ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Análise Comparativa dos Algoritmos de Controle de Congestionamento do TCP

Trabalho de Conclusão de Curso

Engenharia da Computação

Nome do Aluno: Juliana Lima Cavalcanti
Orientador: Prof. Adriano Lorena Inácio de Oliveira

Recife, Julho de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Análise Comparativa dos Algoritmos de Controle de Congestionamento do TCP

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Nome do Aluno: Juliana Lima Cavalcanti
Orientador: Prof. Adriano Lorena Inácio de Oliveira

Recife, Julho de 2005

UNIVERSIDADE
DE PERNAMBUCO

Juliana Lima Cavalcanti

Análise Comparativa dos Algoritmos de Controle de Congestionamento do TCP

Resumo

O mecanismo de controle de congestionamento do TCP é vital para o bom desempenho de redes como a Internet, pois força os transmissores a reduzir a taxa de transmissão sempre que algum congestionamento é detectado. Os algoritmos mais comuns do TCP utilizam a perda de pacotes na rede como indicação de congestionamento; a maioria das implementações reage à presença de congestionamento. Outros como o TCP Vegas, fazem um controle de forma pró-ativa, tentando evitar o congestionamento. Este trabalho provê um estudo comparativo entre algumas implementações do TCP com relação ao controle de congestionamento. Essas implementações englobam o TCP Tahoe, Reno, New Renoe Vegas. A simulação foi a técnica escolhida para o estudo do desempenho dos protocolos pois possui várias vantagens em relação a outras técnicas existentes, principalmente devido ao baixo custo de implementação dos cenários de redes utilizados. Através da técnica de simulação, realizada utilizando-se o simulador NS (*Network Simulator*), é constatada que por ser uma das primeiras implementações e possuir um mecanismo muito simples de reagir ao congestionamento, o TCP Tahoe, não reage bem a presença de congestionamento obtendo um baixo desempenho quando utilizado. As implementações Reno e New Reno, por agregarem outras funcionalidades para reagir ao congestionamento, possuem um desempenho superior ao TCP Tahoe, aproveitando, assim, melhor a banda disponível. Já o TCP Vegas, por possuir mecanismo de controle de congestionamento totalmente diferente das demais implementações abordadas neste estudo, utilizando um melhor esquema para estimar a largura de banda disponível, possui um desempenho superior às outras implementações. Porém em situações em que estão presentes conexões TCP Reno e conexões TCP Vegas, o Vegas pode ter baixa utilização em relação ao *buffers* disponíveis nos roteadores devido ao agressivo mecanismo de reagir ao congestionamento do TCP Reno.

Abstract

The TCP congestion control mechanisms are vital for the good performance of nets like the Internet since they force the transmitters to reduce the amount of data transmitted whenever a congestion detected. The most common TCP algorithms use the loss of packets as signal of congestion. The vast majority of the implementations reacts to the presence of a congestion. Others, as the TCP Vegas, perform a pro-active check trying to prevent the congestion. The purpose of this work is to provide a comparative study over some TCP implementations related to congestion control. These implementations include TCP Tahoe, Reno, New Reno and Vegas. The technique chosen to study protocols performances is simulation since it provides several advantages if compared to other existing techniques and this is mainly because of the scenery implementation low costs. With the simulation technique, using the simulator NS (Network Simulator), we find that, by being one of the first implementations to employ a mechanism that reacts in a very simple way to a congestion, the TCP Tahoe does not react too well to the presence of a congestion providing low performances while in use. The implementations Reno and New Reno, by aggregating other features to react to a congestion, responded with superior performance compared to the TCP Tahoe thus better disposing of the available bandwidth. The TCP Vegas, having a completely different congestion control mechanism compared to the other implementations taken into account in this study, using a better structure to estimate the available bandwidth, presents performances better than the other implementations. However, in situations where both the TCP Reno and the TCP Vegas are present, the TCP Vegas can suffer in reference to the use of the available *buffers* in the routers due to the TCP Reno's aggressive congestion reaction mechanism.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

ACK – *Acknowledgement*
RTT – *round-trip-time*
CWND – *congestion window*
RWND – *Reception window*
QoS – *Quality of service*

Agradecimentos

- A Deus.
- A minha família e ao meu namorado, Jorge Tasso, pelo apoio que me deram e têm me dado em toda a caminhada, que não foi fácil.
- A Fernando Aires, que foi meu amigo de estágio e faculdade e com ele muito aprendi. Agradeço pela força que me deu nas horas que tive dúvidas sobre NS ou em qualquer momento, pois ele sempre estava lá disposto a me ajudar.
- Aos professores do Departamento de Sistemas Computacionais que sempre fizeram o possível e o impossível para que o nosso curso obtivesse reconhecimento dentro da Escola Politécnica e fora dela.
- A Adriano Lorena por ter me orientado nesse trabalho e ter passado seus conhecimentos acadêmicos e profissionais objetivamente.
- Aos meus amigos de turma, em especial, Adélia Barros, Carolina Baldisserotto, Maíra Paschoalino, Polyana Lima e Bruna Bunzen, que sempre suportaram meus momentos de estresse durante toda a graduação e durante a produção deste trabalho.

Introdução

O TCP (*Transmission Control Protocol*) [1] é um protocolo da camada de transporte da arquitetura TCP/IP, o qual é o mais utilizado na Internet e que tem como principal objetivo a entrega confiável de dados. Além desses, o TCP inclui um mecanismo de controle de congestionamento que visa a melhor utilização das conexões da Internet, limitando a capacidade de transmissão de um cliente e/ou servidor, quando a rede está congestionada.

Durante várias décadas, pesquisas em algoritmos de controle de congestionamento [3][11][30][31]têm sido feitas. Esses algoritmos continuam a ser aperfeiçoados de forma a atingir uma melhor média de aproveitamento do uso da rede. O foco dado é essencialmente ao controle do tráfego de dados confiável na Internet. O custo de descarte de pacotes é alto, pois a capacidade de transmissão que é utilizada em cada um dos roteadores anteriores para repassar o pacote até o ponto em que foi descartado acaba sendo desperdiçada. Vários estudos foram feitos em [3] para tentar promover uma reação ao congestionamento e a partir disso começaram a desenvolver várias implementações cada vez mais sofisticadas para controlar esse estado da rede.

Geralmente, o TCP utiliza apenas a perda de pacotes na rede como indicador de congestionamento e responde a essa indicação com a diminuição da taxa de transmissão. Como a capacidade da rede varia constantemente, de acordo com o uso compartilhado dos seus recursos, o protocolo volta a aumentar gradualmente sua taxa de transmissão visando a aproveitar os momentos de maior disponibilidade. Isso ocorre até que ele receba nova indicação de congestionamento e volte a baixar essa taxa.

O congestionamento é um dos problemas mais pesquisados na área de redes de computadores. Ele é um estado atingido pela rede que degrada a qualidade de serviço das conexões estabelecidas e, por isso, os algoritmos de controle de congestionamento [6] nas implementações do TCP tentam minimizar e controlar esse estado da rede. A presença de congestionamento significa que, temporariamente, a carga é maior do que os recursos de uma parte do sistema podem suportar. Sem o controle de congestionamento uma rede pode ficar travada, ou seja, pouco ou nenhum dado é transportado fim a fim.

Inicialmente em [3], Jacobson propôs um mecanismo de controle de congestionamento que em seguida, com a adição da *Fast Retransmit*, deu origem ao TCP Tahoe. Desde então, muitas modificações foram feitas no TCP em relação ao controle de congestionamento e assim deram origem às várias implementações do TCP. A segunda versão foi o TCP Reno que incorporou o algoritmo *Fast Recovery*[9]. A terceira foi uma modificação no algoritmo *Fast Recovery*que deu origem ao TCP New Reno[12].

Vários estudos sobre controle de congestionamento tem sido feitos nos útimos anos para analisar essas implementações. No entanto a maioria deles avaliam, geralmente, poucos aspectos possíveis. Há portanto, uma necessidade de estudar vários aspectos, a fim de obter um estudo abrangente, que permita compreender melhor as implementações do TCP em relação ao controle de congestionamento.

Fall e Floyd em [8] mostraram como algumas implementações do TCP reagem a uma determinada quantidade de perdas de pacotes, visto que a perda de pacotes é um indicativo de

ocorrência de congestionamento para as implementações estudadas nesse artigo. Porém, em [8] só são analisados implementações que utilizam perdas para detectar o congestionamento.

Neste trabalho, assim como em [8], é analisada a quantidade de pacotes perdidos por cada implementação, porém, na análise, foi acresentado a implementação TCP Vegas proposta em [14], a qual possui um mecanismo de controle de congestionamento diferente, não utiliza a perda de pacotes para detectar o congestionamento, ao invés disso, esta versão faz um monitoramento da banda disponível para estimar a capacidade da rede. Dessa forma, com o intuito de obter uma avaliação comparativa mais criteriosa, torna-se conveniente mostrar também, como o TCP Vegas se comporta em períodos de congestionamento, para podermos comparar duas maneiras que o TCP possui para reagir ao congestionamento e não apenas uma.

Este trabalho tem como objetivo realizar um estudo comparativo entre as diversas implementações dos algoritmos de controle de congestionamento do TCP, limitando-se à análise das versões Tahoe, Reno, New Reno e Vegas, pois as mesmas abrangem a implementação inicial, Tahoe, e as outras são modificações feitas para se melhorar o controle de congestionamento nas redes. O TCP Tahoe foi escolhido por ser a primeira versão do TCP a realizar o controle de congestionamento. O TCP Reno e New Reno são estudados por serem as mais utilizadas na Internet. Por fim, o TCP Vegas foi escolhido por ter um mecanismo de congestionamento diferenciados das outras implementações.

Para analisar o desempenho das diversas implementações do TCP foram criados cenários de redes simples e a análise foi feita através de simulação. As simulações são realizadas com tráfego FTP (*File Transfer Protocol*) e tráfego CBR (*Contant bit rate*) para estudar a utilização do canal, o *throughput*, as perdas de pacotes e a variação no tempo de entrega dos pacotes (*jitter*). O protoclo UDP (*User Datagram Protocol*) foi utilizado para fins de comparção, porque este permite a utilização de aplicações que geram taxas de transmissão constantes(CBR) como é analisado no parâmetro *jitter* com a simulação de tráfego multimídia.

O NS (*Network Simulator*) [22] foi o simulador utilizado neste trabalho, pois ele tem demonstrado ser de extrema importância no que se refere a área de redes. Essa é uma ferramenta reconhecida internacionalmente e amplamente utilizada por ser de domínio público. Seu público alvo é vasto incluindo estudantes que podem efetuar análises através de programação em Tcl e C++. O NS possui outras ferramentas agregadas, como o NAM (*Netwokr Animator*), utilizada para uma melhor visualização da dinâmica das simulações.

As simulações são feitas tanto em ambiente homogêneos; onde só existem conexões de uma mesma versão, como em ambientes heterogêneos; onde existem conexão de mais de uma versão do TCP. Neste trabalho apenas coexistem no máximo 2 conexões compartilhando um canal.

O controle de congestionamento também é realizado em outras camadas da arquitetura TCP/IP [7], como a camada de rede e a camada de enlace, por exemplo, porém, neste trabalho, será dada ênfase ao controle de congestionamento realizado pelo TCP, ou seja, pela camada de transporte.

No Capítulo 1 são abordados os aspectos mais importantes do funcionamento do protocolo TCP, apresentando suas características gerais. Essa camada é essencial para o funcionamento da rede fornecendo às aplicações um serviço confiável e orientado a conexão independente da tecnologia de rede e do hardware. O controle de congestionamento promovido pelo TCP evita que as conexões TCP entre os sistemas finais preencham com excessiva quantidade de tráfego os enlaces e os comutadores entre esses sistemas finais. Tendo em vista esse problema, foram desenvolvidas diversas implementações do protocolo TCP para controle de congestionamento. Nesse Capítulo, também foram identificados os quatro algoritmos [5] que são utilizados para compreender como o TCP trata o controle de congestionamento [6]. As diferentes implementações do TCP são baseadas na presença ou ausência desses algoritmos e nas formas

particulares de implementá-los.

No Capítulo 2 são mostrados conceitos sobre simulação [18], o porquê da escolha da técnica de simulação de redes, e as vantagens e desvantagens de se utilizar simulação como ferramenta no estudo em questão. Além disso, o Capítulo 2 aborda também as características e as vantagens de se utilizar o simulador *Network Simulator*, simulador de propósito específico para experimentos com ambientes de redes computadores, o qual foi escolhido para o estudo deste trabalho.

No Capítulo 3 são descritos os cenários de redes que foram utilizados nos experimentos, assim como as variáveis utilizadas que foram necessárias para analisar cada implementação do TCP. As variáveis analisadas são o *jitter*, perdas de pacotes de cada implementação, o *throughput* e a utilização dos canais. Os resultados das simulações realizadas foram analisados nesse Capítulo, a fim de comparar o desempenho dos protocolos.

Por fim, na conclusão, é mostrada a aplicabilidade das implementações estudadas, ou seja, em que tipos de redes as implementações são mais viáveis, a fim de que obtenham um melhor desempenho.

Capítulo 1

Protocolo TCP

Este Capítulo mostra as diversas funcionalidades do protocolo TCP (*Transmission Control Protocol*) e como ele está posicionado na arquitetura TCP/IP.

1.1 Características Gerais do Protocolo TCP

O TCP está implementado na camada de transporte da arquitetura TCP/IP. Os protocolos da camada de transporte são implementados nos sistemas finais, não estando presente nos roteadores da rede. A função da camada de transporte é aceitar os dados vindos da camada de aplicação, dividí-los em segmentos e garantir que todos os segmentos cheguem corretamente na outra extremidade. O modelo TCP/IP pode ser visto na Figura 1.

A camada de transporte é uma camada muito importante na pilha de protocolos do TCP/IP [7], pois fornece serviço de comunicação confiável e eficiente de dados de uma máquina origem para uma máquina de destino, independente da rede e do meio físico existentes. Problemas podem ocorrer na rede de comunicação e pacotes podem ser perdidos durante o seu trajeto, dessa maneira os sistemas finais não ficariam sabendo dessa perda se não existisse a camada de transporte, que é fim-a-fim e exige que uma confirmação seja enviada para o emissor. Para obter melhor desempenho na transferência confiável de dados, o TCP utiliza-se do mecanismo de janela deslizante, em que o emissor pode enviar uma quantidade de pacotes sem receber um ACK (*acknowledgment*), ou seja, uma confirmação, e também utiliza-se de ACK's cumulativos em que um único ACK pode reconhecer vários pacotes que foram enviados anteriormente. Esses mecanismos são explicados ao longo desse Capítulo. Portanto um protocolo de transporte pode oferecer serviço de transferência confiável a uma aplicação mesmo quando o protocolo da camada de rede não é confiável, isto é, mesmo quando o protocolo de rede perde ou duplica pacotes.

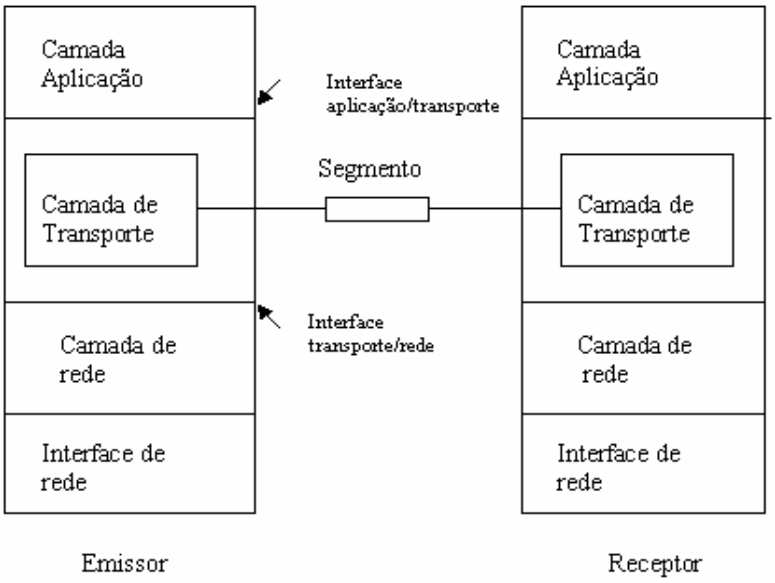

Figura 1. Modelo de referência TCP/IP.

O TCP é um protocolo que foi, inicialmente, definido na RFC 793 [1]. Esse protocolo oferece facilidades às aplicações que o utilizam como protocolo de transporte. Essas aplicações geralmente requerem entrega confiável de dados no destino. O TCP, além da confiabilidade na entrega, também é responsável por controle de fluxo, controle de congestionamento e controle de erros. Para garantir essas funcionalidades, o TCP utiliza retransmissões de pacotes que não foram entregues com sucesso, temporizadores para cada pacote enviado e campos no seu cabeçalho como número de seqüência e número de reconhecimento. O TCP define o formato e a ordem das mensagens trocadas entre dois sistemas finais, bem como as ações realizadas na emissão e no recebimento de um pacote. Muitas vezes, uma aplicação necessita ter certeza de que todos os dados que foram passados para o TCP foram transmitidos para o destinatário. Duas típicas aplicações que utilizam o TCP como protocolo de transporte são o FTP (*File Transfer Protocol*), bastante utilizado pelos usuários da Internet para tranferência de arquivos, e o TELNET, utilizado para acesso remoto.

Os dados trocados pela camada de transporte que utilizam o TCP são chamados de segmentos. Um segmento consiste de um cabeçalho seguido de dados provenientes da aplicação. O cabeçalho do TCP tem o tamanho de 20 bytes e é nele que se encontram os campos que auxiliam na realização de tarefas essenciais do protocolo TCP. O cabeçalho TCP pode ser visto na Figura 2.

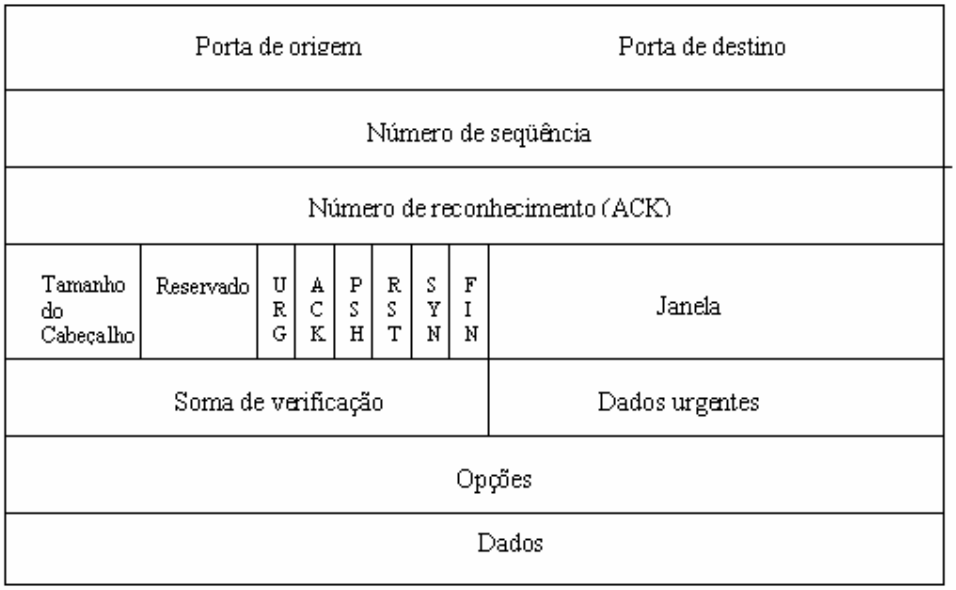

Figura 2. Segmento TCP.

