução dos cenários da rede Antiga da RNP.

Figura 10: Descarte de pacotes: cenários com 50% de Geradores de Tráfego - Rede RNP Antiga

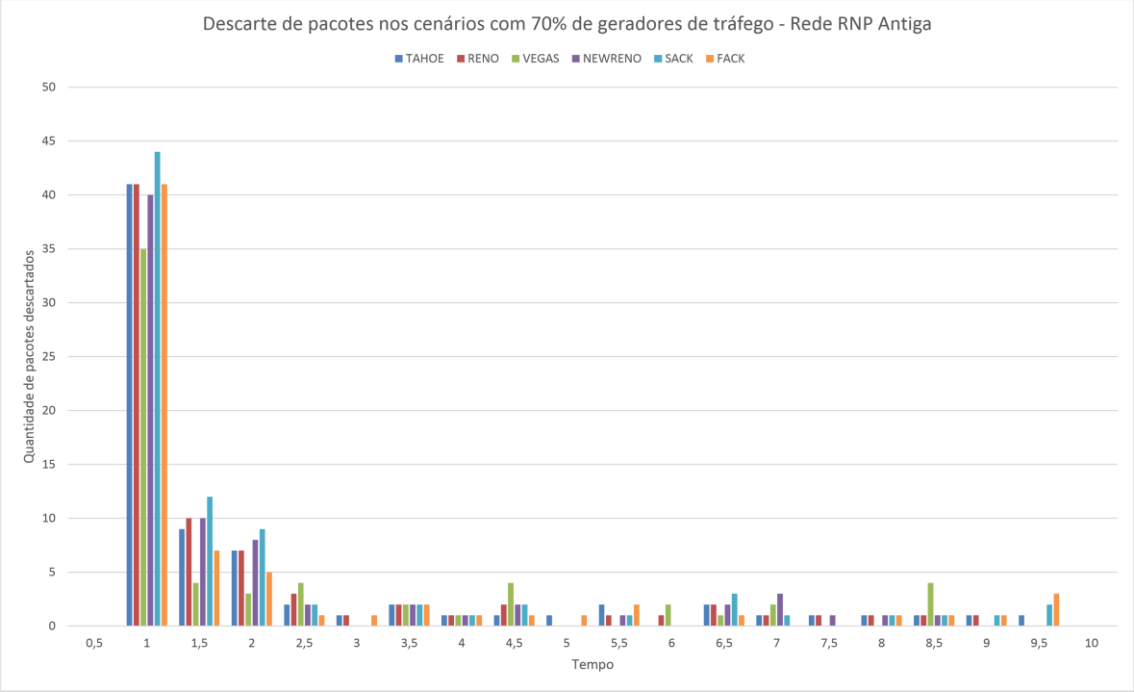


Fonte: Elaborado pelo autor

Como pode ser verificado na Figura 10, todos os algoritmos com meio segundo de execução, não tiveram nenhum pacote de dados descartado. No entanto, quando os geradores de tráfegos CBR começam a transmitir simultaneamente com os tráfegos FTP, há um descarte significativo na rede. O algoritmo de controle de congestionamento TCP Vegas teve melhor desempenho, pois ele identifica o congestionamento na rede de forma pró-ativa, ocasionando menos perdas nesse cenário.

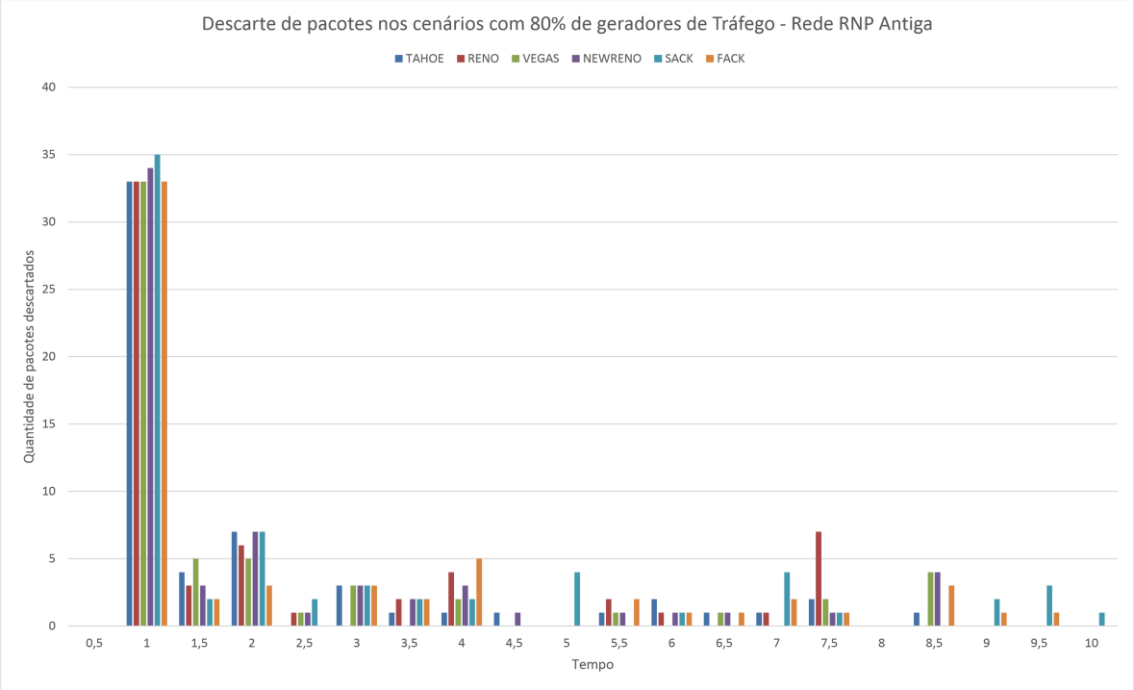
Esse melhor desempenho do algoritmo TCP Vegas pode ser visto também nas Figuras 11 e 12 a seguir:

Figura 11: Descarte de pacotes: cenários com 70% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Figura 12: Descarte de pacotes: cenários com 80% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Mesmo com o aumento de geradores de trafego na rede, o TCP Vegas obteve o melhor desempenho com relação às outras implementações. A Tabela 2 mostra o total de pacotes que foram perdidos nos cenários:

Tabela 2: Quantidade de pacotes descartados – Rede RNP Antiga

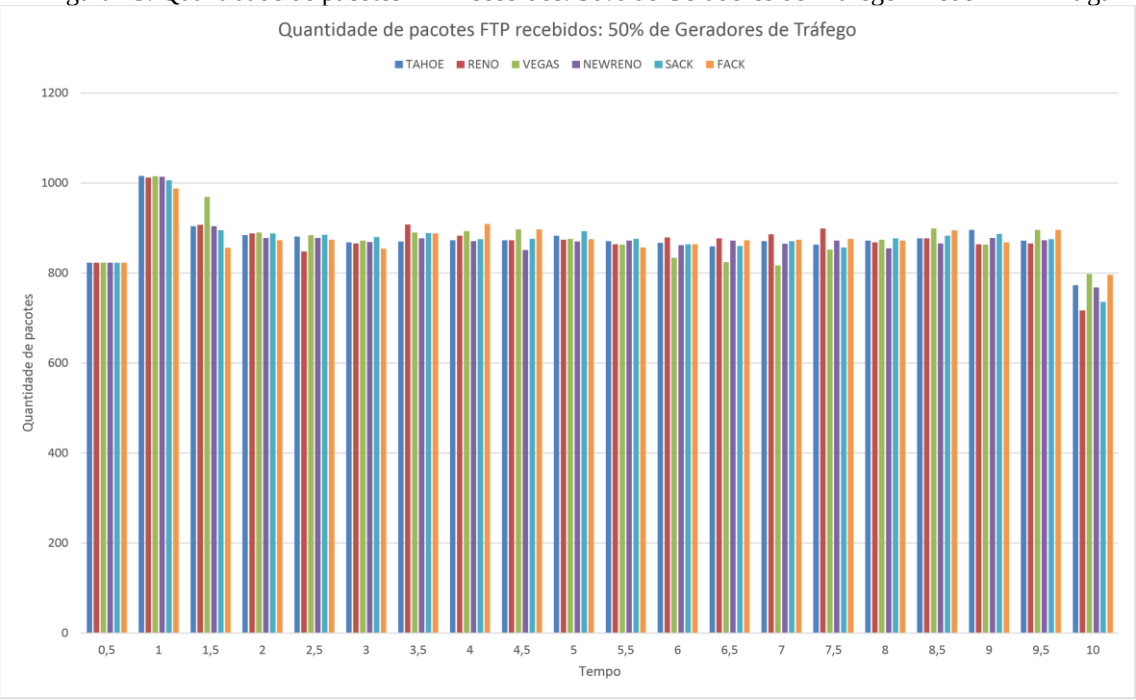
Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	47	53	43	50	46	44
70%	72	76	62	74	82	69
80%	58	60	57	62	69	60

Como pode ser visto na Tabela 2, o TCP Vegas obteve as menores taxas de descarte por conta de seus mecanismos de previsão de congestionamento antes mesmo de seu acontecimento. Outro fator importante para se verificar, é a quantidade baixa de descartes no TCP Tahoe com relação às implementações TCP Reno e TCP NewReno que foram criadas a partir dela. A justificativa deste resultado é o mecanismo de controle de congestionamento em que o Tahoe utiliza, o *Slow Start*, inicia sempre o envio de dados com a janela de congestionamento igual 1, ocasionando poucas perdas.

6.1.2 Pacotes Recebidos

As Figuras 13, 14 e 15 mostrarão a quantidade de pacotes FTP recebidos no decorrer da execução dos cenários da rede Antiga da RNP.

Figura 13: Quantidade de pacotes FTP recebidos: 50% de Geradores de Tráfego - Rede RNP Antiga

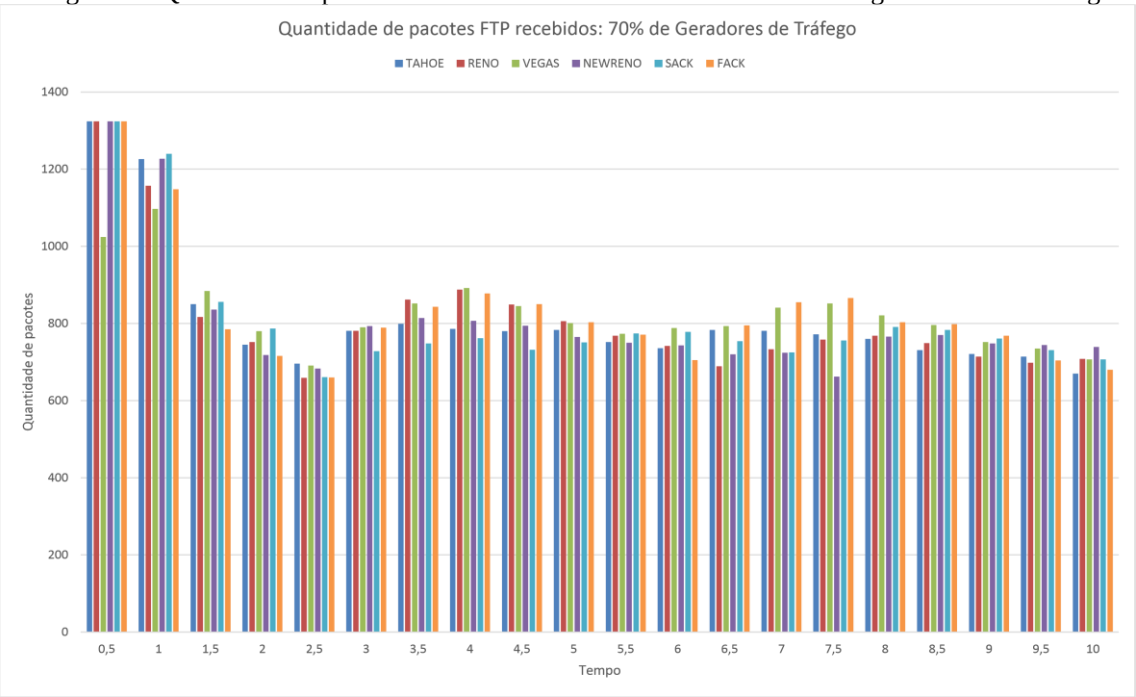


Fonte: Elaborado pelo autor

Pode ser visto na Figura 13 que os algoritmos recebem quantidades parecidas no primeiro segundo de simulação, após esse intervalo, há uma pequena redução na quantidade pacotes recebidos. Neste caso, a justificativa são os tráfegos CBR enviados na rede. Pode ser visto que o TCP Vegas tem intervalos de recebimento de pacotes maior que as outras implementações, por conta do seu controle de congestionamento monitorar as variáveis vazão real e vazão esperada para só depois ajustar a janela CWND, ao invés de diminuir o envio de dados a cada descarte como as outras implementações.

