Leandro Soares de Sousa

Avaliação e implementação de uma variação do protocolo TCP, projetada para redes de alto desempenho, visando à distribuição de objetos multimídia nas unidades de armazenamento do Servidor RIO

NITERÓI

2007

Leandro Soares de Sousa

Avaliação e implementação de uma variação do protocolo TCP, projetada para redes de alto desempenho, visando à distribuição de objetos multimídia nas unidades de armazenamento do Servidor RIO

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Orientador:
Anna Dolejsi Santos

Universidade Federal Fluminense

NITERÓI

2007

Avaliação e implementação de uma variação do protocolo TCP, projetada para redes de alto desempenho, visando à distribuição de objetos multimídia nas unidades de armazenamento do Servidor RIO

Leandro Soares de Sousa

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Banca Examinadora:

Profª. Anna Dolejsi Santos, Dr.Sc. - Orientadora
UFF - Universidade Federal Fluminense

Prof. Edmundo Albuquerque de Souza e Silva , Ph.D.
UFRJ - Universidade Federal do Rio de Janeiro

Profª. Rosa Maria Meri Leão, Dr.
UFRJ - Universidade Federal do Rio de Janeiro

Profª. Morganna Carmen Diniz, Dr.Sc.
UNIRIO - Universidade Federal do Estado do Rio de Janeiro

Agradecimentos

Agradeço a Deus por toda a força que me foi dada para vencer todas as dificuldades e desafios para concluir este trabalho.

Agradeço a minha orientadora Professora Anna que me deu a oportunidade de participar deste trabalho, que acreditou em mim, me ajudou com seu profundo conhecimento nesta área, nas correções do texto da dissertação e colaborou com todos os recursos possíveis para alcançar este objetivo.

Agradeço aos meus familiares pela paciência, compreensão, pelos momentos ausentes, e pelo sacrifício imposto pela minha dedicação ao curso de pós-graduação, especialmente à minha esposa Ana Beatriz.

Agradeço ao Bernardo a toda ajuda para o entendimento e aos procedimentos corretos para a adequação ao Servidor RIO. Agradeço ao professor Edmundo as orientações necessárias para a condução correta deste trabalho.

Com certeza, estarei pecando ao não citar tantas outras pessoas que direta ou indiretamente me ajudaram a trilhar este caminho do saber na UFF, para que eu pudesse estar aqui hoje apresentando este trabalho, e a todas peço humildemente muito obrigado.

Resumo

O objetivo deste trabalho foi estudar o comportamento de variações do protocolo TCP desenvolvidas para redes de alto desempenho. Para tanto, investigamos o quanto o TCP pode se tornar vantajoso, em relação ao UDP, em redes com baixo retardo de propagação e baixo congestionamento, ou seja, modificações na implementação do TCP que maximizam a ocupação da banda passante disponível. Por outro lado, foi importante, também, averiguar se o TCP é, ou não, eficiente para a transferência de uma quantidade massiva de dados, um exemplo disto é a transferência que ocorre entre os servidores do serviço multimídia do projeto - Servidor RIO.

O Servidor RIO utiliza uma modificação do protocolo UDP para implementar um serviço de transferência de dados confiável, no entanto esta confiabilidade também é provida pelo TCP, o que tornou este um bom caso para a avaliação efetuada. Este UDP modificado realiza a comunicação em tempo "não real" do Servidor RIO, que é responsável pela criação de novos objetos multimídia e pela recuperação de objetos para armazenamento. Para a distribuição ou recuperação de um grande número de objetos entre os servidores RIO, os protocolos TCP atuais, Reno e Vegas, que adotam medidas conservadoras para o controle de congestionamento, podem trazer problemas, pois na ocorrência de um evento de perda de pacote, estas variações do TCP realizam uma forte redução da janela de transmissão induzindo a uma baixa utilização da banda passante. No caso das transmissões de grandes volumes estas medidas podem trazer limitações e instabilidade para a taxa de transmissão de dados. Desta forma, investigamos e implementamos sete variações do protocolo TCP, duas delas são as estratégias mais utilizadas atualmente na Internet, TCP Reno e TCP Vegas, e as cinco restantes modificam o algoritmo de controle de congestionamento para minimizar as limitações e instabilidade na taxa de transmissão de dados, estas estão: (i) TCP Westwood, (ii) BIC-CUBIC TCP, (iii) FAST TCP, (iv) Scalable TCP e (v) HighSpeed TCP.