As funções de cada campo do TCP estão listadas abaixo[1,7]:

- Porta de origem: porta de origem da aplicação, utilizada no momento da multiplexação e demultiplexação dos dados da aplicação.
- Porta de destino: porta de destino da aplicação, utilizada também para multiplexação e demultiplexação dos dados da aplicação.
- Número de seqüência*:* campo de 32 bits, destinado ao número de seqüência do segmento. Campo utilizado para permitir ao receptor identificar unicamente um pacote e saber se ele é uma re-transmissão ou não.
- Número de reconhecimento*:* campo que indica o número do próximo byte aguardado pelo emissor.
- Tamanho do cabeçalho TCP*:* informa o tamanho do cabeçalho.
- Reservado*:* 6 bits reservados para uso futuro.
- *Flags:* bits de controle
  URG: indica a presença de dados urgentes.
  ACK: indica que o segmento é um segmento de confirmação.
  PSH: indica que os dados precisam ser entregues imediatamente à aplicação.
  RST: reinicia uma conexão que não está funcionando corretamente.
  SYN: indica a solicitação de estabelecimento de conexão.
  FIN: indica solicitação de encerramento de conexão.
- Tamanho da janela: campo de 16 bits que indica o tamanho da janela, ou seja, quantos bytes podem ser enviados sem receber uma confirmação.
- Soma de verificação (*checksum*)*:* utilizado para detecção de erros no segmento.
- Dados Urgentes (*Urgent point):* indica a posição do segmento onde os dados urgentes terminam.
- Opções*:* campo para fornecer recursos que não foram previsto no cabeçalho inicial.
- Dados*:* campo que contém os dados que são procedentes da aplicação.

O TCP também promove multiplexação e demultiplexação de aplicações. Quando várias

aplicações estão rodando numa máquina, o TCP receptor, através da multiplexação, sabe a que aplicação tem que ser entregue um segmento recebido da camada de rede. Dessa forma também, o TCP identifica na máquina de origem os dados vindos de diferentes processos de aplicação através da demultiplexação. Para promover a multiplexação e a demultiplexação o TCP utiliza dois campos especiais no cabeçalho do segmento, o campo número de porta de origem e número da porta de destino. Esses campos identificam exclusivamente um processo de aplicação rodando na máquina de destino. Dessa forma, o TCP não tem problemas de entregar um segmento a uma aplicação errada nem misturar dados de aplicações diferentes.

A cada byte que é transmitido o TCP atribui um número de seqüência. A máquina de destino verifica esse número para determinar se o pacote recebido é ou não uma re-transmissão. Se o pacote contém um número de seqüência já recebido, esse pacote é uma re-transmissão. Para cada byte emitido é esperado uma confirmação do receptor, ou seja, um ACK. O campo de a*cknowledgement number* (ACK) é igual ao próximo número de seqüência que o receptor espera receber. Como cada pacote enviado é disparado um temporizador, se um ACK não é recebido dentro desse intervalo de tempo o segmento é re-transmitido. Como os dados são transferidos em segmentos, somente o primeiro número de bytes de dados é enviado ao destino novamente. É através do número de seqüência que o TCP receptor reorganiza os segmentos quando um segmento chega fora de ordem, já que um segmento que foi enviado primeiro pode chegar depois devido as várias rotas. Assim, o TCP garante à aplicação que todos os dados cheguem no receptor.

Como o TCP é um protocolo de transporte confiável, ele necessita que todo pacote que foi enviado ao receptor receba uma confirmação (ACK). Porém, se a cada pacote enviado o emissor tivesse que esperar uma confirmação, a banda disponível não seria utilizada da melhor maneira. Se ocorresse uma perda e um ACK não for recebido num intervalo de tempo, ele é re-transmitido, mas todo a largura de banda durante esse tempo da espera não estaria sendo utilizada. Apesar desse mecanismo garantir confiabilidade na medida em que todo pacote enviado é recebido, mesmo que ocorresse re-transmissão, ele somente usa uma parte da banda de rede disponível.

Para que a banda seja utilizada da melhor forma possível, o TCP utiliza o mecanismo de janela deslizante, onde a janela é o número de segmentos que podem ser enviados, mas não reconhecidos. O TCP fornece o serviço de controle de fluxo fazendo com que remetente mantenha uma variável chamada de janela de recepção. Em uma conexão *full-duplex*, o remetente de cada lado mantém uma janela de recepção diferente e, como ela é dinâmica, seu valor muda durante uma conexão. A janela inicial do TCP é determinada de forma dinâmica e o impacto na rede produzido pela variação no seu tamanho pode ser visto em [2]. O mecanismo de janela deslizante funciona da seguinte maneira:

- O emissor pode enviar vários pacotes dentro da janela sem receber um ACK, mas tem que iniciar um temporizador para cada um dos pacotes.
- O receptor tem que reconhecer cada pacote recebido, indicando o número de seqüência do pacote que espera receber.
- O emissor desloca a janela para cada ACK recebido.

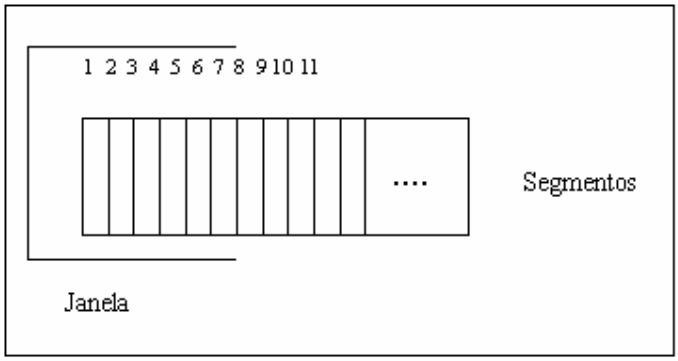

Figura 3. Mecanismo de janela deslizante.

Na Figura 3 o emissor pode transmitir pacotes de 1 a 8 sem esperar por um ACK.

Esse mecanismo de janela deslizante garante:

- Transmissão confiável.
- Uso da banda disponível otimizada, pois aumenta o *throughput*, já que o *throughput* é o número de bytes de dados transmitido por segundo durante um intervalo de tempo.
- Controle de fluxo já que o receptor pode informar a quantidade de memória disponível e o tamanho da janela de comunicação.

O TCP é um protocolo orientado à conexão, onde as máquinas que precisam se comunicar inicialmente estabelecem uma conexão, utilizam a conexão para enviar dados e em seguida, liberam a conexão. Para estabelecer essa conexão, eles trocam pacotes de controle antes de enviar pacotes de dados. Ao estabelecer uma conexão, o TCP está alocando recursos e variáveis de estados nos sistemas finais que serão necessários para a troca de mensagens confiáveis, garantindo que todos os dados sejam entregues sem erros e na ordem correta. A conexão TCP fornece transferência de dados *full-duplex* e é sempre ponto-a-ponto, ou seja, entre duas máquinas apenas.

Para estabelecer uma conexão, o TCP utiliza uma apresentação de três vias conhecida como *3-way handshake.* O estabelecimento da conexão pode ser visto na Figura 4.

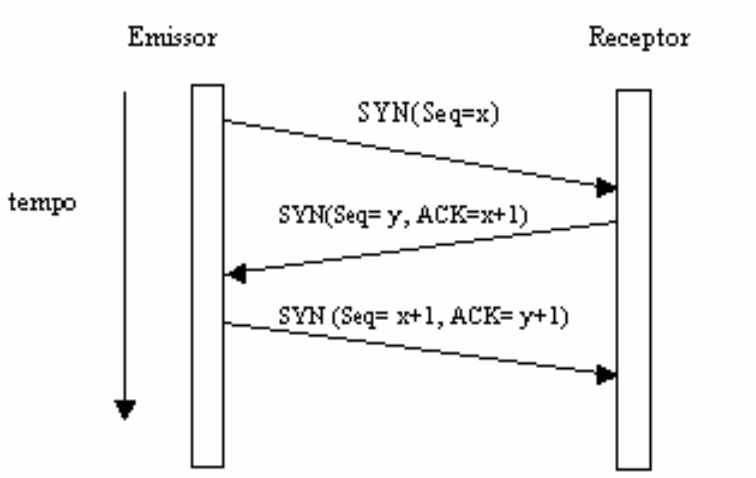

Figura 4. *Threeway handshake.*

No estabelecimento da conexão, a máquina que deseja estabelecer a conexão, envia um segmento com o campo SYN do cabeçalho ativado e com o campo ACK desativado. Apesar de não conter dados, esse segmento é enviado com o campo de número de seqüência ativado para que o mesmo seja confirmado sem ambigüidade. Ao chegar no receptor, se a conexão for aceita, o mesmo envia um segmento concordando com o estabelecimento da conexão. Esse segmento possui o campo SYN do cabeçalho ativado, um número de seqüência para evitar ambigüidade na confirmação e o campo ACK ativado com um número do próximo byte que o receptor espera receber. Dessa maneira, o TCP receptor está indicando que recebeu o segmento anterior corretamente e informando o próximo byte que espera receber. Finalmente o emissor, como resposta ao receptor, envia um segmento SYN com o campo ACK ativado. Esse segmento diferente dos dois primeiros, já pode conter dados. Quando o receptor receber esse segmento, a conexão é considerada estabelecida pelo emissor, o solicitante do pedido. A partir desse momento, quaisquer das duas máquinas pode iniciar o envio de dados através conexão, não importando quem solicitou a conexão, já que a conexão TCP é *full-duplex*.

Quando uma conexão é estabelecida é preciso gerenciá-la para que um transmissor rápido não sobrecarregue um receptor que não tem a capacidade de receber e processar os dados na velocidade enviada pelo transmissor. Para isso, o protocolo TCP realiza um controle de fluxo entre os sistemas finais. O controle é realizado através de um campo no cabeçalho do segmento TCP que informa a quantidade de *buffer* disponível na máquina de destino, fazendo com que o transmissor saiba a capacidade do receptor e não envie mais dados do que ele possa receber.

O encerramento de uma conexão pode ser solicitado por qualquer das duas máquinas da conexão. Em qualquer momento, a máquina que desejar encerrar a conexão envia um segmento com o campo FIN ativado, indicando que não tem mais dados para ser enviado. A partir desse momento todos os recursos da conexão foram liberados. O outro lado da conexão, porém, pode continuar a enviar dados. A conexão é encerrada somente quando os dois lados da conexão forem encerrados. O encerramento da conexão pode ser visto na Figura 5.

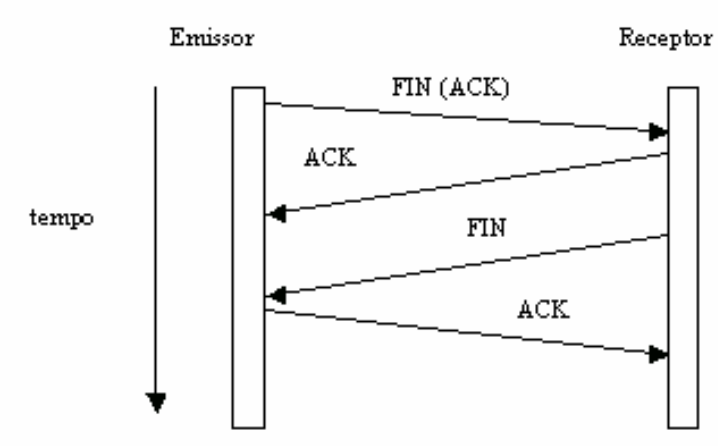

Figura 5. Encerramento de conexão.

1.2 Mecanismos de Controle de Congestionamento

Um problema que surge numa rede como a Internet é o congestionamento, que ocorre quando a memória dos roteadores fica cheia, ou seja, com uma quantidade além da sua capacidade e, dessa forma, os seus *buffers* não têm capacidade para armazenar os pacotes que chegam até eles para

posterior processamento. Portanto, esse estado da rede pode levar à perda de pacotes e também causar longos atrasos de pacotes durante o seu trajeto até o destino. Uma situação em que uma rede pode entrar em estado de congestionamento é quando um roteador recebe vários fluxos de redes diferentes e esse não possui uma quantidade de memória suficiente para armazenar os pacotes vindos dessas várias redes, levando a uma perda de pacotes por falta de memória disponível. Essa perda de pacotes, para algumas implementações, indica que a rede está congestionada e a partir daí tenta diminuir a taxa de envio de pacotes. Nessa situação, quando vários pacotes são perdidos eles precisam ser re-transmitidos pelos remetentes, já que não chegaram no destino, e a largura de banda utilizada na primeira transmissão, antes de chegar a esse roteador sobrecarregado, é desperdiçada.

Para identificar a presença de congestionamento, o TCP entre outros mecanismos, utiliza-se de estouros de temporizadores e reconhecimentos duplicados. O estouro de temporizadores ocorre quando o reconhecimento de um pacote que foi enviado não chega ao receptor em um tempo pré-determinado. A Figura 6 ilustra esse mecanismo.

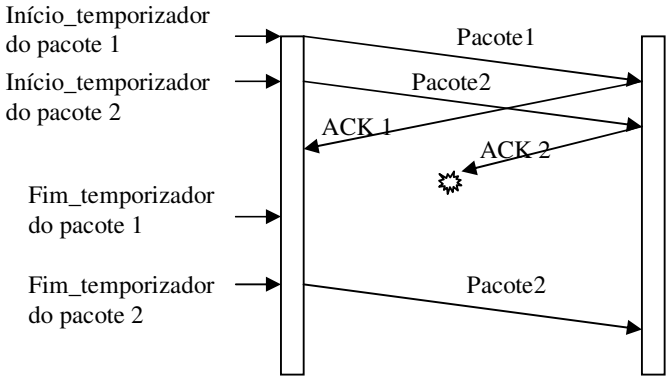

Figura 6. Estouro de temporizador para um pacote.

Nessa Figura pode-se ver que a cada pacote enviado é acionado um temporizador. Para o pacote 1, o reconhecimento chegou antes do fim do temporizador do seu pacote, não necessitando ser re-enviado. Porém, para o pacote 2, o seu reconhecimento é perdido no caminho. Portanto, se o reconhecimento do pacote não chegar ao emissor até que o tempo do seu temporizador estoure, isso indica que o pacote não chegou ao destino ou o seu reconhecimento (ACK) foi perdido durante a transmissão. A perda de um ACK pode ser vista no pacote 2 da Figura 6, onde o ACK é perdido durante o caminho de volta e o temporizador do pacote 2 estoura, com isso o pacote 2 é re-enviado. Já os reconhecimentos duplicados ocorrem quando o emissor receber três ACK's para um mesmo segmento, isso indica que o segmento anterior ao segmento que está sendo reconhecido foi perdido e o receptor informa isso mandando confirmações para o último pacote recebido. Os mecanismos de reconhecimentos duplicados podem ser vistos na Figura 7.

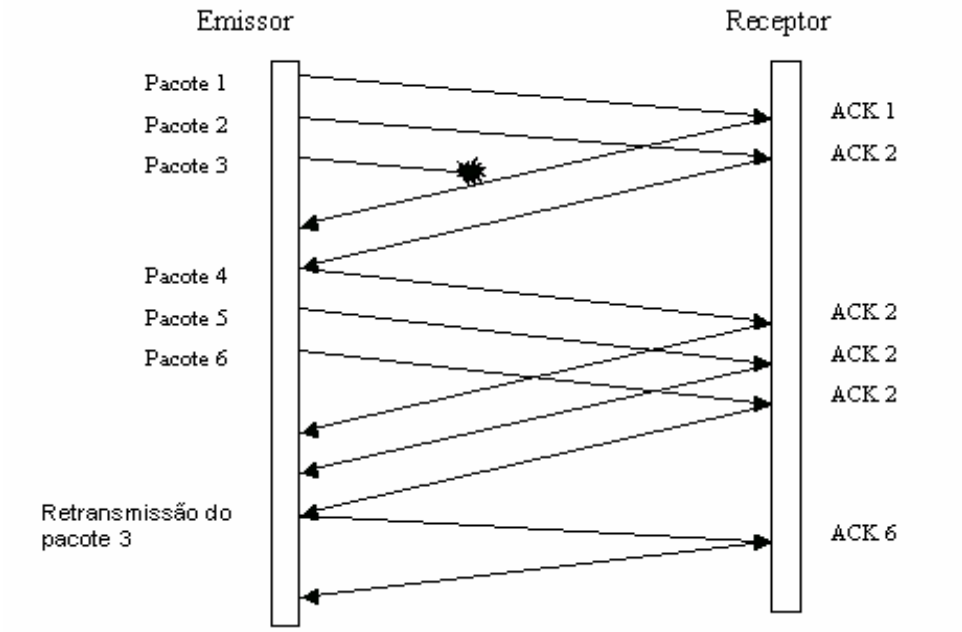

Figura 7. Reconhecimentos duplicados.

Nessa Figura, o pacote 3 é perdido durante a sua transmissão. O emissor sabe que houve uma perda do pacote 3, quando o mesmo recebe confirmações duplicadas, ou seja, o emissor continua a receber ACK para o pacote 2, último pacote que chegou corretamente ao emissor.

A perda de segmentos apontada pelo estouro do temporizador ou por reconhecimentos duplicados é vista como uma indicação de que a rede está congestionada. Erros provocados por meios físicos são muito improváveis, pois a maioria dos meios físicos dos *backbones* da Internet é de fibra óptica e a taxa de erros nesses meios são baixíssimas ou quase nulas.

Para a determinação do *timeout* e, conseqüentemente, das retransmissões, é fundamental fazer uma estimativa do tempo total de transmissão de ida e volta, ou seja, uma estimativa do RTT – *round trip time* [3] em uma determinada conexão. A estimativa do RTT muda com o tempo, devido a mudanças de rotas e do tráfego, por isso o TCP precisa monitorar essas mudanças e modificar o tempo de *timeout* apropriadamente, para que não ocorra estouro de temporizadores indevidamente. Para realizar esse monitoramento, o TCP calcula constantemente o RTT entre o envio de um segmento e o recebimento da confirmação desse segmento. Uma média é calculada sobre vários RTT para ser usada como um valor de *timeout* para o próximo segmento que for enviado. Isto é uma reação muito importante, porque atrasos podem acontecer na rede por muitos motivos, como a saturação de *buffers* de roteadores intermediários da rede.

Algumas observações podem ser feitas com relação ao cálculo da estimativa de RTT. Quando ocorre um *timeout* e um pacote é re-transmitido não se pode atualizar a estimativa do RTT quando uma confirmação de um segmento re-transmitido chega, porque não se sabe a que segmento a confirmação corresponde: se ao primeiro segmento ou ao segmento que foi re-transmitido. O primeiro pacote pode ter sofrido um atraso na rede e não foi descartado ou até mesmo o ACK da primeira transmissão foi retido em algum lugar da rede. Maiores detalhes podem ser vistos em [4].