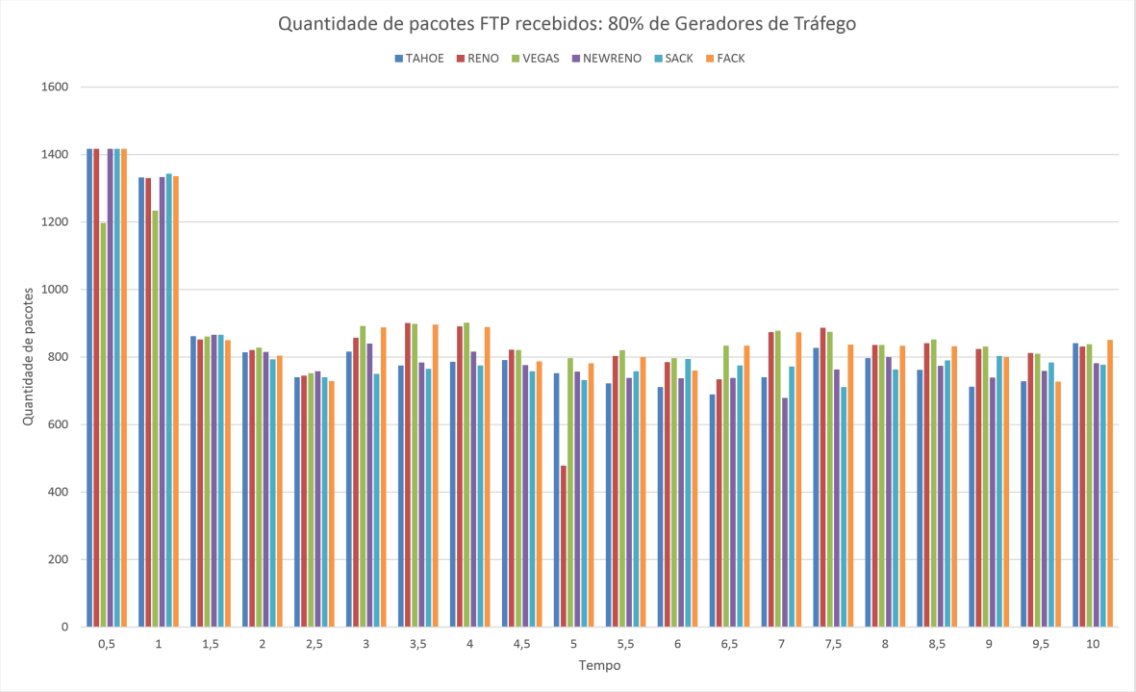
Existem mudanças na quantidade de dados recebidos, que podem ser vistas nas Figuras 14 e 15:

Figura 14: Quantidade de pacotes FTP recebidos: 70% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Figura 15: Quantidade de pacotes FTP recebidos: 80% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Pode ser visto nas Figuras 14 e 15 que a quantidade de pacotes recebidos foi superior com relação aos cenários com 50% de geradores de trafego. Pode ser verificado também que essa quantidade cai abruptamente depois que os geradores CBR estão sendo transmitidos na rede. Nota-se que o algoritmo TCP Vegas envia menos pacotes enquanto a rede não estava congestionada, mas quando ocorre o congestionamento, o algoritmo apresenta melhores resultados, a justificativa deste acontecimento é que os outros algoritmos só reduzem a transmissão quando acontecem descartes ou estouro de *timeouts*, ocasionando o envio maior de dados no primeiro segundo. A seguir, na Tabela 3, é mostrado o total de pacotes recebidos no decorrer das simulações.

Tabela 3: Quantidade de pacotes FTP Recebidos – Rede RNP Antiga

Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	17496	17479	17529	17498	17496	17508
70%	16190	16222	16514	16197	16149	16541
80%	16614	17341	17553	16671	16666	17524

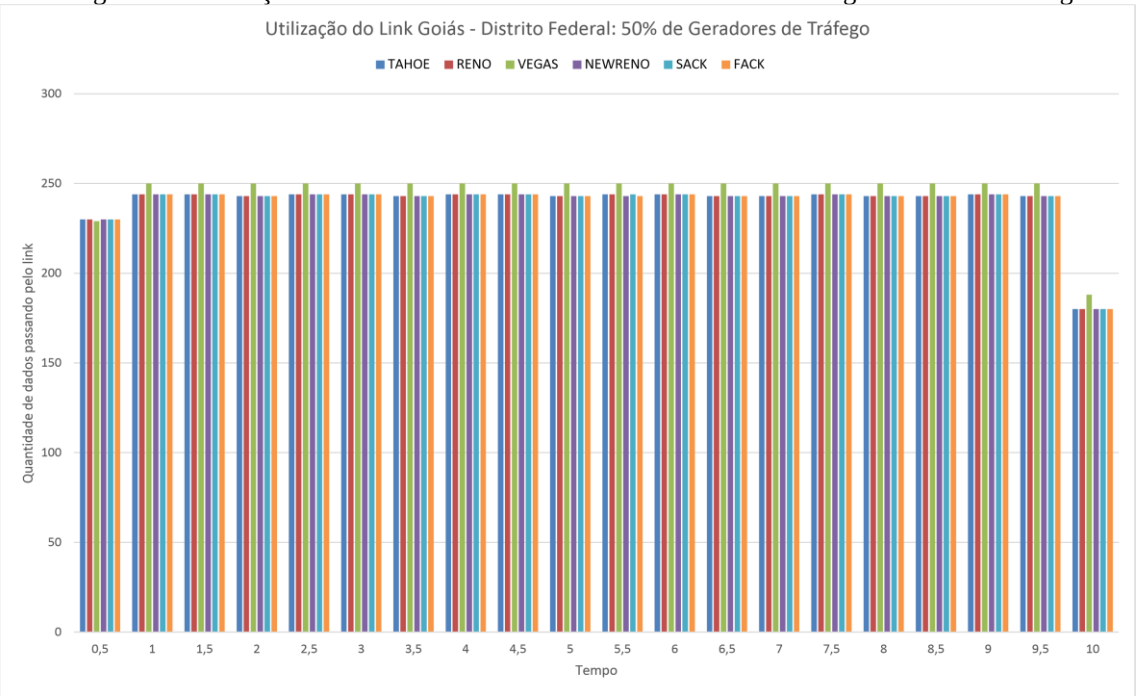
Mesmo com a quantidade inferior no primeiro segundo das simulações com geradores de tráfego 70% e 80%, o TCP Vegas obteve uma maior quantidade de pacotes recebidos. Com o aumento dos geradores de trafego, pode-se verificar que a diferença da

quantidade de pacotes recebidos aumentou nas outras implementações com relação ao TCP Tahoe por conta de seu mecanismo diminuir a janela de congestionamento quando ocorre perdas.

6.1.3 Utilização dos *links*

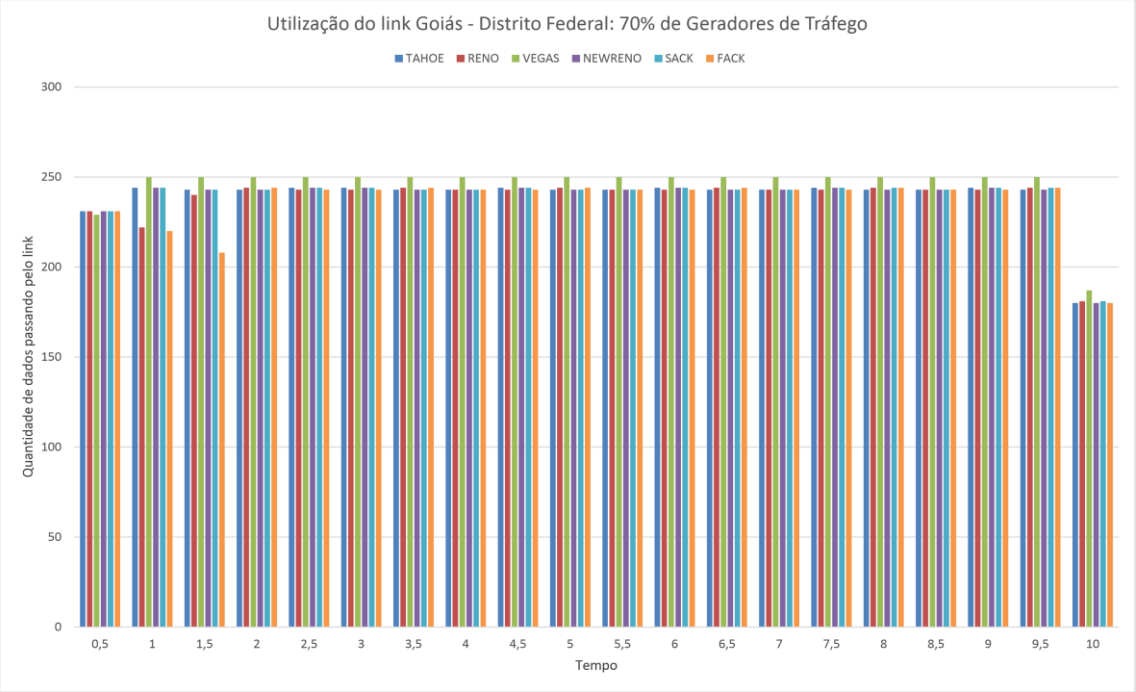
Será mostrado nas Figuras 16, 17 e 18 a quantidade de dados que utilizaram a ligação entre o nó Goiás e o nó Distrito Federal na topologia Antiga da RNP. Vale ressaltar que foram capturados todos os dados que trafegaram neste link.

