UERJ – UNIVERSIDADE DO ESTADO DO RIO DE JANEIRO
CTC – CENTRO DE TECNOLOGIA E CIÊNCIAS
FEN – FACULDADE DE ENGENHARIA
DEPARTAMENTO DE ELETRÔNICA E TELECOMUNICAÇÕES

PROJETO DE GRADUAÇÃO

AVALIAÇÃO EXPERIMENTAL DE PROTOCOLOS DE TRANSPORTE EM REDES GIGABIT

Autor: Wallace de Souza Espíndola
Orientador: Prof. Marcelo Gonçalves Rubinstein
Coordenador da Disciplina: Prof. Joel Martins de Medeiros

Maio de 2007

ESPINDOLA, WALLACE DE SOUZA

Protocolos de Transporte em Redes Gigabit [Rio de Janeiro] 2007.

IX, 68p. 29,7cm (FEN/UERJ, Engenheiro, Engenharia Elétrica – ênfase em Telecomunicações, 2007).

Projeto de Graduação – Universidade do Estado do Rio de Janeiro – UERJ.

1. Redes Gigabit
2. Redes de Comunicação
3. Redes de Alta Velocidade
4. Protocolos de Transporte

I. FEN/UERJ II. Título (série)

Dedicatória

Ao meu filho querido ofereço este trabalho como exemplo de dedicação, disciplina e persistência, por tudo que tive que abnegar em prol de um objetivo a ser alcançado. Como prova do que trabalho e esforço juntos podem realizar.

Agradecimentos

Primeiramente agradeço a Deus, por ter me guiado em cada passo da vida, tornando-me capaz de chegar até aqui.

Aos meus pais, que muito se sacrificaram e se esforçaram para que eu fosse vencedor, e por todo o apoio que me ofereceram durante toda a vida, fazendo com que eu conseguisse até esse momento chegar.

À minha esposa e filho, pela paciência e compreensão pelos muitos momentos de ausência da convivência familiar.

Ao Prof. Marcelo Gonçalves Rubinstein, por ter sido sempre solícito quando necessário, e por ter me orientado de forma tão competente e profissional.

Ao Prof. Luís Henrique Maciel Kosmalski Costa pela presença na banca avaliadora e pela detalhada revisão do trabalho.

Ao amigo Antônio João do Nascimento Dantas, grande conhecedor do Linux, pelo auxílio oferecido nas configurações e nos scripts de teste.

Aos amigos e familiares, que ora apoiaram, ora animaram, e nunca me deixaram desistir no meio do caminho.

A todos aqueles que fazem parte da minha vida e que, de uma forma ou outra, me incentivaram e torceram pelo meu sucesso.

"Se a presença da eletricidade pode se tornar visível em qualquer parte de um circuito, não vejo razão por que a inteligência não possa ser transmitida instantaneamente pela eletricidade".

Samuel F. B. Morse, inventor do código Morse e do telégrafo.

"O que as redes de ferrovias, rodovias e canais foram em outra era, as redes de telecomunicações, informações e computação... são hoje em dia".

Bruno Kreisky, chanceler austríaco.

"... atualmente a questão é como acessar gigabits de informações através de linhas de velocidade ridícula".

J.C.R. Licklider, idealizador de importantes conceitos em computação e redes.

I. Resumo

O protocolo TCP (*Transmission Control Protocol*) tem sido usado há mais de vinte anos na Internet, e apresenta certas limitações quando usado em redes de alta velocidade. Neste trabalho são abordados os efeitos de protocolos de transporte na transmissão de dados em redes gigabit. Este trabalho apresenta um estudo comparativo entre o mais usado protocolo de transporte confiável e orientado à conexão, o TCP, e dois de seus possíveis sucessores o HSTCP (*High-Speed TCP*) e o STCP (*Scalable TCP*).

Inicialmente, as principais características do TCP são apresentadas, assim como as do HSTCP e do STCP. As principais diferenças entre eles são mostradas. Então são realizados testes comparativos de desempenho dos protocolos selecionados. Os resultados são analisados detalhadamente, levando em consideração as vazões alcançadas pelos protocolos em função do atraso e perdas.

No geral, o STCP apresentou o melhor desempenho nos cenários de média e alta latência, enquanto o HSTCP foi melhor nos cenários de baixa latência. Os protocolos HSTCP e STCP demonstraram serem capazes de alcançar vazões muito superiores àquelas alcançadas pelo TCP, chegando a ser entre 2 a 3 vezes maiores, provando que podem oferecer uma melhor utilização da largura de banda disponível nas redes de alta velocidade.

Palavras Chave:

– TCP

– HSTCP

– STCP

– Redes Gigabit

– Redes de Alta Velocidade

II. Abstract

The TCP protocol (*Transmission Control Protocol*) has been used on the Internet for over than 20 years, and it presents certain limitations when used in high-speed networks. This work discusses the effects of transport protocols in data transmission on gigabit networks. It also presents a comparative study between the most used reliable and connection-oriented transport protocol, the TCP, and two of its possible successors, the HSTCP (High-Speed TCP) and the STCP (Scalable TCP).

Initially, main TCP features are presented, as well as the HSTCP and STCP features. The main differences between them are showed. Then comparative performance tests of the selected transport protocols are executed. The results are analyzed, taking into consideration the throughput reached by the protocols in function of delay and loss.

In general, STCP had the best performance in medium and high latency scenarios, while HSTCP was the best one on low latency scenarios. The HSTCP e STCP protocols have demonstrated being capable of reaching throughputs much higher than that reached by TCP, with values between 2 to 3 times beyond TCP ones, evidencing that they are able to provide better utilization of available bandwidth on high-speed networks.

Keywords:

– TCP

– HSTCP

– STCP

– Gigabit Networks

– High-Speed Networks

III. Lista de Figuras

IV. Lista de Tabelas

V. Lista de Acrônimos

ACK	*Acknowledgement*
AIMD	*Additive-Increase, Multiplicative-Decrease*
CA	*Congestion Avoidance*
DNS	*Domain Name Service*
DSACK	*Duplicate Selective Acknowledgment*
FAST-TCP	*Fast Active-queue-management Scalable TCP*
FIN	*Finalize*
FTP	*File Transfer Protocol*
GPL	*General Public License*
HSTCP	*High-Speed TCP*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Secure Hypertext Transfer Protocol*
IP	*Internet Protocol*
MSS	*Maximum Segment Size*
MTU	*Maximum Transmission Unit*
NACK	*Negative Acknowledgement*
NFS	*Network File System*
POP3	*Post Office Protocol version 3*
PSH	*Push*
QoS	*Quality of Service*
RFC	*Request For Comments*
RST	*Reset*
RTT	*Round Trip Time*

SACK	*Selective Acknowledgment*
SMTP	*Simple Mail Transfer Protocol*
SS	*Slow Start*
SSH	*Secure Shell*
STCP	*Scalable TCP*
SYN	*Synchronize*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
URG	*Urgent*
WAN	*Wide Area Network*
XCP	*eXplicit Congestion Control Protocol*

SUMÁRIO

1. Introdução

O mundo moderno não pode mais viver sem se comunicar, e as redes de comunicação desempenham um papel fundamental no cenário mundial atual. O enorme crescimento das demandas de informação levou a uma grande necessidade de elevação das taxas de transmissão de dados.

As redes gigabit foram criadas com o objetivo de resolver o problema atual de limitação de banda nos *backbones* de comunicação e redes de alto tráfego, oferecendo uma banda passante extremamente larga, e, por conseguinte, a possibilidade de alcançar altíssimas taxas de dados.

Por outro lado, a existência destas redes também gera uma questão: se estariam as tecnologias atualmente em uso evoluídas ao ponto de suportar tais altas taxas de dados. No presente momento a resposta certamente seria “não”.

1.1. Objetivo

Este trabalho tem como objetivo estudar e avaliar o desempenho do protocolo de transporte mais comumente utilizado atualmente, o TCP (*Transmission Control Protocol*), testando de forma experimental o seu desempenho. Para tanto, serão levados em consideração os mecanismos de controle de fluxo e congestionamento do TCP, comparando este com outros protocolos alternativos adaptados às redes de alta velocidade, os quais são candidatos a serem seu sucessor na disputa pelo alcance de altas taxas de transmissão de dados.

O protocolo TCP foi concebido na década de 70 e tem sido utilizado há muitos anos nas redes de computadores e principalmente na Internet. O controle de congestionamento do TCP é constituído por diferentes mecanismos. Em estado estacionário, o seu mecanismo de controle de congestionamento é baseado principalmente no algoritmo AIMD (*Additive-Increase, Multiplicative-Decrease*). Esse algoritmo de prevenção de congestionamento faz com que a vazão de uma conexão TCP aumente linearmente a cada reconhecimento positivo (ACK) recebido e decresça exponencialmente a cada segmento perdido.

O mecanismo de controle de congestionamento trabalha para adaptar as taxas de transmissão das conexões TCP ao congestionamento da rede. No entanto, este mesmo mecanismo gera alguns problemas de desempenho para redes de alta velocidade, principalmente quando o produto banda passante x atraso é grande, pois a lentidão do aumento da janela de congestionamento aliada à diminuição excessiva no tamanho desta mediante a perda de segmentos provoca uma baixa utilização da banda passante disponível no enlace.

Muitos protocolos de transporte voltados para redes gigabit estão sendo propostos atualmente, visando o desenvolvimento de novas aplicações que envolvam a transmissão confiável de grande quantidade de dados e também uma melhor utilização das capacidades das redes atuais. Dentre os principais protocolos propostos, destacam-se o XCP (*eXplicit Congestion Control Protocol*) [1], o HSTCP (*High-Speed TCP*) [2], o STCP (*Scalable TCP*) [3] e o FAST-TCP (*Fast Active-queue-management Scalable TCP*) [4]. Alguns protocolos, como o protocolo XCP, exigem modificações nos roteadores, enquanto outros, como o HSTCP, o FAST-TCP e o STCP são implementados nos sistemas finais. O STCP é baseado no mesmo tipo de modificação proposta pelo HSTCP, que por sua vez é baseado no TCP tradicional.

Neste trabalho são avaliados dois dos novos protocolos propostos, o HSTCP e o STCP, comparando-os diretamente com o TCP, o protocolo de transporte mais utilizado atualmente na internet e nas grandes redes corporativas e institucionais.

1.2. Motivação

Com o advento das redes gigabit, diversas aplicações tais como as aplicações multimídias distribuídas de tempo real e as aplicações de grade computacional de larga escala estão sendo viabilizadas [5]. De modo a utilizar essas e outras aplicações de grande consumo de banda em redes de longa distância como a Internet, grandes redes corporativas e outras redes com alto tráfego, a escolha de um protocolo de transporte adequado se torna de vital importância.

Uma rede gigabit traz consigo a possibilidade do uso de novas aplicações, bem como melhoria na utilização das aplicações já existentes, e também melhor distribuição dos recursos de rede. Por outro lado, tal recurso implica em grandes

volumes de dados transitando numa rede, resultando em grande impacto no gerenciamento de tráfego, nos mecanismos de controle de fluxo e congestionamento dos protocolos, na oferta e gerenciamento de Qualidade de Serviço (QoS), e na escalabilidade de recursos.

A motivação deste trabalho se dá pelas grandes possibilidades do uso das redes gigabit e pelo fato de hoje em dia não haver ainda uma solução definitiva para o problema de controle de fluxo e congestionamento do protocolo TCP, o que mantém momentaneamente em aberto a solução a ser adotada, sendo esta muito provavelmente definida em breve.

Desta forma, um dos objetivos deste projeto é avaliar o desempenho de alguns dos protocolos citados, através de experimentos, a fim de definir o protocolo que mais se adequa às redes de alta velocidade. Assim sendo, a definição do novo protocolo de transporte se torna de fundamental importância para viabilizar a utilização da banda passante de maneira mais eficiente em tais redes, possibilitando a difusão em larga escala de serviços com alta exigência de banda, assim como o fim dos gargalos nos *backbones* das grandes redes e da Internet.

1.3. Organização do Trabalho

O presente trabalho está dividido em duas partes.

A primeira parte é baseada em uma pesquisa que abrange os fundamentos teóricos de dois dos novos protocolos de transporte sugeridos, HSTCP e STCP, assim como um estudo mais detalhado do protocolo TCP, que é o ponto de partida do problema.

A segunda parte está voltada para testes reais de funcionamento, visando avaliar o desempenho dos três protocolos citados, utilizando-se para tanto dois microcomputadores conectados que efetuarão transmissões de dados a altas taxas.

Assim são constatadas as diferenças e similaridades entre tais protocolos, visando à indicação de um protocolo mais adequado a ser utilizado nas redes gigabit e outras redes de alta velocidade.

Este trabalho está organizado da seguinte forma: o Capítulo 2 descreve o funcionamento geral do protocolo TCP, mostrando suas principais características de funcionamento e seus mecanismos de controle de fluxo e de congestionamento. A seguir, no Capítulo 3, um estudo do protocolo HSTCP é apresentado, com descrição de suas características gerais e seu mecanismo de controle de congestionamento, levando em consideração as diferenças deste para o protocolo TCP. No Capítulo 4 são mostradas as funcionalidades do protocolo STCP, seu mecanismo de controle de congestionamento e as principais diferenças entre este e o protocolo TCP. No Capítulo 5 são descritos os testes realizados com os três protocolos alvo deste estudo, os resultados obtidos são apresentados, e uma análise detalhada dos dados gerados é realizada. Por fim, no Capítulo 6, conclusões gerais sobre o estudo feito e os testes realizados são apresentadas, e são também levantadas propostas de estudo em trabalhos futuros.

2. O Protocolo TCP

2.1. Introdução ao TCP

O protocolo TCP (*Transmission Control Protocol*) [6] tem sido utilizado há mais de vinte anos na Internet. Ele foi criado por dois pesquisadores chamados Vinton Cerf e Robert Khan na década de 70, antes mesmo da invenção dos PCs e estações de trabalho, antes das redes Ethernet e de outras tecnologias de redes locais, e logicamente, antes da *Web*. A princípio ele foi vinculado ao protocolo IP (*Internet Protocol*) [7], apresentado como TCP/IP, um protocolo único que visava uma ampla interoperabilidade entre redes, protocolos e *hosts* (hospedeiros). Mais tarde entendeu-se a necessidade de considerar o TCP e o IP como protocolos separados, com atribuições distintas, sendo então tratados independentemente. O modelo TCP/IP [8] é a base da comunicação na Internet de hoje, sendo globalmente utilizado.

"*O TCP é um protocolo de transferência confiável de dados que é implementado sobre uma camada de rede fim a fim não confiável (IP)*" [9].

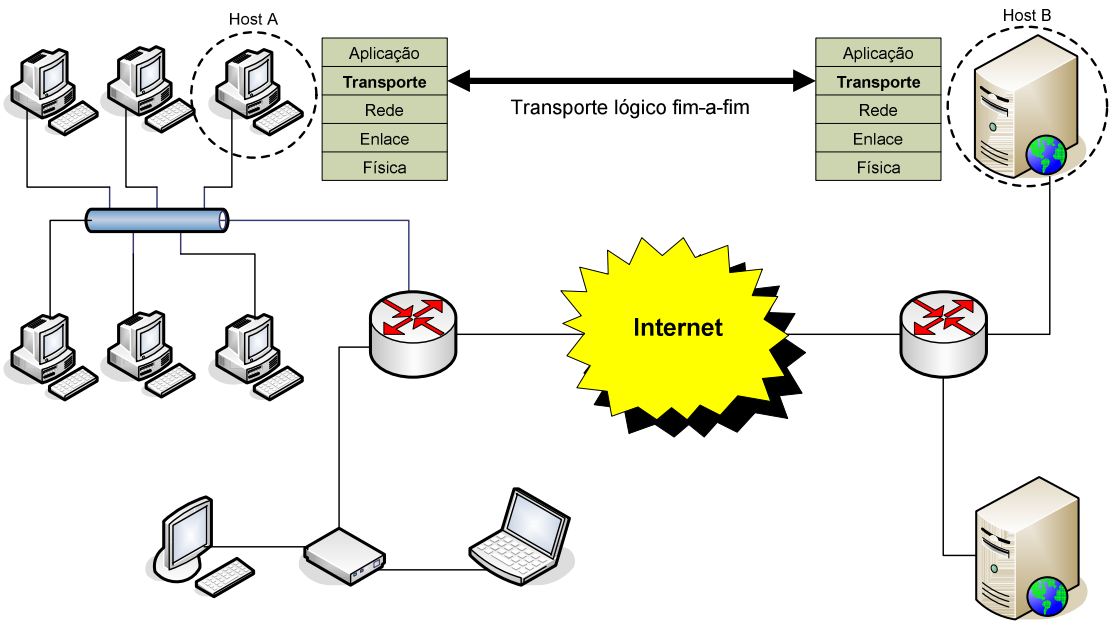

Figura 1: Exemplo de transporte lógico fim-a-fim usando o TCP.

O objetivo do protocolo TCP é fornecer um serviço de transferência confiável de dados entre *hosts* remotos por meio de um transporte lógico fim a fim. Com essa finalidade, ele possui uma série de rotinas e processos, que são transparentes ao usuário e aos protocolos das camadas adjacentes. A Figura 1 mostra um exemplo de conexão lógica fim a fim e conexão física entre dois *hosts* remotos A e B.

Muitas aplicações largamente utilizadas na Internet atualmente usam o TCP como seu protocolo de transporte, visto que este oferece um serviço orientado à conexão e também um serviço de transferência confiável de dados, como pode ser visto nos exemplos listados na Tabela 1.

Tabela 1: Aplicações populares da Internet, seus protocolos de camada de aplicação e protocolos de camada de transporte subjacentes [9, com modificações].

Aplicação	Protocolo de Aplicação	Protocolo de Transporte
Correio eletrônico	SMTP [10]	TCP
Acesso terminal remoto	Telnet [11]	TCP
Web (WWW)	HTTP [12]	TCP
Transferência de arquivos	FTP [13]	TCP
Servidor remoto de arquivos	NFS [14]	TCP ou UDP
Multimídia em tempo real	Quase sempre proprietário (Ex.: *Real Networks*)	TCP ou UDP

As características técnicas do protocolo TCP são descritas de modo mais aprofundado nas seções seguintes.

2.2. A Estrutura do Segmento TCP

A unidade de transferência entre o TCP de duas máquinas é chamado um segmento. O TCP vê o fluxo de dados como uma seqüência de bytes, que ele divide em segmentos para a transmissão. Os segmentos são trocados para estabelecer conexões, transferir dados, enviar reconhecimentos e fechar conexões.

A estrutura do segmento TCP consiste basicamente de campos de cabeçalho e campo de dados (que provêm da aplicação da camada superior).

O comprimento de cabeçalho típico é de 20 bytes. O restante é ocupado pelos campos de opções e de dados. O tamanho total do segmento TCP pode ser variável, dependendo do que se encontra nos campos de opções e de dados.

O que limita o campo de dados é o MSS (*Maximum Segment Size* – Tamanho Máximo de Segmento). Ele define o tamanho máximo da mensagem vinda da aplicação. Se quisermos transferir um arquivo grande, por exemplo, uma imagem, o arquivo de imagem (dado) será fragmentado em tantas partes quanto as forem necessárias para que caiba no campo de dados (com tamanho do MSS), acrescidas pelos cabeçalhos dos protocolos superiores, que para o protocolo de transporte são também dados. Valores comumente utilizados de MSS são de 1460 bytes, 536 bytes e 512 bytes [9].