Numa rede como a Internet, formada por meios físicos confiáveis e que são utilizados por múltiplas conexões, um dos motivos principais de perda de pacotes e conseqüentemente uma diminuição na taxa efetiva de transmissão é o congestionamento, que afeta todas as conexões que compartilham um dado meio físico. O TCP promove um controle de congestionamento fim-a-

fim, onde a presença de congestionamento é descoberta pelos sistemas finais com base na observação do comportamento da rede, como por exemplo, perda de pacotes e atrasos.

Os algoritmos de controle de congestionamento do TCP evitam o problema do congestionamento, forçando os sistemas finais a diminuir a velocidade com que enviam pacotes para a rede durante os períodos em que a rede está congestionada.

Como o protocolo TCP utiliza confirmações e re-transmissões para garantir a confiabilidade na entrega dos segmentos, os sistemas finais são alertados sobre congestionamento quando deixam de receber as confirmações dos pacotes enviados. Tais sistemas reagem a isso, re-transmitindo os pacotes dos quais não receberam a confirmação.

A todo pacote enviado para a rede é associado um temporizador no emissor. Caso o emissor não receba uma confirmação de recebimento do receptor, o temporizador irá estourar e, na maioria das implementações do TCP, isto é interpretado como uma perda de segmento, indicando que a rede está congestionada. Para regular o emissor, os pacotes de confirmação enviados pelo receptor possuem um campo no cabeçalho, com o nome tamanho de janela, que indica a capacidade de seus *buffers* para que o emissor certifique-se da capacidade do receptor e não envie mais pacotes do que eles podem processar. Além disso, é necessário especificar uma janela de congestionamento. Portanto, além da janela de recepção, RWND (*Receiver Window*), que indica a quantidade de segmentos que o receptor pode receber sem confirmação, o transmissor mantém também uma janela de congestionamento,CWND (*Congestion Window*), essa janela impõe uma limitação adicional à quantidade de dados que uma máquina pode enviar numa conexão. A justificativa para a existência da uma janela de congestionameto é que quando uma conexão é estabelecida, deve-se escolher um tamanho de janela de recepção adequado, que pode ser estimado de acordo com o tamanho do *buffer* do receptor. Se o emissor se mantiver sempre dentro desta janela, nunca haverá problemas de sobrecarga dos *buffers* no receptor, mas pode haver sobrecarga por causa do congestionamento da rede. Por isso o emissor pode transmitir dados até o mínimo entre a janela de congestionamento e a janela de recepção.

$$\text{BytesEmTrânsito} \leq \min(\text{CWND}, \text{RWND}).$$

A janela de congestionamento age como um controle de fluxo imposto pelo rede e a janela de recepção é um controle de fluxo imposto pelo receptor.

O controle de congestionamento da maioria das implementações do TCP é realizado por 4 algoritmos [5]: *Slow Start, Congestion Avoidance, Fast Retransmit e Fast Recovery*. Apesar de serem independentes, geralmente são implementados de forma conjunta. A seguir esses algoritmos são detalhados.

1.2.1 *Slow Start* e *Congestion Avoidance*

Através da janela de recepção o TCP faz um controle de fluxo fim-a-fim entre as máquinas que estão se comunicando. Porém é preciso lembrar que durante a comunicação, os pacotes passam pela rede IP, a qual não garante confiabilidade na entrega. Os roteadores que estão na rede IP podem perder pacotes devido a estouros de *buffers*, causando re-transmissões desnecessárias e, conseqüentemente, degradando o desempenho da rede. Para tentar resolver esse problema, o TCP implementa algoritmos para que não sejam injetados na rede mais pacotes do que ela pode processar.

Para fazer um eficiente uso da largura de banda e evitar o congestionamento, o TCP

controla a taxa de transmissão usando um *feedback* da rede.

Nas primeiras implementações do TCP, a perda era percebida através do estouro dos temporizadores, mas até que ocorresse o *timeout*, o TCP continuava a transmitir numa rede possivelmente congestionada, piorando ainda mais a situação e diminuindo o desempenho da rede. A solução para esse problema foi proposta em [3] com a adoção da *Slow Start* e *Congestion Avoidance.*

O algoritmo *Slow Start* (inicialização lenta) é usado pelo TCP emissor para controlar a quantidade de dados que são colocados na rede. Seu mecanismo é baseado na observação de que a taxa em que novos pacotes são colocados na rede deve ser a mesma em que as confirmações são enviadas pelo receptor. Esse algoritmo juntamente com o *Congestion Avoidance,* utiliza duas variáveis além da janela de recepção (RWND) que é mantida pelo TCP emissor. As variáveis são a janela de congestionamento (CWND) e o *threshold.* O *threshold* é utilizado para determinar que algoritmo vai conduzir a transmissão dos dados, se o *Slow Start* ou o *Congestion Avoidance.*

No algoritmo *Slow Start* a CWND inicialmente é pequena e aumenta exponencialmente. Quando um segmento é enviado para a rede, espera-se a confirmação do mesmo, se a confirmação chegar antes do *timeout*, são colocados na rede dois segmentos e novamente espera-se as confirmações, chegando antes do *timeout*, quatro segmentos são enviados para a rede e assim sucessivamente, aumentando, portanto, a CWND exponencialmente. A fase de crescimento exponencial é a fase de inicialização lenta (S*low Start*) [1] [5]. Ela tem esse nome porque começa com uma janela de congestionamento pequena.

O crescimento exponencial acontece até que:

1. Ocorra uma perda de segmento, apontada por um *timeout*.
2. A CWND seja igual a RWND.
3. A CWND estiver abaixo do *threshold.*

Como descrito na RFC 2581 [6], o valor inicial do *threshold* pode ser alto, mas ele diminui em resposta ao congestionamento da rede e segundo Tanembaum [7], o valor inicial do *threshold* é 64KB.

Quando ocorre congestionamento (indicado por *timeout* ou por recebimentos de ACK's duplicados), ao *threshold* é atribuído a metade da janela de congestionamento atual e a janela de congestionamento é re-inicializada para um segmento. O emissor novamente passa a aumentar a janela de congestionamento com rapidez exponencial, usando o algoritmo *Slow Start,* para determinar novamente o que a rede é capaz de suportar. O crescimento exponencial é interrompido quando esse novo *threshold* é alcançado.

O algoritmo de *Slow Start* é utilizado quando

CWND < *threshold.*

Enquanto que o *Congestion Avoidance* é utilizado quando

CWND > *threshold.*

Quando a CWND e o *threshold* são iguais o emissor pode usar ou a *Slow Start* ou *Congestion Avoidance,* dependendo da implementação.

Durante a *Congestion Avoidance* [5] [6], a CWND é incrementada de um segmento a cada RTT, ou seja, o crescimento da CWND é linear e ele segue até que os reconhecimentos continuarem chegando antes do esgotamento de suas temporizações correspondentes.

O algoritmo de *Congestion Avoidance* trata da prevenção de congestionamento, que pode ocorrer quando vários fluxos de dados chegam a um determinado roteador cuja capacidade de envio é menor do que a soma dos fluxos individuais que chegam a ele, sobrecarregando-o e causando perdas de segmentos.

A fase de crescimento linear é denominada de prevenção do congestionamento (*Congestion Avoidance*). A Figura 8 ilustra o comportamento dos algoritmos *Slow Start* e *Congestion Avoidance.*

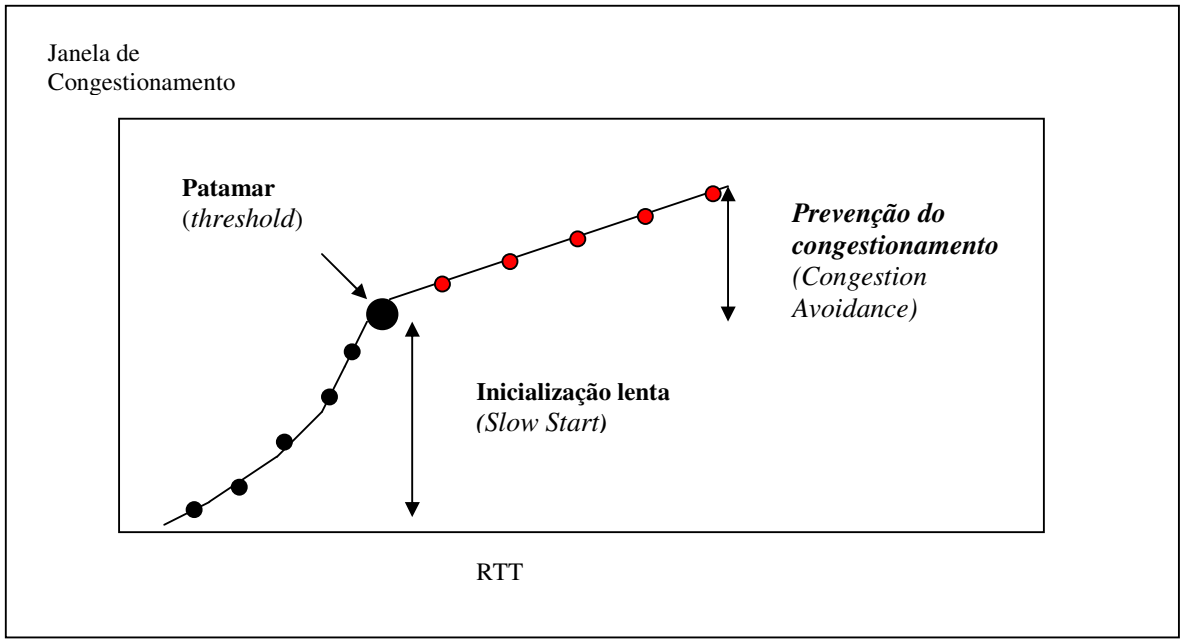

Figura 8. Evolução da janela de congestionamento.

Esses dois algoritmos são muito importante porque no início da transmissão dos pacotes, não se conhece as condições da rede. Lentamente, o TCP vai determinando a capacidade disponível da rede para evitar o congestionamento.

1.2.2 Fast Retransmit e Fast Recovery

O protocolo TCP utiliza o algoritmo *Fast Retransmit* (re-transmissão rápida) [5] [6] para detectar e reparar perdas, baseado no recebimento de confirmações duplicadas.

Quando um *timeout* ocorre, o TCP aciona o algoritmo *Slow Start* e re-inicia a transmissão partindo do segmento perdido. Isso ocasiona uma baixa eficiência do TCP. Muitas vezes os segmentos foram recebidos fora de ordem e estavam no *buffer* do receptor e a re-transmissão do segmento seria desnecessária, pois seriam enviados segmentos que chegaram ao receptor corretamente e que apenas estavam fora de ordem.

Quando o transmissor receber uma confirmação duplicada, ou seja, duas confirmações para um mesmo pacote, isso indica que o receptor recebeu um pacote fora da ordem. O objetivo dessa confirmação é informar ao transmissor que um segmento foi recebido fora de ordem e também informar qual o segmento correto que o receptor está esperando.

O recebimento fora de ordem pode ter sido causado porque, provavelmente, segmentos

foram perdidos e, dessa forma, o receptor recebeu um segmento posterior àquele que ele estava esperando, ou pode ter sido causado pela re-ordenação dos segmentos, pois os mesmo estão trafegando na rede e podem seguir por caminhos diferentes.

Como o TCP não trabalha com confirmações negativas, o receptor reconhece novamente (gerando um segundo ACK) o último segmento de dados que recebeu na ordem. Após o remetente receber três ACK's duplicados para o mesmo segmento ele conclui que o segmento que se seguiu ao segmento reconhecido três vezes foi perdido. Esta perda é atribuída a presença de congestionamento, então o emissor reduz a janela de congestionamento para a metade do seu valor atual, e armazena esse valor na variável *threshold.* Com isso o TCP inicia a fase de *Fast Retransmit,* e começa a transmitir esse segmento perdido antes do *timeout* do segmento perdido acontecer. Se o segmento é re-transmitido antes do temporizador expirar, a fase de *Slow Start* é evitada possibilitando a melhor utilização da capacidade da banda do canal, elevando o desempenho do TCP.

A principal vantagem do algoritmo F*ast Retransmit* é a re-transmissão imediata do segmento considerado perdido, evitando a espera do *timeout* para re-transmissão.

A fase de *Fast Recovery* [5] [6] é utilizada logo após a fase de *Fast Retransmit,* com a finalidade de reduzir o congestionamento da rede evitando a necessidade do acionamento da técnica de *Slow Start.* Nessa fase, ao invés de realizar a *Slow Start,* é iniciado o algoritmo C*ongestion Avoidance.* A razão pela qual este processo é usado, ao invés *Slow Start*, é que o recebimento de ACK's duplicados diz mais do que simplesmente um segmento foi perdido, pois se sabe que o receptor só pode gerar ACK's duplicados quando outro segmento for recebido. Logo, ainda existem dados trafegando entre os dois nós. Talvez o receptor possa informar que ainda existem segmentos intermediários não recebidos e a única maneira do receptor enviar essa informação é ao receber novos segmentos. Portanto, não é aconselhável reduzir o fluxo abruptamente, usando *Slow Start.*

Durante a *Fast Recovery*, a janela de congestionamento aumenta com o recebimento de uma confirmação duplicada. Esse processo é conhecido como inflar a janela, pois com isso o receptor pode continuar recebendo pacotes que tenham sido perdidos e possam corrigir suas falhas. Quando o emissor recebe o primeiro ACK de um segmento enviado durante a fase *Fast Recovery*, a CWND é ajustada para o valor do *threshold* e, novamente, o algoritmo de C*ongestion Avoidance* é acionado. Se o tempo de re-transmissão do segmento perdido ultrapassar o *timeout*, o TCP passa a operar com a técnica *Slow Start.*

1.3 Implementações do TCP

Nesta seção, é realizado um estudo sobre as versões mais utilizadas do TCP, mostrando sua evolução com relação aos algoritmos de controle de congestionamento.

1.3.1 TCP Tahoe

As primeiras implementações do TCP não incluíam qualquer mecanismo de controle de congestionamento. O TCP Tahoe [3] [5] foi a primeira implementação do TCP a incluir controle de congestionamento. Sendo a primeira tentativa de controlar o congestionamento, o TCP Tahoe não possui um mecanismo complicado para garantir um maior desempenho do TCP. Desde então, muitas modificações têm sido feitas no TCP para aumentar seu desempenho. As implementações

posteriores foram agregando outros algoritmos como o *Fast Recovery* e diferenciando o seu modo de operar.

A perda de pacotes e as confirmações duplicadas para um segmento são utilizadas como sinalização para verificar a presença de congestionamento e estimar a largura de banda disponível na rede.

O TCP Tahoe funciona da seguinte maneira. Inicialmente, é utilizado o *Slow Start,* onde a janela de congestionamento (CWND) começa como um segmento e vai aumentando exponencialmente até que ocorra uma perda. Ao ser detectada a perda de pacote, indicada por um *timeout*, a CWND volta a ser de um segmento apenas e começa a crescer novamente. Dessa forma o Tahoe previne que seja enviado para a rede, que se encontra congestionada, mais pacotes do que ela pode processar.

O Tahoe tem duas fases de crescimento da CWND quando do recebimento de um ACK corretamente. Quando a CWND < *threshold*, a janela continua a crescer exponencialmente; caso a CWND > *threshold*, o Tahoe passa a operar no C*ongestion Avoidance,* crescendo linearmente. Esse crescimento é indefinido; ocorre até que uma perda ocorra novamente, ocasião em que a janela cai para um segmento.

Para melhorar seu desempenho, o Tahoe também utiliza o algoritmo *Fast Retransmit* que entra em ação quando chegam confirmações duplicadas para um segmento. Esse acontecimento indica que um segmento foi perdido durante seu trajeto até o destino e precisa ser re-transmitido, sem ter que esperar o seu *timeout* expirar, o que confere um maior utilização do canal e um maior *throughput* da conexão.

Quando uma perda é determinada por um *timeout* ou por confirmações duplicadas, o *threshold* é atualizado da seguinte maneira:

$$threshold = CWND / 2$$

ou seja, o *threshold* cai para a metade do valor da janela de congestionamento no momento em que ocorreu a perda, evitando assim que a janela cresça indefinidamente na próxima fase de *Slow Start*. O TCP Tahoe não utiliza o algoritmo *Fast Recovery.* O avanço da janela de congestionamento no TCP Tahoe é mostrado na Figura 9.

O ponto fraco da implementação Tahoe ocorre quando há muitos segmentos perdidos. Isso aciona o algoritmo de *Slow Start* diversas vezes, ocasionando uma baixa utilização da banda fornecida pela rede.

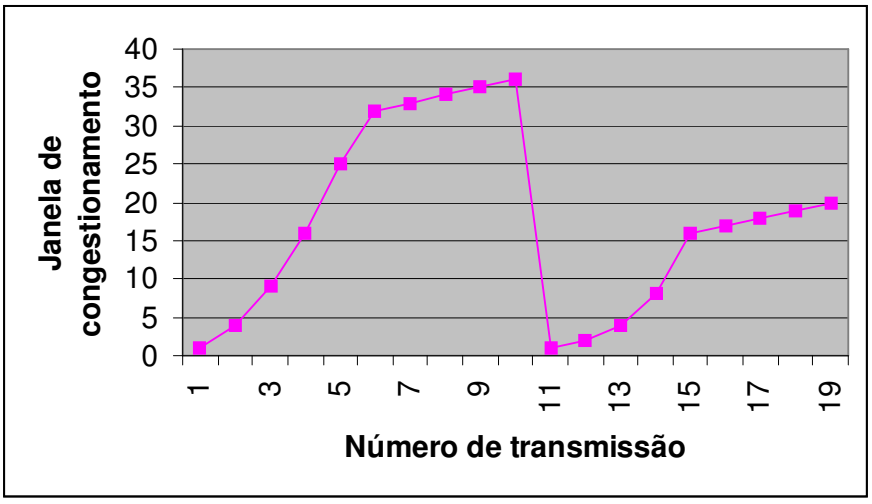

Figura 9. TCP Tahoe.

1.3.2 TCP Reno

O TCP Reno [9] foi idealizado com o intuito de melhorar o desempenho do TCP Tahoe, já que o Tahoe não responde bem a perdas de múltiplos pacotes, diminuindo a CWND para um segmento toda vez que ocorre uma perda. Assim como o Tahoe, o Reno também utiliza a perda de pacotes para estimar a disponibilidade de banda da rede e utiliza a *Fast Retransmit,* porém a fase subseqüente a esta é feita de modo diferente.

O Reno é incrementado com o algoritmo F*ast Recovery* [5] [9] que é acionada após a fase de *Fast Retransmit.* Quando há uma perda ou a chegada de três ACK's duplicados, o TCP Reno observa que a rede está congestionada, retransmite o pacote perdido e armazena em *threshold* a metade da janela de congestionamento no momento da perda.

$$threshold = CWND / 2$$

Esse mecanismo é chamado de *additive increase and multiplicative decrease* [10] [11], que mostra que esse algoritmo leva a uma justa alocação da banda.

Na fase *Fast Recovery,* o Reno aumenta a janela de congestionamento de um segmento cada vez que um ACK duplicado chega ao emissor. O Reno não recorre ao algoritmo *Slow Start* como acontece no TCP Tahoe (esse mecanismo é conhecido como inflar a janela). O comportamento do TCP Reno é mostrado a Figura 10.