Figura 16: Utilização do Link: cenários com 50% de Geradores de Tráfego - Rede RNP Antiga



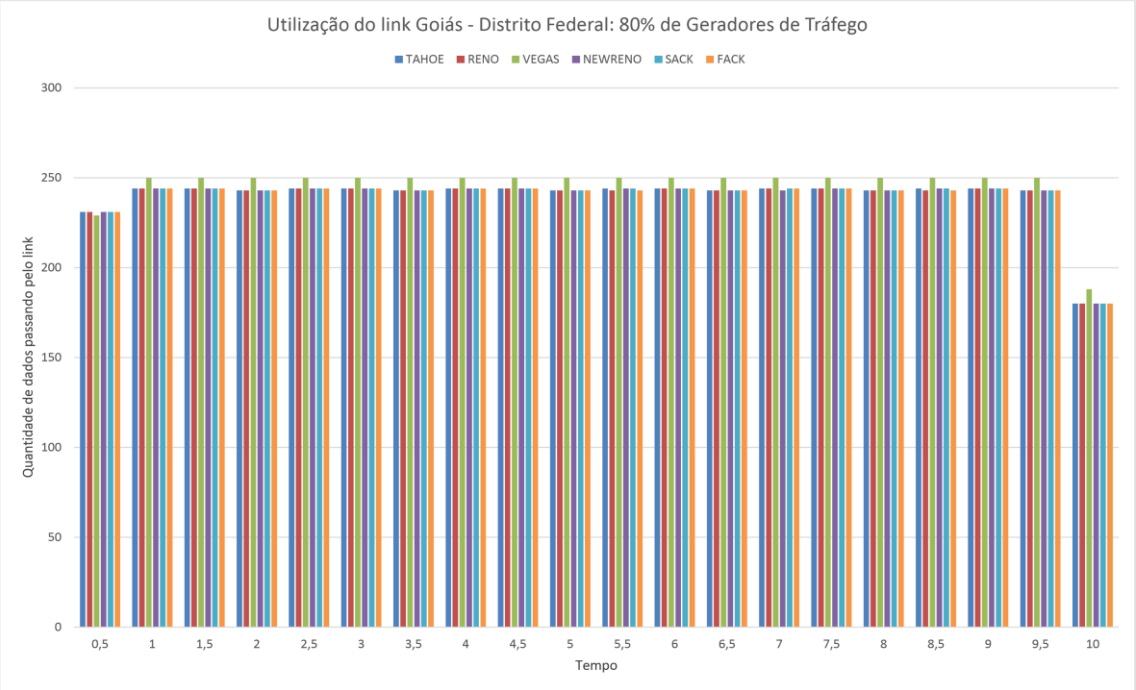
Fonte: Elaborado pelo autor

Figura 17: Utilização do Link: cenários com 70% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Figura 18: Utilização do Link: cenários com 80% de Geradores de Tráfego - Rede RNP Antiga



Fonte: Elaborado pelo autor

Pode ser visto nas Figuras 16, 17 e 18 que nos três tipos de cenários com geradores de tráfego diferentes, as implementações tiveram resultados parecidos na utilização do link, mas o algoritmo TCP Vegas teve um pouco a frente dos demais quando a rede estava com congestionamento, conseguindo utilizar melhor o link, trafegando mais pacotes. A justificativa do melhor desempenho do TCP Vegas é sua maneira eficiente de detectar o

congestionamento na rede com seu mecanismo pró-ativo, levando a uma melhor utilização do *link*.

A seguir na Tabela 4, é mostrada a quantidade de pacotes que passaram pelos nós Goiás e Distrito Federal nos cenários executados:

Tabela 4: Utilização do Link: Quantidade total de dados – Rede RNP Antiga

Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	4794	4794	4917	4793	4794	4793
70%	4792	4768	4916	4792	4795	4733
80%	4797	4795	4917	4796	4797	4795

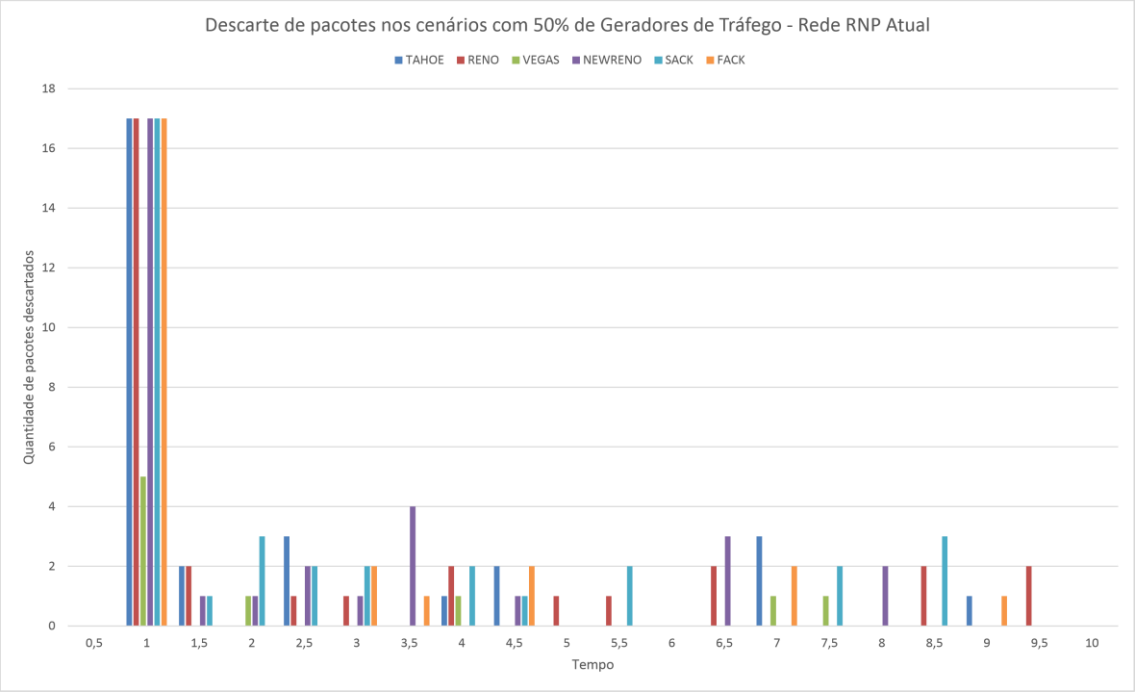
Como foi visto nas Figuras 16, 17 e 18 mesmo com o aumento de geradores de tráfego, o algoritmo TCP Vegas conseguiu utilizar melhor o link entre Goiás e Distrito Federal devido a sua maneira mais eficiente de detecção de congestionamento na rede.

6.2 REDE RNP ATUAL

6.2.1 Descarte

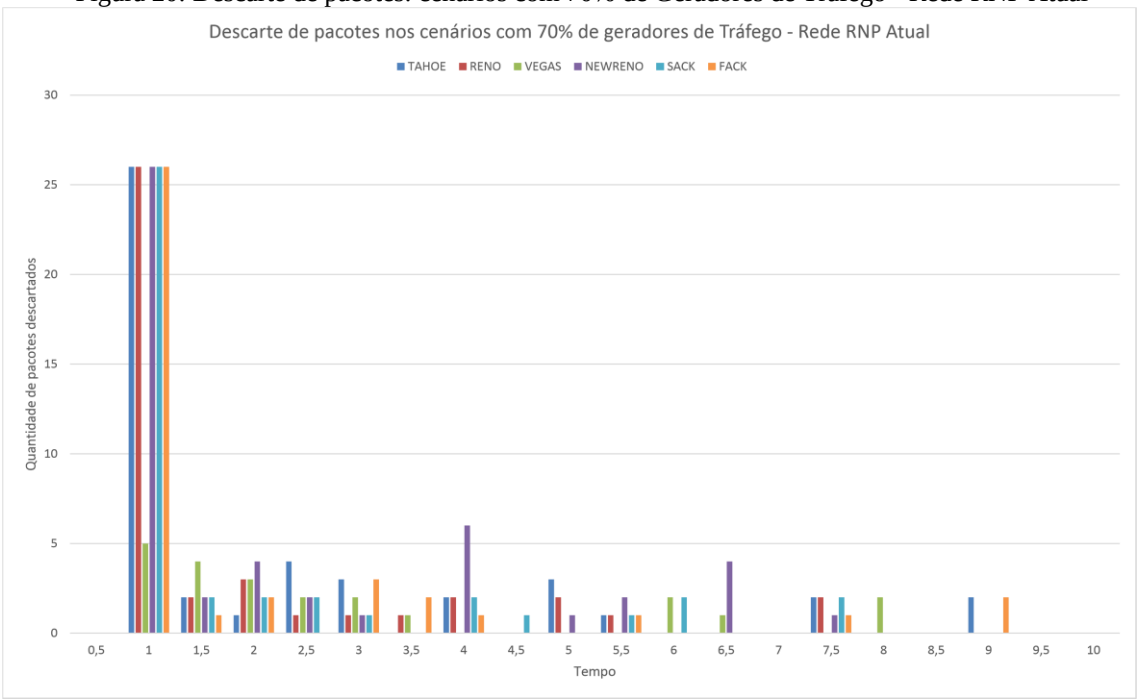
As Figuras 19, 20 e 21 mostram como os algoritmos de controle de congestionamento se comportaram no decorrer da execução dos cenários da topologia de rede utilizada atualmente pela RNP.

Figura 19: Descarte de pacotes: cenários com 50% de Geradores de Tráfego - Rede RNP Atual



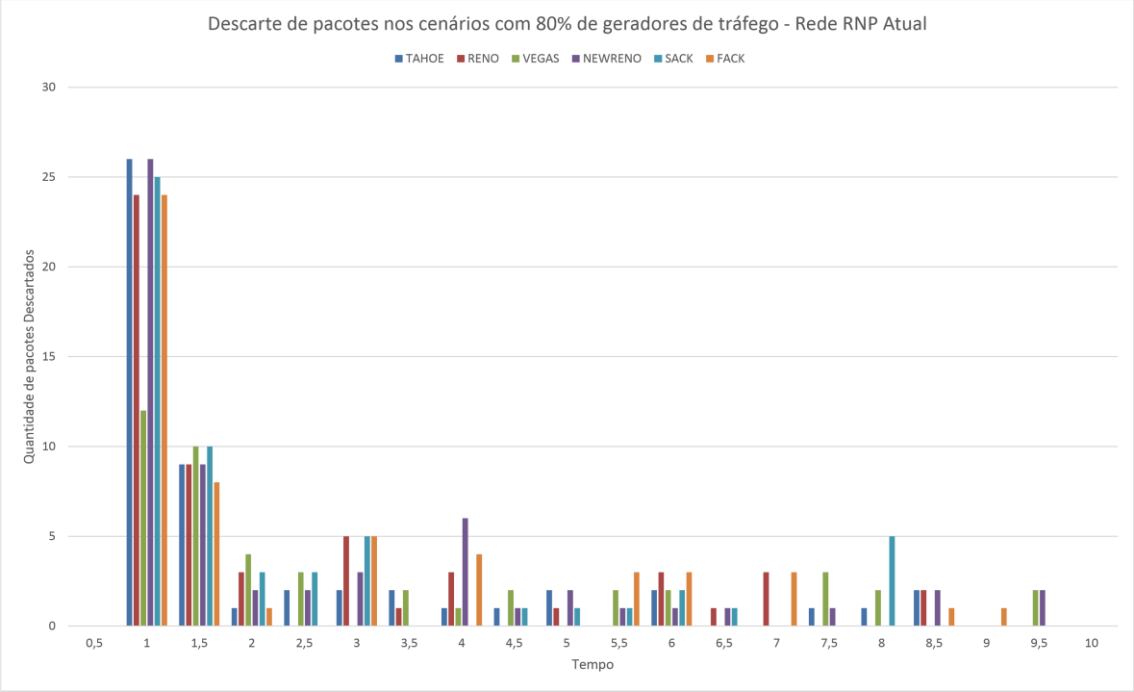
Fonte: Elaborado pelo autor

Figura 20: Descarte de pacotes: cenários com 70% de Geradores de Tráfego - Rede RNP Atual



Fonte: Elaborado pelo autor

Figura 21: Descarte de pacotes: cenários com 80% de Geradores de Tráfego - Rede RNP Atual



Fonte: Elaborado pelo autor

De acordo com as Figuras 19, 20 e 21, mesmo com a atualização da Rede da RNP, podemos observar que o TCP Vegas teve um melhor desempenho com relação aos outros algoritmos.