Os sete TCPs foram avaliados através de uma série de experimentos, nos quais foram efetuadas transmissões de arquivos, e destes foram medidos: (i) o tempo total de transmissão do arquivo, (ii) a estabilidade do fluxo de dados e (iii) a ocupação da banda passante. Nos experimentos, também, avaliamos as estratégias de controle de congestionamento dos TCPs, para tal implementamos o protocolo IP, da camada de rede da pilha de protocolos da Internet, desta forma interferimos no fluxo de dados forçando a perda de pacotes e, também, pudemos coletar dados mais precisos para as avaliações. Um programa foi construído especificamente para este fim, e em conseqüência, tornou-se necessária a implementação das APIs *socket*, que serviram de interface entre este programa e os TCPs.

Finalmente, substituímos a comunicação em tempo "não real" do Servidor RIO pelo FAST TCP, que obteve a melhor avaliação na bateria de experimentos realizados neste trabalho. Através da avaliação dos resultados obtidos nesta implementação, para o Servidor RIO, nos foi possível extrair conclusões e indicações para trabalhos futuros.

Abstract

The objective of this work was to study the behavior variations of the TCP protocol that were designed to perform in high speed networks. For this, we investigated if the TCP can become advantageous, in relation to UDP, in networks with low propagation delay and low traffic, in other words, modifications of TCP projected to maximize the occupation of the available bandwidth. On the other hand, it was important, also, to discover if TCP is, or is not, efficient for the transfer of a massive amount of data, an example of that is the transfer that happens among the servers of the service multimedia of the project - Servidor RIO.

The Servidor RIO uses a modification of the UDP protocol to implement a reliable data transfer service, however this reliability is also provided by TCP, what turned this as a good case for the made evaluation. For the distribution of a great number of videos among Servidores RIO, the current TCP protocols, Reno and Vegas, that adopt conservative measures for the congestion control, can bring problems, because in the occurrence of a package loss event, these variations of TCP accomplish a strong reduction of the transmission window inducing to a low use of the bandwidth. In the case of the extensive volumes of transmissions these measures can bring limitations and instability to the data transmission rate. This way, we investigated and implemented seven variations of the TCP protocol, two of them are the most used strategies in Internet, TCP Reno and TCP Vegas, and the remaining five modify the congestion control algorithm to minimize these limitations and instability to the data transmission rate, these are: (i) TCP Westwood, (ii) BIC-CUBIC TCP, (iii) FAST TCP, (iv) Scalable TCP and (v) HighSpeed TCP.

The seven TCPs were implemented and they also were evaluated through a series of experiments, in which transmissions of files were made, and were measured: (i) the total time of the file transmission, (ii) the stability of the data flow and (iii) the bandwidth occupation. In the experiments, also, were evaluated the strategies of congestion control of TCPs, for such we implemented the IP protocol, of the network layer of the Internet protocols stack, this way we interfered in the data flow forcing the packages loss and, also, we could collect the necessary data with more precision. A program was specifically built for this end, as a consequence, became necessary to implement the socket API, which served as interface between this program and TCPs.

Finally, we substituted the communication in “non real” time of the Servidor RIO by the FAST TCP, that obtained the best evaluation in the battery of experiments accomplished in this work. Through the evaluation of the results obtained in this implementation, for the Servidor RIO, was possible to extract conclusions and indications for future works.

Palavras-chave

1. Ensino a Distância
2. Servidor Multimídia RIO
3. A Real-Time Multimedia Object Server
4. Rede GIGA
5. Nós de Armazenamento
6. TCP de alta performance

Glossário

QoS	:	*Quality of Service* (Qualidade de Serviço);
RT	:	*Real Time* (Tempo Real);
NRT	:	*Non Real Time* (Não é de Tempo Real);
RIO	:	*Randomized I/O* (Operações de entrada e saída aleatórias);
DIVERGE	:	Distribuição de vídeo em larga escala sobre redes Giga com aplicações na educação;
TCP	:	*Transmission control protocol* (Protocolo de controle da transmissão);
UDP	:	*User Datagram Protocol* (Protocolo de Datagramas do Usuário);
IP	:	*Internet Protocol* (Protocolo da Internet);
RTT	:	*Round Trip Time* (Tempo medido desde o envio de um pacote até a chegada da confirmação do recebimento do mesmo);
ACK	:	*Acknowledgment* (Este pacote é utilizado para confirmar a recepção de dados no TCP pela parte receptora comunicação);
API	:	*Application Programming Interface* (ou Interface de Programação de Aplicativos, é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por programas aplicativos - isto é: programas que não querem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços)