Assim que uma confirmação parcial chega ao emissor, ou seja, a confirmação do segmento re-transmitido e não de todos os segmentos que foram transmitidos na fase *Fast Recovery*, a janela de congestionamento é desinflada e retorna ao valor do *threshold* que foi armazenado no momento da perda. A chegada do ACK para segmento perdido é vista pelo TCP Reno como uma indicação de fim de congestionamento, desinflando a janela. A partir daí, a CWND começa a crescer de acordo com o algoritmo *Congestion Avoidance,* linearmente.

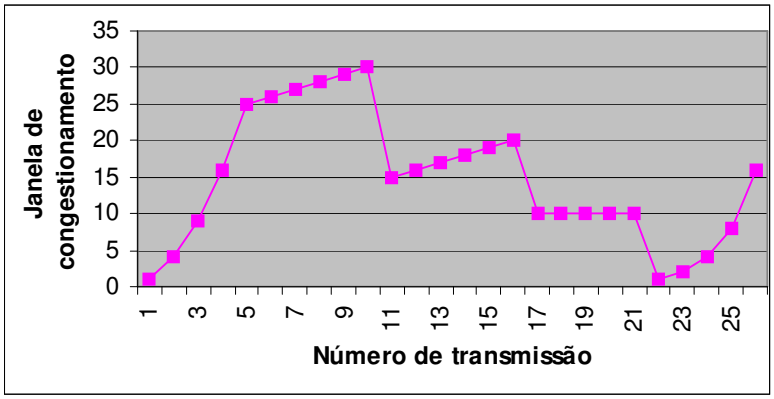

Figura 10. TCP Reno.

Essa implementação proporciona um aumento no desempenho do TCP, prevenindo o esvaziamento do canal após a fase *Fast Retransmit.* O TCP Reno provê um aumento do desempenho do TCP Tahoe quando um único pacote é perdido numa janela, mas não responde tão bem quando múltiplos pacotes são perdidos numa mesma janela de congestionamento.

1.3.3 TCP New Reno

O TCP New Reno [12] é uma otimização do TCP Reno, para o caso onde ocorrem múltiplas perdas de pacotes em uma janela de transmissão. Essa variante do Reno utiliza o algoritmo *Fast Recovery* de forma diferenciada, eliminando a necessidade que o Reno possui de esperar por um *timeout* no caso de múltiplas perdas.

A mudança no comportamento do emissor ocorre quando um ACK parcial chega. A chegada do reconhecimento parcial [12] (parte de todos os segmentos enviados de uma CWND) não faz o New Reno sair da fase *Fast Recover;* ele só abandonará essa fase quando receber um ACK que confirme todos os pacotes que foram enviados desde o início desta fase. Quando ocorrem múltiplas perdas de segmentos o receptor precisa reconhecer todos os segmentos enviados na *Fast Recovery* até a próxima perda. Ou seja, os ACK's parciais não levam ao TCP New Reno a sair da fase *Fast Recovery,* pelo contrário, esses ACK's são tratados como uma indicação de que o segmento seguinte ao reconhecido dentro da seqüência de segmentos enviados na CWND foi perdido e deve ser re-transmitido.

Quando múltiplos segmentos são perdidos em um CWND o New Reno pode recuperar os dados perdidos sem ter que esperar o *timeout* expirar, retransmitindo um segmento a cada RTT até que todos os segmentos perdidos da CWND sejam enviados novamente.

Ao receber as confirmações de todos os segmentos o emissor entrar na fase de *Congestion Avoidance* de forma idêntica ao TCP Reno. O funcionamento do TCP New Reno pode ser visto na Figura 10. A Figura 11 mostra que TCP New Reno somente sai da fase *Fast Recovery* e entra na fase *Congestion Avoidance* , apartir do *threshold*, quando recebe a confirmação de todos os pacotes que foram enviados na fase *Fast Recovery*, diferente do TCP Reno que sai dessa fase quando recebe um reconhecimento de apenas um pacote enviado na *Fast Recovery*.

A idéia principal do New Reno consiste em fazer com que, quando múltiplos segmentos são perdidos em uma janela, o fluxo de transmissão se recupere sem ter que esperar por um estouro de temporizador.

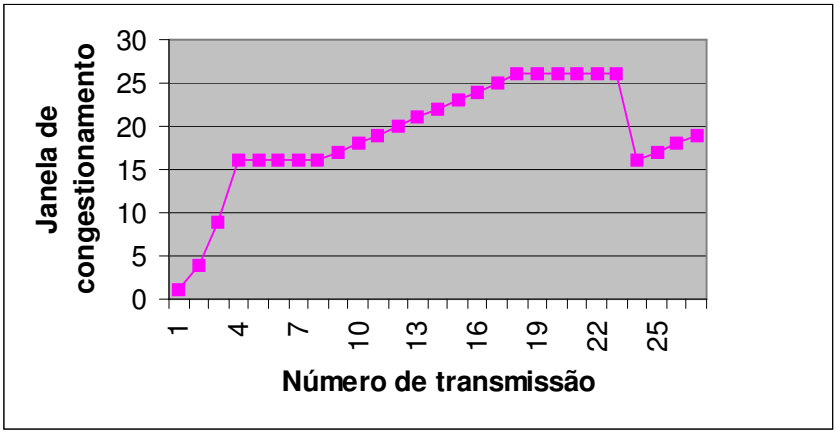

Figura 11. TCP New Reno.

1.3.4 TCP Vegas

O TCP Vegas [13] [14] possui modificações mais profundas no que diz respeito ao

controle de congestionamento. Seu mecanismo é bem diferente das outras implementações do TCP, não utilizando perdas de pacotes para detectar congestionamento. Essa implementação tenta evitar o congestionamento de forma pró-ativa. O Vegas tenta detectar o congestionamento nos roteadores entre o emissor e o receptor antes de ocorrer uma perda.

A implementação Vegas visa a monitorar a banda passante disponível na conexão de modo a detectar e evitar perdas de segmentos na fase *Slow Start* . O algoritmo de *Slow Start* implementado no TCP Vegas inicia-se como nas outras implementações com um segmento apenas, porém o aumento exponencial só ocorre a cada dois RTT. A razão para isso é que o TCP Vegas utiliza um RTT para fazer o cálculo do fluxo da rede para estimar a capacidade disponível da rede. Durante um dos RTT a CWND permanece inalterada.

O algoritmo baseia-se na comparação do valor da vazão real e o valor da vazão esperada. A fase C*ongestion Avoidance* do TCP Vegas utiliza um método de achatamento da taxa de envio em caso de indicativo de congestionamento. O controle da CWND é observado pela mudança no RTT, se o RTT for grande, o Vegas reconhece que a rede está congestionada e só então a CWND diminui, se os RTT's forem pequenos, o TCP Vegas verifica que a rede está livre de congestionamento e aumenta a CWND.

O Algoritmo limita o valor máximo e mínimo para a vazão de uma conexão TCP, mantendo a eficiência dentro desse limite, baseando-se na comparação entre o valor da vazão atual e o valor da vazão esperada [11] [16].

$$Diff = (TaxaEsperada - TaxaAtual) \times minRTT$$

A TaxaEsperada é calculada pela divisão da CWND e o valor do menor RTT encontrado. A TaxaAtual, por sua vez, é calculada pela divisão da CWND e o valor do RTT atual.

Para ajustar o tamanho da CWND, o TCP Vegas utiliza dois limitantes (α e β), cujos valores são respectivamente 1 e 3. Em [15] pode ser visto o desempenho do TCP Vegas com a variação dos valores desses limitantes. O TCP Vegas tenta manter o mínimo de α segmentos e não mais que β segmentos na fila e durante a fase de C*ongestion Avoidance* atualiza a CWND da seguinte maneira:

$$CWND = CWND + 1, \text{ se } Diff < \alpha$$
$$CWND = CWND - 1, \text{ se } Diff > \beta$$
$$CWND = CWND, (\alpha <= Diff <= \beta)$$

O mecanismo para lidar com a perda de segmentos e conseqüentemente com as re-transmissões causadas pelas perdas, também se diferencia dos mecanismos utilizados pelo TCP Tahoe e Reno. Para cada segmento transmitido, o tempo despendido no seu envio é lido e registrado pelo TCP na variável *timestamp*. Quando o ACK é recebido novamente, o Vegas armazena o tempo de chegada e então calcula o RTT com base na diferença desse tempo de chegada do ACK e o *timestamp*. O Vegas usa essa estimativa de RTT para re-transmitir segmentos nas duas situações:

(1) Se um ACK é recebido, sendo esse o primeiro ou o segundo após uma re-transmissão, compara o instante do envio do último segmento e o instante tempo atual. Se a diferença for maior que o valor do *timeout*, o segmento é re-transmitido.

(2) Se um ACK for um ACK duplicado e a diferença de tempo entre o segmento enviado e o tempo atual for maior que o *timeout* o segmento é re-transmitido sem a necessidade de

esperar três ACK's duplicados.

Os resultados experimentais apresentados em [10] [16] mostram que o TCP Vegas obtém um melhor desempenho e perdas menores do que o TCP Reno em diversos ambientes de redes. Também pode ser visto em [16] que o TCP Vegas tem alguns problemas com roteamento e com persistência de congestionamento que podem causar sérios impactos no desempenho dessa implementação.

Cada algoritmo possui vantagens e desvantagens sob determinadas situações de congestionamento. A Tabela 1 apresenta resumidamente as diferenças entre as implementações TCP com relação aos mecanismos de controle congestionamento e as políticas de reconhecimento.

Tabela 1. Características das implementações no TCP.

	Slow Start	***Congestion Avoidance***	***Fast Retransmit***	***Fast Recovery***	**ACK**
Tahoe	Sim	Sim	Sim	Não	Cumulativo
Reno	Sim	Sim	Sim	Sim	Cumulativo
New Reno	Sim	Sim	Sim	Sim	Cumulativo – Diferencia ACK's parciais
Vegas	Mais suave	Linear *Increase-decrease*	Não se aplica	Não se aplica	Cumulativo

Capítulo 2

Simulação de Redes de Computadores com o NS

Esse Capítulo apresenta conceitos relativos a simulação e a sua importância no estudo de redes de computadores. Mostra também as características do simulador de redes *Network Simulator* (NS) utilizado no estudo.

2.1 Visão Geral de Simulação

A simulação [18] [19] atualmente é uma técnica utilizada para solucionar e estudar problemas relacionados à pesquisa acadêmica e muito utilizada também no meio profissional [17]. A técnica de simulação surgiu no início do século XX e seu uso manual teve relativo sucesso enquanto o computador não era utilizado nessa área. No início eram criadas Tabelas de simulação, as quais, dependendo do tamanho, apresentavam um grande trabalho para a pessoa que necessitava tirar informações de um determinado sistema. Pequenas simulações ainda poderiam ser realizadas sem muito esforço, mas existia um limite para a complexidade de problemas que podiam ser solucionados manualmente, pois o trabalho era muito grande.

A crescente complexidade dos sistemas vem tornando a simulação uma ferramenta cada vez mais utilizada em várias áreas do conhecimento. A simulação é usada para descrever e analisar o comportamento de um sistema e pode ser vista como uma imitação da operação de um sistema real ao longo do tempo, envolvendo uma geração da história artificial desse sistema, o modelo. Com o auxílio do computador nas simulações pode-se obter níveis de detalhes que com as técnicas manuais, jamais poderiam se obter, permitindo que diferenças sutis de comportamento possam ser notadas. Além disso, com ferramentas de simulação pode-se utilizar animações,

permitindo que se visualize o comportamento dos sistemas.

O principal atrativo ao uso de simulação é que questões como "o que aconteceria se?" podem ser respondidas sem que os sistemas que estão sendo estudado sofram nenhuma modificação ou até mesmo não existam, já que as simulações são feitas no computador. Algumas das razões para se fazer experimentos com simulação são listados abaixo:

- O sistema real não existe: a simulação permite que estudos sejam feitos sobre sistemas que ainda não existem, levando ao desenvolvimento de projetos mais eficientes, pois todas as possibilidades podem ser testadas antes do projeto ir para execuçaõ.

- Experimentar com o sistema real é custoso: criar ambientes reais para fazer experimentos podem não compensar os resultados obtidos. Criar ambientes de redes com equipamentos que geralmente são caros é uma dos grandes dificuldades em estudos em redes de computadores. Simuladores específicos de redes chegaram para ajudar esses estudos.

- Experimentar com o sistema real é inapropriado: fazer modificações nos sistemas reais pode causar mudanças irreversíveis, o mesmo não ocorre quando se utiliza modelo de simulação.

Para poder fazer simulações é preciso criar um modelo de simulação que é a representação do sistema real. Existem outros tipos de modelos como modelos matemáticos, que são representados por relações e equações matemáticas; modelos visuais, que são representados através de gráficos; e modelos de entrada e saída. A decisão de qual modelo usar depende de vários fatores, como a complexidade do sistema. Se o sistema é simples, ou seja, se as relações entre os elementos estão bem descritas e estruturadas, o uso de modelos matemáticos com o auxílio dos conceitos da teoria das filas pode trazer resultados satisfatórios. No entanto os sistemas do mundo real costumam ser mais complexos e simplificações sobre esses sistemas podem levar a soluções pouco confiáveis.

A maioria dos modelos de simulação é do tipo entrada e saída e orientados a eventos discretos, ou seja, são modelos interativos aos quais se fornecem dados de entradas obtendo-se respostas específicas e os estados da simulação mudam com o tempo de acordo com os eventos. Os modelos orientados a eventos são modelos dinâmicos, diferente dos modelos matemáticos que representam sistemas em pontos fixos. A Figura 12 mostra um modelo de entrada e saída.

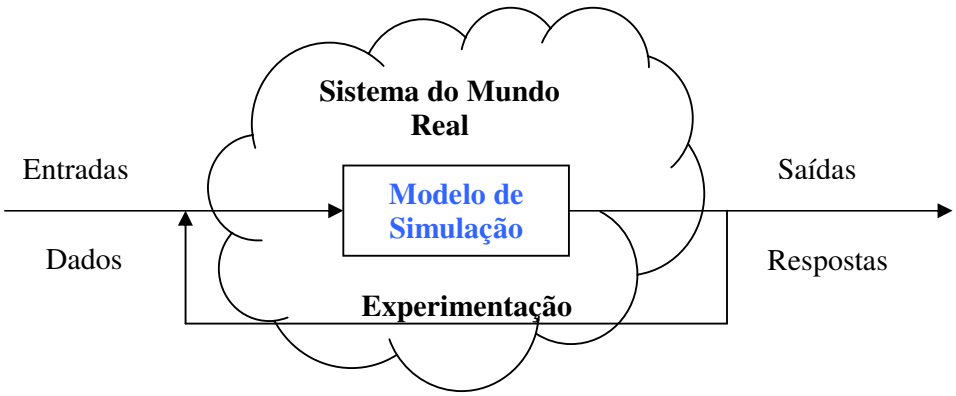

Figura 12. Modelo Entrada e Saída. Fonte: [21]

O primeiro passo para se construir um modelo é descobrir qual é o propósito da modelagem, ou seja, porquê criar um modelo para simulação. Baseado nisso, os limites e os detalhes são estabelecidos. Para isso o modelador tem que entender a estrutura e a as regras de operação do sistema e saber retirar a essência do sistema sem incluir detalhes desnecessários. Muitas informações obtidas durante a modelagem são essenciais para que os resultados de saídas sejam mais relevantes e que através dessas medidas possam ser tomadas decisões.

Existem vários tipos de modelos de simulação e eles devem ser empregados de acordo com o que se quer estudar. Os modelos voltados à previsão são modelos para prever o estado de um sistema no futuro. Já os modelos voltados à investigação, como o nome já diz, estão interessados em investigar o sistema em busca de informações do sistema e a reação do mesmo a certos estímulos. Existem também modelos específicos e modelos genéricos. Os modelos específicos são modelos criados para resolver problemas de alguma área específica, onde os resultados das simulações servem para tomadas de decisões estratégicas. Já os modelos genéricos são modelos mais flexíveis que são usados periodicamente, por longos períodos, para mais de uma finalidade.

Durante a modelagem do sistema é preciso saber alguns conceitos que são essenciais para o entendimento da simulação.

- **Variáveis de estado do sistema**: são informações necessárias para determinar o que está ocorrendo no sistema num determinado instante de tempo. Determinar essas variáveis é muito importante para obtenção das medidas de desempenho desejadas.

- **Entidades**: representam objetos do sistema. Podem ser estáticas ou dinâmicas. As entidades dinâmicas movem-se pelo sistema consumindo recursos e as entidades estáticas servem às entidades dinâmicas. As entidades também possuem atributos que identificam-nas unicamente.

- **Recurso**: é uma entidade que provê serviços as entidades dinâmicas, ou seja, recursos são entidades estáticas.

- **Atividades**: tarefa com período de tempo predeterminado, ou seja, seu final pode ser programado previamente.

- **Eventos**: acontecimentos que ocorrem em conseqüência de atrasos ou de conclusão de atividades.

As aplicações de simulação são extensas, porém, para se realizar uma simulação de forma que se obtenha resultados mais confiáveis, existem alguns passos a serem seguidos.

Inicialmente, deve-se ter clareza do propósito e dos objetivos do estudo e verificar se o problema pode ser solucionado através de simulação. Em seguida, deve-se fazer um planejamento do projeto para verificar a disponibilidade de recursos de software, hardware para realizar o trabalho. Após essa etapa, inicia-se a formulação do modelo conceitual; este ainda não é o modelo propriamente dito, mas é uma etapa que tornará a etapa de modelagem mais fácil. Nessa etapa, faz-se um esboço do sistema, definindo componentes, variáveis, interações e qual o método de modelagem que será utilizado. Aqui também é muito importante a presença do usuário do sistema que dará dicas sobre o funcionamento do mesmo. A próxima etapa é a coleta de dados, onde são extraídas informações e dados para serem utilizados na simulação. As fontes dos dados podem ser de arquivos históricos, vindos de observações do sistema, determinados com base em estimativas de operadores do sistema. Os dados, geralmente, são tratados antes de

utilizados. Em seguida o modelo conceitual é traduzido para modelo real e colocado no computador através de uma ferramenta de simulação ou de alguma linguagem de simulação [20].

Após o modelo ser construído no computador, é necessário submetê-lo à etapa de verificação e validação. Na construção do modelo, várias simplificações são feitas a partir do sistema real, então na etapa de verificação será avaliado se essas simplificações foram corretamente implementadas no modelo. A verificação é usada para se construir modelos sem erros. Existem várias técnicas específicas para fazer verificação que podem ser melhor estudadas em [21]. Na etapa de validação, é verificado se, apesar das simplificações feitas, o modelo se comporta de maneira semelhante ao sistema real; ou seja, produz resultados semelhantes aos observados no sistema real. Por fim são feitas as experimentações e em seguida uma análise estatística dos resultados obtidos.

Outras técnicas de análise de desempenho de redes existem, como a abordagem analítica. Na abordagem analítica podemos citar as Cadeias de Markov e a Teoria das Filas, por exemplo. Para utilizar Teoria das Filas é preciso estimar valores e o uso de valores médio pode levar futuramente a conclusões não muito precisas. É preciso reconhecer que tipo de sistema se quer estudar para então escolher um conjunto de equações adequado. Entretanto essas equações somente são apropriadas quando se considera um grande período de observação, para assim ter uma melhor precisão. O emprego de teoria das filas costuma acontecer quando se quer observar diferenças mais grosseiras nos sistemas.