A Tabela 5 mostra o total de pacotes que foram descartados nos cenários da Rede utilizada atualmente pela RNP:

Tabela 5: Quantidade de pacotes descartados – Rede RNP Atual

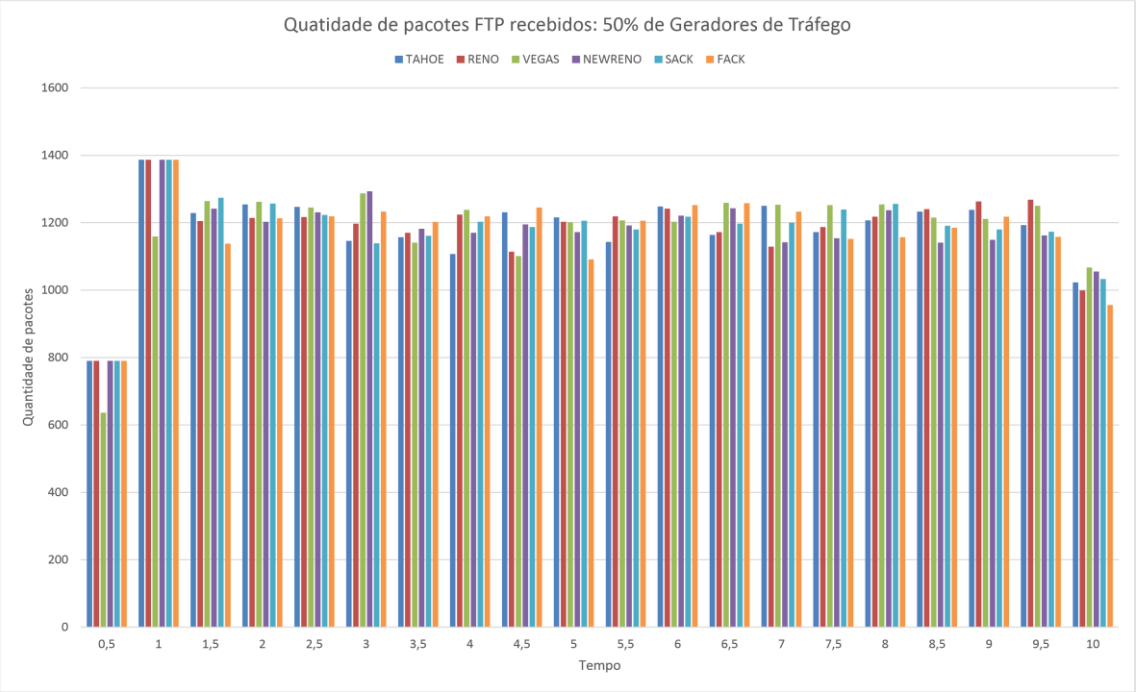
Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	29	31	9	32	35	25
70%	46	41	22	49	41	39
80%	52	55	45	59	57	53

Pode ser visto na Tabela 5 que o TCP Vegas tem um desempenho superior às demais implementações e que o TCP Tahoe obteve os mesmos resultados das análises dos cenários na Rede RNP Antiga, conseguindo menos descartes que suas implementações sucessoras.

6.2.2 Pacotes Recebidos

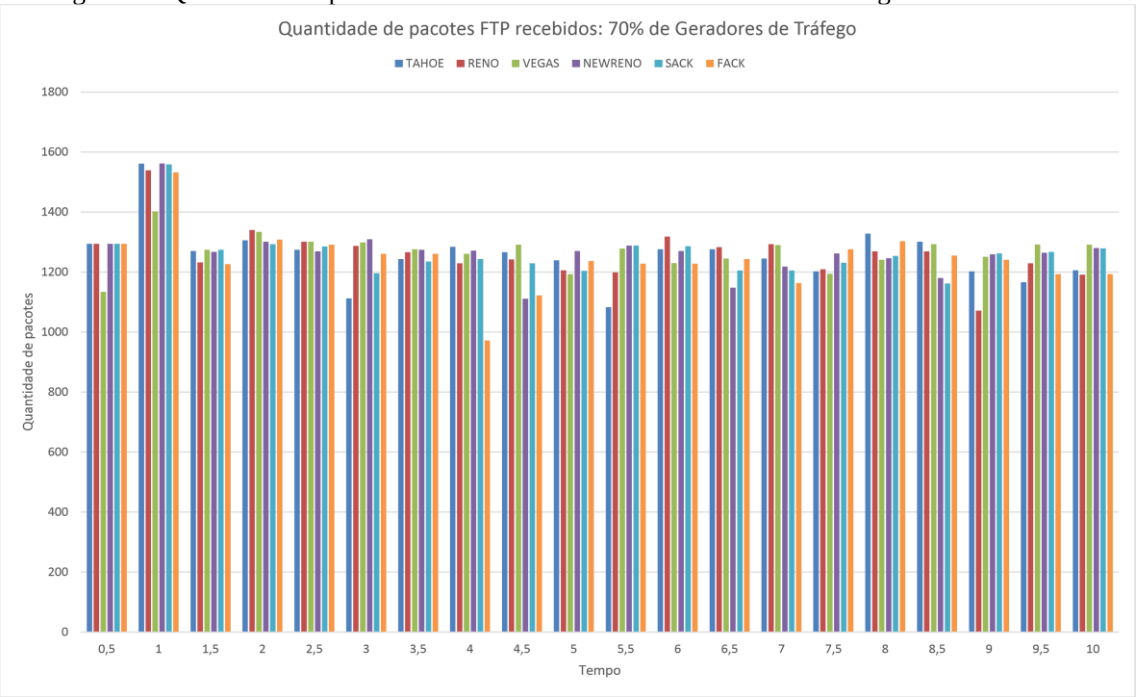
Será mostrado nas Figuras 22, 23 e 24, a quantidade de pacotes FTP recebidos no decorrer da execução dos cenários da Rede utilizada atualmente pela RNP.

Figura 22: Quantidade de pacotes FTP recebidos: 50% de Geradores de Tráfego - Rede RNP Atual



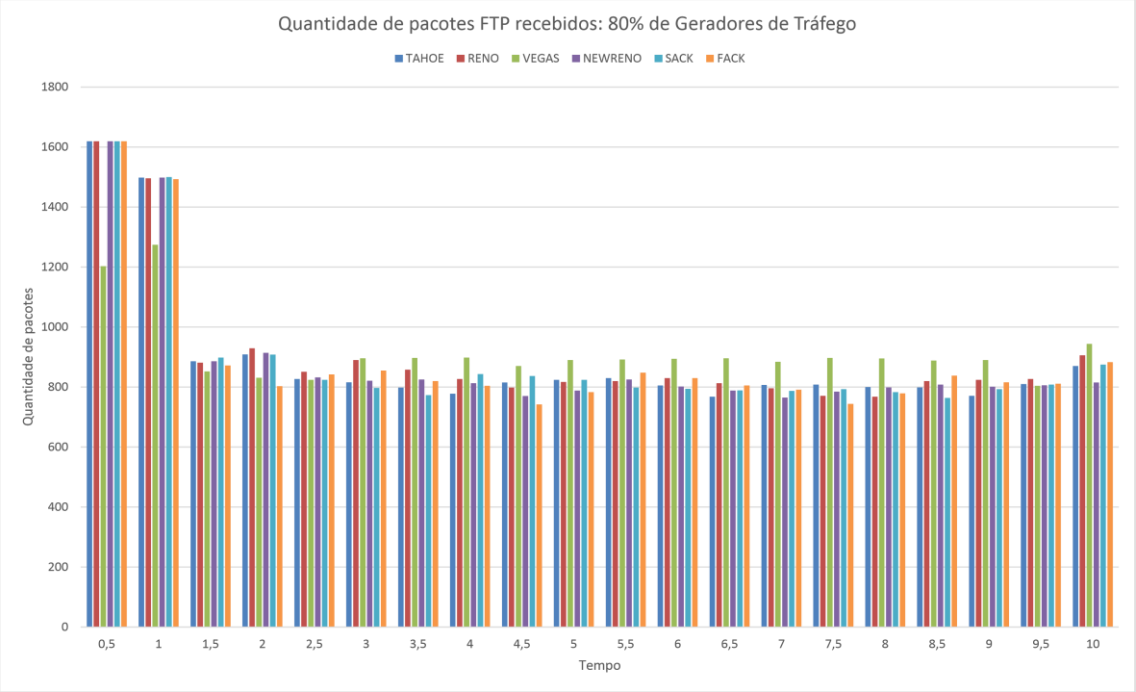
Fonte: Elaborado pelo autor

Figura 23: Quantidade de pacotes FTP recebidos: 70% de Geradores de Tráfego - Rede RNP Atual



Fonte: Elaborado pelo autor

Figura 24: Quantidade de pacotes FTP recebidos: 80% de Geradores de Tráfego - Rede RNP Atual



Fonte: Elaborado pelo autor

Pode-se verificar que a quantidade de pacotes recebidos aumentou em relação à topologia antiga da RNP, isso se deve ao fato de ter mais ligações para transmitir dados e o aumento do número de geradores de tráfego na rede. É percebido também que a quantidade cai significativamente depois que os geradores CBR estão sendo transmitidos pela rede. Nota-se que ocorre o mesmo que aconteceu nos cenários da topologia Antiga, em que o algoritmo TCP Vegas envia menos pacotes quando a rede não está congestionada, mas quando ocorre o congestionamento, o algoritmo consegue se sobressair em relação aos outros algoritmos.

A seguir, na Tabela 6, é mostrado o total de pacotes recebidos no decorrer das simulações, podemos notar que o TCP Vegas consegue o maior numero de pacotes FTP recebidos em todos os cenários.

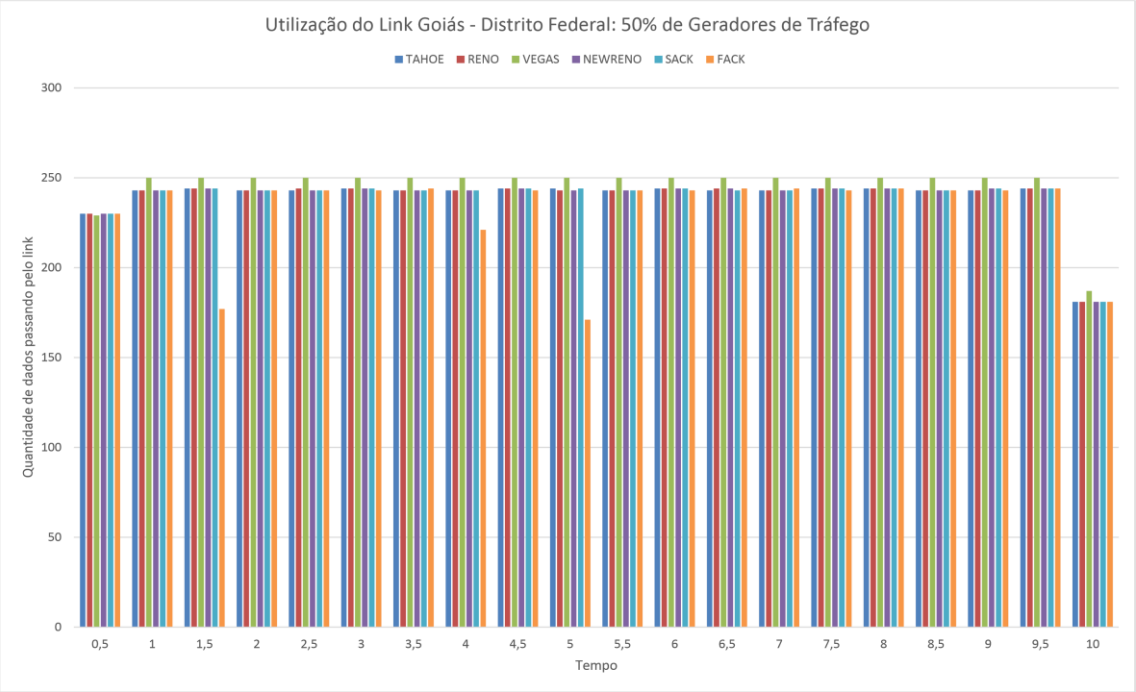
Tabela 6: Quantidade de pacotes FTP Recebidos – Rede RNP Atual

Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	23635	23658	23705	23561	23694	23512
70%	25134	25267	25370	25343	25251	24827
80%	17838	18141	18319	17760	17807	17778

6.2.3 Utilização dos *links*

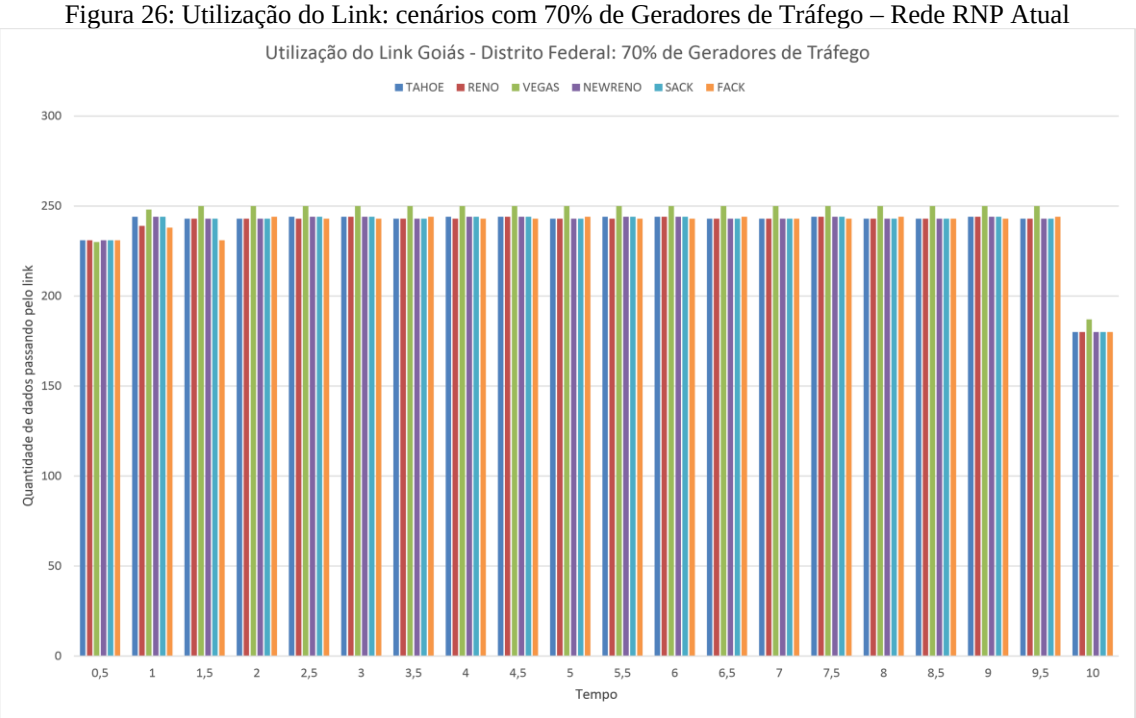
É mostrado nas Figuras 25, 26 e 27 a quantidade de dados que passaram pela ligação entre o nó Goiás e o nó Distrito Federal na topologia utilizada atualmente pela RNP.

Figura 25: Utilização do Link: cenários com 50% de Geradores de Tráfego - Rede RNP Atual



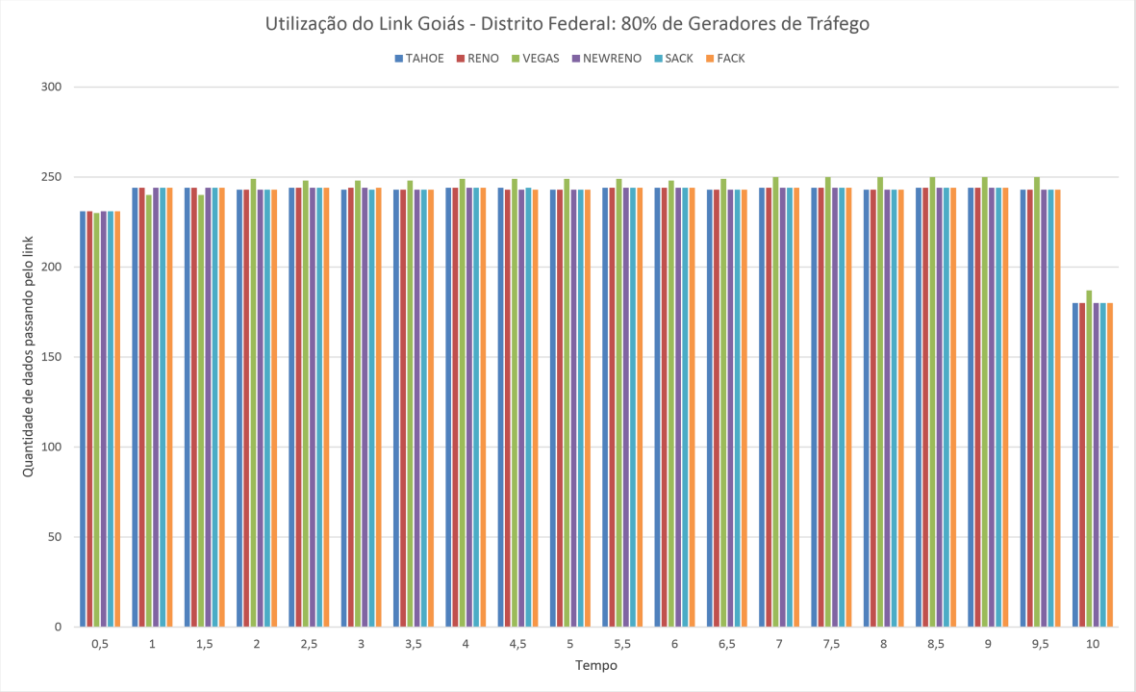
Fonte: Elaborado pelo autor

Figura 26: Utilização do Link: cenários com 70% de Geradores de Tráfego – Rede RNP Atual



Fonte: Elaborado pelo autor

Figura 27: Utilização do Link: cenários com 80% de Geradores de Tráfego – Rede RNP Atual



Fonte: Elaborado pelo autor

Como pode ser visto nas Figuras 25, 26 e 27 que os resultados são os mesmos da rede da RNP utilizada antigamente. As implementações tiveram resultados parecidos na utilização do *link*, mas o algoritmo TCP Vegas ficou à frente dos demais algoritmos quando a rede estava com congestionamento, conseguindo utilizar melhor o *link*, trafegando mais pacotes.

A seguir na Tabela 7, é mostrada a quantidade de pacotes que passaram pelos nós Goiás e Distrito Federal, podendo notar que os resultados não foram modificados de acordo com os resultados da Rede antiga da RNP, que mesmo aumentando os geradores de tráfego, o algoritmo TCP Vegas conseguiu utilizar melhor o *link* entre Goiás e Distrito Federal, devido à sua maneira mais eficiente de detecção de congestionamento na rede.

Tabela 7: Utilização do Link: Quantidade total de dados – Rede RNP Atual

Geradores de tráfego (%)	**TAHOE**	**RENO**	**VEGAS**	**NEWRENO**	**SACK**	**FACK**
50%	4793	4794	4916	4794	4794	4630
70%	4794	4786	4915	4794	4794	4774
80%	4796	4796	4883	4796	4796	4796

7 CONCLUSÃO

Este trabalho realizou uma análise de desempenho nos algoritmos de controle de congestionamento TCP Tahoe, TCP Reno, TCP Vegas, TCP NewReno, TCP Sack e TCP Fack. Através dos resultados apresentados, foi comprovado que a utilização de algoritmos de controle de congestionamento TCP trazem vantagens ao protocolo de transporte mais utilizado na Internet, em que o controle de envio de dados na rede, reduz a taxa de descarte.

Os resultados obtidos nas simulações indicam que o algoritmo de congestionamento TCP Vegas tem o melhor desempenho comparado aos outros algoritmos nas três métricas analisadas, tanto na rede antiga da RNP quanto na rede atualmente utilizada. Isso se deve a sua pró-atividade em monitorar o estado da rede, utilizando as variáveis vazão real e vazão esperada, ajustando a janela de congestionamento CWND e controlando o congestionamento na rede.

Foi observado que o primeiro algoritmo de controle de congestionamento TCP Tahoe e os outros algoritmos que se baseiam em alguns mecanismos desta implementação tiveram um desempenho pior quando comparado ao TCP Vegas. Isso comprova a importância da criação de um algoritmo de controle de congestionamento que não se baseia na perda de dados para diminuir o envio de dados, mas sim um algoritmo de controle de congestionamento que se baseia no resultado de variáveis que informam a situação da rede.

Foi visto que houve uma melhora na topologia da RNP com sua atualização, que suporta o envio de mais dados na rede e tem mais opções de redundância com o aumento do número de *links*.

REFERÊNCIAS

ABED, G. A., ISMAIL, M., & JUMARI, K. **A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions.** Australian Journal of Basic & Applied Sciences, *5*(12), 2011.

ALLMAN M.; PAXSON V.; STEVENS W. **TCP Congestion Control**. IETF Request for Comments (RFC) 2581, 1999.

BENÍTEZ, CÁCERES, D.B. **Aplicação de Algoritmos de Controle para o tratamento de congestionamento em Redes de Computadores.** 2010. 159 f. Tese de Doutorado (Pós-Graduação em Engenharia Elétrica) – Universidade do Rio de Janeiro, Rio de Janeiro, 2010.

CAVALCANTI, J.L **Análise comparativa dos algoritmos de controle de congestionamento do TCP.** 2005. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia da Computação) - Universidade Federal de Pernambuco, Recife, 2005.

FALL, Kevin; VARADHAN, Kannan. **The *ns* Manual (formerly *ns* Notes and Documentation),** 2013**.** Available in <http://www.isi.edu/nsnam/ns/doc/index.html>. Last accessed em 30/10/2013.

FLOYD, S. **TCP Selective Acknowledgment Options.** IETF Request for Comments (RFC) 2018, 1996.

HASSAN, M.; JAIN, R. **High Performance TCP/IP Networking.** 1ª.ed. India: Pearson, 2004.

KUROSE, J. F; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down.** 5ª. ed. São Paulo: Addison-Wesley, 2010.

MATHIS, M.; MAHDAVI, J. **Forward acknowledgement: Refining TCP congestion control**. ACM SIGCOMM Computer Communication Review, v. 26, n. 4, p. 281-291, 1996.

PORTNOI, M.; ARAÚJO, R. **Network Simulator – Visão Geral da Ferramenta de Simulação de Redes.** Salvador: UNIFACS, 2003.

PRETE, L. R; SHINODA, A. A. **Análise do Comportamento das Variações do Protocolo TCP.** In: Congresso Nacional de Matemática Aplicada e Computacional, 32. 2009, Cuiabá. **Anais do CNMAC**, Cuiabá: Editora Cubo, 2009. p. 7-6.

TANENBAUM, A. S. **Redes de Computadores.** 5ª. ed. São Paulo: Campus, 2011.