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Objetivo

O objetivo deste trabalho foi de utilizar um aspecto real do Servidor RIO para realizar um estudo do comportamento de variações do protocolo TCP desenvolvidas para redes de alto desempenho, principalmente em relação aos seus mecanismos de controle de congestionamento. Para alcançar este objetivo, selecionamos, implementamos e avaliamos, através de experimentos, sete variações do protocolo TCP. Entre os sete TCPs selecionados temos dois entre os tradicionais (TCP Reno e TCP Vegas) e cinco projetados exclusivamente para redes de alto desempenho. Após este processo de avaliação, um dos TCPs foi selecionado para dar suporte a um tipo específico de fluxo de dados dos servidores RIO. Nas subseções a seguir detalharemos a motivação para este trabalho e como ele foi organizado.

1.2 Motivação

No Servidor RIO [1, 2] temos dois tipos distintos de comunicação: (i) em "tempo real", para a transmissão das mídias entre os servidores RIO e os clientes, que faz uso do protocolo UDP; e (ii) em "tempo não real", usadas para a transmissão de objetos multimídia para os Nós de Armazenamento dos servidores RIO ou para retirar dos Nós de Armazenamento um objeto para arquivamento, que utiliza um protocolo similar ao UDP, que envia algumas confirmações esporádicas. Este protocolo, que no Servidor RIO é chamado de "UDP modificado", foi implementado para atender a necessidade de transferência confiável de dados de forma simples e com características de confiabilidade.

A comunicação em "tempo real" do Servidor RIO, que oferece o suporte à suas aplicações multimídia, é efetuada por meio do protocolo UDP, isto se deve ao fato de que as aplicações multimídia, em geral, são sensíveis aos retardos nas transmissões e tolerantes a perda de pacotes, até um certo limite. Estas características, principalmente no

caso dos clientes atendidos pelos servidores RIO, e que não utilizam redes de alta velocidade [3], tornam a utilização do UDP fundamental para possibilitar o atendimento à suas necessidades. O TCP, neste caso, pode tornar-se inviável por garantir a transmissão íntegra das informações, gerando possíveis retransmissões, e conseqüentemente dificultando o atendimento às características descritas anteriormente.

No Servidor RIO, não raras vezes, é realizada a transmissão de grandes objetos para serem gravados em seus Nós de Armazenamento. À medida que maior massa de dados passar a ser transferida, através da rede de alto desempenho, a confiabilidade, obtida atualmente pelo "UDP modificado", poderá ser alcançada por meio do protocolo TCP.

Contudo, o protocolo TCP Reno, que sendo o mais utilizado atualmente tomaremos como um TCP "padrão" para efeito de comparações com outras estratégias de implementação do protocolo TCP, sofre com os problemas de seu algoritmo para o controle de congestionamento [4]. Este algoritmo é acionado, pelo protocolo TCP, quando ocorre algum evento de perda de pacote e visa controlar a taxa de transmissão, e desta forma evitar o congestionamento em redes de computadores. No TCP, a perda de pacote é definida pela ocorrência de um entre dois eventos, listados a seguir:

- *timeout*: Este evento ocorre quando dados são transmitidos e o tempo que o TCP aguarda pelo pacote de confirmação[1] de seu recebimento, enviado pela parte receptora comunicação, é extrapolado, sem que o ACK tenha sido recebido; e
- Recepção de três ACKs em duplicata: Isto pode ocorrer, durante uma transmissão de dados no TCP, porque os pacotes podem ser recebidos fora de ordem, e quando isto ocorre o lado receptor envia um ACK para o último pacote recebido na ordem esperada, mas quando o lado transmissor recebe o terceiro ACK em duplicata, para um mesmo pacote, o TCP considera que algum pacote foi perdido e aciona o mecanismo de controle de congestionamento.

O controle de congestionamento atua sobre a janela de congestionamento, também designada por "janela de transmissão", que limita o número de segmentos[2] TCP transmitidos mas ainda não confirmados através de um ACK, pelo lado receptor da comunicação. Quando algum dos eventos acima ocorre o TCP Reno atua de forma distinta, para cada um deles, estas formas de atuação foram relacionadas a seguir:

[1] Este pacote de confirmação, enviado pela parte receptora comunicação, é designado ACK.

[2] Os segmentos TCP são compostos por dois elementos: (i) parte dos dados da comunicação, que é opcional, e (ii) o cabeçalho do protocolo TCP.

- *timeout*: A janela de transmissão é reduzida a 1 MSS; e
- três ACKs em duplicata: A janela de transmissão é reduzida à metade, desde que não caia abaixo de 1 MSS[3].