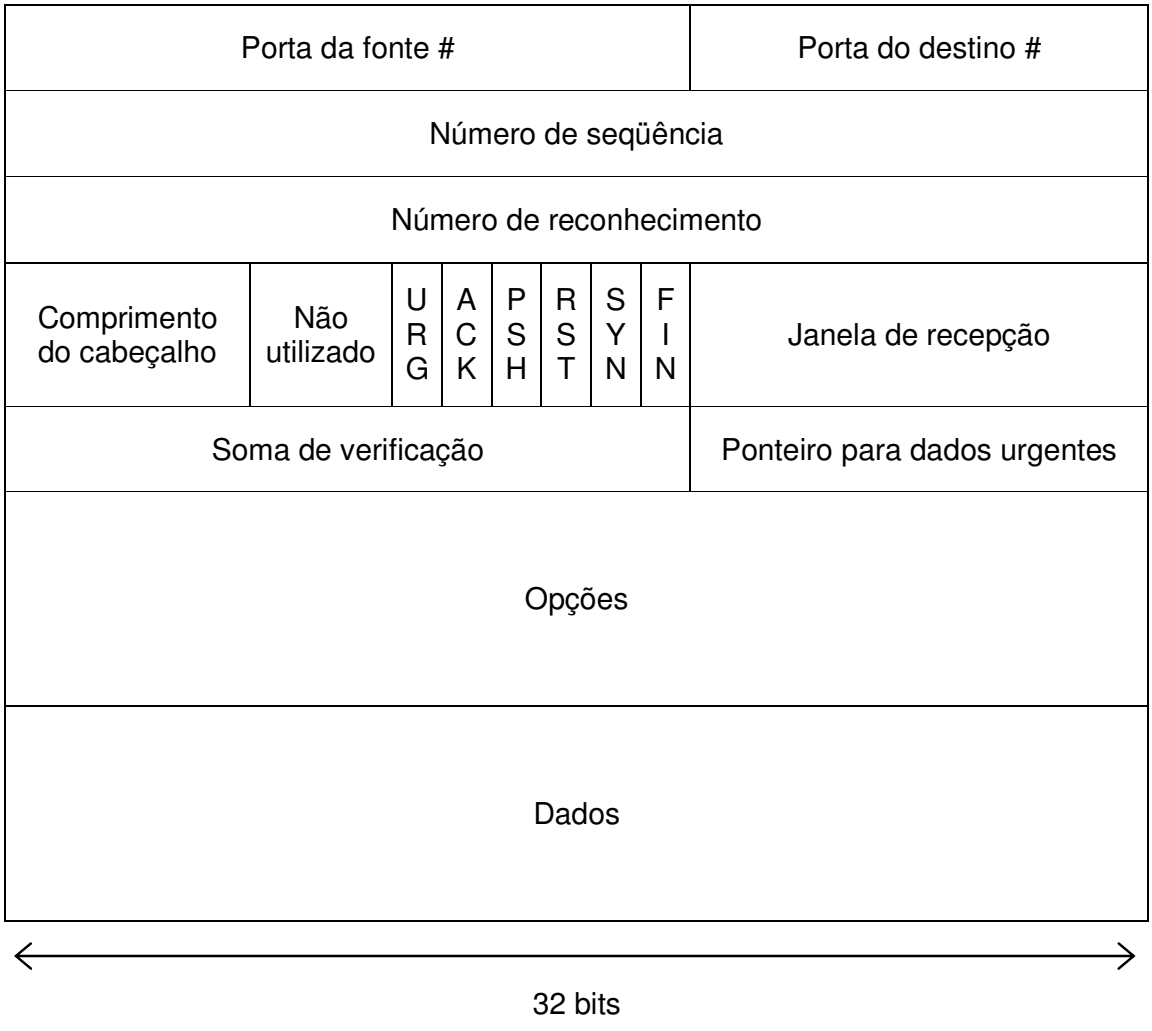

Figura 2: A estrutura de um segmento TCP.

A Figura 2 mostra a estrutura completa do segmento TCP e são também descritos os campos dessa estrutura, baseando-se em [9].

Os campos de **número de porta da fonte** e **número de porta do destino** são usados para multiplexação e demultiplexação dos dados de/para aplicações de camadas superiores.

Os campos de **número de seqüência** e **número de reconhecimento** possuem 32 bits cada e são usados pelos TCPs remetente e destinatário, para se ter um serviço confiável de dados.

O campo de **comprimento do cabeçalho** do segmento TCP define o tamanho do cabeçalho, que pode ser variável devido ao campo de opções – que pertence ao cabeçalho. O campo de opções normalmente está vazio, o que nos dá um valor típico de cabeçalho de 20 bytes.

O **campo de *flags*** contém 6 bits e é formado pelos *flags*: **URG**, **ACK**, **PSH**, **RST**, **SYN** e **FIN**.

O bit **URG** (*Urgent*) indica que os dados contidos no segmento corrente foram marcados como urgentes pela aplicação do lado remetente. A aplicação remetente deverá informar à aplicação destinatária que há dados urgentes a serem entregues e também passar um ponteiro para o final desses dados.

O bit **ACK** (A*cknowledgement*) é usado para reconhecimento de segmentos recebidos corretamente. Ele indica que o segmento que consta no número de reconhecimento do corrente segmento recebido foi entregue corretamente.

O bit **PSH** (*Push*) indica que o destinatário deve passar os dados para a camada superior imediatamente após o recebimento do segmento.

Os bits **RST** (*Reset*), **SYN** (*Synchronize*) e **FIN** (*Finalize*) são usados no estabelecimento e também no encerramento de conexões, para reiniciar, sincronizar e finalizar uma conexão, respectivamente.

O campo **janela de recepção** possui 16 bits e é usado para definir o tamanho da janela deslizante de dados que será usada pelo mecanismo de controle de fluxo do TCP, permitindo que haja um número limitado de segmentos não reconhecidos circulando na rede, sem ultrapassar o tamanho do *buffer* de recepção.

O campo de **opções**, que é opcional e tem comprimento variável, é usado quando dois *hosts* negociam o MSS, como fator de escala de janela de dados em redes de alta velocidade, entre outros casos.

O campo de **soma de verificação** tem 16 bits e provê uma soma dos bits incluídos no segmento, de forma que o receptor pode verificar se os dados do segmento recebido estão corretos. Ela é um dos mecanismos utilizados pelo TCP para implementar o serviço de transferência confiável de dados.

O campo de **ponteiro para dados urgentes** tem 16 bits e é usado para indicar o final (último byte) dos dados urgentes mencionados anteriormente.

O campo de **dados** possui os dados recebidos da camada superior. Esses dados são compostos de dados de aplicação propriamente ditos e cabeçalhos de protocolos de camadas superiores.

A princípio havia previsão de uso dos bits **URG**, **PSH**, e o **ponteiro para dados urgentes**, mas, na prática, não são usados. Suas descrições foram incluídas para tornar a descrição mais completa.

2.3. Modelo de Serviço do TCP

Como já dito, o protocolo TCP é orientado à conexão e fornece um serviço de transferência confiável de dados fim a fim entre *hosts* remotos.

A conexão TCP fornece transferência de dados *full-duplex*, ou seja, pode haver simultaneamente na rede segmentos enviados pelos dois lados conectados, gerando assim um melhor aproveitamento da rede [9].

Uma conexão TCP é sempre ponto-a-ponto, ou seja, só há o envolvimento de um remetente e um destinatário, ou um cliente e um servidor, diferente do que ocorre em conexões *multicast*, onde é possível ter um único remetente para vários destinatários, em uma única operação.

O protocolo TCP é orientado à conexão, o que significa que ele sempre estabelece uma sessão entre as partes comunicantes, de antes de enviar e receber segmentos dados.

O estabelecimento de uma conexão TCP se dá em três etapas, e é comumente conhecido como apresentação em três vias (*3-way handshake*), onde são enviados três segmentos especiais, entre cliente (quem solicita conexão) e servidor (quem responde a essa solicitação) da conexão, antes de quaisquer segmentos com carga útil de dados.

A finalização de uma conexão TCP se dá de forma simples, com um pedido de finalização (FIN) enviado pelo cliente, reconhecimento (ACK) e envio do pedido também pelo servidor (outro FIN), com reconhecimento pelo cliente. Em seguida há uma temporização de fechamento. Após esse tempo todos os recursos estão totalmente liberados, inclusive os números de portas entre processos (*sockets*).

A comunicação no TCP se dá entre processos de aplicação que rodam nos dois lados (*hosts*) envolvidos na conexão. Os *sockets* são portas de comunicação entre os processos de aplicação que estão rodando nos dois lados. Quando um *host* A deseja enviar um dado, usando o protocolo de transporte, para um *host* B (considerando que haja uma conexão entre eles), ele simplesmente direciona estes à porta de comunicação ou *socket* do *host B*, entregando assim os dados à aplicação final em B. O mesmo processo vale se o *host* B deseja enviar um dado para o *host* A.

Cada número de porta é um número de 16 bits que vai de 0 a 65535. Os números de porta bem conhecidos são os que vão de 0 a 1023 [9]. Alguns exemplos de números de porta bem conhecidos são: FTP (portas 20-dados e 21-controle), SSH (porta 22), Telnet (porta 23), SMTP (porta 25), DNS (porta 53), HTTP (porta 80), POP3 (porta 110) e HTTPS (porta 443) [15].

A Figura 3 mostra um exemplo de utilização de *sockets* em uma conexão TCP. Como pode ser visto, existe um processo de aplicação rodando em ambos os lados da conexão. Os *sockets* funcionam como portas de comunicação entre a aplicação e a camada de transporte. Esta última, por sua vez, faz uma conexão lógica fim a fim entre cliente e servidor, de modo transparente ao usuário.

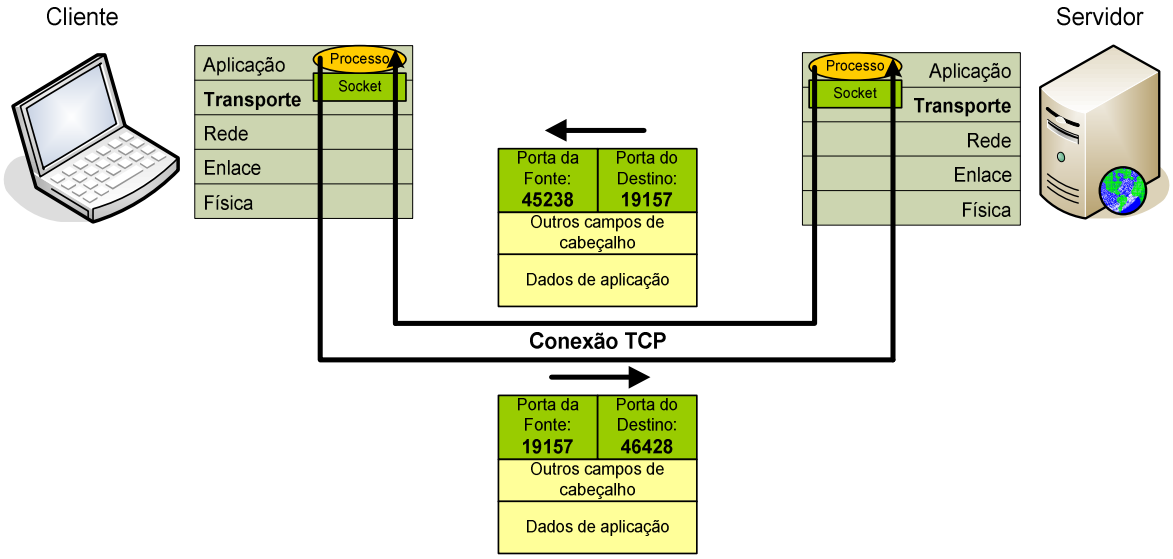

Figura 3: Exemplo do uso de *sockets* e processos de aplicação numa conexão.

O TCP trabalha com *buffers* de transmissão e recepção, que são reservados durante o processo de estabelecimento da conexão, durante a apresentação dos *hosts*. De tempos em tempos o TCP retira dados dos *buffers*, seja para o envio (transmissão) seja para a recepção (entrega para a camada superior da pilha de protocolos – em se considerando o modelo Híbrido de 5 camadas, esta será a camada de aplicação).

A temporização de um segmento TCP enviado ou recebido é baseada na estimativa do RTT (tempo de ida e volta do segmento na rede). Conhecendo os tempos médios de ida e volta dos segmentos, o TCP pode estipular estaticamente o valor da temporização sem que este seja longo demais, evitando baixa utilização da rede, nem curto demais, evitando retransmissões desnecessárias devido ao esgotamento prematuro da temporização. Maiores detalhes sobre a estimativa do RTT e do esgotamento de temporização podem ser encontrados em [9, Seção 3.5.3].

Em resumo, uma conexão TCP é composta basicamente por *buffers* de transmissão e recepção, *sockets* de conexão (que são as portas entre processos de aplicação e transporte dos dois lados da conexão [9]), e variáveis que guardam o estado da conexão (variáveis de estado).

Algumas variáveis de estado do TCP são mostradas abaixo com suas respectivas descrições, baseadas em [9]:

SendBase: Base de envio, isto é, o número de seqüência mais baixo de segmento enviado, mas ainda não reconhecido.

NextSeqNum: Número de seqüência do próximo byte da janela a ser enviado.

RcvBase: Base de recebimento, isto é, o número de seqüência mais baixo de segmento recebido, mas ainda não reconhecido.

RcvBuffer: *Buffer* de recepção, isto é, a janela de tamanho N no receptor.

RcvWindow: Janela de recepção, isto é, a parte ainda disponível para recepção.

LastByteRcvd: Número de seqüência do último byte recebido pelo lado receptor.

LastByteRead: Número de seqüência do último byte lido pela aplicação do receptor.

LastByteSent: Número de seqüência do último byte enviado pelo lado transmissor.

LastByteAcked: Número de seqüência do último byte enviado para o receptor e já reconhecido por ele, ou seja, um ACK relativo ao segmento enviado já foi recebido.

CongWin: Tamanho da Janela de Congestionamento do TCP.

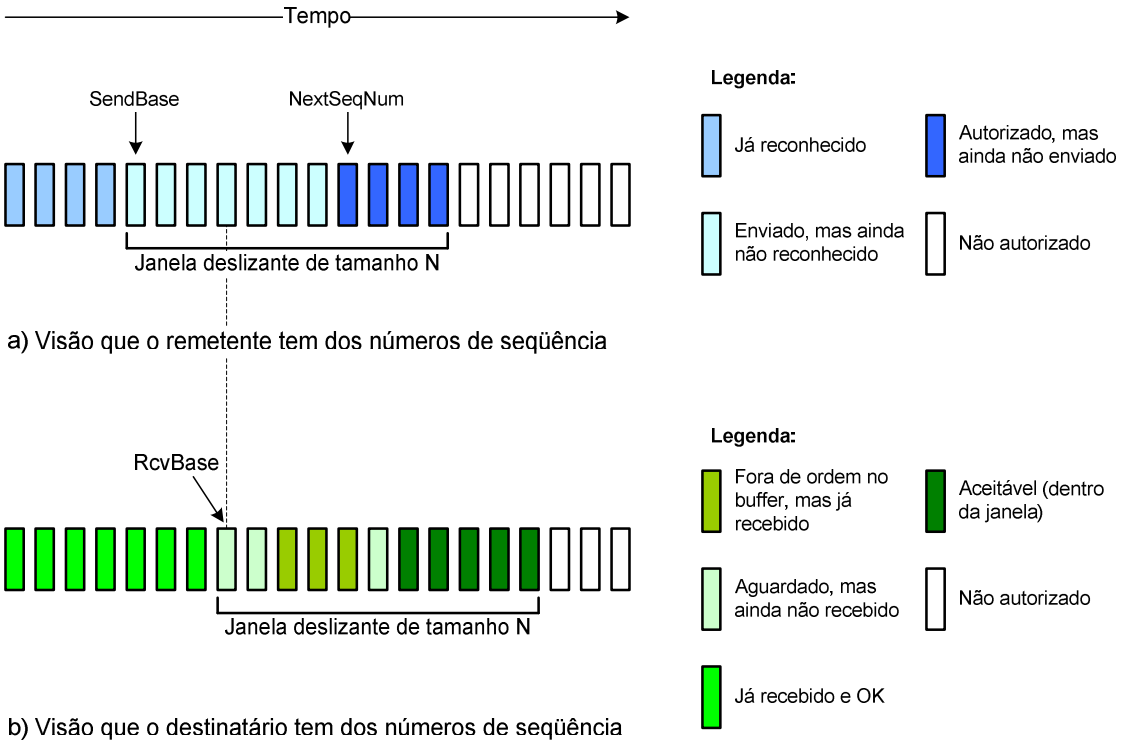

Figura 4: *Buffers* de transmissão e recepção, janela e variáveis de estado.

2.4. Mecanismos de Transferência Confiável de Dados

O TCP possui vários mecanismos que visam à transferência confiável de dados, os quais são enumerados abaixo:

1) Soma de Verificação: usada para detectar possíveis erros em um segmento, garantindo que os dados não tenham sido alterados ou corrompidos durante o trajeto entre a origem e o destino. O número de soma de verificação é formado pelo complemento a 1 da soma (em complemento a 1) de todas as palavras de 16 bits. Ao chegar ao destinatário é calculado o complemento a 1 da soma (em complemento a 1) de todas as palavras de 16 bits, incluindo a soma de verificação, de forma que o resultado dessa soma deverá ser uma cadeia de bits 1. Caso exista algum bit 0 na cadeia de bits resultante dessa soma isto significará que um erro foi encontrado nos dados e o segmento precisa ser retransmitido;

2) Temporizador: usado para controlar a temporização e retransmitir um segmento quando necessário. Isso se dá porque possivelmente o segmento (ou seu ACK) foi perdido dentro do canal. Como pode ocorrer o esgotamento da temporização quando um segmento está atrasado mas não perdido (esgotamento de temporização prematuro), ou quando um segmento foi recebido pelo destinatário mas seu ACK destinatário-remetente foi perdido, existe a possibilidade de um destinatário receber cópias duplicadas de um mesmo segmento;

3) Número de Seqüência: usado para numeração seqüencial de segmentos de dados que transitam do remetente para o destinatário. Lacunas nos números de seqüência de segmentos recebidos permitem que o destinatário detecte um segmento perdido. Segmentos com números de seqüência repetidos permitem que o destinatário detecte cópias duplicadas de um segmento. O número de seqüência do segmento é o número do primeiro byte do campo de dados deste segmento;

4) Número de Reconhecimento: usado pelo destinatário para avisar ao remetente que um segmento ou conjunto de segmentos foi recebido corretamente. Reconhecimentos normalmente portam o número de seqüência do segmento, ou segmentos, que estão sendo reconhecidos. Os reconhecimentos podem ser individuais (por segmento recebido) ou cumulativos (no caso de recebimento de segmentos, mas com perda dos respectivos ACKs, então o primeiro ACK com

número de seqüência posterior que for recebido corretamente informará que todos os segmentos anteriores já foram recebidos corretamente). Considerando dois *hosts* A (cliente) e B (servidor) conectados via TCP, o número de reconhecimento que o *host* A atribui a seu segmento ACK é o número de seqüência do próximo byte que ele estiver aguardando do *host* B;

5) Reconhecimento Negativo (implícito): usado pelo destinatário para avisar ao remetente que um determinado segmento não foi recebido corretamente. Reconhecimentos negativos normalmente portam o número de seqüência do segmento que não foi recebido corretamente. No caso específico do TCP Reno, ao invés de enviar um reconhecimento negativo explícito (NACK), este reconhece novamente o último byte (número de seqüência) recebido corretamente e em ordem, enviando um ACK duplicado toda vez que ele recebe segmentos com número de seqüência maior que o último reconhecido, mas com lacunas. Quando recebe de volta três ACKs duplicados (isso funciona como um NACK implícito), o TCP considera que perdeu o segmento em questão, então ele efetua a retransmissão deste segmento perdido;

6) Janela e Paralelismo: O remetente fica restrito a enviar somente uma determinada faixa de valores de números de seqüência (janela de transmissão). Isso permite que vários segmentos sejam transmitidos, sem esperar pelo reconhecimento individual de cada segmento (paralelismo). O paralelismo pode melhorar em muito o rendimento de uma sessão TCP quando a razão entre o tamanho do segmento e o RTT (atraso de ida e volta) é pequena. O tamanho da janela pode ser estabelecido com base na capacidade do *buffer* do destinatário (armazenamento temporário) de mensagens ou no nível de congestionamento da rede, ou em ambos. O número específico de segmentos não reconhecidos na janela (seu tamanho) é determinado pelos mecanismos de controle de fluxo e de controle congestionamento do TCP, que serão discutidos nas seções a seguir.

Considerando-se ainda o papel do reconhecimento de segmentos, o TCP pode adicionalmente usar dois outros mecanismos para auxílio no reconhecimento de segmentos recebidos, o SACK (*Selective Acknowledgement*) [16] e o DSACK (*Duplicate Selective Acknowledgement*) [17]. O SACK faz reconhecimento seletivo de blocos de segmentos que já foram recebidos corretamente, embora ainda haja lacunas nos números de seqüência de segmentos entregues no buffer de recepção.

O DSACK faz o reconhecimento seletivo de segmentos que tenham sido recebidos em duplicata, enviando um reconhecimento em bloco como no SACK, evitando o estouro de temporização por recebimento três ACKs duplicados. Ambos os mecanismos trabalham de forma a economizar retransmissões desnecessárias de segmentos e ACKs, bem como estouros de temporização (que gerariam outras retransmissões desnecessárias), fatores que aumentariam ainda mais o congestionamento da rede.

2.5. Controle de Fluxo do TCP

Como mencionado anteriormente, os *hosts* de uma conexão TCP (remetente e destinatário) reservam *buffers* de transmissão e recepção durante o processo de apresentação em três vias, quando bem sucedido.

O *buffer* de recepção do lado destinatário da conexão recebe uma quantidade de dados, os quais são temporariamente armazenados (considerando que estão corretos e em ordem) e serão retirados pela aplicação de camada superior de tempos em tempos. O problema com esse processo é que a aplicação pode estar ocupada processando outra tarefa e não retirar os dados do *buffer* por um determinado período de tempo. Considerando-se o fato de que o remetente pode estar enviando dados continuamente, existe a possibilidade de que o *buffer* de recepção fique cheio (e não é uma possibilidade remota) e não consiga mais armazenar os segmentos que chegam, isto é, perca os segmentos entrantes. Isso gera um problema de congestionamento no destinatário.

O serviço de controle de fluxo do TCP visa resolver este problema, por meio de um mecanismo de controle de emissão de segmentos de dados no transmissor, eliminando assim a possibilidade de saturação do *buffer* do receptor [9].