APÊNDICES

APÊNDICE A – Scripts utilizados nas simulações

cenario01.tcl

```
#CRIANDO OBJETO SIMULADOR
set ns [new Simulator]

#ABRINDO O ARQUIVO TRACE NAM
set nf [open saida01.nam w]
$ns namtrace-all $nf

#GRAVAR OS TRACES EM UM ARQUIVO PARA USAR PARA A ANÁLISE
set tf [open analise01.tr w]
$ns trace-all $tf

#CRIAÇÃO DOS NÓS
set acre [$ns node]
set rondonia [$ns node]
set mgrosso [$ns node]
set mgrossosul [$ns node]
set parana [$ns node]
set rgrandesul [$ns node]
set scatarina [$ns node]
set spaulo [$ns node]
set rjaneiro [$ns node]
set tocantins [$ns node]
set goias [$ns node]
set dfederal [$ns node]
set amazonas [$ns node]
set mgerais [$ns node]
set bahia [$ns node]
set sergipe [$ns node]
set alagoas [$ns node]
set esanto [$ns node]
set ceara [$ns node]
set roraima [$ns node]
set rnorte [$ns node]
set pb_jpa [$ns node]
set pb_cge [$ns node]
set pernambuco [$ns node]
set maranhao [$ns node]
set amapa [$ns node]
set para [$ns node]
set piaui [$ns node]

#CRIAÇÃO DAS LIGAÇÕES DOS NÓS
$ns duplex-link $acre $rondonia 2Mb 10ms DropTail
$ns duplex-link $spaulo $mgerais 5Mb 10ms DropTail
$ns duplex-link $rondonia $mgrosso 2Mb 10ms DropTail
$ns duplex-link $mgrosso $mgrossosul 2Mb 10ms DropTail
$ns duplex-link $mgrosso $goias 2Mb 10ms DropTail
$ns duplex-link $mgrossosul $parana 2Mb 10ms DropTail
$ns duplex-link $rgrandesul $scatarina 5Mb 5ms DropTail
$ns duplex-link $goias $dfederal 5Mb 5ms DropTail
$ns duplex-link $parana $spaulo 5Mb 10ms DropTail
$ns duplex-link $parana $rgrandesul 5Mb 10ms DropTail
$ns duplex-link $scatarina $spaulo 5Mb 10ms DropTail
```

```
$ns duplex-link $goias $tocantins 2Mb 10ms DropTail
$ns duplex-link $dfederal $rjaneiro 5Mb 15ms DropTail
$ns duplex-link $mgerais $bahia 5Mb 10ms DropTail
$ns duplex-link $dfederal $amazonas 2Mb 20ms DropTail
$ns duplex-link $bahia $esanto 5Mb 10ms DropTail
$ns duplex-link $amazonas $roraima 2Mb 10ms DropTail
$ns duplex-link $mgerais $ceara 5Mb 20ms DropTail
$ns duplex-link $roraima $ceara 1Mb 25ms DropTail
$ns duplex-link $ceara $rnorte 5Mb 10ms DropTail
$ns duplex-link $ceara $maranhao 2Mb 10ms DropTail
$ns duplex-link $bahia $sergipe 5Mb 10ms DropTail
$ns duplex-link $sergipe $alagoas 5Mb 5ms DropTail
$ns duplex-link $alagoas $pernambuco 5Mb 5ms DropTail
$ns duplex-link $rnorte $pb_jpa 5Mb 5ms DropTail
$ns duplex-link $pb_jpa $pb_cge 5Mb 5ms DropTail
$ns duplex-link $pb_cge $pernambuco 5Mb 5ms DropTail
$ns duplex-link $maranhao $para 2Mb 10ms DropTail
$ns duplex-link $para $amapa 1Mb 10ms DropTail
$ns duplex-link $para $piaui 2Mb 15ms DropTail
$ns duplex-link $piaui $pernambuco 2Mb 10ms DropTail
$ns duplex-link $rjaneiro $esanto 5Mb 10ms DropTail
$ns duplex-link $spaulo $rjaneiro 5Mb 10ms DropTail
$ns duplex-link $mgerais $dfederal 5Mb 10ms DropTail

#FILA NAS LIGAÇÕES
$ns duplex-link-op $acre $rondonia queuePos 0.5
$ns duplex-link-op $spaulo $mgerais queuePos 0.5
$ns duplex-link-op $rondonia $mgrosso queuePos 0.5
$ns duplex-link-op $mgrosso $mgrossosul queuePos 0.5
$ns duplex-link-op $mgrosso $goias queuePos 0.5
$ns duplex-link-op $mgrossosul $parana queuePos 0.5
$ns duplex-link-op $rgrandesul $scatarina queuePos 0.5
$ns duplex-link-op $goias $dfederal queuePos 0.5
$ns duplex-link-op $parana $spaulo queuePos 0.5
$ns duplex-link-op $parana $rgrandesul queuePos 0.5
$ns duplex-link-op $scatarina $spaulo queuePos 0.5
$ns duplex-link-op $goias $tocantins queuePos 0.5
$ns duplex-link-op $dfederal $rjaneiro queuePos 0.5
$ns duplex-link-op $mgerais $bahia queuePos 0.5
$ns duplex-link-op $dfederal $amazonas queuePos 0.5
$ns duplex-link-op $bahia $esanto queuePos 0.5
$ns duplex-link-op $amazonas $roraima queuePos 0.5
$ns duplex-link-op $mgerais $ceara queuePos 0.5
$ns duplex-link-op $roraima $ceara queuePos 0.5
$ns duplex-link-op $ceara $rnorte queuePos 0.5
$ns duplex-link-op $ceara $maranhao queuePos 0.5
$ns duplex-link-op $bahia $sergipe queuePos 0.5
$ns duplex-link-op $sergipe $alagoas queuePos 0.5
$ns duplex-link-op $alagoas $pernambuco queuePos 0.5
$ns duplex-link-op $rnorte $pb_jpa queuePos 0.5
$ns duplex-link-op $pb_jpa $pb_cge queuePos 0.5
$ns duplex-link-op $pb_cge $pernambuco queuePos 0.5
$ns duplex-link-op $maranhao $para queuePos 0.5
$ns duplex-link-op $para $amapa queuePos 0.5
$ns duplex-link-op $para $piaui queuePos 0.5
$ns duplex-link-op $piaui $pernambuco queuePos 0.5
$ns duplex-link-op $rjaneiro $esanto queuePos 0.5
$ns duplex-link-op $spaulo $rjaneiro queuePos 0.5
$ns duplex-link-op $mgerais $dfederal queuePos 0.5
```

```
#############TCP TAHOE
set tcp [new Agent/TCP]
$tcp set fid_ 1
$tcp set windows_ 2000
set sink [new Agent/TCPSink]
$ns attach-agent $acre $tcp
$ns attach-agent $ceara $sink
$ns connect $tcp $sink
set ftp [$tcp attach-source FTP]
$ns at 0.1 "$ftp start"

set tcp1 [new Agent/TCP]
$tcp1 set fid_ 1
$tcp1 set windows_ 2000
set sink1 [new Agent/TCPSink]
$ns attach-agent $spaulo $tcp1
$ns attach-agent $acre $sink1
$ns connect $tcp1 $sink1
set ftp1 [$tcp1 attach-source FTP]
$ns at 0.1 "$ftp1 start"

set tcp2 [new Agent/TCP]
$tcp2 set fid_ 1
$tcp2 set windows_ 2000
set sink2 [new Agent/TCPSink]
$ns attach-agent $rondonia $tcp2
$ns attach-agent $rgrandesul $sink2
$ns connect $tcp2 $sink2
set ftp2 [$tcp2 attach-source FTP]
$ns at 0.1 "$ftp2 start"

set tcp3 [new Agent/TCP]
$tcp3 set fid_ 1
$tcp3 set windows_ 2000
set sink3 [new Agent/TCPSink]
$ns attach-agent $mgrosso $tcp3
$ns attach-agent $amazonas $sink3
$ns connect $tcp3 $sink3
set ftp3 [$tcp3 attach-source FTP]
$ns at 0.1 "$ftp3 start"

set tcp4 [new Agent/TCP]
$tcp4 set fid_ 1
$tcp4 set windows_ 2000
set sink4 [new Agent/TCPSink]
$ns attach-agent $mgrossosul $tcp4
$ns attach-agent $rnorte $sink4
$ns connect $tcp4 $sink4
set ftp4 [$tcp4 attach-source FTP]
$ns at 0.1 "$ftp4 start"

set tcp5 [new Agent/TCP]
$tcp5 set fid_ 1
$tcp5 set windows_ 2000
set sink5 [new Agent/TCPSink]
$ns attach-agent $rgrandesul $tcp5
$ns attach-agent $maranhao $sink5
$ns connect $tcp5 $sink5
set ftp5 [$tcp5 attach-source FTP]
$ns at 0.1 "$ftp5 start"
```

```
set tcp6 [new Agent/TCP]
$tcp6 set fid_ 1
$tcp6 set windows_ 2000
set sink6 [new Agent/TCPSink]
$ns attach-agent $goias $tcp6
$ns attach-agent $dfederal $sink6
$ns connect $tcp6 $sink6
set ftp6 [$tcp6 attach-source FTP]
$ns at 0.1 "$ftp6 start"

set tcp7 [new Agent/TCP]
$tcp7 set fid_ 1
$tcp7 set windows_ 2000
set sink7 [new Agent/TCPSink]
$ns attach-agent $para $tcp7
$ns attach-agent $spaulo $sink7
$ns connect $tcp7 $sink7
set ftp7 [$tcp7 attach-source FTP]
$ns at 0.1 "$ftp7 start"

set tcp8 [new Agent/TCP]
$tcp8 set fid_ 1
$tcp8 set windows_ 2000
set sink8 [new Agent/TCPSink]
$ns attach-agent $parana $tcp8
$ns attach-agent $ceara $sink8
$ns connect $tcp8 $sink8
set ftp8 [$tcp8 attach-source FTP]
$ns at 0.1 "$ftp8 start"

#####TRÁFEGO CBR

set udp0 [new Agent/UDP]
$ns attach-agent $scatarina $udp0
set null0 [new Agent/Null]
$ns attach-agent $spaulo $null0
$ns connect $udp0 $null0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 1000
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
$ns at 0.5 "$cbr0 start"
$ns at 9.5 "$cbr0 stop"

set udp1 [new Agent/UDP]
$ns attach-agent $goias $udp1
set null1 [new Agent/Null]
$ns attach-agent $para $null1
$ns connect $udp1 $null1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"

set udp2 [new Agent/UDP]
$ns attach-agent $dfederal $udp2
set null2 [new Agent/Null]
$ns attach-agent $mgrosso $null2
$ns connect $udp2 $null2
```

```
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 1000
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2
$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"

set udp3 [new Agent/UDP]
$ns attach-agent $mgerais $udp3
set null3 [new Agent/Null]
$ns attach-agent $amapa $null3
$ns connect $udp3 $null3
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 1000
$cbr3 set interval_ 0.005
$cbr3 attach-agent $udp3
$ns at 0.5 "$cbr3 start"
$ns at 9.5 "$cbr3 stop"

set udp4 [new Agent/UDP]
$ns attach-agent $rjaneiro $udp4
set null4 [new Agent/Null]
$ns attach-agent $pb_jpa $null4
$ns connect $udp4 $null4
set cbr4 [new Application/Traffic/CBR]
$cbr4 set packetSize_ 1000
$cbr4 set interval_ 0.005
$cbr4 attach-agent $udp4
$ns at 0.5 "$cbr4 start"
$ns at 9.5 "$cbr4 stop"

set udp5 [new Agent/UDP]
$ns attach-agent $dfederal $udp5
set null5 [new Agent/Null]
$ns attach-agent $para $null5
$ns connect $udp5 $null5
set cbr5 [new Application/Traffic/CBR]
$cbr5 set packetSize_ 1000
$cbr5 set interval_ 0.005
$cbr5 attach-agent $udp5
$ns at 0.5 "$cbr5 start"
$ns at 9.5 "$cbr5 stop"

set udp6 [new Agent/UDP]
$ns attach-agent $ceara $udp6
set null6 [new Agent/Null]
$ns attach-agent $rgrandesul $null6
$ns connect $udp6 $null6
set cbr6 [new Application/Traffic/CBR]
$cbr6 set packetSize_ 1000
$cbr6 set interval_ 0.005
$cbr6 attach-agent $udp6
$ns at 0.5 "$cbr6 start"
$ns at 9.5 "$cbr6 stop"

set udp7 [new Agent/UDP]
$ns attach-agent $piaui $udp7
set null7 [new Agent/Null]
$ns attach-agent $spaulo $null7
$ns connect $udp7 $null7
set cbr7 [new Application/Traffic/CBR]
```

```
$cbr7 set packetSize_ 1000
$cbr7 set interval_ 0.005
$cbr7 attach-agent $udp7
$ns at 0.5 "$cbr7 start"
$ns at 9.5 "$cbr7 stop"

$ns at 10.0 "finish"
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam saida01.nam &
        exit 0
}
```