A modelagem analítica em geral, implica em posse de profundo conhecimento matemático e às vezes, necessita que sejam feitas algumas suposições, podendo resultar em um afastamento do modelo real. Apesar da fundamentação matemática ser vista como uma vantagem nessa abordagem, a validação da modelagem pode ser mais difícil.Um estudo feito analiticamente das versões do TCP Tahoe, Reno e Vegas pode ser encontrado em [11].

Outra técnica que pode ser utilizada para fazer análise de desempenho de redes é a experimentação em laboratórios. Através da experimentação pode-se construir laboratórios com equipamentos de redes reais, realizar as experimentações e obter as métricas necessárias para a análise. Com esta técnica pode-se obter resultados mais realísticos, porém o custo para a construção desses laboratórios pode não compensar os resultados. Dessa forma a simulação pode ser uma alternativa barata e valiosa, se for feita de maneira correta e utilizando-se simuladores validados e reconhecidos. Como é o caso do NS, simulador utilizado nesse trabalho.

A simulação é mais fácil de aplicar e de se obter resultados do que modelos analíticos. Nos modelos analíticos, as análises são observadas geralmente para um conjunto limitado de medidas de desempenho e requer bastante conhecimento matemático. Com o uso de simulação pode-se compreender melhor quais variáveis são mais importantes no estudo. Porém, a construção de modelos de simulação não são fáceis e requerem bastante estudo e os resultados da simulação são, às vezes, de difícil interpretação, necessitando de ferramentas auxiliares.

Para empregar a simulação como técnica é preciso ter um conhecimento de alguns conceitos estatísticos fundamentais. Essa base estatística é importante para que se possa ter uma compreensão dos resultados obtidos. Os resultados das simulações são amostras de comportamento de um sistema e para se tomar qualquer decisão com base nos resultados deve-se sempre realizar inferências.

O crescimento no uso de ferramentas de simulação atualmente, se deve à facilidade de uso e sofisticação dos ambientes de desenvolvimento juntamente com o crescente poder de processamento dos computadores. As interfaces gráficas cada vez mais amigáveis e destinadas a diversas áreas específicas cada vez mais atraem usuários.

Atualmente, existem dois grupos de ferramentas que auxiliam as simulações. As ferramentas de propósito geral e as ferramentas de propósito específico. As de propósito geral servem para simular sistemas que podem pertencer a qualquer área de aplicação. Porém, por ter

essa flexibilidade, essas ferramentas podem se tornar difícil de serem utilizadas se o sistema a ser simulado for mais complexo e necessitar de detalhe peculiar. Como referência de uma ferramenta de propósito geral está o Arena [21], ferramenta bastante utilizada em várias áreas.

As ferramentas de propósito específicos são ferramentas utilizadas para solucionar problemas de uma determinada área. Essas ferramentas possuem módulos pré-existentes que facilitam a criação do modelo. Como exemplo desse tipo de ferramenta está o *Network Simulator,* ferramenta específica para simulação de redes de computadores [22].

Em estudos de redes de computadores especificamente, o uso de simulação também tem suas limitações. Geralmente, o tamanho da rede a ser simulada depende de restrições da capacidade do processador da máquina e do tamanho da memória do computador. As simulações são modelos da realidade que sofrem algumas simplificações do modelo do mundo real. Essas abstrações melhoram o desempenho do simulador, mas ao mesmo tempo pode esconder detalhes importantes.

2.1.2 Importância da Simulação de Redes

A simulação de redes é uma ferramenta muito importante para explorar rápida e economicamente o comportamento de vários protocolos em uma grande variedade de topologias e tráfegos, para tentar mostrar a interação desses vários aspectos numa rede e principalmente na Internet. Nos últimos anos a Internet tem crescido significativamente em tamanho e utilização, como conseqüência desse crescimento, novos protocolos e algoritmos estão sendo desenvolvidos para satisfazer as condições das mudanças.

Devido à complexidade inerente das redes e dos protocolos, a simulação desempenha um papel vital na tentativa de caracterizar o comportamento das redes, assim como os possíveis efeitos das modificações propostas no seu funcionamento. Várias novas necessidades da Internet como qualidade de serviço (Quality of Sercive), segurança, gerenciamento, transporte multicast e experimentos com novos meios para transmissão de dados podem ser estudadas através de simulação.

Desenvolver e avaliar novos protocolos e algoritmos requer muitos experimentos e, embora o uso de laboratórios possa ser utilizado, essas alternativas possuem alguns inconvenientes. Essa abordagem geralmente falha quando se quer testar vários tipos de tráfego e vários tamanhos de topologias, pois pode ser custoso criar esses ambientes. Pode ser difícil de reconfigurar e compartilhar os equipamentos e também difícil de representar alguns fenômenos como interferências de ondas eletromagnéticas, por exemplo. A repetição de experimentos sob condições diferentes também é difícil de alcançar.

Já simuladores de redes que simulam vários protocolos, tráfegos, topologias, podem prover um rico ambiente para experimentos a baixo custo, entender o comportamento dos protocolos existentes e de novos protocolos, e claro, dar oportunidade para estudar interação entre os protocolos em larga escala, pois através dos simuladores nenhum equipamento de rede real é necessário.

As vantagens da adoção de simulação são:

- as simulações não necessitam de muitos equipamentos já que é necessária uma máquina apenas para efetuar as simulações e analisar os resultados
- os simuladores permitem examinar um vasto conjunto de cenários num período de tempo relativamente pequeno
- as simulações oferecem meios para testar os protocolos em vários tipos de redes que muitas vezes seriam difíceis de criar e também podem dar acesso a uma rede específica

mesmo antes de estar construída.

No entanto, a modelagem e a simulação das redes não é sempre tão fácil de realizar e, claro, possui algumas desvantagens:

- Alguns simuladores usam implementações de protocolos abstratas que não são as implementações que estão na maioria dos sistemas operacionais reais, precisando tratar esse tipo de fato, pois os resultados podem não corresponder exatamente aos testes equivalentes aos feitos em uma rede real.

- Alguns simuladores às vezes, não modelam os eventos que não são de redes e que podem ter impactos no desempenho dos protocolos.

Portanto, é necessário acompanhar todos os estudos de simulação por processos de validação, que permitem demonstrar com rigor se os resultados de uma simulação se aproximam dos resultados reais. O NS é um simulador que já possui seus módulos validados, não necessitando de validações posteriores.

As ferramentas que permitem entender e simular o complexo comportamento dos protocolos em redes são indispensáveis. A utilização de ambientes de simulação vem aumentando de forma significativa uma vez que esses permitem o estudo e avaliação de redes a baixo custo. Os simuladores, geralmente, são fáceis de usar, fáceis de extender novas funcionalidades e explorar vários cenários. Uma ferramenta que vem sendo bastante utilizada é o NS (*Network Simulator*).

Na próxima seção, será dada maior ênfase nas características desse simulador de redes.

2.2 Network Simulator

O simulador *Network Simulator* (NS) [22] é um simulador de propósito específico, focado para o desenvolvimento de pesquisas em redes de computadores. Atualmente apresenta-se como uma das mais poderosas ferramentas de análise, reconhecida internacionalmente e muito utilizada nos principais meios acadêmicos e nas maiores companhias de redes de informação pelas suas facilidades de utilização, criação de cenários de redes e oferecendo suporte aos vários protocolos das várias camadas da arquitetura TCP/IP.

O projeto do NS iniciou como uma alteração do simulador de redes REAL [23], em 1988, e evoluiu consideravelmente nos últimos anos. O simulador não está concluído ainda, mas o resultado de um esforço contínuo de investigação e desenvolvimento por parte da comunidade que o utiliza, já que o mesmo é de código aberto e está em constante atualização; ou seja, suporta novos módulos ou alterações nos módulos já existentes.

O núcleo dessa ferramenta, escrito em C++, pode ser modificado, ou personalizado, de acordo com as necessidades de cada usuário, pois o NS é um simulador de código aberto, porém é preciso re-compilar as classes que foram modificadas. Esse simulador tem um melhor desempenho se instalado em ambiente UNIX, como em FreeBSD, SunOS e Solaris. O NS também roda sobre a plataforma Windows, porém é necessário a instalação de alguns programas auxiliares. A instalação do NS para utilização neste trabalho foi feita no ambiente UNIX, mais especificamente em Linux, na distribuição *Red Hat* 9.

O NS tem demonstrado ser de extrema importância na área de pesquisa de redes e muito esforço tem sido feito para que ele se torne um simulador cada vez mais fácil de utilizar, de modo que seu uso seja cada vez mais abrangente. É a ferramenta mais utilizada na escrita dos mais

variados artigos de investigação na área de redes.

Esse simulador tem várias características especiais para suporte a redes. Simulação de múltiplos protocolos, com uma interface para facilmente construir novos cenários e vizualizadores de simulação como o NAM [24]. O NS implementa uma variedade de mecanismos e dá suporte a simulação de diversas tecnologias de redes como, por exemplo:

- Protocolos da arquitetura TCP/IP: TCP e UDP, protocolo IP, protocolos da camada de enlace (MAC), HTTP.

- Fontes geradoras de tráfego: Exponencial, Pareto, CBR (*Constant bit rate*) VBR (*Variable Bit Rate*).
- Mecanismos de gerenciamento de filas: DropTail (FIFO), RED (*Random Early Detection*), SFQ (*Stochastic Fair Queue*).

- Algoritmos de roteamento: DV (*Distace vector*) e roteamento dinâmico.

- Aplicações : FTP, Telnet, Web.

O NS também implementa tráfego multicast e alguns protocolos da camada MAC para simulação de LAN's e simulação de redes sem fio.

Simulador baseado em eventos discretos [25], o NS, utiliza a idéia de escalonador de eventos. O escalonador seleciona o próximo evento agendado, executa-o e retorna para executar o próximo evento. Atualmente, o simulador executa um evento de cada vez, isto é, em qualquer parte do tempo, somente um evento está sendo executado. Se mais de um evento estiver programado para executar ao mesmo tempo, o primeiro evento agendado será realizado e em seguida o outro. Eventos simultâneos não são reordenados pelos escalonadores e dada uma entrada, a ordem de execução dos eventos deve ser a mesma, sempre que esta entrada for selecionada.

2.2.1 Arquitetura do *NS*

O *Network Simulator* é um simulador orientado a objeto desenvolvido em duas linguagens, C++ e Otcl (*Object-oriented Tool Command Language*) [26] de forma modular, sendo que o seu *frontend* é o interpretador Tcl.

A razão para se ter duas linguagens, ao invés de uma, é devido ao fato do NS ter dois tipos de tarefas. Por um lado, a simulação detalhada de protocolos requer uma linguagem de programação poderosa e rápida, e por outro lado, a criação e modificação dos cenários de redes necessita de um linguagem em que as modificações sejam mais fáceis de realizar. Portanto a divisão em duas linguagens (OTcl e C++) objetiva dar ao simulador tanto velocidade e poder, quanto flexibilidade e facilidade de mudança de parâmetros.

A vantagem de se utilizar duas linguagens no NS é que os usuários podem implementar suas simulações com um alto nível de abstração, pois não precisam se preocupar em implementar detalhes dos protocolos existentes e por outro lado têm a facilidade de modificar os Scripts de simulação rapidamente.

O núcleo do NS é escrito na linguagem C++ , por ser uma linguagem compilada e por executar mais rapidamente e, portanto, mais eficiente no processamento de baixo nível. Porém a linguagem C++ é mais complexa no que diz respeito a modificações, tornando-a mais adequada para implementação detalhada de protocolos. A linguagem C++ permite a construção de

algoritmos que trabalham com grandes conjuntos de dados e está presente em módulos que não são modificados freqüentemente, e que necessitam de alto desempenho. Portanto o uso da linguagem C++ nos módulos confere velocidade e mais praticidade na implementação de protocolos.

A linguagem OTcl roda mais lentamente, mas pode ser alterada muito rapidamente e interativamente, tornando-a ideal para as conFigurações dos cenários de simulação.

A linguagem OTcl é uma versão orientada à objetos da linguagem Tcl e é usada como a linguagem de criação de Scripts de simulação que contém o cenário da rede, os protocolos e o tipo de tráfego. Em uma simulação, geralmente, necessita-se realizar mudanças nos parâmetros do cenário que está sendo estudado para verificar o comportamento da rede simulada em diferentes condições. Por isso o *NS* faz uso da linguagem OTcl que foi desenvolvida pelo MIT. OTcl é uma linguagem de fácil manipulação e interpretada, já que os programas de simulação precisam ser alterados de maneira rápida e novamente executados. O OTcl pode ser visto como a linguagem de interface com o usuário, pois os Scripts de simulação são escritos nesta linguagem, permitindo a manipulação e reconFiguração de parâmetros de uma maneira mais fácil.

Para cada classe do simulador construída em C++ existe uma classe correspondente em OTcl. O simulador suporta uma classe hierárquica em C++, e uma classe hierárquica similar dentro do interpretador Otcl. O conjunto das classes do NS desenvolvido em C++ é chamado de hierarquia compilada já que o C++ é uma linguagem compilada; e o correspondente em OTcl é chamada de hierarquia interpretada, já que a mesma é uma linguagem interpretada. Quando se cria Scripts através do interpretador OTcl esses são associados a um objeto correspondente na hierarquia compilada.

A arquitetura do NS é mostrada na Figura 13. O usuário atua na camada tcl, escrevendo os Scripts e realizando as simulações. O escalonador de eventos e os componentes de rede são implementados em C++ e disponibilizados para o interpretador OTcl pela camada "tclcl", ou seja, é o "tclcl" que provê a forma de unir as duas linguagens. O escalonador de eventos é bastante importante na arquitetura do NS pois ele é responsável por disparar os eventos que estão na fila de eventos e acionar o objeto que vai realizar o tratamento do evento.

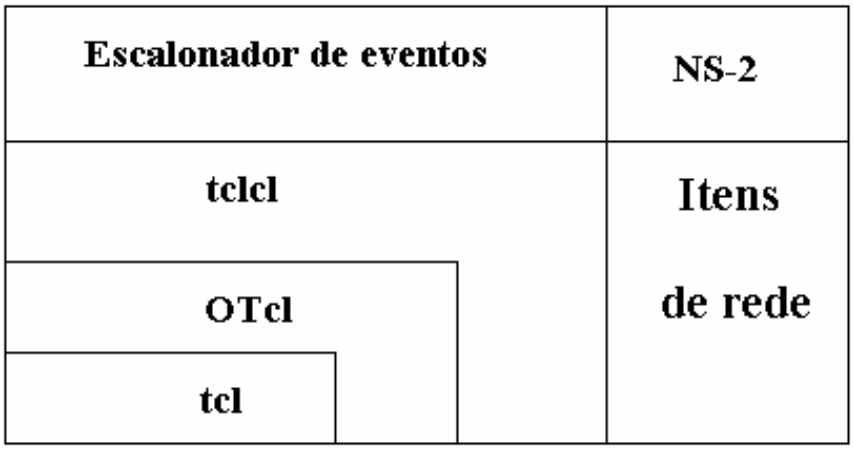

Figura 13. Arquitetura do NS.

A dinâmica de uma simulação na visão do usuário é vista da seguinte maneira: para criar uma simulação é preciso escrever um Script OTcl que será lido por um interpretador e gerará uma saída específica que posteriormente será analisada . O Script pode gerar ainda um arquivo de trace ou pode inicializar o NAM (*Network Animator*) [24] para visualização da simulação caso

seja especificado no Script. Os arquivos de trace são arquivos de saída que são utilizados para analisar os dados gerados na simulação. A dinâmica da simulação é mostrada na Figura 14. A inicialização de um Script Otcl é feita através do comando 'ns <tclscript>', onde <tclscript> é o nome do Script em tcl que define o cenário de simulação. Tudo no NS depende de um Script Tcl.

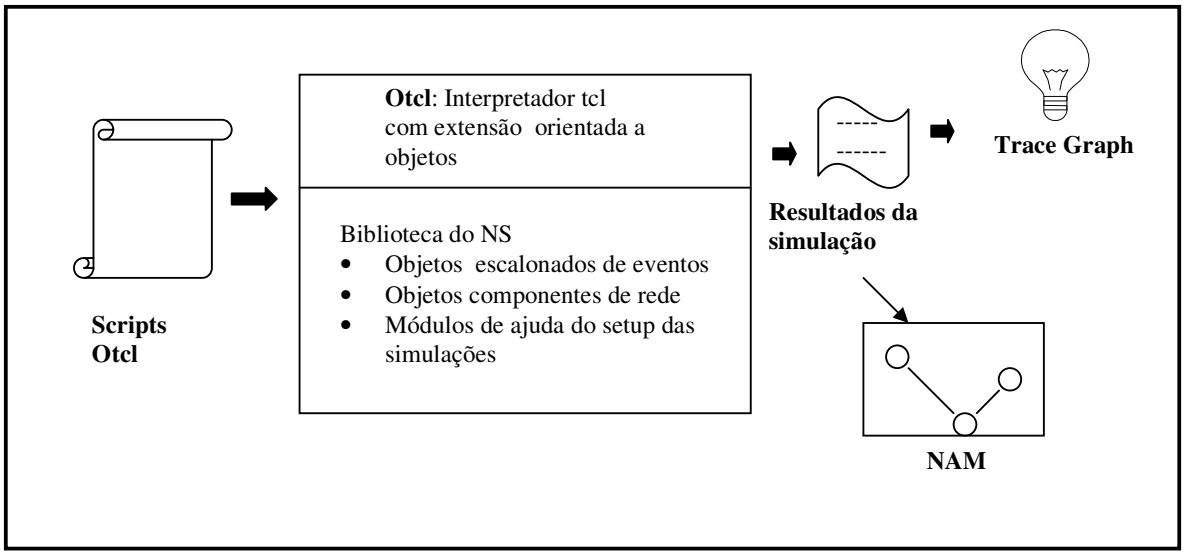

Figura 14. Dinâmica da simulação no NS.

Os Scripts possibilitam o uso de vários conceitos que existem em outras linguagens de programação como variáveis, funções, laços. Esses Scripts possibilitam a criação de uma simulação e de agentes para que se possa simular uma rede com características o mais próximo possível da realidade. São geralmente simples e podem ser aprendidos somente através de exemplos, não apresentando dificuldades para seu entendimento. O Script se orienta também por ações que ocorrem durante o tempo da simulação, já que simula-se tráfegos em uma rede e que esses precisam ter um começo e um fim.

O NS trabalha com o conceito de agentes, que é muito semelhante ao utilizado em orientação a objetos. Os agentes são baseados nas suas respectivas classes, as quais possuem seus parâmetros básicos. Um agente precisa ser conectado a outro para que possa utilizar suas propriedades de envio de pacotes e isso é bastante explícito quando é criado um Scripts de simulação em OTcl.

2.2.2 Ferramentas Auxiliares

Um ponto importante a observar é que o NS não fornece estatísticas de simulação de modo automático; essas devem ser obtidas através de procedimentos específicos no Script da simulação. Pode-se, ainda, usar ferramentas para análise dos arquivos de *trace* gerados durante a simulação, que são os verdadeiros resultados da simulação como pôde ser visto na Figura 13.