Uma conexão TCP mantém algumas variáveis de estado, como já dito, e entre elas a janela de recepção (RcvWindow) e o *buffer* de recepção (RcvBuffer), conforme mostrado na Figura 5.

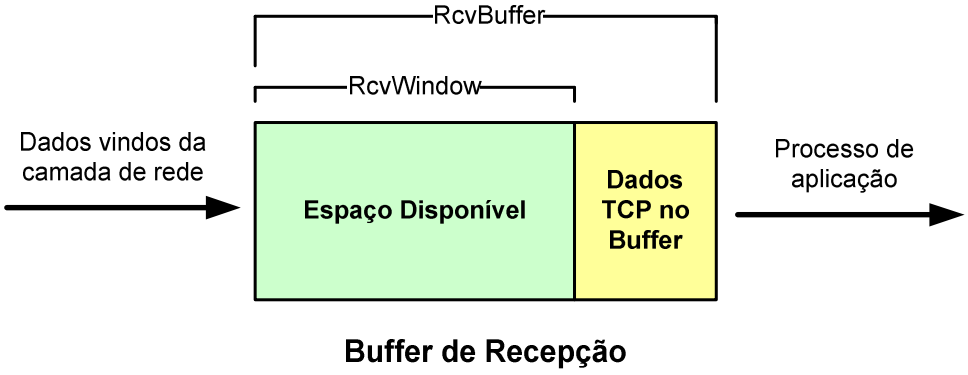

Figura 5: Janela de recepção (RcvWindow) e *buffer* de recepção (RcvBuffer).

O TCP realiza o controle de fluxo fazendo com que estas variáveis sejam atualizadas a cada segmento recebido corretamente pelo lado destinatário. Essa informação é repassada ao remetente e isso dá a ele uma idéia de quanto espaço livre ainda existe no *buffer* do lado receptor. Em uma conexão TCP *full-duplex* cada *host* possui um *buffer* de recepção independente. A janela de recepção é dinâmica, mudando ao longo do tempo de vida dessa conexão [9].

Baseando-se no valor das variáveis, e como o TCP não pode ultrapassar o *buffer* do receptor alocado à conexão, temos que:

$$\left(LastByteRcvd - LastByteRead\right) \leq RcvBuffer\ . \qquad (1)$$

Se (LastByteRcvd – LastByteRead) < RcvBuffer, sabe-se que o *buffer* do receptor ainda possui espaço disponível e o remetente está livre para transmitir. Se (LastByteRcvd – LastByteRead) = RcvBuffer, conclui-se que o *buffer* do receptor está cheio e o remetente deve aguardar para realizar a próxima transmissão.

A janela de recepção, que mede o espaço disponível no *buffer* de recepção, pode ser ajustada de acordo com o espaço livre neste, da seguinte forma:

$$RcvWindow = RcvBuffer - \left(LastByteRcvd - LastByteRead\right). \qquad (2)$$

Por outro lado, o remetente também faz o controle dos dados enviados, com base em variáveis de estado, da seguinte forma:

$$\left(LastByteSent - LastByteAcked\right) \leq RcvWindow\ . \qquad (3)$$

Assim, de posse dessas informações, o TCP garante que não haverá saturação do *buffer* do receptor durante a conexão.

Um problema adicional levantado no controle de fluxo é o tamanho de janela do receptor. Como já mencionado, o segmento TCP possui um campo chamado janela de recepção que tem o tamanho de 16 bits. Esse tamanho limita o tamanho máximo de janela de recepção a 64Kbytes (2^{16} = 64 Kbytes). Visando contornar este problema, foi criado um mecanismo denominado *Window Scale*, descrito na RFC 1323 [18], que realiza a multiplicação do tamanho da janela de recepção, sendo este fator de multiplicação negociado na fase de estabelecimento da conexão [5].

2.6. Controle de Congestionamento do TCP

Quando uma rede fica congestionada, os *buffers* dos roteadores que interconectam os pontos da rede ficam cheios e ocorre inevitavelmente a perda de segmentos, sendo então necessário efetuar retransmissões, o que aumenta ainda mais o congestionamento da rede.

"O congestionamento é um dos 'dez mais' da lista de problemas fundamentalmente importantes no trabalho de rede" [9]. O controle do congestionamento não é um serviço individual voltado especificamente para uma conexão TCP em uma rede, mas sim um serviço de "utilidade pública", visto que ele tem como objetivo aliviar o tráfego excessivo de toda a rede.

Podemos considerar algumas situações com relação ao congestionamento:

1) quando a taxa de dados dos *hosts* transmissores se aproxima da capacidade do canal passam a existir grandes atrasos de fila nos *buffers*, que vão resultar certamente em congestionamento da rede;

2) estando os *buffers* cheios e com filas, haverá perdas de segmentos de dados, logo, retransmissões são realizadas para compensar as perdas de uma rede congestionada, aumentando ainda mais o tráfego de dados na rede, e, por conseguinte, o congestionamento da mesma;

3) quando existem grandes atrasos na rede acontecem esgotamentos de temporização prematuros, seguidos de retransmissão, embora os segmentos ainda

estejam em trânsito na rede. Nesse caso o desempenho cai ainda mais devido ao fato de que os roteadores utilizam a largura de banda da rede para fazer (re)transmissões desnecessárias de segmentos, subutilizando o enlace;

4) no limite do tráfego pesado, quando segmentos são descartados após terem passado por alguns nós (roteadores) da rede devido ao congestionamento, há um desperdício do trabalho realizado pelos nós anteriores para que o segmento chegasse até aquele ponto. Teria sido mais vantajoso para a rede ter descartado o segmento no primeiro nó, aumentando assim sua capacidade de transmissão.

Puderam ser vistas nas situações de congestionamento anteriormente citadas algumas conseqüências de uma rede congestionada: atrasos, perda de segmentos, retransmissões desnecessárias, baixa utilização da largura de banda disponível na rede.

O controle de congestionamento do TCP tem como objetivo principal justamente evitar a perda de segmentos por congestionamento da rede, por meio da regulação da taxa de transmissão dos *hosts* envolvidos numa conexão. O TCP utiliza um mecanismo de controle de congestionamento fim a fim, visto que a camada de rede, que utiliza o protocolo IP (considerando-se o uso do TCP/IP), não oferece realimentação explícita sobre o congestionamento da rede [9]. Neste mecanismo somente os *hosts* finais envolvidos na conexão detectam e fazem o controle do congestionamento da rede.

Num modelo real de rede (considerando a Internet como exemplo mais descritivo), sempre teremos *buffers* finitos, trajetos múltiplos e com vários roteadores, e também com algumas (ou muitas) conexões em andamento. O controle de congestionamento do TCP tenta distribuir igualitariamente as conexões, de forma que a vazão ou taxa de transferência de dados (número de bytes por segundo recebidos pelo destinatário [9]) das conexões tenda à igualdade.

O modelo de controle de congestionamento do TCP está descrito detalhadamente no RFC 2581 [19].

"*A abordagem adotada pelo TCP é obrigar cada remetente a limitar a taxa à qual enviam tráfego para sua conexão como função do congestionamento percebido*" [9]. A percepção ou detecção de congestionamento se dá por meio de

eventos de perda de segmentos de dados. Se um remetente TCP está enviando dados a uma determinada taxa e percebe congestionamento na rede, ele reduz a taxa de envio de dados para a conexão. Por outro lado, se o remetente TCP percebe que há pouco ou nenhum congestionamento na rede, ele aumenta sua taxa de envio de tráfego para a conexão.

Baseando-se no TCP Reno, exceto quando outro tipo de TCP for explicitamente citado, considera-se evento de perda de dados como sendo a perda de um segmento sinalizado pelo recebimento de três ACKs duplicados ou o esgotamento da temporização. Isso se dá porque os *buffers* dos roteadores no meio de caminho estão transbordando e descartam os segmentos recebidos em excesso. Quando a freqüência destes eventos de perda aumenta muito, o TCP considera que o caminho remetente-destinatário está congestionado.

Considerando o caso oposto, quando há transmissões sucessivas sem eventos de perda, a conexão TCP considera que a rede está livre de congestionamento e aumenta a janela de congestionamento CongWin, ou seja, libera o aumento da taxa de envio de dados do remetente para a conexão.

Se o remetente recebe os reconhecimentos de segmentos a uma taxa relativamente baixa, ele aumentará a CongWin a uma taxa relativamente baixa (por exemplo, se há enlace de baixa largura de banda ou há atrasos na rede). Se ele recebe os reconhecimentos a uma taxa alta, ele aumentará sua janela de congestionamento a uma taxa mais alta, ou seja, mais rápida será a elevação da vazão desta conexão.

O TCP armazena o estado da conexão em variáveis específicas (como já mencionado). A variável CongWin (janela de congestionamento) é a responsável pela limitação de taxa de envio de dados para uma conexão TCP. O TCP regula a janela da seguinte forma:

$$\left(LastByteSent - LastByteAcked\right) \leq Min\left(CongWin, RcvWindow\right). \quad (4)$$

A quantidade de dados da conexão não reconhecidos e ainda circulando na rede não poderá exceder o mínimo de CongWin e RcvWindow [9]. Admite-se, na discussão que se segue, que o *buffer* de recepção seja tão grande que a quantidade

de dados não reconhecidos será limitada unicamente pelo tamanho da janela de congestionamento (CongWin). Toda a discussão é baseada em [9].

Ao se limitar o controle por CongWin, limita-se indiretamente a taxa de envio dados do remetente. Considera-se uma conexão TCP com perdas e atrasos de transmissão desprezíveis. Em linhas gerais, no início da transmissão envia-se a quantidade de dados de uma janela de congestionamento para a conexão. Seu tempo de ida e reconhecimento é de 1 RTT. Então ao fim de 1 RTT tem-se o reconhecimento para os segmentos enviados. Pode-se dizer então que:

$$Vazão = \frac{CongWin}{RTT}\ bytes\ \ por\ segundo\ . \qquad (5)$$

Considerando que CongWin é igual a *w* unidades de dados de tamanho MSS, tem-se a seguinte equação para a vazão:

$$Vazão = \frac{w \times MSS}{RTT}\ . \qquad (6)$$

Logo, ao se ajustar o valor da janela CongWin, ou simplesmente *w*, automaticamente ajusta-se a taxa de envio de dados para a conexão de um remetente TCP.

2.6.1. Partida Lenta

A janela inicial de congestionamento do TCP é de 1 MSS [20]. Sendo assim, a taxa inicial de dados é de MSS/RTT, com $w = 1$. Considerando-se o fato de que a largura de banda disponível pode ser muito maior do que isso, o TCP aumenta o valor de sua janela exponencialmente. Essa fase é conhecida como **partida lenta** (*Slow Start* – SS), porque o remetente TCP inicia a transmissão a uma taxa lenta e depois aumenta a janela de forma exponencial [9].

No aumento exponencial o TCP eleva o tamanho da janela de congestionamento duplicando-a a cada RTT. Janela é aumentada de 1 MSS a cada reconhecimento recebido com sucesso [9]. O TCP continua nesse modo de aumento até que seja alcançado um patamar específico ou até que ocorra um evento de

perda de segmento. Um exemplo de janela de congestionamento com aumento exponencial durante a partida lenta do TCP é mostrado na Figura 6.

O patamar específico que é alcançado ao final da fase de aumento exponencial da janela na partida lenta é chamado de *Threshold (SSThresh – Slow Start Threshold)*. *Threshold* (que significa limiar ou patamar) é uma variável do TCP e tem um valor inicial alto (tipicamente 65 Kbytes [9]). Depois de alcançado o valor de *Threshold*, e caso não haja nenhum evento de perda de segmentos, a janela de congestionamento passa a aumentar de forma linear, operando pelo algoritmo AIMD (mais detalhes sobre seu funcionamento na seção seguinte).

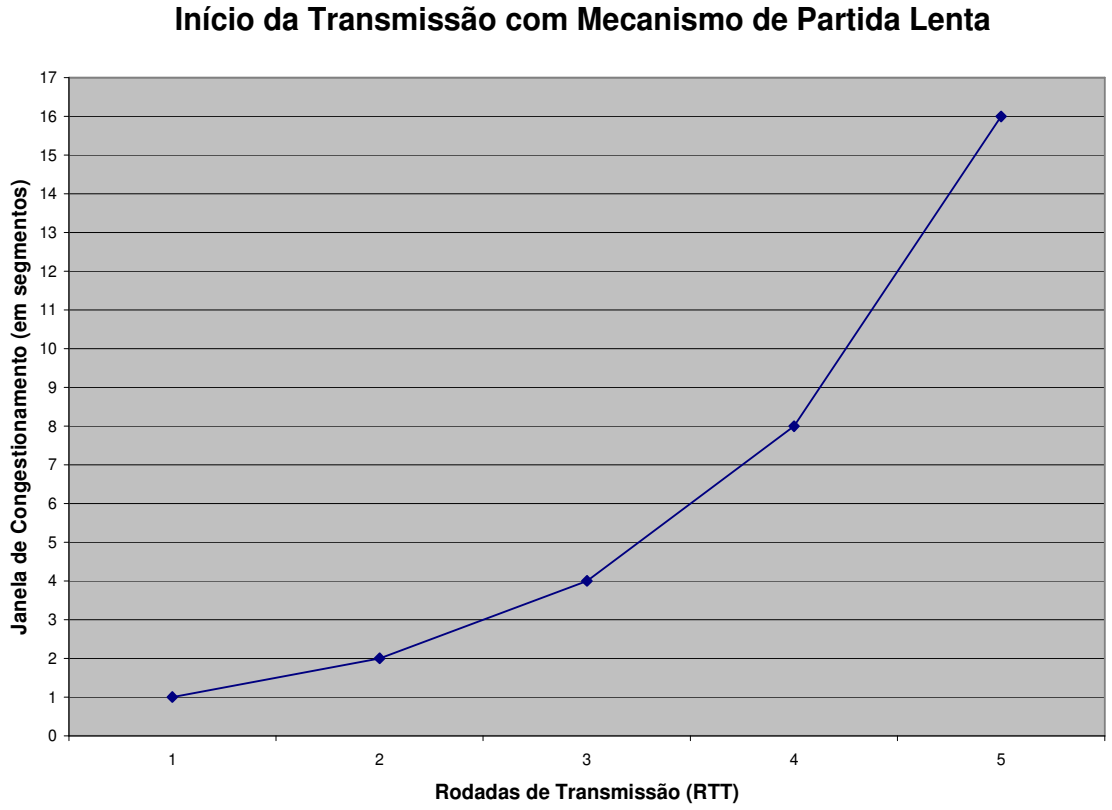

Figura 6: Início da transmissão com o mecanismo de partida lenta.

Após eventos de perda de segmentos por esgotamento de temporização, o controle de congestionamento do TCP faz com que a janela CongWin seja reduzida a 1 MSS [19], e ajustando *Threshold* para metade do valor corrente de CongWin. A partir de então passa a aumentar seu tamanho exponencialmente, utilizando o

mecanismo de partida lenta, até alcançar novamente o valor de *Threshold*. A partir de então passa a operar linearmente pelo algoritmo AIMD (vide Figura 8).

Após eventos de perda de segmentos detectados através do recebimento de três ACKs duplicados, a variável *Threshold* é ajustada para CongWin/2. A janela de congestionamento então reinicia sua evolução de modo linear, operando pelo algoritmo AIMD, iniciando a partir de *Threshold*, até que um novo evento de perda seja detectado [9]. Na ocorrência de novos eventos de perda por três reconhecimentos duplicados a janela cai pela metade novamente (sempre atualizando o valor de *Threshold* para CongWin/2), assumindo o valor atualizado de *Threshold*, e assim por diante (vide Figura 8).

Colocando em termos de equações, na partida lenta, a janela CongWin é aumentada a cada reconhecimento recebido da seguinte forma:

$$CongWin = CongWin + MSS\ . \tag{7}$$

Considerando-se a janela em termos de *w*, tem-se que:

$$w \times MSS = w \times MSS + MSS\ . \tag{8}$$

E, após simplificação em função de MSS, resulta em:

$$w = w + 1\ . \tag{9}$$

Ou, para fins de comparação futura, utilizando o parâmetro *c*, tem-se então:

$$w = w + c\ , \text{ (onde } c = 1). \tag{10}$$

Considerando-se um evento de perda pelo recebimento de três ACKs duplicados, a janela *w* diminui da forma abaixo:

$$w = 0{,}5 \times w\ ,\ w = w/2 \text{ ou } w = w - 0{,}5 \times w\ . \tag{11}$$

Ou, para fins de comparação futura, utilizando o parâmetro *b*, tem-se então:

$$w = w - b \times w\ , \text{ (onde } b = 0{,}5). \tag{12}$$

Deve ser lembrado que, quando ocorre um evento de perda por esgotamento de temporização, a janela CongWin volta para 1 MSS e reinicia a fase de partida lenta.

2.6.2. Prevenção de Congestionamento

O TCP utiliza o algoritmo AIMD (*Additive-Increase, Multiplicative-Decrease* - Aumento Aditivo, Diminuição Multiplicativa) para regular o comportamento da janela de congestionamento mediante eventos de perda por recebimento de três ACKs duplicados. Essa fase é conhecida como **prevenção de congestionamento** (*Congestion Avoidance* – CA), e é representada graficamente por um aumento linear da janela de congestionamento [9].

Mediante eventos de sucesso, ou seja, recebimento de reconhecimentos positivos corretamente (ACKs) e sem perdas de segmentos, o TCP assume que há largura de banda disponível e aumenta um pouco o tamanho de CongWin a cada reconhecimento com sucesso. Ele eleva precavidamente o tamanho de sua janela de congestionamento, aumentando CongWin de 1 MSS a cada tempo de ida e volta (RTT) sem eventos de perda [19].

A janela CongWin é aumentada da seguinte forma:

$$CongWin = CongWin + \left[MSS\left(\frac{MSS}{CongWin} \right) \right], \qquad (13)$$

onde [MSS x (MSS / CongWin)] equivale ao aumento proporcional a cada reconhecimento positivo. Isso nos mostra que, se no tempo de 1 RTT “n” segmentos com carga útil de 1 MSS são enviados, após reconhecimento com sucesso de “n” segmentos de dados, teremos o aumento da janela de congestionamento de 1 MSS.

De uma forma mais simplificada, baseando-se na Equação 13, utilizando a janela *w* em unidades de MSS, tem-se o seguinte:

$$w \times MSS = w \times MSS + \left[MSS\left(\frac{MSS}{w \times MSS} \right) \right], \qquad (14)$$

Que, após simplificação em função de MSS, resulta em:

$$w = w + \frac{1}{w}. \qquad (15)$$

Ou, para fins de comparação futura, utilizando o parâmetro *a*, tem-se então:

$$w = w + \frac{a}{w}, \text{ (onde } a = 1\text{).} \quad (16)$$

Mediante eventos de perda detectados por três ACKs duplicados, o AIMD faz com que a taxa de envio de dados seja diminuída pela metade, ou seja, o tamanho da janela seja multiplicado por 0,5. O valor mínimo ao qual a janela pode ser diminuída é de 1 MSS [9].

Considerando um evento de perda, a janela *w* diminui como abaixo:

$$w = 0{,}5 \times w, \; w = w/2 \text{ ou } w = w - 0{,}5 \times w. \quad (17)$$

Ou, para fins de comparação futura, utilizando o parâmetro *b*, tem-se então:

$$w = w - b \times w, \text{ (onde } b = 0{,}5\text{).} \quad (18)$$

Uma amostra gráfica do funcionamento do controle de congestionamento com o uso do algoritmo AIMD é mostrada na Figura 7. Como pode ser visto, nesta fase ocorrem eventos sucessivos de aumento linear e queda da janela pela metade devido a eventos de perda. Esse comportamento, de aumentos e decaimentos sucessivos, deixa o gráfico com a aparência de dentes de serra ao longo do tempo.

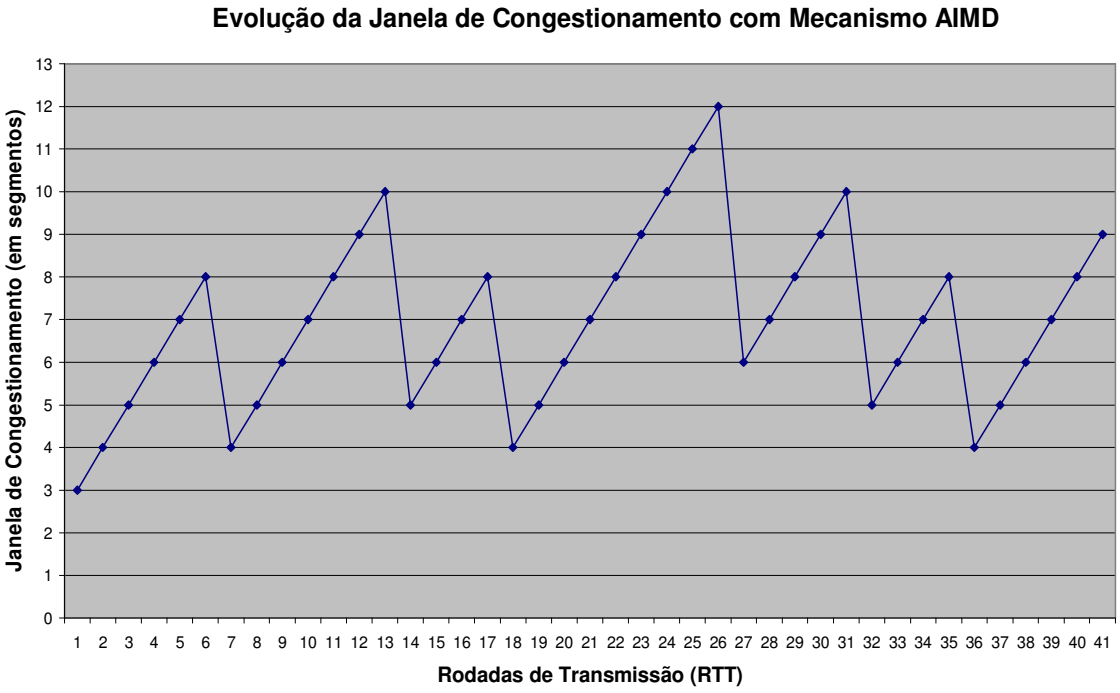

Figura 7: Evolução da janela de congestionamento com uso do algoritmo AIMD.