cenario04.tcl

```
#EXECUTAR A SIMULAÇÃO
$ns run

#CRIANDO OBJETO SIMULADOR
set ns [new Simulator]

#ABRINDO O ARQUIVO TRACE NAM
set nf [open saida04.nam w]
$ns namtrace-all $nf

#GRAVAR OS TRACES EM UM ARQUIVO PARA USAR PARA A ANÁLISE
set tf [open analise04.tr w]
$ns trace-all $tf

#CRIAÇÃO DOS NÓS
set acre [$ns node]
set rondonia [$ns node]
set mgrosso [$ns node]
set mgrossosul [$ns node]
set parana [$ns node]
set rgrandesul [$ns node]
set scatarina [$ns node]
set spaulo [$ns node]
set rjaneiro [$ns node]
set tocantins [$ns node]
set goias [$ns node]
set dfederal [$ns node]
set amazonas [$ns node]
set mgerais [$ns node]
set bahia [$ns node]
set sergipe [$ns node]
set alagoas [$ns node]
set esanto [$ns node]
set ceara [$ns node]
set roraima [$ns node]
set rnorte [$ns node]
set pb_jpa [$ns node]
set pb_cge [$ns node]
set pernambuco [$ns node]
set maranhao [$ns node]
set amapa [$ns node]
```

```
set para [$ns node]
set piaui [$ns node]

#CRIAÇÃO DAS LIGAÇÕES DOS NÓS
$ns duplex-link $acre $rondonia 2Mb 10ms DropTail
$ns duplex-link $spaulo $mgerais 5Mb 10ms DropTail
$ns duplex-link $rondonia $mgrosso 2Mb 10ms DropTail
$ns duplex-link $mgrosso $mgrossosul 2Mb 10ms DropTail
$ns duplex-link $mgrosso $goias 2Mb 10ms DropTail
$ns duplex-link $mgrossosul $parana 2Mb 10ms DropTail
$ns duplex-link $rgrandesul $scatarina 5Mb 5ms DropTail
$ns duplex-link $goias $dfederal 5Mb 5ms DropTail
$ns duplex-link $parana $spaulo 5Mb 10ms DropTail
$ns duplex-link $parana $rgrandesul 5Mb 10ms DropTail
$ns duplex-link $scatarina $spaulo 5Mb 10ms DropTail
$ns duplex-link $goias $tocantins 2Mb 10ms DropTail
$ns duplex-link $dfederal $rjaneiro 5Mb 15ms DropTail
$ns duplex-link $mgerais $bahia 5Mb 10ms DropTail
$ns duplex-link $dfederal $amazonas 2Mb 20ms DropTail
$ns duplex-link $bahia $esanto 5Mb 10ms DropTail
$ns duplex-link $amazonas $roraima 2Mb 10ms DropTail
$ns duplex-link $mgerais $ceara 5Mb 20ms DropTail
$ns duplex-link $roraima $ceara 1Mb 25ms DropTail
$ns duplex-link $ceara $rnorte 5Mb 10ms DropTail
$ns duplex-link $ceara $maranhao 2Mb 10ms DropTail
$ns duplex-link $bahia $sergipe 5Mb 10ms DropTail
$ns duplex-link $sergipe $alagoas 5Mb 5ms DropTail
$ns duplex-link $alagoas $pernambuco 5Mb 5ms DropTail
$ns duplex-link $rnorte $pb_jpa 5Mb 5ms DropTail
$ns duplex-link $pb_jpa $pb_cge 5Mb 5ms DropTail
$ns duplex-link $pb_cge $pernambuco 5Mb 5ms DropTail
$ns duplex-link $maranhao $para 2Mb 10ms DropTail
$ns duplex-link $para $amapa 1Mb 10ms DropTail
$ns duplex-link $para $piaui 2Mb 15ms DropTail
$ns duplex-link $piaui $pernambuco 2Mb 10ms DropTail
$ns duplex-link $rjaneiro $esanto 5Mb 10ms DropTail
$ns duplex-link $spaulo $rjaneiro 5Mb 10ms DropTail
$ns duplex-link $mgerais $dfederal 5Mb 10ms DropTail
$ns duplex-link $ceara $dfederal 5Mb 20ms DropTail
$ns duplex-link $ceara $rjaneiro 5Mb 25ms DropTail

#FILA NAS LIGAÇÕES
$ns duplex-link-op $acre $rondonia queuePos 0.5
$ns duplex-link-op $spaulo $mgerais queuePos 0.5
$ns duplex-link-op $rondonia $mgrosso queuePos 0.5
$ns duplex-link-op $mgrosso $mgrossosul queuePos 0.5
$ns duplex-link-op $mgrosso $goias queuePos 0.5
$ns duplex-link-op $mgrossosul $parana queuePos 0.5
$ns duplex-link-op $rgrandesul $scatarina queuePos 0.5
$ns duplex-link-op $goias $dfederal queuePos 0.5
$ns duplex-link-op $parana $spaulo queuePos 0.5
$ns duplex-link-op $parana $rgrandesul queuePos 0.5
$ns duplex-link-op $scatarina $spaulo queuePos 0.5
$ns duplex-link-op $goias $tocantins queuePos 0.5
$ns duplex-link-op $dfederal $rjaneiro queuePos 0.5
$ns duplex-link-op $mgerais $bahia queuePos 0.5
$ns duplex-link-op $dfederal $amazonas queuePos 0.5
$ns duplex-link-op $bahia $esanto queuePos 0.5
$ns duplex-link-op $amazonas $roraima queuePos 0.5
$ns duplex-link-op $mgerais $ceara queuePos 0.5
$ns duplex-link-op $roraima $ceara queuePos 0.5
```

```
$ns duplex-link-op $ceara $rnorte queuePos 0.5
$ns duplex-link-op $ceara $maranhao queuePos 0.5
$ns duplex-link-op $bahia $sergipe queuePos 0.5
$ns duplex-link-op $sergipe $alagoas queuePos 0.5
$ns duplex-link-op $alagoas $pernambuco queuePos 0.5
$ns duplex-link-op $rnorte $pb_jpa queuePos 0.5
$ns duplex-link-op $pb_jpa $pb_cge queuePos 0.5
$ns duplex-link-op $pb_cge $pernambuco queuePos 0.5
$ns duplex-link-op $maranhao $para queuePos 0.5
$ns duplex-link-op $para $amapa queuePos 0.5
$ns duplex-link-op $para $piaui queuePos 0.5
$ns duplex-link-op $piaui $pernambuco queuePos 0.5
$ns duplex-link-op $rjaneiro $esanto queuePos 0.5
$ns duplex-link-op $spaulo $rjaneiro queuePos 0.5
$ns duplex-link-op $mgerais $dfederal queuePos 0.5
$ns duplex-link-op $ceara $dfederal queuePos 0.5
$ns duplex-link-op $ceara $rjaneiro queuePos 0.5

#############TCP TAHOE
set tcp [new Agent/TCP]
$tcp set fid_ 1
$tcp set windows_ 2000
set sink [new Agent/TCPSink]
$ns attach-agent $acre $tcp
$ns attach-agent $ceara $sink
$ns connect $tcp $sink
set ftp [$tcp attach-source FTP]
$ns at 0.1 "$ftp start"

set tcp1 [new Agent/TCP]
$tcp1 set fid_ 1
$tcp1 set windows_ 2000
set sink1 [new Agent/TCPSink]
$ns attach-agent $spaulo $tcp1
$ns attach-agent $acre $sink1
$ns connect $tcp1 $sink1
set ftp1 [$tcp1 attach-source FTP]
$ns at 0.1 "$ftp1 start"

set tcp2 [new Agent/TCP]
$tcp2 set fid_ 1
$tcp2 set windows_ 2000
set sink2 [new Agent/TCPSink]
$ns attach-agent $rondonia $tcp2
$ns attach-agent $rgrandesul $sink2
$ns connect $tcp2 $sink2
set ftp2 [$tcp2 attach-source FTP]
$ns at 0.1 "$ftp2 start"

set tcp3 [new Agent/TCP]
$tcp3 set fid_ 1
$tcp3 set windows_ 2000
set sink3 [new Agent/TCPSink]
$ns attach-agent $mgrosso $tcp3
$ns attach-agent $amazonas $sink3
$ns connect $tcp3 $sink3
set ftp3 [$tcp3 attach-source FTP]
$ns at 0.1 "$ftp3 start"

set tcp4 [new Agent/TCP]
$tcp4 set fid_ 1
```

```
$tcp4 set windows_ 2000
set sink4 [new Agent/TCPSink]
$ns attach-agent $mgrossosul $tcp4
$ns attach-agent $rnorte $sink4
$ns connect $tcp4 $sink4
set ftp4 [$tcp4 attach-source FTP]
$ns at 0.1 "$ftp4 start"

set tcp5 [new Agent/TCP]
$tcp5 set fid_ 1
$tcp5 set windows_ 2000
set sink5 [new Agent/TCPSink]
$ns attach-agent $rgrandesul $tcp5
$ns attach-agent $maranhao $sink5
$ns connect $tcp5 $sink5
set ftp5 [$tcp5 attach-source FTP]
$ns at 0.1 "$ftp5 start"

set tcp6 [new Agent/TCP]
$tcp6 set fid_ 1
$tcp6 set windows_ 2000
set sink6 [new Agent/TCPSink]
$ns attach-agent $goias $tcp6
$ns attach-agent $dfederal $sink6
$ns connect $tcp6 $sink6
set ftp6 [$tcp6 attach-source FTP]
$ns at 0.1 "$ftp6 start"

set tcp7 [new Agent/TCP]
$tcp7 set fid_ 1
$tcp7 set windows_ 2000
set sink7 [new Agent/TCPSink]
$ns attach-agent $para $tcp7
$ns attach-agent $spaulo $sink7
$ns connect $tcp7 $sink7
set ftp7 [$tcp7 attach-source FTP]
$ns at 0.1 "$ftp7 start"

set tcp8 [new Agent/TCP]
$tcp8 set fid_ 1
$tcp8 set windows_ 2000
set sink8 [new Agent/TCPSink]
$ns attach-agent $parana $tcp8
$ns attach-agent $ceara $sink8
$ns connect $tcp8 $sink8
set ftp8 [$tcp8 attach-source FTP]
$ns at 0.1 "$ftp8 start"


#####TRAFEGO CBR
set udp0 [new Agent/UDP]
$ns attach-agent $scatarina $udp0
set null0 [new Agent/Null]
$ns attach-agent $spaulo $null0
$ns connect $udp0 $null0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 1000
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
$ns at 0.5 "$cbr0 start"
$ns at 9.5 "$cbr0 stop"
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $goias $udp1
set null1 [new Agent/Null]
$ns attach-agent $para $null1
$ns connect $udp1 $null1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"

set udp2 [new Agent/UDP]
$ns attach-agent $dfederal $udp2
set null2 [new Agent/Null]
$ns attach-agent $mgrosso $null2
$ns connect $udp2 $null2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 1000
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2
$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"

set udp3 [new Agent/UDP]
$ns attach-agent $mgerais $udp3
set null3 [new Agent/Null]
$ns attach-agent $amapa $null3
$ns connect $udp3 $null3
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 1000
$cbr3 set interval_ 0.005
$cbr3 attach-agent $udp3
$ns at 0.5 "$cbr3 start"
$ns at 9.5 "$cbr3 stop"

set udp4 [new Agent/UDP]
$ns attach-agent $rjaneiro $udp4
set null4 [new Agent/Null]
$ns attach-agent $pb_jpa $null4
$ns connect $udp4 $null4
set cbr4 [new Application/Traffic/CBR]
$cbr4 set packetSize_ 1000
$cbr4 set interval_ 0.005
$cbr4 attach-agent $udp4
$ns at 0.5 "$cbr4 start"
$ns at 9.5 "$cbr4 stop"

set udp5 [new Agent/UDP]
$ns attach-agent $dfederal $udp5
set null5 [new Agent/Null]
$ns attach-agent $para $null5
$ns connect $udp5 $null5
set cbr5 [new Application/Traffic/CBR]
$cbr5 set packetSize_ 1000
$cbr5 set interval_ 0.005
$cbr5 attach-agent $udp5
$ns at 0.5 "$cbr5 start"
$ns at 9.5 "$cbr5 stop"
```

```
set udp6 [new Agent/UDP]
$ns attach-agent $ceara $udp6
set null6 [new Agent/Null]
$ns attach-agent $rgrandesul $null6
$ns connect $udp6 $null6
set cbr6 [new Application/Traffic/CBR]
$cbr6 set packetSize_ 1000
$cbr6 set interval_ 0.005
$cbr6 attach-agent $udp6
$ns at 0.5 "$cbr6 start"
$ns at 9.5 "$cbr6 stop"

set udp7 [new Agent/UDP]
$ns attach-agent $piaui $udp7
set null7 [new Agent/Null]
$ns attach-agent $spaulo $null7
$ns connect $udp7 $null7
set cbr7 [new Application/Traffic/CBR]
$cbr7 set packetSize_ 1000
$cbr7 set interval_ 0.005
$cbr7 attach-agent $udp7
$ns at 0.5 "$cbr7 start"
$ns at 9.5 "$cbr7 stop"

set udp8 [new Agent/UDP]
$ns attach-agent $pb_cge $udp8
set null8 [new Agent/Null]
$ns attach-agent $rgrandesul $null8
$ns connect $udp8 $null8
set cbr8 [new Application/Traffic/CBR]
$cbr8 set packetSize_ 1000
$cbr8 set interval_ 0.005
$cbr8 attach-agent $udp8
$ns at 0.5 "$cbr8 start"
$ns at 9.5 "$cbr8 stop"


$ns at 10.0 "finish"
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam saida04.nam &
        exit 0
}
#EXECUTAR A SIMULAÇÃO
$ns run
```