O NAM (*Network Animator*) [24], juntamente com o NS, forma um conjunto de ferramentas poderosas para o estudo das simulações de redes. Com o NAM, os protocolos podem ser visualizados como animações. Essa ferramenta gera desenhos dos nós e links e através dela também é possível observar o comportamento do fluxo dos pacotes sobre cada nó da rede, bem como o comportamento e o estado dos protocolos e visualizar os eventos mais importantes que estão ocorrendo.

O NS tem a possibilidade de gerar uma saída específica para a simulação no NAM, porém é necessário deixar explícito nos Scripts OTcl os comandos que o invocam. Os arquivos de trace do NAM contém todas as informações necessárias para animação, tanto do esquema estático da rede quanto dos eventos dinâmicos, como saída e chegada de pacotes para que seja possível a criação da topologia da rede, com seus nós, links e fluxo dos pacotes. Com a utilização do NAM, os usuários podem editar uma topologia, o *layout* e verificar o andamento da simulação dinamicamente, bem como criar novas visões para a mesma animação. Na Figura 15, pode-se ver a tela principal do NAM com uma topologia ilustrativa.

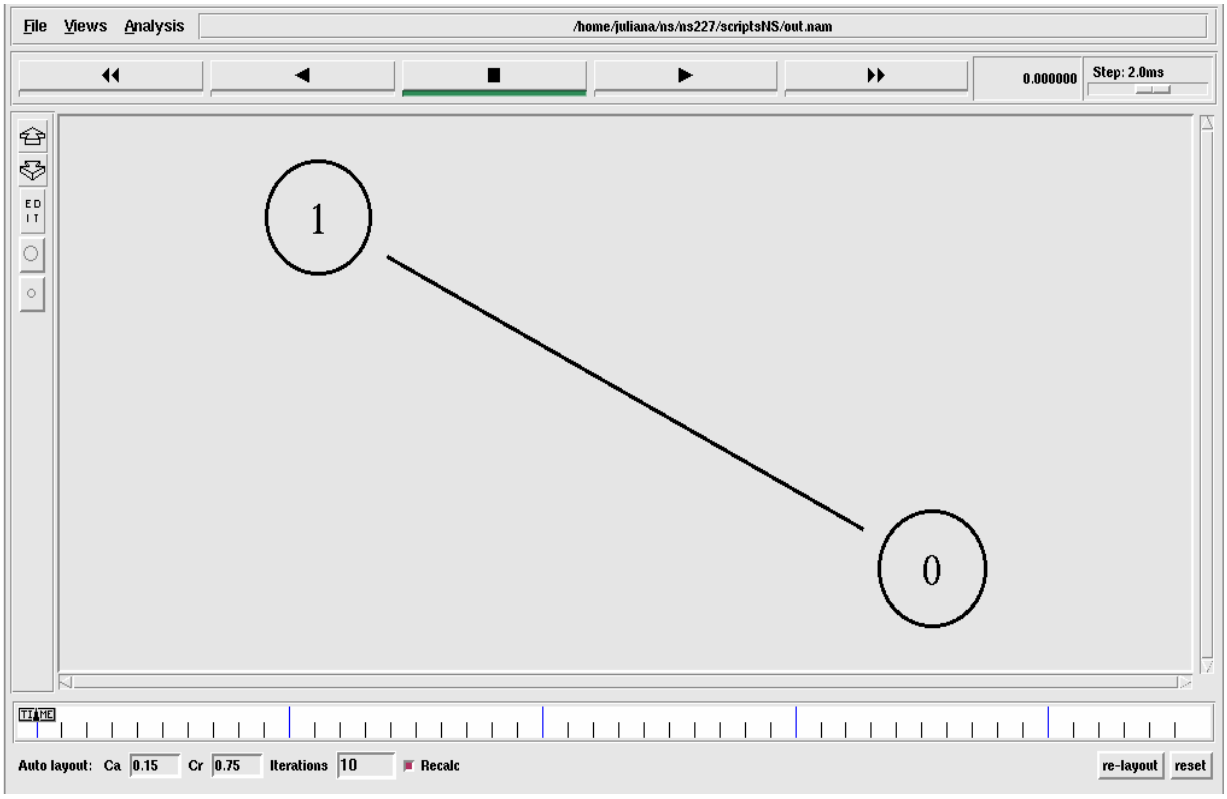

Figura 15. Tela do NAM

O uso de ferramentas para análise dos arquivos de *trace* gerados durante a simulação, que são os verdadeiros resultados da simulação, é muito importante . A partir dos arquivos de *trace* é possível fazer medidas estatísticas que são de interesses para o estudo. Esses arquivos, com formatação específica, registram cada evento gerado pelos escalonadores. As ferramentas para análise dos arquivos de *trace* são capazes de ler os dados gravados nesses arquivos e efetuar os cálculos desejados. Uma destas ferramentas, muito utilizada, é o *awk* [27]; uma linguagem construída para buscar padrões dentro de um arquivo e efetuar ações programadas.

A ferramenta *TraceGraph* disponível em [28] é uma ferramente que auxilia na extração de informações dos arquivos de trace que são gerados durante as simulações e é de fácil utilização. Essa ferramenta é bastante interativa e possui grande utilidade para diminuir os esforços de extrair informações dos arquivos de *trace* que só contem números. Ela dispões de variados tipos de gráficos que ajudam na compreensão dos aspectos estudados. Ela foi utilizada para extrair a maioria das informações contidas nos arquivos de trace para obtenção dos resultados deste trabalho. O arquivo de *trace* gerado pela simulação e que é a entrada para o *Trace Graph* é

mostrado na Figura 16. Esse arquivo contém toda os dados dos eventos da simulação gerados pelo simulador e a partir dele pode-se extrair as informações desejadas.

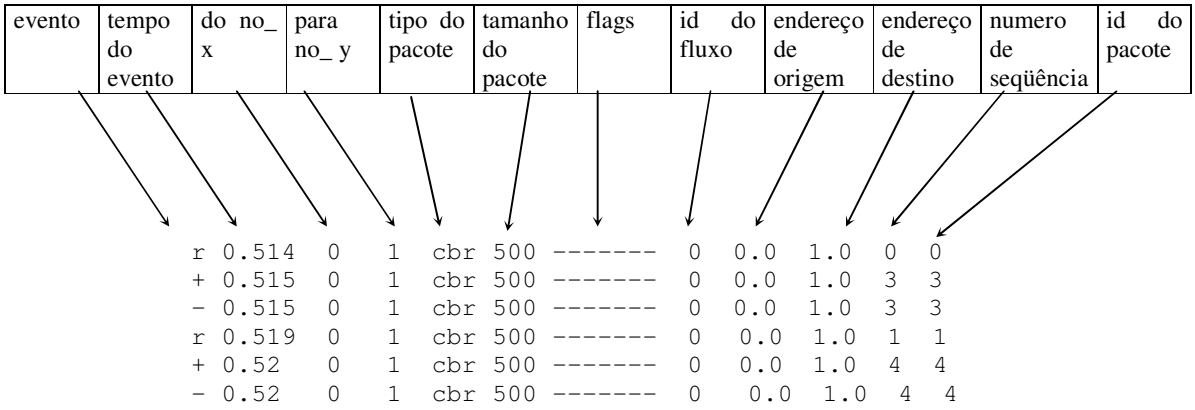

Figura 16. Arquivo de *trace*.

Esses arquivos de *trace* possuem os vários eventos que são realizados no decorrer da simulação. Os eventos são representados por +, -,d, r.

1. Entra na fila de espera (+)
2. Sai da fila (-)
3. Prossegue para o nó seguinte – *receive* (r)
4. É feito o descarte – *Drop* (d)

O significado de cada coluna do arquivo também está mostrado na Figura 16.

Existem outras ferramentas que podem ser utilizadas para a mesma finalidade, como o *Xgrap*[29] ou o *Gnuplot*[29], porém a ferramenta *TraceGraph* foi escolhida por ser de fácil manipulação e por disponibilizar uma maior variedade de gráficos.

Capítulo 3

Experimentos e Resultados

Para mostrar o desempenho e as diferenças entre as implementações de algoritmos de controle de congestionamento estudadas, foi utilizada a técnica de simulação, detalhada no Capítulo anterior. Para isso, são criados os cenários de redes a serem experimentados, através dos Scripts Otcl e, em seguida são mostrados os experimentos e resultados das simulações. Através de simulações, pode-se explorar o comportamento das diversas implementações dos algoritmos do TCP para controlar o congestionamento, explorando o comportamento dos protocolos de redes em algumas topologias com alguns tipos de tráfegos e a interação entre os nós da rede.

3.1 Modelos e Parâmetros de Redes

Esta seção descreve os modelos e parâmetros de redes utilizados para atingir os objetivos.

3.1.1 Modelos

Para comparar as várias implementações foram criados cenários de redes, simulando a existência de transmissores, a existência da rede e dos receptores, interligados por enlaces com diferentes larguras de banda.

Uma das topologias de rede que foi utilizada para analisar e simular o comportamento quanto a perdas de pacotes, foi a mesma referenciada em [8] e pode ser vista na Figura 17. O cenário de rede consiste de um transmissor, indicado pelo nó 0, um receptor, indicado pelo nó 2 e um roteador, indicado pelo nó 1. O roteador 1 possui *buffer* finito, com pequena capacidade, e as simulações são realizadas utilizando a disciplina de fila FIFO (*first-in first-out*), ou seja, o

primeiro pacote que chegar é o primeiro sair. Nessa topologia, os nós são interligados através de enlaces. O nó 0 está ligado ao nó 1 por um enlace de 10Mbps e possui um atraso de propagação de 2ms. O nó 1 está ligado ao nó 2 por um enlace de 1Mbps e possui um atraso de propagação de 1ms.

Uma conexão TCP é estabelecida entre o nó 0 e 2. Essa conexão é feita de tal forma que pacotes são enviados apenas pelo nó 0; o nó 2 apenas envia um ACK para cada pacote preveniente do nó 0.

A idéia da simulação é gerar uma situação de congestionamento fazendo com que a soma dos tráfegos gerados pelas fontes, exceda a capacidade do enlace que chega ao nó de destino.

Esse cenário consiste numa rede simples, mas que apesar de simples, já é suficiente para caracterizar um congestionamento no enlace 1->2. O congestionamento vai ocorrer no enlace 1->2, porque esse enlace recebe um tráfego de um enlace com capacidade de 10Mbps e ele somente tem capacidade de apenas 1Mbps. Esse cenário vai levar a perda de pacotes pelas implementações do TCP no enlace 1->2 pois esse enlace não vai suportar a carga que chega do enlace 0->1, levando a um congestionamento do enlace 1->2.

Com esse cenário podemos analisar variáveis que são interessantes para mostrar o desempenho dos algoritmos de controle de congestionamento que são estudados. A topologia descrita aqui foi utilizada para o estudo dos protocolos com respeito a perdas de pacotes, utilização do canal e variação do atraso entre chegadas dos pacotes. A descrição dessa topologia pode ser encontrada no Script topologia3nostahoe.tcl que está no Apêndice A.

Nesse cenário as simulações foram feitas durante 50 segundos, tempo esse suficiente para iniciar o congestionamento no enlace 1->2 e a partir desse estado, analisar as métricas necessárias. Foram realizadas 20 simulações para cada parâmetro estudado através desse cenário, essa quantidade foi escolhida afim de obter um nível de confiança maior possível e considerado bom pelo simulador, e dessa forma obter resultados mais representativos.

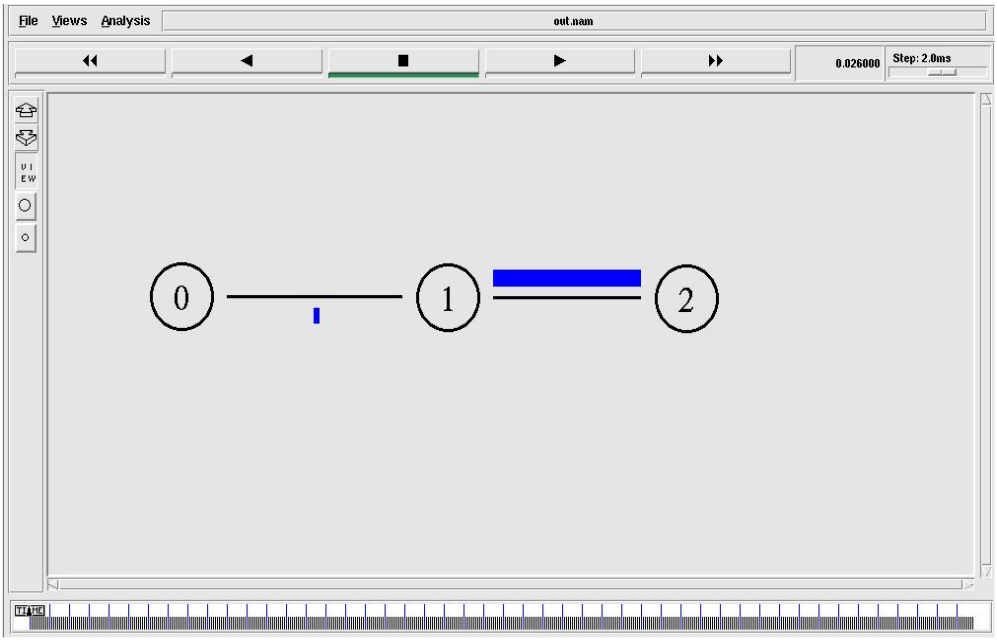

Figura 17. Topologia de rede 1 (Extraída do NAM).

É considerado outro cenário, mostrado na Figura 18, baseado numa topologia genérica tendo como referencia [15] [16]. Essa topologia foi utilizada no estudo do parâmetro *throughput,*

ou seja, a quantidade efetiva de pacotes que chegaram ao receptor. A topologia encontra-se no Script renovegasThroughput.tcl que encontra-se no Apêndice A. As duas fontes (nó 0 e nó 1), executam a aplicação FTP, onde sempre existem dados disponíveis para serem enviados, ou seja, enquanto o valor da janela permitir, dados são transmitidos com a taxa de transmissão permitida pelo canal.

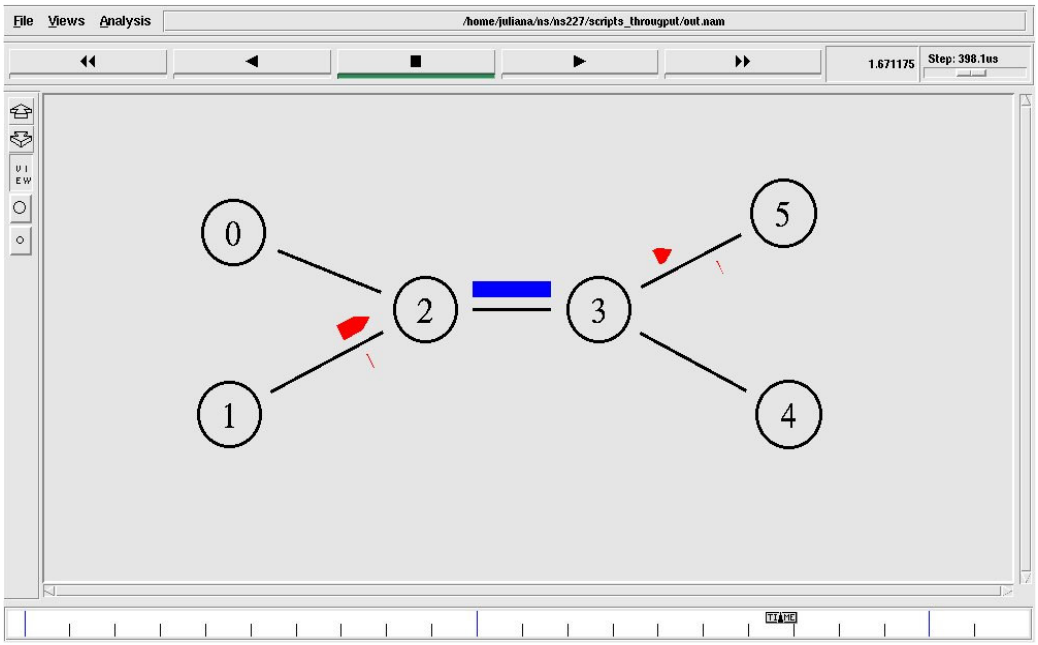

Figura 18. Topologia de rede 2.(Extraída do NAM)

Os nós 2 e 3 simulam dois roteadores da rede. Os parâmetros da rede 2 são mostrados na Tabela 2.

Tabela 2. Parâmetros da rede 2.

Network	**Topologia 2**
Quantidade de nós	4
Links 0 → 2 – Enlace, Atraso.	10Mbps, 2ms
Links 1 → 2 – Enlace, Atraso.	10Mbps, 3ms
Links 2 → 3 – Enlace, Atraso.	1,5Mbps, 1ms
Links 3 → 5 – Enlace, Atraso.	10Mbps, 2ms
Links 3→ 4 – Enlace, Atraso.	10Mbps, 3ms

Um ambiente heterogêneo com duas conexões TCP foram criadas: uma conexão TCP Reno, envia pacotes do nó 0 para o nó 4; e a outra conexão TCP Vegas envia pacotes do nó 1 para o nó 5. Dessa forma o enlace 2->3 é compartilhado por duas conexões de duas diferentes versões do TCP. Na simulação foi considerado a situação em que a a conexão TCP Reno inicia-se primeiro e em seguida a conexão Vegas.

As simulações tiveram duração de 20s, que foi um tempo suficiente para que os efeitos esperados pudessem acontecer. Foram realizadas 25 replicações variando as sementes geradoras de tráfego para obter resultados mais precisos. Essa quantidade de replicações foi escolhida para que o custo computacional das simulações não seja grande demais e que pudéssemos obter um nível de confiança estatítica dos resultados aceitável.

Essa topologia, foi utilizada para caracterizar outra forma de congestionamento. O congestionamento nessa rede ocorre devido ao compartilhamento do enlace 2->3 por duas conexões. O enlace 2->3 fica congestionado pois nele chegam dois enlaces de 10Mbps e ele só possui uma capacidade de 1,5Mbps. Isso leva a um cenário de congestionamento porque o enlace 2->3 não tem a capacidade de enviar os seus pacotes na mesma velocidade que chega até ele, ocorrendo assim perdas de pacotes devido ao congestionamento.

3.1.2 Parâmetros

É importante verificar as variáveis que se pode estudar para analisar o desempenho dos algoritmos de controle de congestionamento. As variáveis que são importantes para esse estudo foram identificadas com base nas referências [8][15] e são analisadas ao longo do estudo. A variável *jitter*, em particular, foi sugerida para confirmar que o TCP apesar de ser confiável e garantir um controle de congestionamento, não obtem bom desempenho quando carrega aplicações multimídias. As variáveis são:

1. Perda de pacotes
2. Utilização do canal
3. *jitter* (Atraso entre chegada de pacotes)
4. *Throughput*

Sabe-se que a fração dos pacotes perdidos aumenta com o aumento da intensidade de tráfego, portanto esse fator torna-se bastante importante no estudo de controle de congestionamento, visto que a perda de pacotes é o primeiro sintoma de que está ocorrendo congestionamento na rede.

A utilização do canal é um aspecto a ser estudado para mostrar como cada implementação utiliza a banda disponível, ou seja, como elas fazem para obter um maior aproveitamento do canal. A utilização diz respeito à forma como as implementações do TCP fazem para que os canais de transmissão estejam sempre com a quantidade de pacotes no canal próxima da sua capacidade.