Equacionando *Threshold* e CongWin para eventos de perda detectados pelo recebimento de três ACKs duplicados, tem-se primeiramente:

$$Threshold = \frac{CongWin}{2} \text{ ou } Threshold = \frac{w \times MSS}{2}. \quad (19)$$

E, após atualizar o valor de *Threshold*, atualiza-se o valor da janela de congestionamento:

$$CongWin = Threshold \quad \text{ou simplesmente } w = \frac{w}{2}. \quad (20)$$

Em eventos de perda por esgotamento de temporização, a janela CongWin volta para o início, como abaixo:

$$CongWin = 1\,MSS \quad \text{ou simplesmente } w = 1. \quad (21)$$

O valor da variável *Threshold* é então ajustado para a metade do valor da janela de congestionamento, como na Equação 20, e a janela passa a operar de exponencialmente até alcançar *Threshold*, e a partir de então de modo linear pelo algoritmo AIMD.

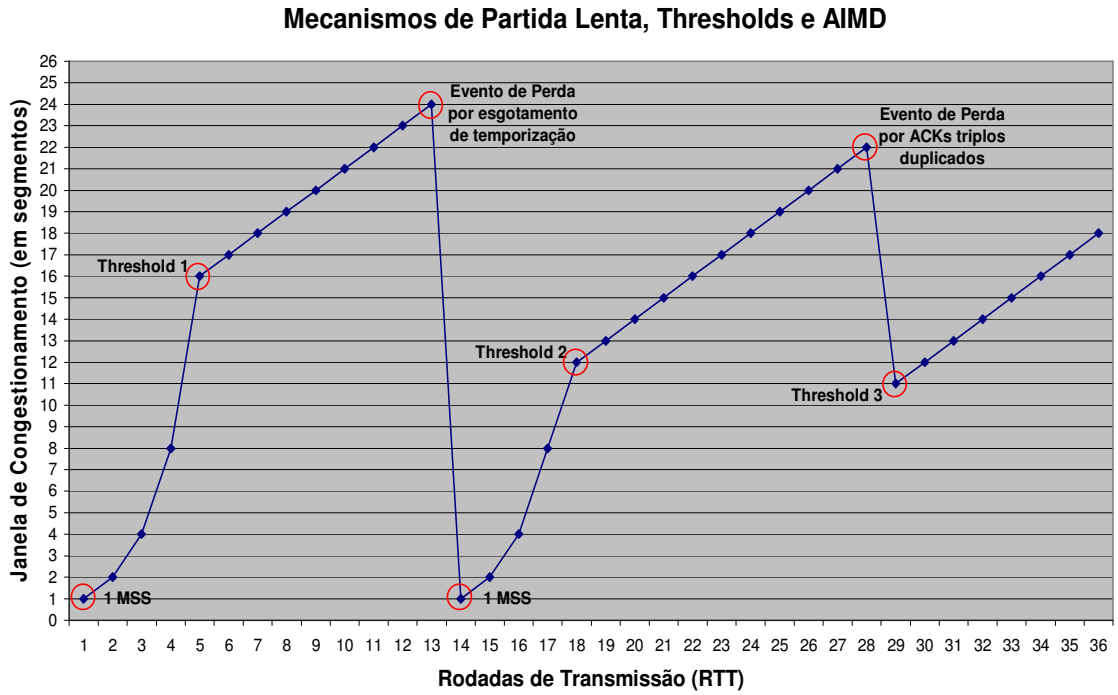

Figura 8: Evolução da janela CongWin com partida lenta, AIMD e *thresholds*.

A Figura 8 mostra a evolução da janela de congestionamento do TCP com aumentos exponenciais delimitados pelos *Thresholds*, e aumentos lineares delimitados por eventos de perda de segmentos.

2.7. Considerações Adicionais sobre o TCP

Na Tabela 2 pode ser visto um resumo dos principais eventos e suas respectivas ações de controle de congestionamento.

Tabela 2: Resumo de eventos e ações do controle de congestionamento de um remetente TCP [9, com modificações].

Estado	Evento	Ação de controle de congestionamento do remetente TCP	Comentário
Partida Lenta (SS)	Recebimento de ACK para dados não reconhecidos anteriormente	CongWin = CongWin + MSS Ou $w = w + c$, (onde c = 1) Se (CongWin > Threshold) ajustar para CA	Resulta em uma duplicação de CongWin a cada RTT, elevando a janela em 1 MSS para cada ACK recebido
Prevenção de Congestionamento (CA)	Recebimento de ACK para dados não reconhecidos anteriormente	CongWin = CongWin + MSS x (MSS/CongWin) Ou $w = w + \frac{a}{w}$, (onde a =1)	Aumento linear, resultando em aumento de CongWin de 1 MSS a cada RTT
SS ou CA	Evento de perda detectado por três ACKs duplicados	*Threshold* = CongWin/2, CongWin = Threshold, ajustar para CA	Recuperação rápida, executando diminuição multiplicativa (CongWin não cairá abaixo de 1 MSS)
SS ou CA	Evento de perda detectado por esgotamento de temporização	*Threshold* = CongWin/2, CongWin = 1 MSS, ajustar para SS	Entra na fase de partida lenta (SS) novamente, iniciando a janela em 1 MSS
SS ou CA	Recebimento de ACK duplicado	Incrementa a contagem de ACK duplicado para segmento que está sendo reconhecido	CongWin e *Threshold* não mudam

Uma conexão TCP ao longo do tempo não é uma constante, ou seja, existem oscilações em sua vazão devido à perda de segmentos e seus efeitos, partida lenta e AIMD. A vazão média de uma conexão em função da perda de segmentos deve portanto ser considerada.

Por exemplo, para o caso de uma conexão TCP com segmentos de 1500 bytes e RTT de 100 ms, visando alcançar uma vazão de 10 Gbps, e baseando-se na Equação 6 e em [21], o tamanho da janela de congestionamento deveria ser de 83.333 segmentos, ou seja, w = 83.333 segmentos de tamanho MSS. São muitos segmentos para uma única janela, o que somente seria alcançado numa situação próxima do ideal, ou seja, quase sem perdas.

A Tabela 3 mostra exemplos de como as taxas de perda de segmentos precisam ser cada vez mais baixas a medida que são aumentados os RTTs e a janela de congestionamento, visando alcançar altas vazões.

Tabela 3: Taxas de perda irrealmente baixas no TCP para alcançar altas vazões, dada uma conexão com segmentos de 1500 bytes e RTTs de 0,1 segundos [2].

Vazão (Mbps)	RTTs necessários entre Perdas	Janela de Congestionamento (segmentos)	Taxa de Perda
1	5,5	8,3	0,02
10	55,5	83,3	0,0002
100	555,5	833,3	0,000002
1.000	5.555,5	8.333,3	0,00000002
10.000	55.555,5	83.333,3	0,0000000002

De [9] tem-se que a relação de estado estacionário entre o tamanho médio de janela e a taxa de perda de segmentos é dada por:

$$Vazão\ média = \frac{1{,}22.MSS}{RTT\sqrt{p}}, \qquad (22)$$

onde *p* é a taxa média de perda de segmentos. Simplificando-se pela Equação 6, tem-se que a janela média *w* em função da taxa de perda de segmentos (sua função resposta), é dada por:

$$w = \frac{1{,}22}{\sqrt{p}} \quad \text{ou} \quad w = \frac{1{,}22}{p^{0{,}5}} \,. \tag{23}$$

Sendo assim, conforme o exemplo anteriormente citado, para alcançar uma vazão média de 10 Gbps em estado estacionário, baseando-se na Equação 23, a taxa de perda de segmentos *p* não deveria ultrapassar 2×10^{-10}, ou seja, no máximo um segmento perdido a cada 5.000.000.000 segmentos enviados (ou de forma equivalente, 1 perda de segmento a cada 100 minutos ou 1:40 h), valores de perda irrealmente baixos, o que para as redes atuais ainda não é possível alcançar [21].

Essa taxa de perda é muito baixa, sendo bem inferior à taxa de perda alcançada utilizando-se as tecnologias atuais de fibras óticas [22], o que seria da ordem de 10^{-7} ou, equivalentemente, 10^{-5}% [3]. Isto torna claro porque uma solução para o problema de baixo desempenho do TCP em redes de alta velocidade deve rapidamente ser encontrada.

A Tabela 4 mostra algumas características de uma conexão TCP. Devem ser notados os imensos tempos de recuperação, assim como as taxas mínimas de perda que são necessárias, quando a conexão trabalha a altas taxas.

Tabela 4: Características de uma conexão com segmentos de 1500 bytes e RTTs de 200 ms, utilizando o controle de congestionamento tradicional do TCP [3].

Vazão	Janela de Congestionamento (segmentos)	Tempo de Recuperação de um evento de perda	Taxa de Perda
1 Mbps	17	1,7 s	$5{,}2 \times 10^{-3}$
10 Mbps	170	17 s	$5{,}2 \times 10^{-5}$
100 Mbps	1.700	2 min 50 s	$5{,}2 \times 10^{-7}$
1 Gbps	17.000	28 min	$5{,}4 \times 10^{-9}$
10 Gbps	170.000	4 h 43 min	$5{,}4 \times 10^{-11}$

3. O Protocolo HSTCP

3.1. Introdução ao HSTCP

O protocolo HSTCP (*High-Speed* TCP, ou TCP de alta velocidade), é uma versão modificada do protocolo TCP. Ele foi proposto preliminarmente em meados de 2002 [21] e oficialmente no final de 2003 por Sally Floyd através do RFC3649 [2].

Como mencionado anteriormente, o TCP não consegue utilizar 100% da largura de banda em redes de alta velocidade devido ao seu mecanismo de controle de congestionamento, que necessita de taxas de perda irrealmente baixas para alcançar altas vazões. O HSTCP foi projetado com o objetivo de atingir uma taxa de 10 Gbps com taxas de perda mais aceitáveis, ou seja, valores de perda dentro da realidade das redes atuais. A Tabela 3 ilustra como as taxas de perda devem ser cada vez mais baixas a medida que são aumentados os RTTs e a janela de congestionamento, para possibilitar o alcance de altas vazões.

O HSTCP não muda o funcionamento geral do TCP, ele apenas propõe alterações em seu mecanismo de controle de congestionamento, no lado remetente, que a partir de um determinado patamar da janela de congestionamento passa a operar num modo mais agressivo, levando a um desempenho melhorado.

O HSTCP realiza uma pequena modificação nos parâmetros de aumento e diminuição de sua janela de congestionamento, de modo a permitir que o protocolo seja capaz de alcançar altas vazões com requisitos mais realistas para a taxa de perda de segmentos em estado estacionário, utilizando também requisitos mais realistas para o número de RTTs entre eventos de perda [2].

Em cenários de alto congestionamento o protocolo HSTCP se comporta da mesma forma que o TCP, sem alterações. Desse modo, seu algoritmo de congestionamento trabalha com mais cautela, sem altas elevações de velocidade, evitando o agravamento do congestionamento da rede.

3.2. Modelo de Serviço do HSTCP

O objetivos do protocolo HSTCP são os seguintes [2]:

- alcançar uma alta vazão por conexão requerendo taxas realísticas de perdas de segmentos;
- alcançar uma alta vazão por conexão de forma razoavelmente rápida durante a partida lenta;
- alcançar uma alta vazão sem longos atrasos quando em recuperação de múltiplas retransmissões por fim de temporização, ou em recuperação de períodos com pequenas janelas de congestionamento;
- trabalhar sem realimentação explícita por parte dos roteadores a respeito do estado de congestionamento da rede;
- não requerer informações adicionais por parte dos receptores;
- possuir desempenho compatível com o do protocolo TCP em ambientes com médio ou alto grau de congestionamento (ou seja, taxas de perdas de segmentos maiores ou iguais a 1%);
- não gerar carga adicional de congestionamento para a rede (aumento da taxa de perda) em redes com médio ou alto grau de congestionamento;
- o seu desempenho deve ser no mínimo tão bom quanto o do protocolo TCP em redes com médio ou alto grau de congestionamento;
- variação aceitável de desempenho em termos de aumento da janela de congestionamento em um RTT, resposta a congestionamentos severos e também quanto aos tempos de convergência para a justiça entre conexões de uma mesma rede.

O HSTCP trabalha quase que em sua totalidade com o mesmo modelo de serviço do protocolo TCP. Somente algumas pequenas modificações são inseridas, visando efetivar as alterações dadas pela melhoria do controle de congestionamento em relação ao TCP padrão.

O HSTCP utiliza alguns parâmetros novos em relação ao TCP no seu controle de congestionamento:

Low_Window: patamar inferior da janela de congestionamento onde a modificação do protocolo foi projetada para começar a operar. Inicialmente seu valor foi definido como 38 (em unidades de MSS);

High_Window: patamar superior da janela de congestionamento para a qual o protocolo foi projetado para atingir. Inicialmente esse parâmetro teve seu valor foi definido como 83000 (em unidades de MSS);

Low_P: taxa de perda de segmentos para a qual a modificação do protocolo foi projetada para começar a operar. Inicialmente o parâmetro teve seu valor foi definido como 0,001 ou 10^{-3}, o que equivale a uma taxa de perda de 0,1%;

High_P: taxa de perda de segmentos para a qual o protocolo foi projetado para atingir a vazão de 10 Gbps. Inicialmente o valor deste parâmetro foi definido como 10^{-7} ou o equivalente a 10^{-5}%. Esta taxa é comparável àquela atingida por enlaces de longa distância com fibras ópticas, no interior de dispositivos de rede, ou em sistemas finais [3];

High_Decrease: parâmetro de decréscimo da janela de congestionamento após um evento de congestionamento. Inicialmente seu valor foi definido como 0,1 ou 10%.

Detalhes sobre os cálculos envolvendo o uso dos parâmetros anteriormente citados serão vistos na seção a seguir.

3.3. Controle de Congestionamento do HSTCP

O controle de congestionamento foi a grande área de modificação do protocolo HSTCP em relação ao TCP. A fase de prevenção de congestionamento (*Congestion Avoidance* – CA) foi modificada para agir de forma mais "agressiva" quando a janela de congestionamento se encontra acima do patamar dado pela variável Low_Window, começando efetivamente este a trabalhar para o alcance de velocidades mais altas, como será mostrado.

3.3.1. Partida Lenta

O mecanismo de partida lenta do TCP (*Slow Start* – SS) não foi modificado no HSTCP. Foi considerado em sua proposição, um tanto conservadora, que a

evolução exponencial da janela de congestionamento nessa fase é suficientemente rápida, levando-se em conta o fato de que não há realimentação explícita sobre o estado de congestionamento da rede por parte dos roteadores existentes ao longo do caminho da conexão.

3.3.2. Prevenção de Congestionamento

Conforme já mencionado, a fase de prevenção de congestionamento é baseada no mecanismo AIMD (*Additive-Increase, Multiplicative-Decrease*). A modificação do funcionamento deste mecanismo foi a grande mudança deste novo protocolo em relação ao TCP.

Em resumo, o protocolo TCP utiliza as seguintes equações para ajustar o tamanho da janela de congestionamento (*w*) no mecanismo AIMD, conforme apresentado anteriormente nas Equações 16 e 18 respectivamente:

Reconhecimento: $w = w + \frac{a}{w}$.

Evento de Perda: $w = w - b \times w$ (por três ACKs duplicados).

Onde a e b são definidas em unidades de MSS, sendo a = 1 e b = 0,5.

A proposta do HSTCP é modificar estas equações de forma que *a* e *b* sejam variados em função do tamanho da janela de congestionamento *w*, da seguinte forma:

Reconhecimento: $w = w + \frac{a(w)}{w}$. (24)

Evento de Perda: $w = w - b(w) \times w$ (por três ACKs duplicados). (25)

O objetivo é que, quando a janela de congestionamento *w* possui grandes valores, *a* seja aumentado em maior escala nos reconhecimentos, visando alcançar uma maior vazão, e que *b* seja diminuído em menor escala nos eventos de perda, visando um menor retrocesso no tamanho da janela *w* [5].

Tem-se então para o HSTCP as seguintes equações para *a(w)* e *b(w)* [5]:

$$a(w) = \frac{w^2 \times 2 \times b(w)}{w^{1,2} \times 12{,}8 \times (2 - b(w))} \quad (26)$$

e

$$b(w) = \frac{(High\_Decrease - 0{,}5)(\log w - \log Low\_Window)}{(\log High\_Window - \log Low\_Window)} + 0{,}5\,; \quad (27)$$

onde o parâmetro High_Decrease determina o valor *b(w)* para o maior valor da janela de congestionamento (*w* = High_Window). Os valores *a(w)* e *b(w)* assumem os valores padrões do TCP quando *w* ≤ Low_Window, ou seja, abaixo da janela mínima o HSTCP terá uma função resposta idêntica a do TCP.

Na Figura 9 é apresentada uma amostra dos valores que *a(w)* e *b(w)* podem assumir em uma janela de congestionamento dinâmica, para os valores padrões de Low_Window, High_Window, Low_P, High_P e High_Decrease. Uma tabela com os valores de assumidos por *a(w)* e *b(w)*, assim como os valores de *w* correspondentes, pode ser encontrada no Apêndice A.

Pode-se observar que para um mesmo intervalo da janela *w*, o coeficiente de elevação *a(w)* possui uma inclinação muito mais agressiva em relação ao seu coeficiente de descida *b(w),* ou seja, em eventos de reconhecimento a janela *w* aumenta consideravelmente mais e em eventos de perda ela decai menos quando se comparando com o comportamento do TCP padrão.

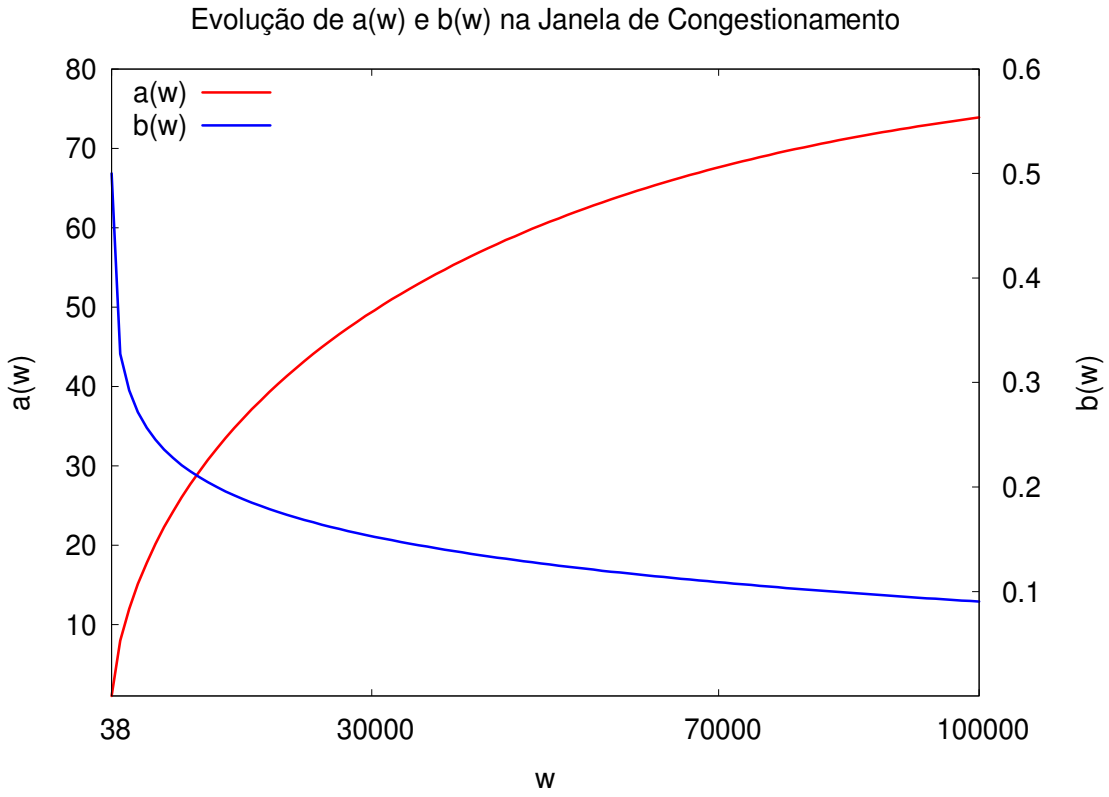

Figura 9: Evolução de *a(w)* e *b(w)* numa janela, utilizando-se o HSTCP.