APÊNDICE B – Script utilizado nos arquivos trace

perda_pacotes.sh

```
#!/bin/bash
echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 0.1 A 0.5"
echo
cat $1 | awk '$2$5 >= "0.1tcp" && $2$5 <= "0.5tcp"' | grep tcp >> resultTime
```

```
PackLost1=`cat resultTime | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost1"
echo "Quantidade de pacotes perdidos no TEMPO 0.1 A 0.5 : $PackLost1" >>
rt$1.txt
rm resultTime

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 0.6 A 1.0"
echo
cat $1 | awk '$2$5 >= "0.6tcp" && $2$5 <= "1.0tcp"' | grep tcp >> resultTime1
PackLost2=`cat resultTime1 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost2"
echo "Quantidade de pacotes perdidos no TEMPO 0.6 A 1.0 : $PackLost2" >>
rt$1.txt
rm resultTime1

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 1.1 A 1.5"
echo
cat $1 | awk '$2$5 >= "1.1tcp" && $2$5 <= "1.5tcp"' | grep tcp >> resultTime2
PackLost3=`cat resultTime2 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost3"
echo "Quantidade de pacotes perdidos no TEMPO 1.1 A 1.5 : $PackLost3" >>
rt$1.txt
rm resultTime2

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 1.6 A 2.0"
echo
cat $1 | awk '$2$5 >= "1.6tcp" && $2$5 <= "2.0tcp"' | grep tcp >> resultTime3
PackLost4=`cat resultTime3 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost4"
echo "Quantidade de pacotes perdidos no TEMPO 1.6 A 2.0 : $PackLost4" >>
rt$1.txt
rm resultTime3

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 2.1 A 2.5"
echo
cat $1 | awk '$2$5 >= "2.1tcp" && $2$5 <= "2.5tcp"' | grep tcp >> resultTime4
PackLost5=`cat resultTime4 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost5"
echo "Quantidade de pacotes perdidos no TEMPO 2.1 A 2.5 : $PackLost5" >>
rt$1.txt
rm resultTime4

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 2.6 A 3.0"
echo
cat $1 | awk '$2$5 >= "2.6tcp" && $2$5 <= "3.0tcp"' | grep tcp >> resultTime5
PackLost6=`cat resultTime5 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost6"
echo "Quantidade de pacotes perdidos no TEMPO 2.6 A 3.0 : $PackLost6" >>
rt$1.txt
rm resultTime5

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 3.1 A 3.5"
echo
cat $1 | awk '$2$5 >= "3.1tcp" && $2$5 <= "3.5tcp"' | grep tcp >> resultTime6
```

```
PackLost7=`cat resultTime6 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost7"
echo "Quantidade de pacotes perdidos no TEMPO 3.1 A 3.5 : $PackLost7" >>
rt$1.txt
rm resultTime6

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 3.6 A 4.0"
echo
cat $1 | awk '$2$5 >= "3.6tcp" && $2$5 <= "4.0tcp"' | grep tcp >> resultTime7
PackLost8=`cat resultTime7 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost8"
echo "Quantidade de pacotes perdidos no TEMPO 3.6 A 4.0 : $PackLost8" >>
rt$1.txt
rm resultTime7

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 4.1 A 4.5"
echo
cat $1 | awk '$2$5 >= "4.1tcp" && $2$5 <= "4.5tcp"' | grep tcp >> resultTime8
PackLost9=`cat resultTime8 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost9"
echo "Quantidade de pacotes perdidos no TEMPO 4.1 A 4.5 : $PackLost9" >>
rt$1.txt
rm resultTime8

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 4.6 A 5.0"
echo
cat $1 | awk '$2$5 >= "4.6tcp" && $2$5 <= "5.0tcp"' | grep tcp >> resultTime9
PackLost10=`cat resultTime9 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost10"
echo "Quantidade de pacotes perdidos no TEMPO 4.6 A 5.0 : $PackLost10" >>
rt$1.txt
rm resultTime9

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 5.1 A 5.5"
echo
cat $1 | awk '$2$5 >= "5.1tcp" && $2$5 <= "5.5tcp"' | grep tcp >>
resultTime10
PackLost11=`cat resultTime10 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost11"
echo "Quantidade de pacotes perdidos no TEMPO 5.1 A 5.5 : $PackLost11" >>
rt$1.txt
rm resultTime10

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 5.6 A 6.0"
echo
cat $1 | awk '$2$5 >= "5.6tcp" && $2$5 <= "6.0tcp"' | grep tcp >>
resultTime11
PackLost12=`cat resultTime11 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost12"
echo "Quantidade de pacotes perdidos no TEMPO 5.6 A 6.0 : $PackLost12" >>
rt$1.txt
rm resultTime11

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 6.1 A 6.5"
```

```
echo
cat $1 | awk '$2$5 >= "6.1tcp" && $2$5 <= "6.5tcp"' | grep tcp >>
resultTime12
PackLost13=`cat resultTime12 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost13"
echo "Quantidade de pacotes perdidos no TEMPO 6.1 A 6.5 : $PackLost13" >>
rt$1.txt
rm resultTime12

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 6.6 A 7.0"
echo
cat $1 | awk '$2$5 >= "6.6tcp" && $2$5 <= "7.0tcp"' | grep tcp >>
resultTime13
PackLost14=`cat resultTime13 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost14"
echo "Quantidade de pacotes perdidos no TEMPO 6.6 A 7.0 : $PackLost14" >>
rt$1.txt
rm resultTime13

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 7.1 A 7.5"
echo
cat $1 | awk '$2$5 >= "7.1tcp" && $2$5 <= "7.5tcp"' | grep tcp >>
resultTime14
PackLost15=`cat resultTime14 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost15"
echo "Quantidade de pacotes perdidos no TEMPO 7.1 A 7.5 : $PackLost15" >>
rt$1.txt
rm resultTime14

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 7.6 A 8.0"
echo
cat $1 | awk '$2$5 >= "7.6tcp" && $2$5 <= "8.0tcp"' | grep tcp >>
resultTime15
PackLost16=`cat resultTime15 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost16"
echo "Quantidade de pacotes perdidos no TEMPO 7.6 A 8.0 : $PackLost16" >>
rt$1.txt
rm resultTime15

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 8.1 A 8.5"
echo
cat $1 | awk '$2$5 >= "8.1tcp" && $2$5 <= "8.5tcp"' | grep tcp >>
resultTime16
PackLost17=`cat resultTime16 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost17"
echo "Quantidade de pacotes perdidos no TEMPO 8.1 A 8.5 : $PackLost17" >>
rt$1.txt
rm resultTime16

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 8.6 A 9.0"
echo
cat $1 | awk '$2$5 >= "8.6tcp" && $2$5 <= "9.0tcp"' | grep tcp >>
resultTime17
PackLost18=`cat resultTime17 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost18"
```

```
echo "Quantidade de pacotes perdidos no TEMPO 8.6 A 9.0 : $PackLost18" >>
rt$1.txt
rm resultTime17

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 9.1 A 9.5"
echo
cat $1 | awk '$2$5 >= "9.1tcp" && $2$5 <= "9.5tcp"' | grep tcp >>
resultTime18
PackLost19=`cat resultTime18 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost19"
echo "Quantidade de pacotes perdidos no TEMPO 9.1 A 9.5: $PackLost19" >>
rt$1.txt
rm resultTime18

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - TEMPO 9.6 A 10.0"
echo
cat $1 | awk '$2$5 >= "9.6tcp" && $2$5 <= "10.0tcp"' | grep tcp >>
resultTime19
PackLost20=`cat resultTime19 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLost20"
echo "Quantidade de pacotes perdidos no TEMPO 9.6 A 10.0 : $PackLost20" >>
rt$1.txt
rm resultTime19

echo
echo "SCRIPT PARA PERDA DE DADOS TCP - PERDA TOTAL DE DADOS (FTP E CBR)"
echo
cat $1 | awk '$2 >= "0.1" && $2 <= "10.0"' >> resultTime20
PackLostTotal=`cat resultTime20 | grep d | wc -l`
echo "Quantidade de pacotes perdidos: $PackLostTotal"
echo "Quantidade de pacotes perdidos na PERDA TOTAL DE DADOS (FTP E CBR) :
$PackLostTotal" >> rt$1.txt
rm resultTime20

SOMAPackLost=$(($PackLost1+$PackLost2+$PackLost3+$PackLost4+$PackLost5+$Pac
kLost6+$PackLost7+$PackLost8+$PackLost9+$PackLost10+$PackLost11+$PackLost12
+$PackLost13+$PackLost14+$PackLost15+$PackLost16+$PackLost17+$PackLost18+$P
ackLost19+$PackLost20))
echo "Soma de pacotes TCP Perdidos : $SOMAPackLost" >> rt$1.txt
cat rt$1.txt
```