O *jitter* é uma variação dos atrasos na entrega dos pacotes, ou seja, o tempo decorrido entre o momento em que um pacote é gerado na fonte e o momento em que é recebido no destino pode variar de pacote para pacote. Em aplicações com tráfego multimídia, por exemplo, essa variação não pode ser grande, pois compromete a qualidade da imagem e vídeo. A qualidade pode se deteriorar até níveis inaceitáveis assim que a remessa de pacotes de voz e imagem atinge um enlace congestionado, pois irá perder muitos pacotes.

O *throughput* é a vazão propriamente dita; é a quantidade de pacotes que realmente chega ao receptor.

3.2 Resultados das Simulações

Esta seção descreve os resultados mais significativos dos vários experimentos a partir dos cenários mostrados na seção 3.1.1. As experimentações foram feitas em ambientes homogêneos e heterogêneos. Ambientes homogêneos são ambientes onde apenas conexões de uma única versão do TCP existe, enquanto em ambientes heterogêneos existem conexões de diferentes versões do TCP. As conexões heterogêneas são necessárias visto que na Internet podem estar sendo usadas várias versões do TCP num mesmo link. Apenas as Figuras relevantes que ajudem a entender o comportamento das implementações foram extraídas da ferramenta *Trace Graph*.

3.2.1 Perdas de Pacotes

Os primeiros experimentos estudam o comportamento das implementações com relação a perdas de pacotes.

Para estudar o comportamento das implementações no que diz respeito a perdas de pacotes a topologia de rede da Figura 17 foi utilizada. As conexões entre os nós 0, 1 e 2 para cada implementação TCP foram criadas através de Scripts Otcl. Os Scripts para cada implementação são semelhantes, mudando apenas a implementação, portanto somente o Script da simulação do TCP Tahoe encontra-se no Apêndice A (topologia3nostahoe.tcl).

Em cada Script somente o nó 0 envia dados para o nó 2, apesar do enlace ser *full-duplex*. Isso se torna necessário para melhor analisar as perdas ocorridas.

O tráfego utilizado nesses Scripts foi a aplicação simulada FTP que já é uma aplicação implementada e validada pelo NS. O agente TCP criado nos Scripts da simulação não gera tráfego, para isso é necessário criar um módulo gerador de tráfego que é anexado ao agente TCP, esse módulo então gera os dados. O FTP é uma aplicação bastante utilizada em redes como a Internet e é caracterizada pela transferência de uma grande massa de dados.

O cenário da Figura 17 foi simulado separadamente para cada implementação do TCP em questão. Em cada Script da simulação foi gerado um arquivo de *trace*. Para sua geração foi necessário deixar explícito no Script Otcl a abertura e o fechamento de um arquivo desse tipo.

O escopo da análise dos dados gerados na simulação está restrito às informações relacionadas com os descartes de pacotes produzidos pelo simulador. Assim, foi feita uma análise somente nos pacotes descartados que continham dados, sendo os demais (ACK's, pacotes de sincronização e de estabelecimento de conexão) ignorados. Em função disso, os registros selecionados dos arquivos de *trace* foram apenas do tipo "d". Para coletar do arquivo de *trace* apenas os pacotes descartados, foi necessário utilizar nos Scripts a linguagem AWK para retirar somente os pacotes que possuíam a primeira coluna do arquivo de *trace* igual a "d".

A Tabela 3 mostra as quantidades de pacotes transmitidos, as recebidas no destino e as perdidas durante a transmissão. É importante lembrar que essas quantidades referem-se somente a pacotes com dados. Para efeito de análise os pacotes de reconhecimento, de sincronização e de estabelecimento de conexão não foram considerados. Da mesma forma, os registros do arquivo de *trace* relacionados com a movimentação de pacotes entre nós intermediários da rede foram ignorados.

Os valores das colunas da Tabela 3 foram obtidos pela média de pacotes transmitidos, recebidos e descartados em cada uma das 20 simulações realizados, a fim de obter um valor mais representativo.

Tabela 3. Quantidade de pacotes de cada implementação.

	Pacotes transmitidos	**Pacotes recebidos**	**Pacotes descartados**
Tahoe	11.982	11.766	216
Reno	12.015	11.784	231
New Reno	12.262	12.027	234
Vegas	12.507	12.500	7

Como pode ser visto na Tabela 3 o TCP Tahoe obteve menor número de perdas, comparado com o Reno e o New Reno. Isso é devido ao mecanismo de controle de congestionamento do Tahoe que, apesar de utilizar *Fast Retransmit*, quando um pacote que foi re-transmitido é reconhecido, o Tahoe vai para a fase de *Slow Start* iniciando o envio de pacotes com uma janela de congestionamento igual a 1. Dessa forma, ocorre menos perda, porém a

utilização do canal é muito baixa pois o canal fica praticamente vazio ao começar a S*low Start.* Com é constatado na próxima Seção. Portanto o TCP Tahoe possui um desempenho inferior ao TCP Reno e New Reno.

O TCP Reno e New Reno tiveram um número de perdas de pacotes bem próximos. Isso é devido à adição da *Fast Recovery* nessas duas implementações, prevenindo que o canal não se esvazie completamente. Porém, como eles continuam enviando pacotes numa rede que ainda pode estar congestionada, um número maior de pacotes foram perdidos.

A implementação do TCP Vegas controla o congestionamento na rede monitorando os resultados da diferença entre a taxa real e a taxa esperada para então ajustar a janela de congestionamento e, assim, impede que a janela cresça de forma que a rede não possa procesá-los, levando a um menor número de perdas como pode constatado na Tabela 3.

3.2.2 Utilização do canal

Para investigar e comparar como cada implementação utiliza a largura de banda disponível na rede foi utilizado o cenário de rede da Figura 17.

A utilização do canal de transmissão é muito importante, pois os vários mecanismos utilizados pelas implementações precisam utilizar a largura de banda da melhor forma possível para obter maior desempenho, visto que os enlaces são compartilhados por diversas transmissões, simultaneamente. Apesar de estar sendo simulado em um cenário simples, nesse cenário já se verifica a diferença de utilização em cada implementação.

Para obter como cada implementação utiliza a banda disponível foram criados Scripts de simulações para cada implementação e, em seguida, foi obtido a quantidade de pacotes que trafegaram no enlace 1 -> 2.

É coletado apenas a quantidade de pacotes desse enlace, pois esse enlace é o enlace que encontra-se congestionado e cada implementação tem que tentar manter sempre a maior quantidade possível de pacotes no enlace congestionado. Dessa forma, cada implementação visa utilizar esse enlace da melhor maneira possível.

Foi utilizado o tráfego FTP em cada simulação. Como os Scripts de simulação para esse estudo são semelhantes, mudando apenas a versão do TCP, um dos Scripts utilizados nessas simulações encontra-se no Apêndice A (topologia3nosUtiliz.tcl).

A Tabela 4 mostra a quantidade de pacotes que trafegaram no enlace 1 -> 2 para cada implementação, obtida de uma média feita das 20 simulações realizadas no estudo desse parâmetro.

Tabela 4. Utilização do canal.

	Pacotes no enlace n1-n2
Tahoe	4.583
Reno	5.892
New Reno	6.013
Vegas	6.253

Como pode ser visto na Tabela 4, o TCP Vegas possui a maior quantidade de pacotes trafegando no enlace 1 -> 2, mostrando que essa implementação possui a melhor utilização do canal, devido a sua mais eficiente maneira de detectar o congestionamento da rede.

Entre as implementações Tahoe, Reno e New Reno, a diferença na utilização do canal é devida também aos seus mecanismos de controle de congestionamento.

O Tahoe, por não utilizar o mecanismo *Fast Recovery* retorna para a fase de S*low Start* toda vez que um pacote é re-transmitido, isso ocasiona uma pequena utilização da banda fornecida pela rede, pois, toda vez que ocorrer re-transmissão, ele irá re-começar a enviar pacotes com a janela de congestionamento com valor 1. Se ocorrerem várias perdas seguidas, o Tahoe volta a *Slow Start* diversas vezes, apresentando assim essa baixa utilização e, portanto, um baixo desempenho.

Já o TCP Reno utiliza a *Fast Recovery*, obtendo assim maior utilização do canal, pois com essa etapa ele não precisa esperar um ACK do pacote que foi perdido para enviar mais dados. A maior utilização do canal em relação ao Tahoe é também em conseqüência da não redução da janela de congestionamento para 1 como ocorre no Tahoe. Quando ocorre uma re-transmissão o TCP Reno diminui a sua janela de congestionamento para metade do seu valor atual e não para 1, e continua a enviar pacotes, partindo da fase C*ongestion Avoidance.* Dessa maneira, ele previne o esvaziamento do canal de transmissão após a fase de *Fast Retransmit,* aumentando a utilização do canal.

A maior utilização do canal, em relação ao Tahoe e Reno, promovida pelo TCP New Reno é devida a uma modificação na sua fase de *Fast Recovery* tentando aproveitar ainda mais a banda passante. Se ocorrerem várias perdas numa mesma janela de congestionamento, o TCP New Reno não sai da *Fast Recovery*, ele somente sai dessa fase quando todos os pacotes que foram enviados durante essa fase forem reconhecidos.

O TCP Vegas novamente obtém o melhor desempenho. A sua maior utilização do canal é devida ao seu mecanismo pró-ativo de controle de congestionamento. Levando a uma baixa perda nesse enlace e uma melhor utilização do mesmo.

3.2.3 Variação no atraso da entrega dos pacotes (*Jitter*)

Nessa análise será mostrada como o TCP se comporta com tráfego CBR (*Constant Bit Rate*), ou seja, tráfego multimídia. Para isso é utilizada a topologia da Figura 17. Os Scripts utilizados nessa parte do estudo encontram-se no Apêndice A (topo3UDPCBR.tcl e topo3TahoeCBR.tcl).

No Script topo3TahoeCBR.tcl, a implementação TCP Tahoe foi simulada com o tráfego CBR. O tráfego CBR simula aplicações multimídias, como vídeo e áudio; ou seja, aplicações em que os pacotes não requerem confirmação. Essas são aplicações de tempo real, em que é melhor ignorar a perda de um pacotes ao invés do pacote ser re-transmitido, pois isso levaria a uma má reprodução da aplicação no receptor. Com essas aplicações, a variação do tempo na entrega(*jitter*) não pode ser grande, pois compromete a qualidade da imagem e vídeo.

Diferente das aplicações como correio eletrônico e transferência de arquivos (FTP), o tráfego CBR simula aplicações tolerantes à perda, pois podem tolerar alguma perda de dados no caminho e essas perdas ocasionais causam somente pequenas perturbações na recepção de áudio e vídeo. Nessas aplicações, os dados perdidos podem resultar em falhas durante a execução de um áudio/vídeo, por exemplo, porém isso não é considerado um grande prejuízo.

Ao simular o tráfego CBR para cada implementação do TCP foram obtidos os gráficos das variações no atraso das entregas dos pacotes recebidos pelo receptor (*jitter*). A Figura 19 ilustra o *jitter* da implementação do TCP Tahoe. Não são mostrados os gráficos obtidos com as outras implementações, pois são todos semelhantes.

Na Figura 19, pode-se ver que a variação na entrega é muito grande, variando de 0ms a 8ms. Isso ocorre porque o TCP fornece um serviço de transporte confiável e toda vez que ocorre uma perda, o pacote é re-enviado resultando em um grande variação de tempo na chegada dos pacotes no receptor . A entrega confiável de dados não é interessante para aplicações multimídias, pois o re-envio de um pacote causa no receptor uma não sincronização no momento da

reprodução, ocorrendo “paradas” na reprodução devido à espera dos pacotes que estão sendo re-transmitidos quando ocorre uma perda.

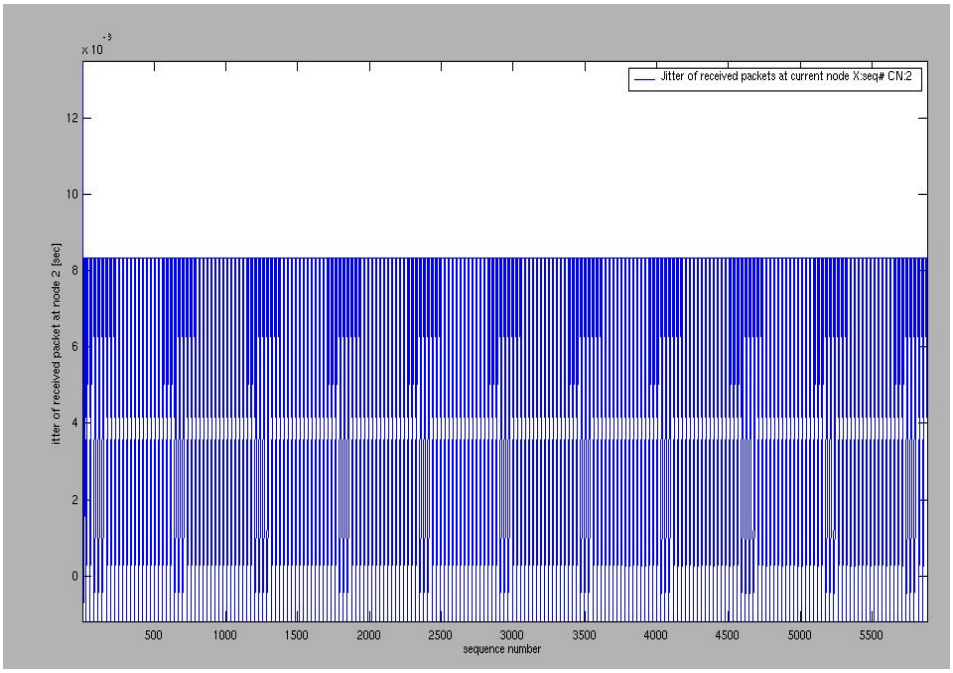

Figura 19. *Jitter* produzido pelo TCP Tahoe. (Extraída do *Trace Graph*)

Foi feita uma simulação com o mesmo cenário de rede, porém com o protocolo de transporte UDP ao invés do TCP, para tentar mostrar uma outra maneira de transmitir aplicações multimídias. A Figura 20 mostra o *jitter* resultante quando o tráfego CBR é transportado pelo protocolo UDP. O Script da simulação do tráfego CBR é mostrado no Apêndice A (topo3UDPCBR.tcl). Esse Script gera tráfego CBR que é levado pelo protocolo de transporte UDP.

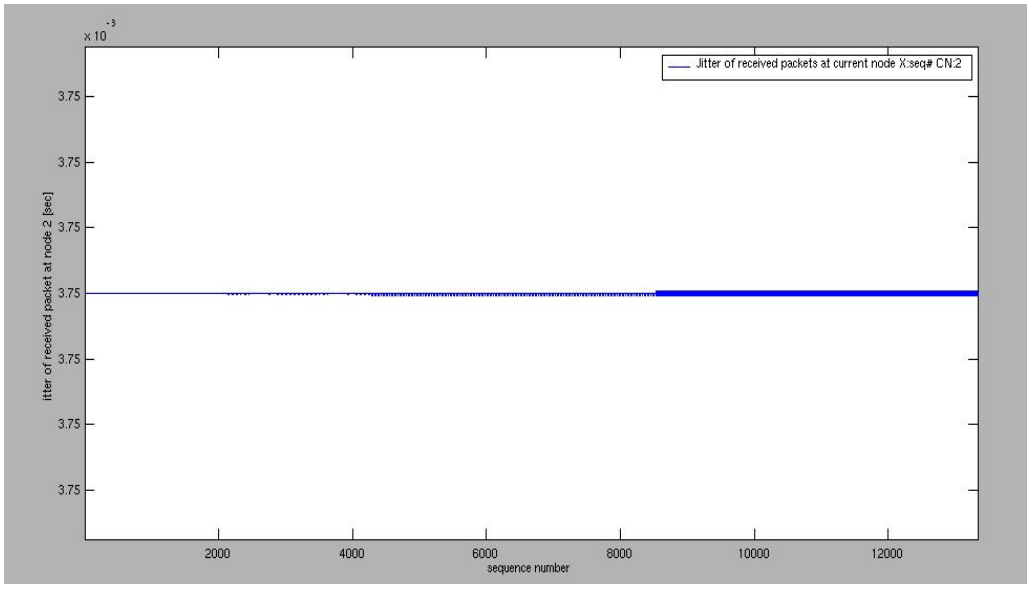

Figura 20. *Jitter* produzido pelo UDP. (Extraída do Trace Graph)

O protocolo UDP não é um protocolo que fornece serviço confiável à aplicação, sendo um protocolo bem mais leve e simplificado, porém ele é muito utilizado para o transporte de aplicações multimídias, pois as mesmas não requerem entrega confiável.

O UDP não inclui controle de congestionamento, portanto um transmissor pode transmitir dados no enlace na taxa que desejar. Apesar de nem todos os pacotes alcançarem o receptor, uma grande parte deles chegarão. Dessa maneira, as aplicações multimídias freqüentemente não requerem utilização do protocolo TCP, visto que essas aplicações são tolerantes à perda e não necessitam de um serviço de transporte confiável.

Na Figura 20 podemos ver que a variação no atraso dos pacotes recebidos (*jitter*) é bem pequena, chegando a no máximo de 3.75ms. Esse valor pode ser comparado com o valor do *jitter* que foi obtido quando o tráfego CBR foi levado pelo TCP (Figura 19). O *jitter* do tráfego CBR levado pelo TCP é bem maior, chegando a 8ms, e isso resulta numa má reprodução da aplicação no receptor. Esse atraso quer dizer que, alguns pacotes levados por TCP podem ter um atraso de 8ms e com UDP esse atraso é no máximo 3,75ms. Isso acontece porque o UDP não realiza re-transmissões, se um pacote for perdido é melhor ignorar uma perda a esperar que esse pacote seja re-transmitido. Essas re-transmissões realizadas pelo TCP pode levar ao receptor da aplicação a ter que esperar intervalos maiores para "ouvir" uma música ou "ver" um vídeo. Portanto, pudemos comprovar que o protoloco UDP tem um desempenho melhor quando carrega tráfego multimídia..

3.2.4 *Throughput*

Para esse estudo é utilizado o cenário da Figura 18, com as conexões apresentadas na Tabela 5. Nesse cenário é comparado o *throughput* do TCP Reno e Vegas e verificado também a compatibilidade entre eles, ou seja, o que ocorre quando duas versões do TCP são colocadas para compartilhar um canal congestionado. O Script dessa simulação está no Apêndice A (renovegasThroughput.tcl).

Tabela 5. Configuração das conexões

	Trajeto	**Versão do TCP**
Conexão 1	1 → 5	Reno
Conexão 2	2 → 4	Vegas

Nesse estudo só são comparadas as versões Reno e Vegas, pois as duas versões possuem mecanismos de reação ao congestionamento diferentes: o Reno utiliza perdas de pacotes para detectar a presença de congestionamento enquanto que o TCP Vegas detecta fazendo uma estimativa da banda disponível. Portanto é mostrado como as conexões Vegas competem com as conexões Reno.