A função de resposta do HSTCP, assim como no TCP, será linear na escala log-log, mas a inclinação desta reta será maior para este protocolo, como mostrado na Figura 10. Para uma dada taxa de perdas *p* a janela de congestionamento *w* possuirá um valor mais alto no protocolo HSTCP que no TCP [5].

Esta função resposta, diferenciada para o intervalo onde as modificações do protocolo atuarão, se baseia nos pontos Low_Window e High_Window e para os quais equivalem os pontos Low_P e High_P. Para taxas médias de perda maiores que Low_P ou valores médios de janela menores que Low_Window, o protocolo HSTCP se comportará exatamente da mesma forma que o TCP padrão [2]. Tem-se então para função resposta:

$$w = 10^{S(\log p - \log Low\_P) + \log Low\_Window}. \quad (28)$$

A inclinação desta reta será dada por:

$$S = \frac{\log High\_Window - \log Low\_Window}{\log High\_P - \log Low\_P}. \quad (29)$$

Sendo assim:

$$w = \left(\frac{p}{Low\_P} \right)^{S} Low\_Window. \tag{30}$$

Substituindo os valores padrão (mencionados anteriormente) nas equações acima, temos que $S = -0,83$ e w será então:

$$w = \left(\frac{p}{10^{-3}} \right)^{-0,83} \times 38. \tag{31}$$

O que finalmente resulta em:

$$w = \frac{0,12}{p^{0,83}}. \tag{32}$$

Esse será então o valor aproximado para função resposta do protocolo HSTCP na fase de prevenção de congestionamento.

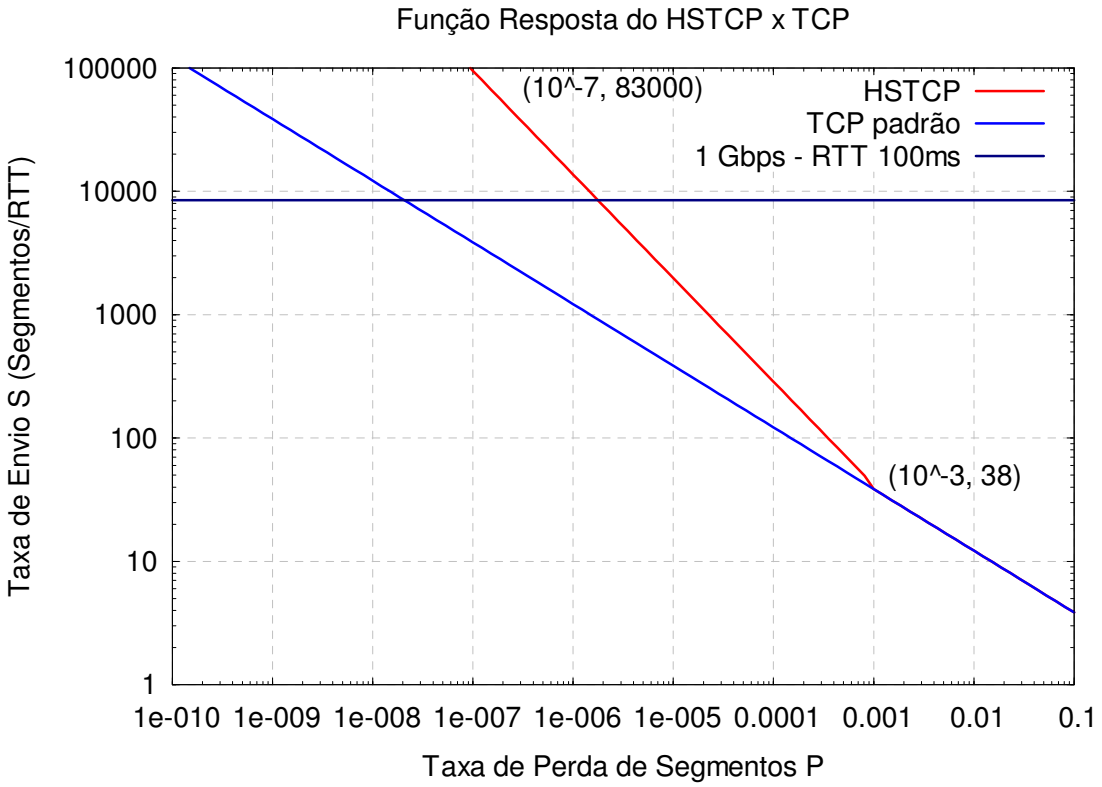

Figura 10: Funções resposta para o TCP padrão e o HSTCP.

Na Figura 10 uma comparação entre as funções resposta do protocolo TCP e do HSTCP para uma dada janela de congestionamento é apresentada. Fica claro que a mudança do HSTCP em relação ao TCP só começa a operar a partir de um determinado ponto, antes disso ele se comporta exatamente igual ao TCP.

Na Tabela 5 é apresentada uma comparação entre as funções resposta do protocolo TCP e do HSTCP em função da taxa de perda de segmentos.

Tabela 5: Tamanho da janela de congestionamento *w* e RTTs entre eventos de perda do TCP e do HSTCP, em função da taxa de perda *p* [2].

Taxa de Perda de Segmentos *p*	TCP		HSTCP	
	Janela *w* (segmentos)	RTTs entre perdas	Janela *w* (segmentos)	RTTs entre perdas
10^{-2}	12	8	12	8
10^{-3}	38	25	38	25
10^{-4}	120	80	263	38
10^{-5}	379	252	1795	57
10^{-6}	1.200	800	12.279	83
10^{-7}	3.795	2.530	83.981	123
10^{-8}	12.000	8.000	574.356	180
10^{-9}	37.948	25.298	3.928.088	264
10^{-10}	120.000	80.000	26.864.653	388

A Figura 11 mostra um gráfico de [22], que faz uma comparação experimental da janela de congestionamento no TCP e no HSTCP, usando o algoritmo AIMD:

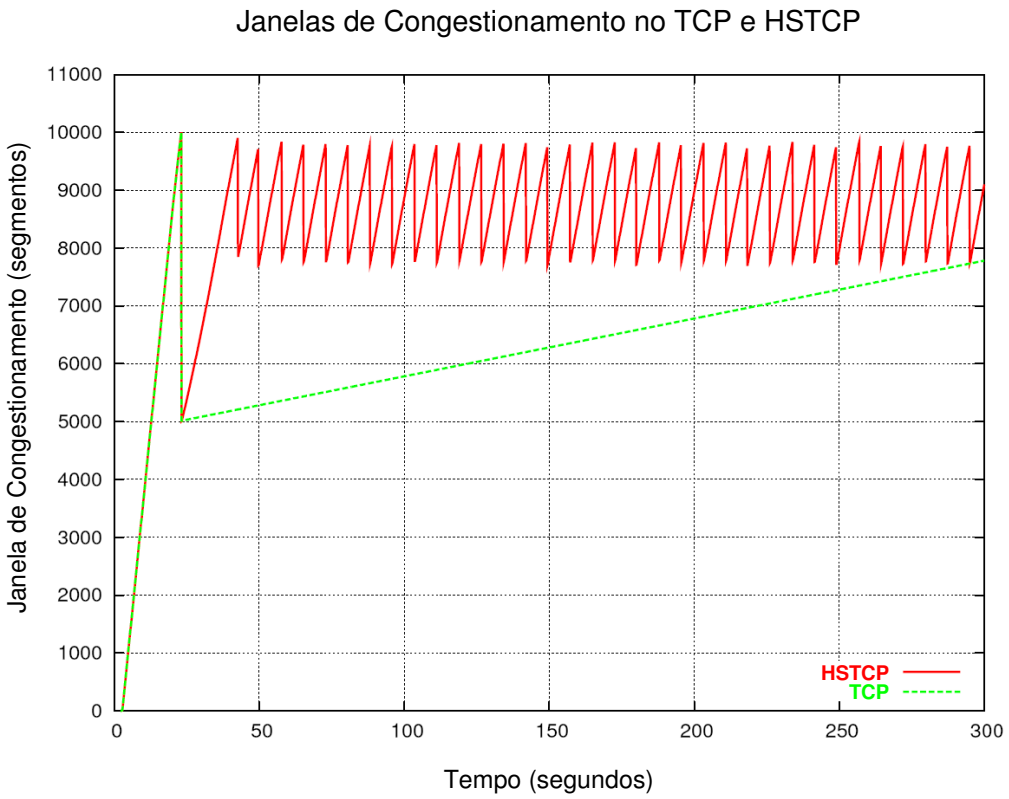

Figura 11: Comparação das janelas de congestionamento no TCP e HSTCP [22].

Pode-se se notar que com o HSTCP se torna possível alcançar valores muito maiores de janela de segmentos, e conseqüentemente vazões mais altas, utilizando taxas de perdas e número de RTTs entre eventos de perda muito mais realistas, possíveis de ser encontrados nas redes de alta velocidade atuais.

3.4. Considerações Adicionais sobre o HSTCP

Um outro modo de comparar os protocolos é visualizar uma conexão HSTCP como sendo equivalente, em termos de tamanho de janela alcançado, a N conexões TCP paralelas, o que demonstra, de forma aproximada, o ganho de performance do protocolo HSTCP em relação ao TCP.

Considerando-se N como sendo inicialmente 1 e também que seu valor aumenta conforme aumenta a janela de congestionamento do HSTCP, pode-se dizer que N(*w*) é o número de conexões TCP paralelas, onde N(*w*) varia com o tamanho da janela *w*. Com base na equação da janela de congestionamento média

do TCP (Equação 23) pode-se relacionar o número de conexões paralelas como sendo igual a N x janela *w* do TCP. Levando-se em conta a relação anterior e a equação da janela de congestionamento média do HSTCP (Equação 32), e com base em [2], tem-se:

$$N(w) = 0{,}23w^{0{,}4}. \quad (33)$$

Este é o número de conexões TCP equivalente a uma conexão HSTCP, com base em sua função resposta e para N(*w*) ≥ 1.

A relação é demonstrada na Tabela 6. Os valores de N(*w*) mostram que uma conexão HSTCP pode ser equivalente a até 23 conexões TCP, para uma janela de congestionamento com 100.000 segmentos. Sendo assim, fica comprovado que o HSTCP representa um alto ganho de desempenho em relação ao seu antecessor, para um mesmo cenário.

Tabela 6: Número de conexões TCP paralelas emuladas pela função resposta do HSTCP, de forma aproximada [2].

Janela de Congestionamento (segmentos)	Número N(*w*) de conexões TCP paralelas
1	1
10	1
100	1,4
1.000	3,6
10.000	9,2
100.000	23,0

4. O Protocolo STCP

4.1. Introdução ao STCP

O protocolo STCP (*Scalable* TCP, ou TCP escalável) é, assim como o HSTCP, uma versão modificada do protocolo TCP. Ele foi proposto preliminarmente no final do ano de 2002 por Tom Kelly [3], e até o momento da finalização deste trabalho não houve oficialmente nenhuma informação sobre a criação de um RFC para esse novo protocolo de transporte.

Assim como o HSTCP, o STCP se baseia no conhecido fato da baixa taxa de utilização da capacidade de redes de alta velocidade. Ele tem como objetivo oferecer um melhor aproveitamento da parcela de largura de banda disponível e não-utilizada pelo TCP tradicional, sem impactar o tráfego existente na rede.

O STCP toma como base o mesmo princípio utilizado pelo HSTCP, de não mudar o funcionamento geral do TCP, mas sim o de modificar o seu mecanismo de controle de congestionamento no lado remetente (podendo até utilizar destinatários TCP tradicionais), para operar num modo mais agressivo a partir de uma dada taxa de perda de segmentos e uma dada janela de congestionamento.

Ele também visa obter altas vazões nas redes de alto produto atraso x banda passante, mas com taxas de perda aceitáveis, dentro da realidade do que pode ser encontrado nas redes de alta velocidade atuais.

Da mesma forma que o HSTCP, o STCP propõe também modificar somente os parâmetros de aumento e diminuição da janela de congestionamento do TCP, só que de forma mais simples que no HSTCP, utilizando valores constantes para tais parâmetros [3].

O protocolo STCP age da mesma forma que o TCP em cenários de alto congestionamento, devido ao fato de ele utilizar o algoritmo de congestionamento modificado somente quando acima de um patamar específico de taxa de perdas e janela de congestionamento.

4.2. Modelo de Serviço do STCP

O protocolo STCP tem como objetivo alcançar uma alta vazão por conexão utilizando taxas de perdas de segmentos mais realísticas que o TCP. Ele visa também trabalhar de forma escalável tais vazões de forma bem mais rápida que seu antecessor.

O STCP, assim com o TCP e o HSTCP, trabalha sem realimentação explícita pelos roteadores da rede a respeito do estado de congestionamento ao longo do caminho da conexão. Também não requer nenhuma informação adicional sobre congestionamento por parte dos receptores em suas conexões.

Visa possuir um desempenho equivalente a do TCP mediante redes com médio ou alto congestionamento, ou seja, taxas de perdas de segmentos aproximadamente iguais ou maiores que 1%. Da mesma forma, visa também não gerar carga adicional de congestionamento para a rede.

O STCP trabalha com boa parte do modelo de serviço do protocolo TCP tradicional, e o seu desempenho deve ser no mínimo tão bom quanto o do TCP em redes com médio ou alto grau de congestionamento.

Suponha-se o caso de uma conexão TCP num enlace de longa distância, com largura de banda disponível de 1 Gbps e RTTs de 200 ms, o que corresponderia a uma janela de congestionamento de aproximadamente 17.000 segmentos (vide Tabela 4). Ao ocorrer um evento de congestionamento (perda de segmento), a janela de congestionamento seria cortada pela metade (8.500 segmentos) e sua taxa de envio decairia para 500 Mbps. O tempo médio de recuperação, ou seja, o tempo necessário para que a taxa de envio volte a ser de 1 Gbps, é de 8.500 RTTs ou aproximadamente 28 minutos. Em muitas WANs de alta velocidade este tempo de recuperação seria muito maior do que o tempo entre períodos transientes de congestionamento. Tempos de recuperação maiores do que alguns poucos minutos poderiam levar a uma baixa utilização da largura de banda disponível de uma rede mesmo quando já não há congestionamento nesta rede por um longo período de tempo.

Na Tabela 4 são mostradas características de uma conexão TCP. Devem ser observados os imensos tempos de recuperação, assim como as taxas mínimas de perda que são necessárias, quando são utilizadas conexões de alta velocidade.

O protocolo STCP propõe-se a resolver este problema por meio do uso de um algoritmo de controle de congestionamento escalável, com tempos de recuperação muito mais rápidos, necessários à melhoria das taxas de utilização das redes de alta velocidade. Com o STCP o tempo necessário para dobrar qualquer taxa de envio é de aproximadamente 70 RTTs [3].

Para a mesma conexão citada anteriormente, o tempo necessário para dobrar a taxa de envio, ou seja, fazê-la voltar a 1 Gbps com RTTs de 200 ms, seria de aproximadamente 70 RTTs ou 14 segundos. Isto sugere que o STCP pode utilizar de forma muito mais otimizada a largura de banda disponível em redes de alta velocidade que enfrentam congestionamento transiente.

4.3. Controle de Congestionamento do STCP

O controle de congestionamento foi a principal área de modificação do protocolo STCP, assim como no HSTCP, em relação ao TCP. A fase de prevenção de congestionamento (*Congestion Avoidance* – CA) foi modificada para agir de forma mais otimizada quando a janela de congestionamento e a taxa de perdas atingem um patamar especificado, começando este efetivamente a trabalhar para o alcance de vazões mais elevadas, através do uso de uma taxa mais elevada de aumento de sua janela de congestionamento, como será mostrado.

4.3.1. Partida Lenta

O mecanismo de partida lenta do TCP (*Slow Start* – SS) não foi modificado no STCP. Ele segue a mesma linha de desenvolvimento do HSTCP. Foi considerado que a taxa de aumento da janela de congestionamento nessa fase é suficientemente rápida, não sendo a fonte geradora da baixa utilização da banda numa conexão, levando-se em conta o fato de que não há realimentação explícita sobre o estado de congestionamento da rede proveniente dos roteadores ao longo da conexão.

4.3.2. Prevenção de Congestionamento

A principal mudança no controle de congestionamento do STCP em relação ao TCP foi a forma com a qual o algoritmo de atualização da janela de congestionamento, o AIMD (*Additive-Increase, Multiplicative-Decrease*), realiza essa atualização, de forma que se tenha uma utilização mais eficiente da largura de banda disponível na rede.

Relembrando que o protocolo TCP utiliza as seguintes equações para ajustar o tamanho da janela de congestionamento *w* (Equações 16 e 18):

Reconhecimento: $w = w + \frac{a}{w}$.

Evento de Perda: $w = w - b \times w$ (por três ACKs duplicados).

Onde a e b são definidas em unidades de MSS, sendo a = 1 e b = 0,5.

A proposta do STCP é modificar o valor dos parâmetros *a* e *b* de forma que o tamanho de sua janela de congestionamento possa variar escalarmente, da seguinte forma:

Reconhecimento: $w = w + a$. (34)

Evento de Perda: $w = w - b \times w$ (por três ACKs duplicados). (35)

Onde *a* e *b* são definidos em unidades de MSS, sendo o parâmetro aditivo *a* igual a 0,01 (note que não há divisão por *w* neste caso), e o parâmetro de decréscimo *b* igual 0,125, o que representa uma diminuição de um oitavo (1/8) ou 12,5% no tamanho da janela *w* mediante um evento de perda de segmento, detectado através do recebimento de três ACKs duplicados.

O objetivo é que quando o produto atraso x banda passante for alto, visando alcançar uma maior vazão, o algoritmo de congestionamento do STCP possa se recuperar de congestionamentos transientes de forma mais rápida, bem como alcançar altas vazões utilizando melhor a capacidade da rede.

A Figura 12 abaixo mostra a dinâmica das janelas de congestionamento, considerando uma conexão TCP e uma conexão STCP nas mesmas condições, em dois enlaces dedicados de capacidades distintas *c* e *C*, onde a capacidade do primeiro enlace, *c*, é menor que a capacidade do segundo enlace, *C*.

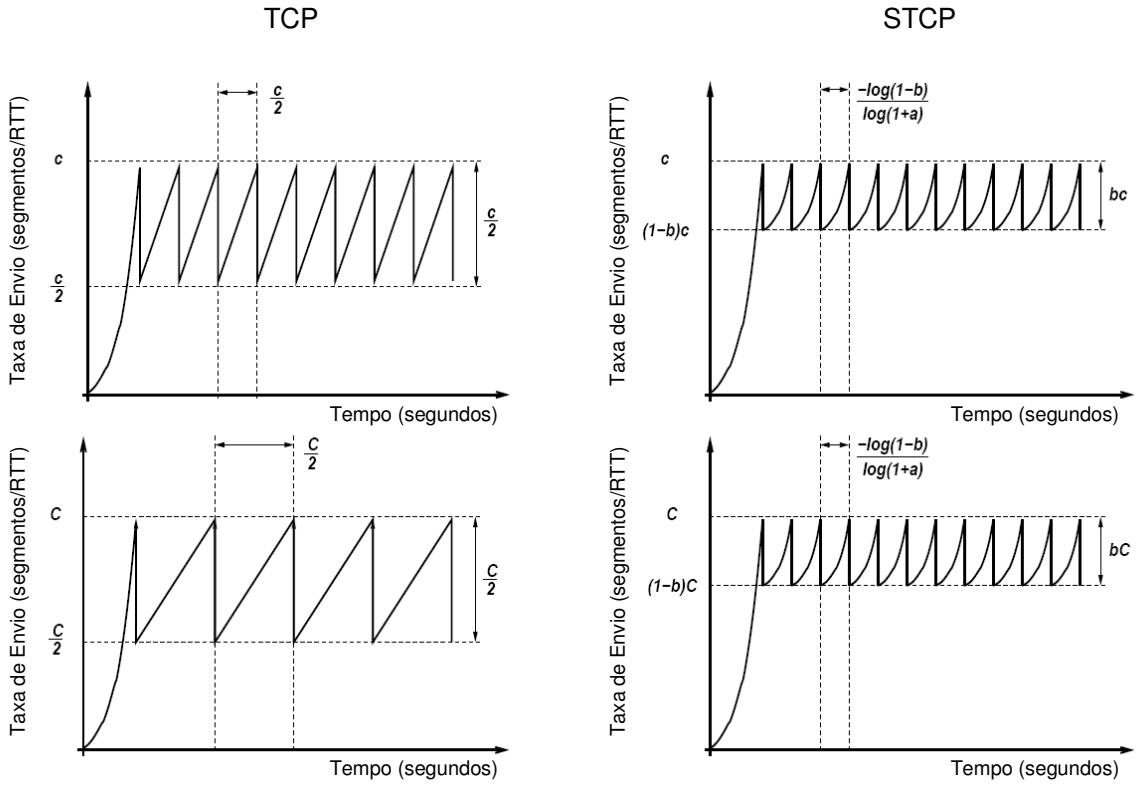

Figura 12: Escalonamento das janelas de congestionamento no TCP e STCP [3].

De acordo com a Figura 12, pode-se observar que os tempos de recuperação de eventos de perda no TCP tradicional são proporcionais ao tamanho da janela da conexão (sua capacidade) e ao seu tempo de ida e volta (RTT). O STCP tem seus tempos de recuperação de perdas unicamente baseados no RTT da conexão. Esta invariância com a capacidade do enlace permite ao STCP ter uma vantagem de desempenho em relação ao TCP tradicional em redes de alta velocidade [3]. O escalonamento da janela de congestionamento no STCP fica então dependente somente dos parâmetros *a*, *b* e da taxa de perdas de segmentos, independentemente da capacidade do enlace.

O algoritmo de atualização da janela de congestionamento se relaciona com esta janela através da sua função resposta. A função resposta generalizada do protocolo STCP, para pequenas taxas de perda fim-a-fim, é definida da seguinte forma abaixo:

$$w = \frac{a}{b} \times \frac{1}{p}, \text{ ou } w = \frac{0{,}01}{0{,}125p}, \text{ ou simplesmente } w = \frac{0{,}08}{p}, \quad (36)$$

onde *p* é a taxa média de perda de segmentos.

Relembrando, a função resposta média do TCP (Equação 23) foi anteriormente definida como:

$$w = \frac{1{,}22}{\sqrt{p}} \text{ ou } w = \frac{1{,}22}{p^{0{,}5}}.$$

Na Figura 13 é mostrada uma comparação entre as funções resposta do protocolo TCP padrão, do HSTCP e do STCP, para conexões em iguais condições. As modificações inseridas pelo protocolo STCP só começam a operar a partir de um determinado ponto da curva, baseado num valor mínimo de taxa de perdas de segmentos e da janela de congestionamento; anteriormente a este ponto ele se comporta exatamente da mesma forma que o TCP tradicional. Como pode ser percebido, o STCP apresenta o mesmo tipo de comportamento que o HSTCP, tornando sua função resposta mais inclinada (comportamento mais agressivo) a partir do ponto selecionado por cada protocolo. O STCP começa a operar antes do ponto selecionado pelo HSTCP, com um ângulo de inclinação pouco maior, o que sugere que este é capaz de alcançar vazões mais altas que o HSTCP (e muito mais altas que o TCP padrão) nas mesmas condições.

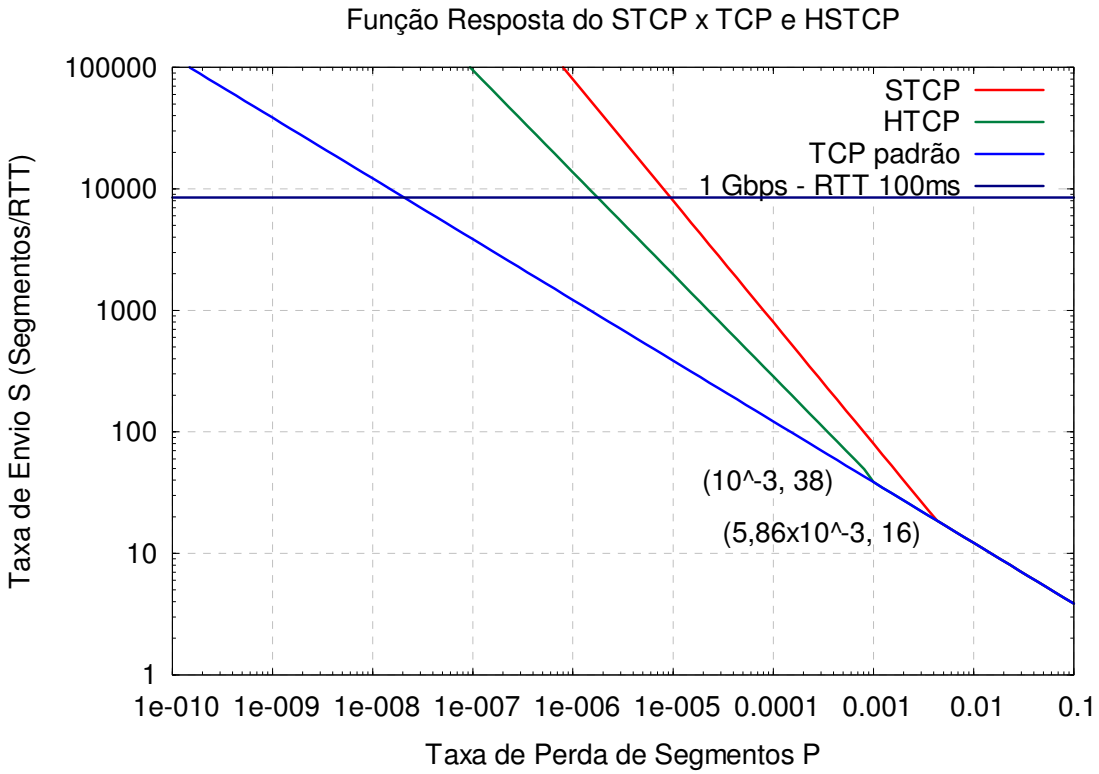

Figura 13: Funções resposta para o TCP padrão, HSTCP e o STCP.

Conexões TCP tradicionais não utilizam efetivamente grandes janelas e na prática sofrem limitações de espaço disponível nos *buffers* de envio e recepção, e tendem a manter um determinado tamanho da janela de congestionamento [3]. Seja este valor limitado da janela de congestionamento chamado de *w'*. A este tamanho de janela está também associado um valor de taxa de perda, que é a taxa de perda máxima necessária para suportar tamanhos de janela iguais ou superiores a *w'*. Seja este valor da taxa de perda chamado de *p'*. Define-se então que a modificação do protocolo STCP irá apenas trabalhar a partir do ponto (*p'*, *w'*).

A escolha de *w'* foi uma decisão estratégica. Escolhendo-se o valor de janela de congestionamento *w'* = 16 no TCP tradicional, seria exatamente o necessário para que, quando o TCP atingir uma janela de congestionamento com tamanho aproximadamente igual a 420 segmentos, o STCP esteja recebendo exatamente o dobro da largura de banda em relação ao TCP, com o mesmo valor de RTT.

Isto sugere também que preocupações sobre o fato do STCP receber larguras de banda muito acima daquelas disponíveis para o TCP, para valores de

janela maiores que *w'*, não devam de forma alguma surgir até que a janela de congestionamento seja grande o suficiente para que não haja problemas com os tempos de recuperação [3].

Para os propósitos deste, fica então definido o valor da janela mínima para início do funcionamento do STCP como sendo *w'* = 16 segmentos. Isto corresponde a 24Kb com segmentos de 1500 bytes, e a taxa de perda de segmentos *p'* = 5,86 x 10^{-3}. O ponto (*p'*, *w'*) pode ser visualizado na Figura 13.

Para assegurar uma função resposta contínua e decrescente, a curva do STCP deve passar pelo ponto (*p'*, *w'*), com a seguinte limitação em *a* e *b*:

$$\frac{a}{b} = p' \times w' \approx \sqrt{1{,}5\,p'}\,. \tag{37}$$

O número de variáveis fica limitado a uma, pois escolhendo *b* fixa-se *a*, já que os valores de *p'* e *w'* foram previamente definidos. Dessa forma, escolhendo-se o valor de *b* = 0,125 resulta então em a = 0,01.

4.4. Considerações Adicionais sobre o STCP

Como mencionado anteriormente, a escolha dos valores de *a* e *b* no algoritmo de controle de congestionamento do STCP foi uma decisão estratégica. A Tabela 7 reforça esta escolha por meio de comparações com outros possíveis valores para estes parâmetros, destacando seus valores ótimos.

Tabela 7: Características de uma conexão STCP com RTTs de 200 ms, usando diferentes valores para os parâmetros *a* e *b* [3, com modificações].

b	a	Tempo de recuperação de um evento de perda	Tempo para diminuir a vazão pela metade	Tempo para dobrar a vazão
1/2 ou 0,5	2/50 ou 0,04	3,54 s	0,20 s	3,54 s
1/4 ou 0,25	1/50 ou 0,02	2,91 s	0,48 s	7,0 s
1/8 ou 0,125	**1/100 ou 0,01**	**2,68 s**	**1,04 s**	**13,9 s**
1/16 ou 0,0625	1/200 ou 0,005	2,59 s	2,15 s	27,8 s

5. Testes dos Protocolos

Nesta seção são descritos os testes realizados para avaliar o desempenho dos protocolos HSTCP e STCP, em comparação ao TCP, baseando-se nas vazões alcançadas por estes mediante variados cenários de atraso e perdas.

Uma rede gigabit foi montada entre dois microcomputadores, onde configurações do núcleo do sistema operacional foram feitas visando melhorar o desempenho, e scripts de automatização foram utilizados, de forma a viabilizar a execução de todos os experimentos propostos.

Os detalhes sobre recursos, configuração, automatização, execução e resultados dos testes encontram-se nas subseções a seguir.

5.1. Recursos Utilizados

Nos experimentos realizados neste trabalho foram utilizados dois (02) microcomputadores com os seguintes recursos:

Especificações do Computador 1 – Cliente:

- 01 Placa Mãe Intel SE7520BD2V Dual Core SATA RAID LAN
- 02 Processadores Xeon 3.0 GHz 800 MHz 1MB de Cache
- 02 Memórias 512 MB DDR 333 MHz ECC Kingston – total 1 GB
- 02 Canais SCSI Ultra 320 com RAID 0 e 1 Integrada
- 02 HDs 36 GB SCSI 10.000 RPM Seagate
- 02 Placas Gigabit Ethernet on-board

Especificações do Computador 2 – Servidor:

- 01 Placa Mãe Intel D865PERLL ATX P4 AGP8X SATA RAID LAN
- 01 CPU Intel Pentium 4 2.8E GHz HT 1MB 800FSB S478
- 02 Memórias 256 MB DDR400 PC3200 Transcend/Kingston – total 512 MB
- 01 HD 80 GB Serial ATA Seagate Barracuda 7.200 RPM
- 01 Placa Gigabit Ethernet on-board

Em ambas as máquinas foi utilizado o Sistema Operacional Linux rodando a distribuição Red Hat Fedora Core 5 [23] com *kernel* atualizado.

Para executar os testes foi utilizado o Iperf 2.0.2 [24], um software de análise de desempenho de banda e cálculo de perda de datagramas na rede que é mantido pela Universidade de Illinois sob licença GPL. O Iperf emula transmissões de dados numa rede cliente-servidor e também possui a facilidade da configuração de vários parâmetros, possibilitando assim uma análise de desempenho bastante eficiente.

Para viabilizar os experimentos, simulando ambientes mais próximos da realidade de uma rede real com congestionamento, atraso e perdas, foi utilizado o Netem (*Network Emulator*) [25] que, entre outras funções, serve para emular atrasos e taxas de perda de segmentos em uma rede. O Netem já está incluso e habilitado nos *kernels* a partir da versão 2.6 das distribuições Linux mais conhecidas.

Os gráficos de vazão foram gerados com a utilização da ferramenta Gnuplot [26], que é um aplicativo de domínio público, destinado à construção de gráficos e superfícies baseando-se em funções ou simplesmente coleções de dados.

5.2. Configuração dos Experimentos

Nos testes foi utilizada uma topologia que consiste em dois computadores interligados diretamente através de uma rede Ethernet de 1 Gbps. Nessa rede foi adotada a arquitetura cliente-servidor para a transmissão/recepção de dados, tendo sido o Computador 1 o cliente e o Computador 2 o servidor.

Algumas configurações nas placas de rede e em diferentes parâmetros do sistema operacional são necessárias para que sejam alcançadas altas vazões [5, 27, 28, 29, 30]. As modificações descritas a seguir foram feitas para todos os testes para os três protocolos alvo de análise neste trabalho.

As placas de rede foram configuradas para operarem em modo *full-duplex* a 1000 Mbps e a auto-negociação de velocidade foi desabilitada. O controle de fluxo realizado na subcamada MAC pelo Ethernet também foi desabilitado para que não influísse nas medidas.

A versão do *kernel* do sistema operacional utilizada foi a 2.6.18, pois esta continha como padrão o TCP Reno nativo, além dos protocolos de transporte alternativos (HSTCP e STCP entre outros) como módulos, sendo necessário que tais módulos fossem corretamente carregados antes da realização dos testes.

As filas de transmissão e recepção entre os subsistemas de rede do *kernel* e o *driver* da interface de rede foram aumentadas, de forma que fossem diminuídas as perdas geradas pelo esgotamento de espaço nessas filas. A fila de transmissão (*txqueuelen*) foi aumentada de 1000 para 2000 segmentos (vide Apêndices B e C) e a fila de recepção (*netdev_max_backlog*) foi aumentada de 1000 para 3000 segmentos.

O *cache* do TCP também foi modificado. O TCP guarda em um *cache* algumas estatísticas de conexões anteriores, o que pode fazer com que as medidas realizadas anteriormente influenciem as novas, portanto tais estatísticas foram eliminadas a cada conexão. Para isso foram habilitadas as variáveis tcp_no_metrics_save e route.flush do TCP de forma a eliminar estatísticas antigas de tamanho de janela e *Threshold* entre sessões TCP, bem como rotas utilizadas.

O tamanho dos *buffers* de transmissão e recepção do sistema operacional e dos *sockets* do TCP foram também aumentados de forma que permitissem que grande quantidades de dados fossem transmitidas sem limitações de memória, bem como o alcance de altos valores da janela de congestionamento. As modificações nos *buffers* de recepção do sistema operacional foram rmem_default = “65536” (valor padrão), rmem_max = "33554432" (valor máximo) e nos nos *buffers* de recepção do TCP foram tcp_rmem = "4096 87380 33554432", onde os 3 valores são respectivamente o mínimo, o padrão e máximo. As modificações nos *buffers* de recepção do sistema operacional foram wmem_default = “65536” (valor padrão), wmem_max = "33554432" (valor máximo) e nos nos *buffers* de transmissão do TCP foram tcp_wmem = "4096 65536 33554432", onde os 3 valores são respectivamente o mínimo, o padrão e máximo. Os valores equivalentes são: 33554432 = 32 Mb, 87380 = 85 Kb, 65536 = 64 Kb, 4096 = 4 Kb.

O valor de MSS usado nos testes foi de 1460 bytes, com MTU (*Maximum Transmission Unit*) de 1500 bytes, ou seja MSS + 20 bytes de cabeçalho transporte TCP + 20 bytes de cabeçalho de rede IP, que resulta no total de 1500 bytes.

As modificações feitas nos arquivos de configuração do Linux, em ambas as máquinas, podem ser vistas no Apêndice D.

Algumas configurações importantes já estavam habilitadas na versão do *kernel* utilizada, tais como TCP *window scaling* (permite a utilização de janelas maiores do que 64 Kbytes), e o uso de SACK e DSACK (ambos fazem o reconhecimento seletivo de segmentos, de forma a economizar retransmissões desnecessárias e estouros de temporização).

5.3. Execução dos Testes

Para realizar os testes foram criados alguns scripts de automatização, que contêm os comandos de execução dos testes propriamente ditos no lado cliente (vide Apêndice B) e no lado servidor (vide Apêndice C) da conexão, além da consolidação dos valores numéricos, geração de intervalo de confiança, e geração dos valores finais escalonados (base para a geração dos gráficos de vazão).

Foram realizadas seqüências de 10 transferências contínuas de dados nos testes, cada uma com 60 segundos de duração, para cada par latência x taxa de perda de segmentos, utilizando-se cada um dos 3 protocolos de transporte avaliados, configurados dinamicamente pelos scripts durante as execuções, através da modificação do valor da variável do TCP chamada tcp_congestion_control.

As latências utilizadas nos testes foram: 0 ms, 15 ms, 30 ms, 45 ms, 60 ms e 75 ms. As taxas de perda segmentos utilizadas foram: 0,0001%, 0,0002%, 0,0005%, 0,001%, 0,002%, 0,005% e 0,01%.

No total foram realizados 1260 testes, acrescidos de comandos de modificação de parâmetros do sistema operacional, configuração da interface de rede, e geração de logs das configurações, executados numa bateria de 21 horas contínuas de testes em laboratório.

5.4. Análise dos Resultados

Durante os testes realizados o parâmetro de desempenho considerado para avaliação foi a vazão alcançada por cada protocolo em diferentes cenários de atraso e perdas. Nesses testes, o atraso foi variado de 0 a 75 ms, e a taxa de perda de segmentos foi variada de 10^{-6} (0,0001%) a 10^{-4} (0,01%).

Como já mencionado, foram colhidas 10 amostras (transferências de dados) com 60 segundos de duração cada. Para cada grupo de amostras dos experimentos foi utilizado um intervalo de confiança de 90% para a média (vide Apêndice E), o que foi representado nos gráficos pelas barras de erro verticais.

Os resultados obtidos nos experimentos realizados são mostrados nas Figuras de 14 a 19.

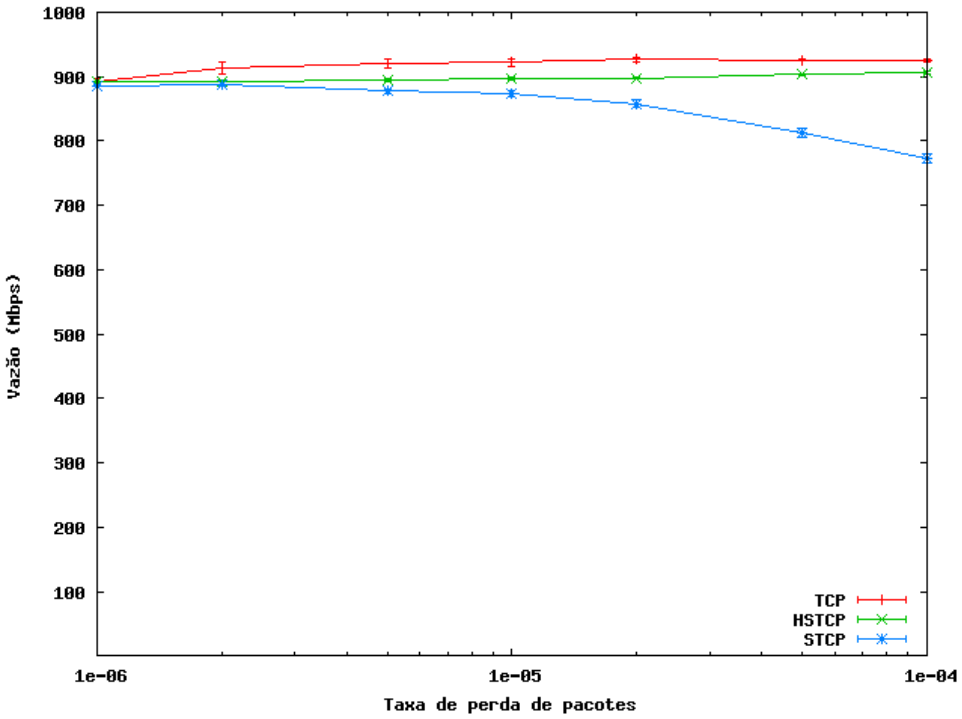

Figura 14: Vazão obtida para latência igual a 0 ms.

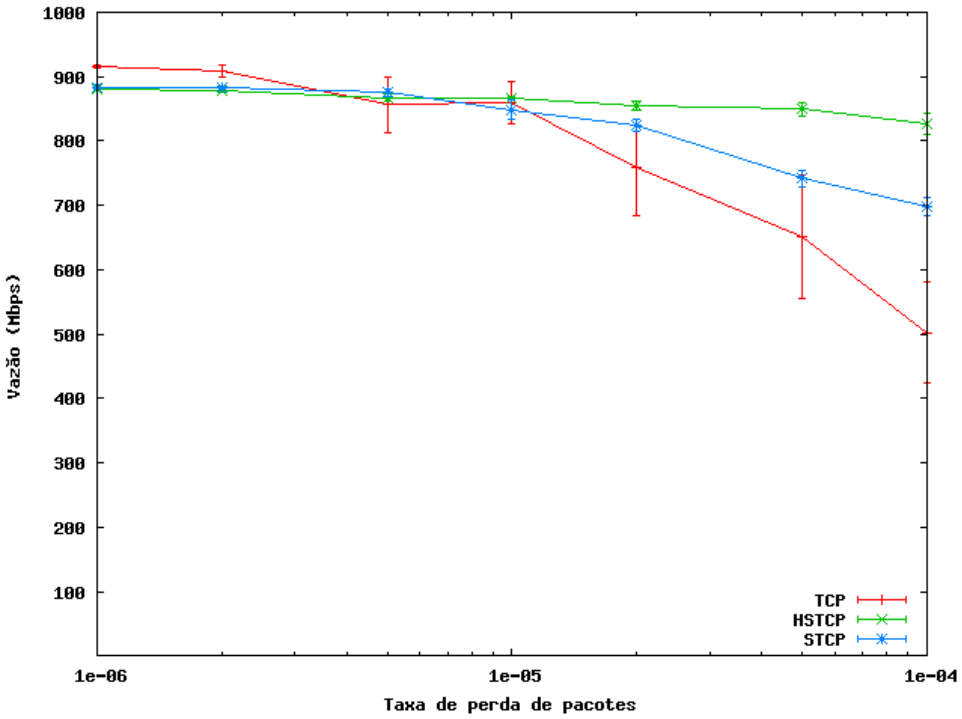

Figura 15: Vazão obtida para latência igual a 15 ms.