Em [13][14] e nesse trabalho pôde-se comprovar que o TCP Vegas tem um melhor desempenho em relação a utilização do canal e em relação a perdas de pacotes. Porém, isso acontece quando existem somente conexões TCP Vegas nos enlaces. No entanto é significativo fazer um estudo de como o TCP Vegas se comporta quando disputa o canal com conexões de outra versão do TCP, como o TCP Reno. Outra justificativa para esse estudo é que na Internet, a maioria dos enlaces usam diferentes versões de TCP e, portanto é válido verificar como ocorre esse compartilhamento de enlaces.

Em [15] pode-se ver um estudo da compatibilidade Reno x Vegas, também através de uma análise heurística.

Assim como em [16] foram medidas a quantidade de ACK's que cada conexão recebeu

para se obter o *throughput*. Portanto, o campo *Throughput* na Tabela 6 é a quantidade de ACK's recebidos pela determinada implementação, pois esse número indica a quantidade real de pacotes que foram enviados e recebidos, ou seja, o *throughput* da conexão.

Para a comparação variou-se o tamanho do *buffer* para cada simulação, com valores de 4, 12, 15 e 20. A variação do tamanho do *buffer* é utilizada para mostrar como as duas implementações reagem a essas mudanças.

O TCP Vegas utilizado na conexão 2, foi utilizado com a sua configuração *default*, ou seja, $\alpha = 1$ e $\beta = 3$. Outros estudos do desempenho do Vegas com outras configurações dos seus parâmetros α e β, podem ser vistos e comentados em [15].

Tabela 6. *Throughput* Reno X Vegas

Buffer	**Throughput Reno T(r)**	**Throughput Vegas T(v)**	**T(r) / T(v)**
4	321	386	0,832
12	319	285	1,111
15	260	83	3,132
20	195	36	5,416

A Tabela 6 mostra o resultado das simulações. Quando o tamanho do *buffer* é pequeno a conexão do TCP, Vegas consegue obter uma maior utilização da banda compartilhada, quando compartilha o canal com o TCP Reno. Porém, quando o tamanho do *buffer* aumenta, o TCP Reno não compartilha a banda de forma justa com o TCP Vegas. Podemos ver que com o tamanho do *buffer* 20, o Reno chega a ter um *throughput* 5 vezes maior que o TCP Vegas. Isso ocorre devido aos seus diferentes mecanismos de controle de congestionamento. O Reno tem uma reação agressiva e tenta utilizar todos os *buffers* disponíveis, não deixando nenhum *buffer* para outras conexões. Já o Vegas, por ter um mecanismo conservativo, ocupa pouco espaço de *buffer*, pois o Reno utiliza muitos *buffers* e o Vegas interpreta isso como um sinal de congestionamento, ou seja, como se não existisse banda disponível para enviar mais pacotes. Dessa maneira o Vegas obtém menor *throughput* quando disputa o canal com o TCP Reno.

Apesar do TCP Reno ter um maior *throughput* isso resulta em algumas conseqüências não muito boas para o TCP Reno. Como ele utiliza todos os *buffers* que estiverem disponíveis, seus pacotes, quando chegam nas filas, apresentam um atraso alto por causa do tamanho das filas no roteador. Já os pacotes levados por conexões com o TCP Vegas não se acumulam na rede e possuem atrasos bem menores.

Conclusões e Trabalhos Futuros

O tema Redes de Computadores possui um imenso conjunto de oportunidades de pesquisa. A utilização de simuladores para testar e validar os modelos propostos é imprescindível em algumas as áreas.

Com o objetivo de obter uma análise comparativa do desempenho do TCP, este trabalho abordou um estudo sobre o controle de congestionamento realizado pelas implementações do TCP Tahoe, Reno, New Reno e Vegas, sobre diversas condições. O TCP desde seu surgimento até o momento é o protocolo mais utilizado em computadores que desejam se comunicar, por propiciar um serviço importante de transmissão confiável de dados através de uma conexão entre duas máquinas.

Embora seu funcionamento básico esteja padronizado, o TCP apresenta diversas variações e aprimoramento visando a um melhor aproveitamento dos recursos de redes. O comportamento do TCP varia de acordo com os algoritmos do controle de congestionamento que implementam. Em redes de alto desempenho, o TCP funciona perfeitamente e traz muitas vantagens, pois sua janela de congestionamento é incrementada exponencialmente e mesmo que ocorra uma perda esporádica o TCP se recupera bem. Em redes congestionadas, a alta freqüência nas perdas de pacotes levam o TCP a reduzir a sua janela para 1 e começar com a *Slow Start* várias vezes. Para que o TCP possa trabalhar beneficiando ao máximo as redes, é muito melhor ter roteadores com bastante memória para poder processar todos os pacotes, mesmo que de forma mais lenta, para que não haja descarte. Esse descarte leva o TCP à *Slow Start* (CWND =1) e o atraso teoricamente é resolvido com a presença da *Fast Retransmit* e *Fast Recovery.*

Pôde-se ver que o TCP Tahoe, por ser a versão mais simples do TCP, para se reagir ao congestionamento na rede, essa versão leva algumas desvantagens em relação às outras nos aspectos estudados, visto que a mesma não possui nem mesmo a fase de *Fast Recovery* não se recuperando bem após ocorrer perdas.

Com o estudo pôde-se confirmar que o tráfego multimídia, por exemplo, áudio e vídeo devem ser levados preferencialmente por UDP já que o mesmo promove menor variação no atraso da entrega dos pacotes.

Uma conclusão que se pode chegar é que as implementações Tahoe, Reno e New Reno por terem seu mecanismo de controle de congestionamento baseado em perdas de pacotes, não apresentam um bom desempenho em rede com alto grau de perdas, uma rede sem fio por exemplo que tem um alto grau de perdas, porém isso não significa necessariamente a presença de

congestionamento e sim perdas que são inerentes ao próprio meio. Por terem sido desenvolvidas para redes fixas, essas versões do TCP sempre deduzem que perdas implicam em congestionamento.

O TCP Vegas apresenta algumas modificações mais profundas para melhorar o desempenho em relação às abordagens do Reno e New Reno e Tahoe. Ele tenta evitar o congestionamento de forma pró-ativa, diferentemente do Tahoe e Reno que reagem ao congestionamento quando ele ocorre. Por ter essa característica, o Vegas, quando em ambientes que só existem conexões TCP Vegas obtém um *throughput* maior que o Reno e o Tahoe. Quando coexistem conexões Reno e Vegas e as duas versões compartilham os *buffers* dos roteadores, o TCP Reno por ter um comportamento mais progressivo em comparação com o comportamento conservativo do Vegas causa um injusto compartilhamento desses *buffers*. O TCP Reno acaba "roubando" banda do TCP Vegas. O estudo detalhado do desempenho do TCP Vegas pode ser visto em [14].

Como trabalhos futuros pode-se realizar um estudo com redes com maior número de nós, com outras características e com outros tipos de tráfegos, já que neste trabalho, para simplificar, foram construídas topologias simples mas que são representativas para o estudo em questão.

Pode-se também fazer um estudo com outras versões do TCP e com outros parâmetros relacionados também ao congestionamento, pois este trabalho abrangeu apenas as versões Tahoe, Reno, New Reno e Vegas.

A principal sugestão seria utilizar este trabalho como auxílio no estudo das implementações apresentadas a fim de propor até mesmo o desenvolvimento de novas implementações que corrijam as limitações que as implementações estudadas aqui possuam.

Bibliografia

[1] ALLMAN,M., STEVENS, W., *Transmission Control Protocol*. IETF Request for Comments (RFC) 793, Setembro,1981.

[2] PODURI, K., NICHOLS, K., *Simulation Studies of Increased Initial TCP Window Size*, Request for Comments (RFC) 2415, 1998.

[3] JACOBSON, v., "*Congestion Avoidance and Control*,". *SIGCOMM Symposium on communications Architectures and Protocols*, pag. 314–329, 1988.

[4] KARN, P., PATRIDGE, C., *"Improving Round-Trip Time Estimates in Reliable Transport Protocols*, "Computer Communication Review, vol. 17, no. 5, pag. 2-7

[5] STEVENS, W., *TCP Slow Start , Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, IETF, Request for Comments (RFC) 2001, 1997.

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

[7] TANENBAUM, S. A., *Redes de Computadores*, Terceira Edição, Editora Afiliada.

[8] FALL K.; FLOYD S. *Simulation-based Comparisons of Tahoe, Reno and SACK TCP*. ACM Computer Communication Review, volume 26, pag. 270-280, Julho de 1996.

[9] JACOBSON, V., "*Modified TCP Congestion Avoidance Algorithm*,". Technical report, 30 Apr. 1990. Email to the end2end-interest Mailing List, URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[10] CHUI, D., JAIN, R., *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*. Computer Networks and ISDN Systems, 1989.

[11] HASEGAWA, Go., MURATA, M., MIYAHARA. H., *Fairnesd and Stability of Congestion Control Mechanisms of TCP*.IEEE, Japão, 1999.

[12] FLOYD S.; HEANDERSNO T. *The New Reno Modification to TCP´s Fast Recovery Algorithm*. IETF Request for Comments (RFC) 2582, 1999.

[13] AHN, J.S.,DANZIG, P. B.,LIU, Z.,YAN, L., "*Experience with TCP Vegas: Emulation*

and experiment," in *Proc. SIGCOMM '95 Symp.,* Agosto de 1995.

[14] BRAKMO, L., PETERSON, L., *TCP Vegas: End to End Congestion Avoidance on a Global Internet*. IEEE Journal on Selected Areas in Communication, Vol 13, No. 8 (Outubro 1995) pag. 1465-1480.

[15] FENG, W., VANICHPUN, s., *Enabling compatibility between TCP Renoe Vegas*, Proceedings of the 2003 Symposium on Application and the Internet. IEEE, pag.1872-9, 2003.

[16] JEONGHOON M.; RICHARD J. L.; ANANTHARAM V.; WALRAND J. *Analysis and Comparison of TCP Reno and Vegas.* In Proceedings of lEEE ICC '97, pág. 495-499, June 1997.

[17] BAJAJ, S., BRESLAU, L., ESTRIN, D., FAAL, K., at all., "*Improving Simulation for Network Research*", USC Computer Science Departament Technical Report

[18] FILHO, P., J., F., *Introdução à modelagem e Simulação de Sistemas*, Visual Books Editora. Novembro, 2001.

[19] JAIN, R., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, Segunda edição.

[20] PEGDEN, C.D., SHANNON, R.E., SADOWSKI, R.P. *Introduction to Simulation Using SIMAN*. 2 ed. New York: McGraw-Hill, 1990.

[21] FILHO, P., J., F., *Introdução à modelagem e Simulação de Sistemas*, Visual Books Editora. Novembro, 2001.

[22] MCCANNE,S., FLOYD,S., *NS (Network Simulator).*

[23] KESHAV, S., *REAL: a network simulator*. Tech. Rep. 88/472, University of California, Berkeley, Dezembro de 1988. (real)

[24] ESTRIN, D., HANDLEY, M., HEIDMANN, J., MC-CANNE, S., XU, Y., YU, H. *Network visualization with the VINT network animator nam*. Tech. Rep. 99-703, University of Southern California, Março. 1999.

[25] BANKS, J. e CARSON, J.S., *Discrete-Event System Simulation*. Prentice-Hall, Englewood Cliffs, NJ, 1996.

[26] OTCL (MIT Object Tool Command Language). Disponível na Internet via URL: http://otcl-tclcl.sourceforge.net/otcl/. Útimo acesso em 27/04/05

[27] BROWN, Brian. *Introduction to awk*. Disponível na Internet via URL: http://allman.rhon.itam.mx/dcomp/awk.html.

[28] MALEK, J. *NS-2 trace files analyzer*. Disponível na Internet. Http://www.geocities.com/tracegraph. Último acesso em 15/06/05.

[29] FALL, K. e VARADHAN, K. *The NS Manual (formely NS Notes and Documentation)*. The VINT Project, 2002.

[30] SERGIO C., COSTA R., *Análise do Protocolo TCP para Redes de Alta Velocidade e Redes Móveis*, PUC-RJ, 2004.

[31] VERES, A., MIKLOS, B., *The Chaotic Nature of TCP Congestion Control*. In *Proc*. Of IEEE INFOCOM 2000, Março de 2000.

Apêndice A

Scripts utilizados nas simulações

topologia3nostahoe.tcl

```
set ns [new Simulator]

$ns color 0 blue

set nf [open out.nam w]
$ns namtrace-all $nf

#Abrir arquivo de trace
#set tf [open topo3tahoe.tr w]
#$ns trace-all $tf

set n0 [$ns node]
set r1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $r1 10Mb 1ms DropTail
$ns duplex-link $r1 $n2 1Mb 1ms DropTail

$ns duplex-link-op $n0 $r1 orient right
$ns duplex-link-op $r1 $n2 orient right

#limitar a fila no roteadores
$ns queue-limit $r1 $n2 5
$ns queue-limit $n2 $r1 5

set tcp0 [new Agent/TCP]
$tcp0 set fid_ 0
$tcp0 set window_ 20
$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0

$ns connect $tcp0 $sink0


set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
proc finish {} {
      global ns nf nt #tf
      $ns flush-trace
      #pacotes perdidos tahoe
            exec awk \{\{ if ($1=="d" && $3==2 && $4==3 && $9==1.0 &&
                  $10=3.1)print$2, 44}} tahoe_util > perdas_tahoe

      close $nf
      #fechar aquivo de trace
      close $tf
      #close $nt
      exec nam out.nam &
      exit 0
}

$ns at 0.0 "$ftp0 start"
$ns at 50.0 "$ftp0 stop"
$ns at 50.1 "finish"

$ns run
```

topologia3nosUtiliz.tcl

```
set ns [new Simulator]

$ns color 0 blue

set nf [open out.nam w]
$ns namtrace-all $nf

#Abrir arquivo de trace
set nt [open tahoe_util.tr w]
$ns trace-all $nt

set n0 [$ns node]
set r1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $r1 10Mb 1ms DropTail
$ns duplex-link $r1 $n2 1Mb 1ms DropTail

$ns duplex-link-op $n0 $r1 orient right
$ns duplex-link-op $r1 $n2 orient right

#limitar a fila no roteadores
$ns queue-limit $r1 $n2 5
$ns queue-limit $n2 $r1 5

set tcp0 [new Agent/TCP]
$tcp0 set fid_ 0
$tcp0 set window_ 20
$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0

$ns connect $tcp0 $sink0
```

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

proc finish {} {
      global ns nf nt #tf
      $ns flush-trace
      #armazena em  "tahoeutil.tr" ; os pacotes q passaram pelo enlace 1 -> 2
           exec awk {{ if ($1=="-" && $3==1 && $4==2 && $9==0.0 &&
                  $10=2.0)print$2, 44}}  tahoe_util.tr > tahoeutil

      close $nf
      #fechar aquivo de trace
      #close $tf
      close $nt
      exec nam out.nam &
      exit 0
}

$ns at 0.0 "$ftp0 start"
$ns at 50.0 "$ftp0 stop"
$ns at 50.1 "finish"

$ns run
```

topo3TahoeCBR.tcl

```
set ns [new Simulator]

$ns color 0 blue


set nf [open out.nam w]
$ns namtrace-all $nf

#Abrir arquivo de trace
set tf [open topo3tahoe_CBR.tr w]
$ns trace-all $tf

set n0 [$ns node]
set r1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $r1 10Mb 1ms DropTail
$ns duplex-link $r1 $n2 1Mb 1ms DropTail


$ns duplex-link-op $n0 $r1 orient right
$ns duplex-link-op $r1 $n2 orient right


set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0

$ns connect $tcp0 $sink0

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $tcp0
```

```
proc finish {} {
      global ns nf tf
      $ns flush-trace

      close $nf
      #fechar aquivo de trace
      close $tf

      exec nam out.nam &
      exit 0
}

$ns at 0.0 "$cbr0 start"
$ns at 50.0 "$cbr0 stop"
$ns at 50.1 "finish"

$ns run
```

topo3UDPCBR.tcl

```
set ns [new Simulator]

$ns color 0 blue


set nf [open out.nam w]
$ns namtrace-all $nf

#Abrir arquivo de trace
set nt [open topo3_UDP.tr w]
$ns trace-all $nt

set n0 [$ns node]
set r1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $r1 10Mb 1ms DropTail
$ns duplex-link $r1 $n2 1Mb 1ms DropTail


$ns duplex-link-op $n0 $r1 orient right
$ns duplex-link-op $r1 $n2 orient right

#limitar a fila no roteadores
$ns queue-limit $r1 $n2 5
$ns queue-limit $n2 $r1 5

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set null0 [new Agent/Null]
$ns attach-agent $n2 $null0

$ns connect $udp0 $null0

set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

```
proc finish {} {
      global ns nf nt #tf
      $ns flush-trace
      close $nf
      #fechar aquivo de trace
      #close $tf
      close $nt
      exec nam out.nam &
      exit 0
}

$ns at 0.0 "$cbr0 start"
$ns at 50.0 "$cbr0 stop"
$ns at 50.1 "finish"

$ns run
```

renovegasThroughput.tcl

```
set ns [new Simulator]

$ns color 0 blue
$ns color 1 red

set nf [open out.nam w]
$ns namtrace-all $nf

set nt [open ack_10reno_vegas.tr w]
$ns trace-all $nt

set n0 [$ns node]
set n1 [$ns node]
set r2 [$ns node]
set r3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n0 $r2 10Mb 2ms DropTail
$ns duplex-link $n1 $r2 10Mb 3ms DropTail
$ns duplex-link $r2 $r3 1.5Mb 1ms DropTail
$ns duplex-link $r3 $n4 10Mb 4ms DropTail
$ns duplex-link $r3 $n5 10Mb 5ms DropTail

$ns duplex-link-op $n0 $r2 orient right-down
$ns duplex-link-op $n1 $r2 orient right-up
$ns duplex-link-op $r2 $r3 orient right
$ns duplex-link-op $r3 $n4 orient right-down
$ns duplex-link-op $r3 $n5 orient right-up

#limitar a fila entre os roteadores
$ns queue-limit $r2 $r3 10
$ns queue-limit $r3 $r2 10

set tcp0 [new Agent/TCP/Reno]
$tcp0 set fid_ 0
$tcp0 set window_ 20
$ns attach-agent $n0 $tcp0
```

```
set sink0 [new Agent/TCPSink]
$ns attach-agent $n4 $sink0

$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

set tcp1 [new Agent/TCP/Vegas]
$tcp1 set fid_ 1
$tcp1 set window_ 20
$ns attach-agent $n1 $tcp1

set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2

$ns connect $tcp1 $sink2

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

proc finish {} {
      global ns nf nt
      $ns flush-trace
      #armazena em  "ackreno.tr" os acks de 4 -> 0
            exec awk {{ if ($1=="r" && $3==2 && $4==1 && $9==5.0 &&
                  $10=1.0)print$2, 44}}  ack_10reno_vegas.tr > ack10renoV
      #armazena em  "ackvegas.tr" os acks de 4 -> 0
            exec awk {{ if ($1=="r" && $3==2 && $4==0 && $9==4.0 &&
                  $10=0.0)print$2, 44}}  ack_10reno_vegas.tr > ack10vegas
      close $nf
      #fechar aquivo de trace

      close $nt
      exec nam out.nam &
      exit 0
}


$ns at 0.0 "$ftp0 start"
$ns at 0.1 "$ftp1 start"
$ns at 20.0 "$ftp0 stop"
$ns at 20.1 "$ftp1 stop"
$ns at 20.2 "finish"

$ns run
```