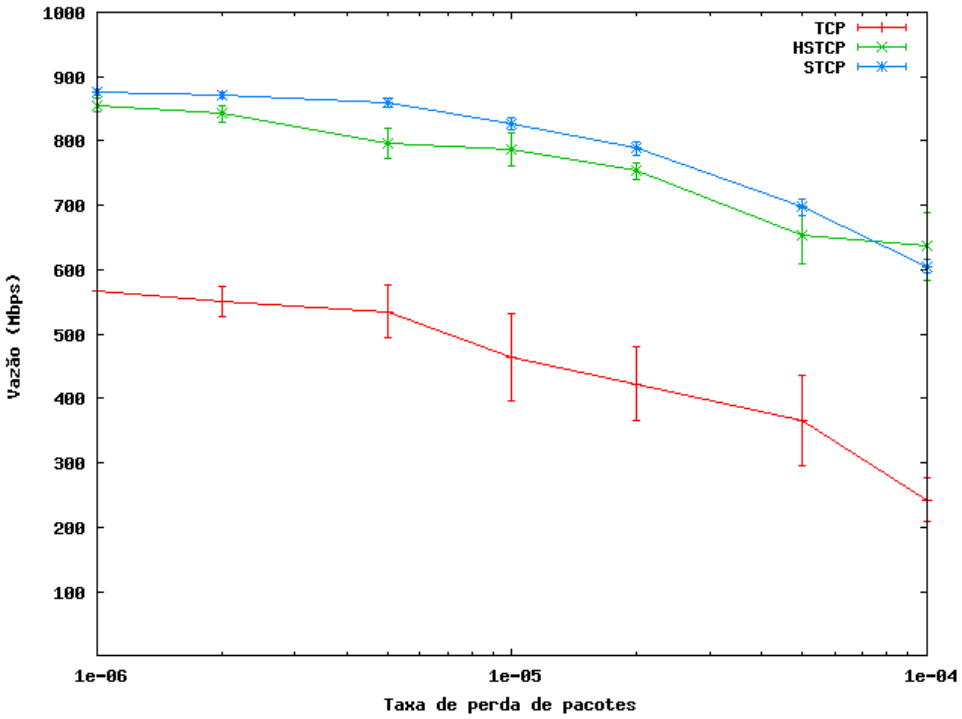

Figura 16: Vazão obtida para latência igual a 30 ms.

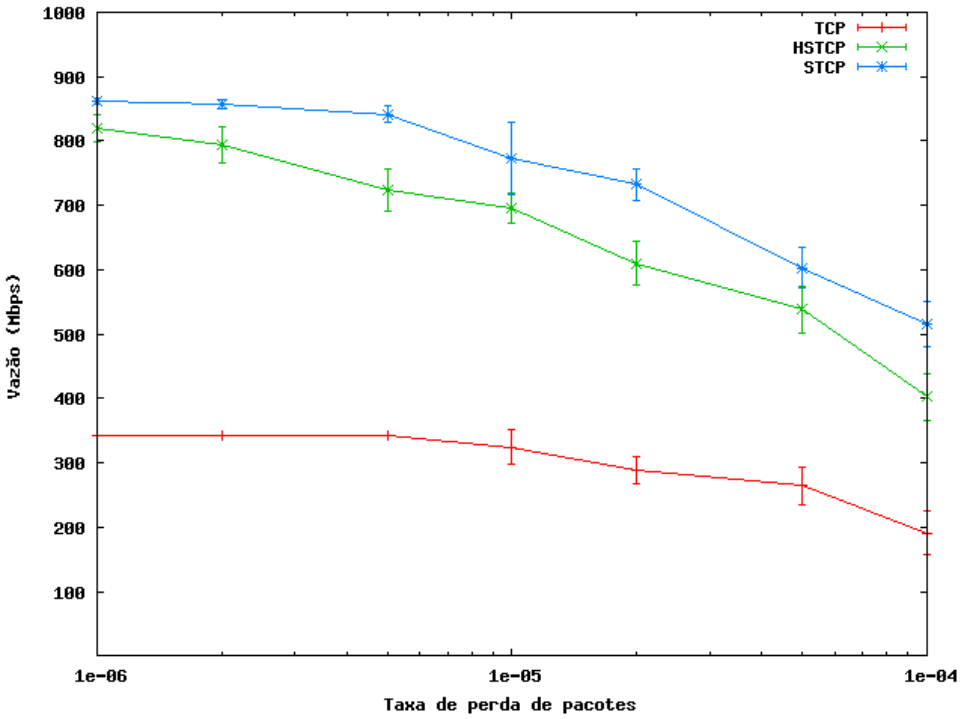

Figura 17: Vazão obtida para latência igual a 45 ms.

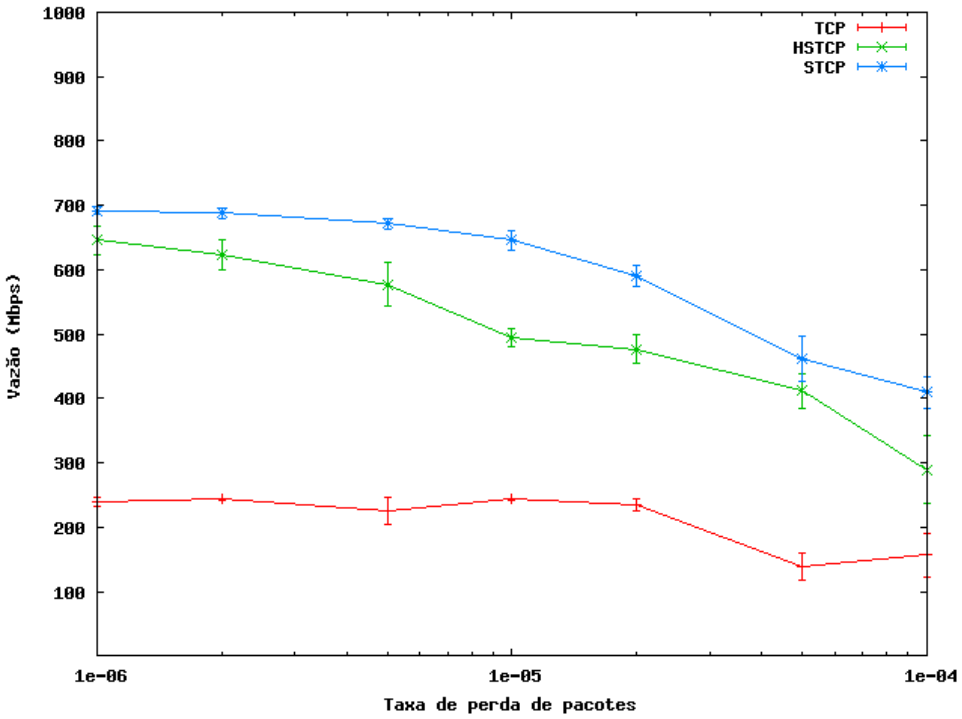

Figura 18: Vazão obtida para latência igual a 60 ms.

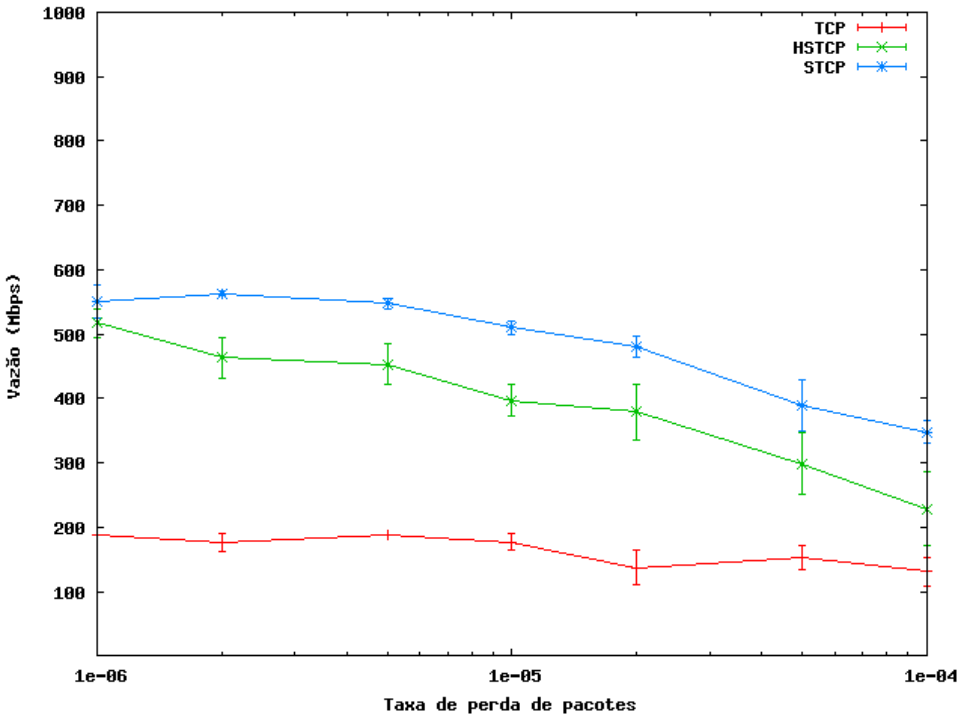

Figura 19: Vazão obtida para latência igual a 75 ms.

Os resultados mostram primeiramente que, como esperado, o protocolo TCP demonstra claramente uma queda significante em seu desempenho à medida que são aumentados o atraso na entrega dos segmentos e a conexão é exposta à uma taxa de perda de dados moderada. De um modo geral isto ocorre onde o produto atraso x banda passante para as conexões é mais alto.

Para os valores de atraso próximos ou iguais a zero, e com moderada taxa de perdas de segmentos, os três protocolos atingiram a vazão máxima do enlace para esse tipo de protocolo de transporte, algo em torno de 900 Mbps. Isso se deve ao fato de as janelas de congestionamento serem aumentadas rapidamente (e com elas a vazão) após a ocorrência de uma perda de segmento pelos mecanismos de controle de congestionamento utilizados pelos protocolos, devido ao baixo atraso. Conforme os segmentos de dados são enviados, os segmentos de reconhecimento são recebidos rapidamente pelo transmissor, gerando uma recuperação rápida das perdas, com aumento proporcionalmente rápido das taxas de transmissão.

Para os valores altos de taxa de perda (a partir de 10^{-4} ou 0,01%), os três protocolos demonstraram a tendência de se igualarem em desempenho, num valor

de vazão baixo; tendência a qual o HSTCP e o STCP foram projetados para executar, até chegar ao seu valor de transição, passando a operar exatamente conforme o TCP.

Os protocolos HSTCP e STCP se portaram de forma bastante estável durante os testes, sem quedas de conexão ou variações abruptas de vazão. As barras de erro mostram que a vazão encontrada nas conexões variou pouco e se portou de forma bastante coerente, com variações próximas ou até menores que as apresentadas pelo TCP em suas conexões com atraso até 45 ms, tendo sido somente um pouco maiores em alguns pontos a partir deste valor de atraso.

De um modo geral, os protocolos HSTCP e STCP comprovaram ter um desempenho superior ao TCP nos cenários onde o atraso e perdas estão presentes e o produto atraso x banda passante é alto. A vazão obtida nos testes dos dois protocolos foi, em média, entre 2 a 3 vezes maior que a vazão do protocolo TCP, tendo estes obtido os melhores resultados nos cenários onde o atraso nos enlaces era maior.

Os gráficos mostram ainda que o protocolo STCP obteve o melhor resultado em termos de desempenho, dentre todos os protocolos testados, nos cenários de médio e alto atraso (de 30 a 75 ms). Em alguns pontos o STCP chegou a obter vazões até 150 Mbps acima (aproximadamente) daquelas obtidas pelo HSTCP (testes de 60 ms), e até aproximadamente 500 Mbps maiores que as obtidas pelo TCP padrão (testes de 45 ms).

O protocolo HSTCP obteve também um desempenho muito bom sobre o TCP, tendo conseguido alcançar vazões até 450 Mbps acima daquelas alcançadas por este. O desempenho do HSTCP foi superior ao do STCP somente nos testes com enlaces de baixa latência (0 e 15 ms), onde ele chegou a alcançar um valor aproximadamente 120 Mbps acima do alcançado pelo STCP (testes de 15 ms).

Como fatores a verificar em trabalhos futuros, destaca-se o fato de o protocolo TCP ter elevado sua vazão com o aumento das perdas, assim como ter obtido um desempenho superior ao HSTCP e o STCP nos testes com latência igual a 0 ms e no início dos testes de 15 ms, já que em teoria este seria inferior ao HSTCP e o STCP em termos de desempenho. Do mesmo modo deve-se verificar o fato de o HSTCP ter se desempenhado melhor que o STCP nos testes de 0 ms e 15 ms.

6. Conclusões

Neste trabalho foi avaliado o desempenho do protocolo TCP em relação a dois de seus candidatos a sucessor, o HSTCP e o STCP, em uma rede de alta velocidade. Foi demonstrado que é possível implementar um rede gigabit capaz de alcançar altas vazões, utilizando de maneira mais eficiente a banda disponível nos enlaces gerados, com o uso de tais protocolos.

Como problema conhecido, o TCP tem baixo desempenho onde o produto atraso x banda passante é alto. Este problema se refletiu claramente durante os testes realizados. O seu mecanismo de controle de congestionamento, baseado no algoritmo AIMD se comporta de forma conservadora em conexões de alta banda passante, não atendendo bem às demandas de taxas de dados atuais, por não conseguir utilizar totalmente a banda disponível em tais enlaces.

Por outro lado, os protocolos alternativos HSTCP e STCP, baseados no TCP, se mostraram soluções bastante eficientes para resolver o problema de baixa utilização de banda do TCP atual. Seus resultados demonstraram que eles podem atingir vazões entre 2 e 3 vezes maiores que as atingidas pelo TCP em cenários idênticos e com grande atraso. Porém, para as taxas de perdas mais altas (a partir de 10^{-4} ou 0,01%), as vazões obtidas pelos três protocolos avaliados demonstraram uma tendência de se igualar, como esperado.

Os resultados mostram ainda que, no geral, o protocolo STCP obteve o melhor desempenho dentre todos os protocolos testados. Sua vantagem em relação aos outros fica mais evidente nos cenários onde o atraso é maior. O protocolo HSTCP também obteve um desempenho muito bom sobre o TCP, ficando na frente do STCP somente nas conexões de baixa latência.

Embora implementadas sobre uma arquitetura simples, as conexões de teste dos protocolos de transporte em questão puderam ser avaliadas de forma consideravelmente eficiente, sem restrições de implementação ou teste, levando em conta diferentes cenários emulados de atraso e taxas de perda de segmentos em uma rede gigabit experimental, atingindo o objetivo proposto por este trabalho.

Como proposta de trabalhos futuros nesse assunto, propõe-se o uso de outros emuladores, como por exemplo o NistNet [31], a fim de verificar os resultados

obtidos neste trabalho. Sugere-se estudos mais aprofundados das vazões alcançadas com a utilização de tráfegos de fundo reais, com suas taxas e atrasos variados, causando efeitos de queda de desempenho nas conexões principais. Sugere-se ainda o estudo de cenários de múltiplas conexões simultâneas e em ambos os sentidos paralelamente, utilizando iguais e diferentes protocolos em cada uma delas. Uma arquitetura de rede diferente com “n” conexões passando por “n” roteadores ao longo do caminho também é recomendada, trazendo tais experimentos para uma realidade mais próxima da que pode ser encontrada nas grandes redes atualmente.

7. Referências Bibliográficas

[1] Katabi, D.; Handley, M. & Rohrs, C. – "Congestion Control for High Bandwidth-Delay Product Networks". Em ACM SIGCOMM, páginas 89 a 102. (2002) – URL alternativa: http://www.sigcomm.org/sigcomm2002/papers/xcp.pdf – Último acesso: Setembro/2006.

[2] Floyd, Sally – "High-Speed TCP for Large Congestion Windows", RFC 3649, Dezembro 2003 – URL: ftp://ftp.rfc-editor.org/in-notes/rfc3649.txt – Último acesso: Setembro/2006.

[3] Kelly, Tom – "Scalable TCP: Improving Performance in Highspeed Wide Area Networks". Em ACM Computer Communication Review, volume 33, páginas 83-91 (2003) – URL alternativa: http://datatag.web.cern.ch/datatag/papers /pfldnet2003-ctk.pdf – Último acesso: Agosto/2006.

[4] Jin, C., Wei; D. X. & Low, S. H. – "Fast TCP: Motivation, Architecture, Algorithms, Performance". Em IEEE INFOCOM (2004) – URL alternativa: http://www.ieee-infocom.org/2004/Papers/52_2.PDF – Último acesso: Setembro/2006.

[5] Costa, Luís Henrique M. K.; Rezende, José F. & Rubinstein, Marcelo G. – Artigo: "Avaliação Experimental e Simulação do Protocolo TCP em Redes de Alta Velocidade" – XXII Simpósio Brasileiro de Telecomunicações - SBrT'05, Setembro 2005, Campinas, SP – URL alternativa: http://www.gta.ufrj.br/ftp /gta/TechReports/RCR05.pdf – Último acesso: Maio/2007.

[6] J. Postel – "Transmission Control Protocol", RFC 793, Setembro 1981 – URL: http://www.rfc-editor.org/rfc/rfc793.txt – Último acesso: Agosto/2006.

[7] J. Postel – "Internet Protocol: DARPA Internet Program Protocol Specification", RFC 791, Setembro 1981 – URL: http://www.rfc-editor.org/rfc/rfc791.txt – Último acesso: Agosto/2006.

[8] T. Socolofsky & C. Kale – "A TCP/IP Tutorial", RFC 1180, Janeiro 1991 – URL: http://www.rfc-editor.org/rfc/rfc1180.txt – Último acesso: Agosto/2006.

[9] Kurose, James F. & Ross, Keith W. – "Redes de Computadores e a Internet: uma abordagem top-down" – 3ª Edição – Editora Pearson Addison Wesley – São Paulo, 2006.

[10] J. Klensin – “Simple Mail Transfer Protocol”, RFC 2821, Abril 2001 – URL: http://www.rfc-editor.org/rfc/rfc2821.txt – Último acesso: Agosto/2006.

[11] J. Postel & J. Reynolds – “TELNET Protocol Specification”, RFC 854, Maio 1993 – URL: http://www.rfc-editor.org/rfc/rfc854.txt – Último acesso: Agosto/2006.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee & R. Feilding – “Hypertext Transfer Protocol - HTTP/1.1”, RFC 2616, Junho 1999 – URL: http://www.rfc-editor.org/rfc/rfc2616.txt – Último acesso: Agosto/2006.

[13] J. Postel & J. Reynolds – “File Transfer Protocol (FTP)”, RFC 959, Outubro 1985 – URL: http://www.rfc-editor.org/rfc/rfc959.txt – Último acesso: Agosto/2006.

[14] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler & D. Noveck – “Network File System (NFS) version 4 Protocol”, RFC 3530, Abril 2003 – URL: ftp://ftp.rfc-editor.org/in-notes/rfc3530.txt – Último acesso: Agosto/2006.

[15] IANA (Internet Assigned Names Authority) – “TCP and UDP Port Numbers” – URL: http://www.iana.org/assignments/port-numbers – Último acesso: Setembro/2006.

[16] S. Floyd, J. Mahdavi, M. Mathis & A. Romanow – “TCP Selective Acknowledgment Options”, RFC 2018, Abril 2003 – URL: http://www.ietf.org/rfc/rfc2018.txt – Último acesso: Maio/2007.

[17] S. Floyd, J. Mahdavi, M. Mathis & M. Podolsky – “An Extension to the Selective Acknowledgement (SACK) Option for TCP”, RFC 2883, July 2000 – URL: http://www.ietf.org/rfc/rfc2883.txt – Último acesso: Maio/2007.

[18] Jacobson, V., Braden, R. & Borman, D. – “TCP Extensions for High Performance”, RFC 1323, Maio 1992 – URL: http://www.rfc-editor.org /rfc/rfc1323.txt – Último acesso: Agosto/2006.

[19] M. Allman, V. Paxson & W. Stevens – “TCP Congestion Control”, RFC 2581, Abril 1999 – URL: http://www.rfc-editor.org/rfc/rfc2581.txt – Último acesso: Agosto/2006.

[20] M. Allman, S. Floyd & C. Partridge – “Increasing TCP's Initial Window”, RFC 3390, Outubro 2002, ftp://ftp.rfc-editor.org/in-notes/rfc3390.txt – Último acesso: Agosto/2006.

[21] S. Floyd, S. Ratnasamy & S. Shenker – “Modifying TCP´s Congestion Control for High Speeds” (2002) – URL: http://www.icir.org/floyd/papers/hstcp.pdf – Último acesso: Agosto/2006.

[22] E. Souza & D. Agarwal – “A HighSpeed TCP Study: Characteristics and Deployment Issues” (2003) – URL: http://dsd.lbl.gov/~evandro/hstcp/ hstcp-lbnl-53215.pdf – Último acesso: Outubro/2006.

[23] Red Hat, Inc. and others – “Red Hat Fedora Core 5” (2006) – URL: http://www.fedora.redhat.com/ – Último acesso: Fevereiro/2007.

[24] A. Tirumala, F. Qin, J. Dugan, J. Ferguson & K. Gibbs – “Iperf” (1999-2005) – URL: http://dast.nlanr.net/projects/Iperf/ – Último acesso: Fevereiro/2007.

[25] Network Emulator – “Netem Network Emulator” (2005) – URL: http://linux-net.osdl.org/index.php/Netem – Último acesso: Fevereiro/2007.

[26] T. Williams & C. Kelley – “Gnuplot” (1986-1993, 1998, 2004) – URL: http://www.gnuplot.info/ – Último acesso: Fevereiro/2007.

[27] TCP Tuning Guide for Linux – URL: http://www-didc.lbl.gov/TCP-tuning/linux.html – Último acesso: Março/2007.

[28] TCP Stacks Testbed – SLAC Stanford Linear Accelerator Center – URL: http://www-iepm.slac.stanford.edu/bw/tcp-eval/ – Último acesso: Março/2007.

[29] TCP Testing – URL: http://linux-net.osdl.org/index.php/TCP_testing – Último acesso: Março/2007.

[30] Gentoo Linux Wiki – “HOWTO TCP Tuning” – URL: http://gentoo-wiki.com/HOWTO_TCP_Tuning – Último acesso: Maio/2007.

[31] Nist Net Home Page – “Nist Net Network Emulator” (2005) – URL: http://www-x.antd.nist.gov/nistnet/ – Último acesso: Maio/2007.

[32] R. K. Jain – “The art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling” – Editora John Wiley & Sons Inc. – USA, 1991.

8. Apêndices

8.1. Apêndice A: Valores de *a(w)* e *b(w)* do HSTCP [2]

A tabela abaixo representa os valores dos parâmetros de aumento e diminuição da janela de congestionamento, *a(w)* e *b(w)*, utilizados pelo mecanismo de controle de congestionamento do HSTCP, para os valores padrões Low_Window = 38, High_Window = 83000, Low_P = 10^{-3}, High_P = 10^{-7}, e High_Decrease = 0,1.

Tabela 8: Valores dos parâmetros *a(w)* e *b(w)* do protocolo HSTCP [2].

w	*a(w)*	*b(w)*	*w*	*a(w)*	*b(w)*
38	1	0,50	17042	38	0,18
118	2	0,44	17981	39	0,18
221	3	0,41	18955	40	0,18
347	4	0,38	19965	41	0,17
495	5	0,37	21013	42	0,17
663	6	0,35	22101	43	0,17
851	7	0,34	23230	44	0,17
1058	8	0,33	24402	45	0,16
1284	9	0,32	25618	46	0,16
1529	10	0,31	26881	47	0,16
1793	11	0,30	28193	48	0,16
2076	12	0,29	29557	49	0,15
2378	13	0,28	30975	50	0,15
2699	14	0,28	32450	51	0,15
3039	15	0,27	33986	52	0,15
3399	16	0,27	35586	53	0,14
3778	17	0,26	37253	54	0,14
4177	18	0,26	38992	55	0,14
4596	19	0,25	40808	56	0,14
5036	20	0,25	42707	57	0,13
5497	21	0,24	44694	58	0,13
5979	22	0,24	46776	59	0,13
6483	23	0,23	48961	60	0,13
7009	24	0,23	51258	61	0,13
7558	25	0,22	53677	62	0,12
8130	26	0,22	56230	63	0,12
8726	27	0,22	58932	64	0,12
9346	28	0,21	61799	65	0,12
9991	29	0,21	64851	66	0,11
10661	30	0,21	68113	67	0,11
11358	31	0,20	71617	68	0,11
12082	32	0,20	75401	69	0,10
12834	33	0,20	79517	70	0,10
13614	34	0,19	84035	71	0,10
14424	35	0,19	89053	72	0,10
15265	36	0,19	94717	73	0,09
16137	37	0,19	101262	74	0,09

8.2. Apêndice B: Script de execução dos testes no cliente

```
#!/bin/bash

################################################################################
#  Script para rodar testes de Vazão x Perda com RTTs fixos em redes gigabit,  #
#          utilizando os protocolos de transporte TCP, HSTCP e STCP            #
#                                                                              #
#    *** Projeto de Graduação VIII-B - Engenharia de Telecom - UERJ ***        #
#                                                                              #
# Autor: Wallace de Souza Espindola                                            #
# Colaboração em linguagem Shell Script: Antônio João N. Dantas                #
#                                                                              #
# Criado em: Fevereiro/2007                                                    #
#                                                                              #
################################################################################

start() {

    ######################################################################
    #                      Configurações Iniciais                        #
    ######################################################################

    # Vai para o diretorio dos testes
    cd /root/Desktop/wallace/Testes

    # Remove antigos arquivos para evitar erros em novas execuções
    rm -rf *.txt

    # Modifica valor de txqueuelen e loga a configuracao em arquivo
    ifconfig eth0 txqueuelen 2000
    ifconfig eth0 > ifconfig.txt

    # Coloca a Placa de Rede em 1000Mbps Full Duplex com Flow Control Off e loga em arquivo
    ethtool -s eth0 speed 1000 duplex full autoneg off
    ethtool -A eth0 autoneg off rx off tx off
    ethtool eth0 > ethtool.txt
    echo "**********************************************************" >> ethtool.txt
    ethtool -a eth0 >> ethtool.txt

    # Verifica o log do driver e1000 da Placa de Rede Intel e loga em arquivo
    dmesg | grep e1000 > dmesg.txt

    # Levanta os modulos dos protocolos extras e loga a config em arquivo
    modprobe tcp_highspeed
    modprobe tcp_scalable
    lsmod | sort > lsmod.txt

    # Inicia o Netem (network emulator) -- o primeiro tem que ser "add"
    tc qdisc add dev eth0 root netem delay 0ms loss 0%

    # Populando matriz de atrasos
    delay[1]="0"
    delay[2]="15"
    delay[3]="30"
    delay[4]="45"
    delay[5]="60"
    delay[6]="75"

    # Populando matriz de perdas
    loss[1]="0.0001"
    loss[2]="0.0002"
    loss[3]="0.0005"
    loss[4]="0.0010"
    loss[5]="0.0020"
    loss[6]="0.0050"
    loss[7]="0.0100"
}

teste() {

    ######################################################################
    #                         Execução dos Testes                        #
    ######################################################################

    for d in `seq 6` do
        for l in `seq 7` do
```

```
            # Seta o atraso e o rtt necessarios para o teste - nesse loop usa "change"
            tc qdisc change dev eth0 root netem delay ${delay[$d]}ms loss ${loss[$l]}%

            for f in `seq 10` do
                # Comandos Iperf para execucao dos testes
                # (-w window size /-t duration (s)/-i interval (s)/-l lenth of buffer)
                case "$tipo" in
                    TCP)
                        flowfile="TCP.vazao.rtt_${delay[$d]}.loss_${loss[$l]}.test$f.txt"
                        consolfile="TCP.consolidado.rtt_${delay[$d]}ms.loss_${loss[$l]}.txt"
                        lossxflowfile="tcplossxflow.rtt_${delay[$d]}ms.txt"
                        iperf -c 10.24.60.71 -w 4M -t 60 -i 5 -l 64k > $flowfile
                        cat $flowfile | tail -n1 >> $consolfile
                        # Salva estatisticas do netem
                        echo "" >> netem_tcp.txt
                        echo "*************** $flowfile ***************" >> netem_tcp.txt
                        echo "" >> netem_tcp.txt
                        tc -s qdisc ls dev eth0 >> netem_tcp.txt
                    ;;
                    HSTCP)
                        flowfile="HSTCP.vazao.rtt_${delay[$d]}.loss_${loss[$l]}.test$f.txt"
                        consolfile="HSTCP.consolidado.rtt_${delay[$d]}ms.loss_${loss[$l]}.txt"
                        lossxflowfile="hstcplossxflow.rtt_${delay[$d]}ms.txt"
                        iperf -c 10.24.60.71 -w 4M -t 60 -i 5 -l 64k > $flowfile
                        cat $flowfile | tail -n1 >> $consolfile
                        # Salva estatisticas do netem
                        echo "" >> netem_hstcp.txt
                        echo "*************** $flowfile ***************" >> netem_hstcp.txt
                        echo "" >> netem_hstcp.txt
                        tc -s qdisc ls dev eth0 >> netem_hstcp.txt
                    ;;
                    STCP)
                        flowfile="STCP.vazao.rtt_${delay[$d]}.loss_${loss[$l]}.test$f.txt"
                        consolfile="STCP.consolidado.rtt_${delay[$d]}ms.loss_${loss[$l]}.txt"
                        lossxflowfile="stcplossxflow.rtt_${delay[$d]}ms.txt"
                        iperf -c 10.24.60.71 -w 4M -t 60 -i 5 -l 64k > $flowfile
                        cat $flowfile | tail -n1 >> $consolfile
                        echo "" >> netem_stcp.txt
                        echo "*************** $flowfile ***************" >> netem_stcp.txt
                        echo "" >> netem_stcp.txt
                        tc -s qdisc ls dev eth0 >> netem_stcp.txt
                    ;;
                    *)
                    ;;
                esac;

            done;

            # Apos rodar o for interno 10 vezes executa lossxflow no for de 7
            lossxflow

        done;
    done;
}

lossxflow() {

    ######################################################################
    #      Gera os arquivos de perda x vazão por tipo de protocolo       #
    ######################################################################

    cat $consolfile |
    while read line;
        do
            flow=`echo $line | awk '{print $7}'`
            loss=`ls $consolfile | awk -F"_" '{print $3}' | cut -f 1,2 -d '.'`
            printf "$loss\t$flow\n" >>/root/Desktop/wallace/Testes/consolidados/$lossxflowfile
        done;
}

testeTCP() {

    ##############################################################################################
    #                                TESTES DO PROTOCOLO TCP                                     #
    ##############################################################################################

    echo ""
    echo "*********************************************************"
```

```
    echo "************** INICIANDO TESTES DO PROTOCOLO TCP **************"
    echo "***************************************************************"
    echo ""

    tipo="TCP"

    # Vai para o diretorio dos testes
    cd /root/Desktop/wallace/Testes/Tests_TCP

    # Remove  arquivos antigos para evitar erros em próximas execuções
    rm -rf *.txt

    # Verifica o sysctl.conf e modifica o sysctl para usar o TCP, logando em arquivo
    echo "***************** Aplica configs do sysctl.conf *****************" > sysctl_TCP.txt
    sysctl -p >> sysctl_TCP.txt
    echo "****************** Pega dados de TCP do sysctl ******************" >> sysctl_TCP.txt
    sysctl -w net.ipv4.tcp_congestion_control=reno
    sysctl -a | grep tcp >> sysctl_TCP.txt
    echo "***************** Pega dados de memoria do sysctl ***************" >> sysctl_TCP.txt
    sysctl -a | grep mem >> sysctl_TCP.txt
    echo "************** Pega dados de route.flush do sysctl **************" >> sysctl_TCP.txt
    sysctl -a | grep flush >> sysctl_TCP.txt

    teste

    echo ""
    echo "***************************************************************"
    echo "************** TESTES DO PROTOCOLO TCP COMPLETOS **************"
    echo "***************************************************************"
    echo ""
}

testeHSTCP() {

    ###############################################################################
    #                              TESTES DO PROTOCOLO HSTCP                      #
    ###############################################################################

    echo ""
    echo "***************************************************************"
    echo "************** INICIANDO TESTES DO PROTOCOLO HSTCP **************"
    echo "***************************************************************"
    echo ""

    tipo="HSTCP"

    # Vai para o diretorio dos testes
    cd /root/Desktop/wallace/Testes/Tests_HSTCP

    # Remove arquivos antigos para evitar erros em próximas execuções
    rm -rf *.txt

    # Verifica o sysctl.conf e modifica o sysctl para usar o HSTCP, logando em arquivo
    echo "***************** Aplica configs do sysctl.conf *****************" > sysctl_HSTCP.txt
    sysctl -p >> sysctl_HSTCP.txt
    echo "****************** Pega dados de TCP do sysctl ******************" >> sysctl_HSTCP.txt
    sysctl -w net.ipv4.tcp_congestion_control=highspeed
    sysctl -a | grep tcp >> sysctl_HSTCP.txt
    echo "***************** Pega dados de memoria do sysctl ***************" >> sysctl_HSTCP.txt
    sysctl -a | grep mem >> sysctl_HSTCP.txt
    echo "************** Pega dados de route.flush do sysctl **************" >> sysctl_HSTCP.txt
    sysctl -a | grep flush >> sysctl_HSTCP.txt

    teste

    echo ""
    echo "***************************************************************"
    echo "************** TESTES DO PROTOCOLO HSTCP COMPLETOS **************"
    echo "***************************************************************"
    echo ""
}

testeSTCP() {

    ###############################################################################
    #                              TESTES DO PROTOCOLO STCP                       #
    ###############################################################################
```

```
    echo ""
    echo "*****************************************************************"
    echo "************* INICIANDO TESTES DO PROTOCOLO STCP ***************"
    echo "*****************************************************************"
    echo ""

    tipo="STCP"

    # Vai para o diretorio dos testes
    cd /root/Desktop/wallace/Testes/Tests_STCP

    # Remove arquivos antigos para evitar erros em próximas execuções
    rm -rf *.txt

    # Verifica o sysctl.conf e modifica o sysctl para usar o STCP, logando em arquivo
    echo "***************** Aplica configs do sysctl.conf *****************" > sysctl_STCP.txt
    sysctl -p >> sysctl_STCP.txt
    echo "****************** Pega dados de TCP do sysctl ******************" >> sysctl_STCP.txt
    sysctl -w net.ipv4.tcp_congestion_control=scalable
    sysctl -a | grep tcp >> sysctl_STCP.txt
    echo "***************** Pega dados de memoria do sysctl ***************" >> sysctl_STCP.txt
    sysctl -a | grep mem >> sysctl_STCP.txt
    echo "************* Pega dados de route.flush do sysctl *************" >> sysctl_STCP.txt
    sysctl -a | grep flush >> sysctl_STCP.txt

    teste

    echo ""
    echo "*****************************************************************"
    echo "************* TESTES DO PROTOCOLO STCP COMPLETOS ***************"
    echo "*****************************************************************"
    echo ""
}

# MAIN
start;
testeTCP;
testeHSTCP;
testeSTCP;
```

8.3. Apêndice C: Script de execução dos testes no servidor

```
#!/bin/bash

################################################################################
#  Script para rodar testes de Vazão x Perda com RTTs fixos em redes gigabit,  #
#          utilizando os protocolos de transporte TCP, HSTCP e STCP            #
#                                                                              #
#    *** Projeto de Graduação VIII-B - Engenharia de Telecom - UERJ ***        #
#                                                                              #
# Autor: Wallace de Souza Espindola                                            #
#                                                                              #
# Criado em: Fevereiro/2007                                                    #
#                                                                              #
################################################################################

start() {

    echo ""
    echo "************************************************************"
    echo "************ INICIANDO TESTES COM O SERVIDOR IPERF ************"
    echo "************************************************************"
    echo ""

    # Vai para o diretorio dos testes
    cd /root/Desktop/wallace/Testes

    # Remove antigos arquivos para evitar erros em novas execuções
    rm -rf *.txt

    # Modifica valor de txqueuelen e loga a configuracao em arquivo
    ifconfig eth0 txqueuelen 2000
    ifconfig eth0 > ifconfig.txt

    # Coloca a Placa de Rede em 1000Mbps Full Duplex com Flow Control Off e loga em arquivo
    ethtool -s eth0 speed 1000 duplex full autoneg off
    ethtool -A eth0 autoneg off rx off tx off
    ethtool eth0 > ethtool.txt
    echo "************************************************************" >> ethtool.txt
    ethtool -a eth0 >> ethtool.txt

    # Verifica o log do driver e1000 da Placa de Rede Intel e loga em arquivo
    dmesg | grep e1000 > dmesg.txt

    # Verifica o sysctl e usa as novas configuracoes do sysctl.conf, logando em arquivo
    echo "************ Aplica configs do sysctl.conf ************" > sysctl_server.txt
    sysctl -p >> sysctl_server.txt
    echo "************** Pega dados de TCP do sysctl **************" >> sysctl_server.txt
    sysctl -a | grep tcp >> sysctl_server.txt
    echo "************ Pega dados de memoria do sysctl ************" >> sysctl_server.txt
    sysctl -a | grep mem >> sysctl_server.txt
    echo "********** Pega dados de route.flush do sysctl **********" >> sysctl_server.txt
    sysctl -a | grep flush >> sysctl_server.txt

    # Comando que roda o servidor Iperf nos testes
    # Redireciona o output para um arquivo texto
    # (-s run server / -w window size /-l length of buffer)
    iperf -s -w 4M -l 64k > ifperf_server_tests.txt

    echo ""
    echo "************************************************************"
    echo "************ TESTES COM O SERVIDOR IPERF CONCLUIDOS ************"
    echo "************************************************************"
    echo ""

}

# MAIN
start;
```

8.4. Apêndice D: Script de configuração usado nos micros

O *script* abaixo é o conteúdo do arquivo de configuração do *kernel* do Linux, sysctl.conf, que é carregado automaticamente toda vez que o Linux é inicializado. Todas as configurações aqui feitas são tratadas como configurações fixas do Linux, já que ele sempre será inicializado da mesma forma, contendo estas mesmas configurações adicionais de parâmetros do sistema.

```
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values, 0 is disabled, 1 is enabled.  See sysctl(8) and
# sysctl.conf(5) for more details.

# Controls IP packet forwarding
net.ipv4.ip_forward = 0

# Controls source route verification
net.ipv4.conf.default.rp_filter = 1

# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0

# Controls the System Request debugging functionality of the kernel
kernel.sysrq = 0

# Controls whether core dumps will append the PID to the core filename.
# Useful for debugging multi-threaded applications.
kernel.core_uses_pid = 1

# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 1

###############################################################################
#    Abaixo estão as configurações feitas para otimizar o funcionamento   #
#      das máquinas Linux para a realização dos testes dos protocolos     #
#                                                                         #
#   *** Projeto de Graduação VIII-B - Engenharia de Telecom - UERJ ***    #
#                                                                         #
# Autor: Wallace de Souza Espindola                                       #
#                                                                         #
# Criado em: Fevereiro/2007                                               #
#                                                                         #
###############################################################################

# TCP congestion control algorithm
#==> Options are: reno (TCP Reno), highspeed (HSTCP), scalable (STCP), or others.
net.ipv4.tcp_congestion_control = reno

# Increase Linux TCP buffer limits (33554432=32Mb / 65536=64Kb)
net.core.rmem_max = 33554432
net.core.wmem_max = 33554432
net.core.rmem_default = 65536
net.core.wmem_default = 65536

# Increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.tcp_wmem = 4096 65536 33554432
net.ipv4.tcp_mem = 33554432 33554432 33554432

# Don't cache ssthresh from previous connection
net.ipv4.tcp_no_metrics_save = 1

# Recommended increasing this for 1000 BT or higher, for 10 Gb using 3000
net.core.netdev_max_backlog = 3000

# Flush all the tcp cache settings
net.ipv4.route.flush = 1
```

8.5. Apêndice E: Intervalo de Confiança

O cálculo do valor médio de um determinado parâmetro por si só não define exatamente a característica do mesmo.

Um intervalo de confiança relativo à média expressa a idéia de que temos um determinado nível de confiança de que a média se encontra naquele intervalo. Além da média, o intervalo de confiança delimita a margem de precisão existente em um determinado conjunto de amostras. Com isso, pode-se ter uma melhor e mais correta idéia do comportamento média de uma medida estatística.

Por exemplo, um intervalo de confiança de 90% indica que com 90% de confiança, a média da população de amostras está entre os limites superior e inferior do intervalo.

O cálculo dos intervalos de confiança utilizados nas amostras dos testes presentes nesse projeto se baseia na seguinte fórmula [32]:

$$Ic = \left(\bar{x} - \left[t_{(1-\alpha/2\,;\,n-1)} \times \frac{s}{\sqrt{n}} \right] \;,\; \bar{x} + \left[t_{(1-\alpha/2\,;\,n-1)} \times \frac{s}{\sqrt{n}} \right] \right)$$

Onde:

$\bar{x}$ – média das amostras;

n – quantidade de amostras;

s – desvio padrão;

α – coeficiente de confiança (se o intervalo de confiança é de 90%, $\alpha = 0{,}1$);

$t_{(1-\alpha/2;n-1)}$ – valor tabelado dependente de α e n.