Estas medidas são muito conservadoras e tem um reflexo direto, e bastante negativo, na utilização dos recursos de comunicação disponíveis. Isto ocorre porque o TCP assume que qualquer evento de perda é provocado por um congestionamento na rede, e isto pode não ser a verdade. Vejamos o caso dos *links* sem fio, nos quais um pacote pode perder sua integridade devido a uma condição momentânea, por exemplo um ruído na comunicação, neste caso o TCP reduzirá a janela de transmissão sem que um congestionamento efetivamente tenha ocorrido. Outro caso, especialmente grave, é o dos *links* de alta performance, por exemplo um *link* de fibra ótica de 10 GigaBits, de alto custo, que pode ser subutilizado devido a uma perda de pacote devido, por exemplo pelo congestionamento momentâneo nos *buffers* do roteador. Estas estratégias para o controle de congestionamento são o principal problema para utilização do TCP Reno em redes de alto desempenho.

Os problemas, descritos anteriormente, nas implementações atuais do protocolo TCP, notadamente no TCP Reno, foram o principal motivo para a elaboração deste trabalho.

1.3 Contribuição do trabalho

Além do objetivo principal, o de substituir o "UDP modificado" pelo protocolo TCP, devemos destacar como contribuição a biblioteca desenvolvida para comportar as variações do protocolo TCP. Esta biblioteca funciona como um arcabouço que possibilitou o desenvolvimento dos TCPs, de tal forma que em um único código objeto temos todas as variações do TCP. Desta forma, para selecionar-mos a variação do TCP para uma determinada transferência de dados, que é definida apenas no momento de sua utilização, basta que utilizamos um simples arquivo de parâmetros. Esta estratégia, para trabalhos futuros, simplificará a implementação e avaliação de novas variações do protocolo TCP.

[3] 1 MSS é o tamanho mínimo para a janela de transmissão no protocolo TCP.

1.4 Organização do trabalho

Este trabalho está subdividido em duas partes principais:

1ª - Parte: Identificação, seleção, implementação e experimentos sobre variações do TCP para redes de alto desempenho. Desta forma, pudemos definir a variação do TCP que mais se adequou às necessidades do Servidor RIO. A primeira parte foi desenvolvida nos Capítulos de 2 à 4.

- No segundo capítulo temos, inicialmente, uma descrição das questões que levaram os pesquisadores a buscar alternativas aos mecanismos de controle de fluxo e congestionamento do protocolo TCP padrão. Em seguida, temos os critérios de seleção das variações do TCP alvo deste trabalho, a definição do formato de apresentação das variações escolhidas e a descrição das mesmas.
- No Capítulo 3, abordamos os aspectos mais relevantes da implementação da biblioteca de funções, desenvolvida inteiramente para este trabalho, que fornece o suporte necessário ao desenvolvimento de aplicações utilizando cada uma das variações do protocolo TCP selecionadas.
- O quarto capítulo é dedicado aos experimentos realizados, nele estão descritos os critérios de avaliação, o programa que suportou os experimentos, a forma como os dados foram coletados e, também, a análise dos resultados obtidos.

2ª - Parte: Esta, descrita no Capítulo 5, se dispõe a substituir o "UDP Modificado", correntemente utilizado no Servidor RIO, nas transmissões em "tempo não real" pela variação do TCP que se demonstrou mais eficiente na primeira parte do trabalho.

- No quinto capítulo, temos a especificação do alvo da implementação no Servidor RIO, como a questão foi abordada, aspectos de sua implementação, experimentos realizados e uma avaliação dos resultados colhidos.

Após as duas partes, apresentadas acima, temos, no Capítulo 6, as conclusões obtidas durante o desenvolvimento desta dissertação e as sugestões para trabalhos futuros.

2 Seleção e descrição das variações do TCP para redes de alta performance

Antes de nos atermos à seleção das variações do protocolo TCP de alta performance, faremos uma breve introdução das principais características que impuseram a criação destas variações do TCP.

O principal temor, dos desenvolvedores do TCP, é que durante a comunicação dos dados, caso não fossem observados os "sinais" de congestionamento na rede, uma rede de computadores fosse levada a um estado tal que toda e qualquer comunicação fosse impossibilitada, devido a uma utilização maior que os recursos disponíveis. Refletindo este temor, foi desenvolvido o algoritmo para o controle de congestionamento, mas a estratégia implementada é muito conservadora e apresenta alguns problemas para a efetiva ocupação dos recursos de comunicação de dados, os principais problemas foram listados a seguir:

- O "sinal" escolhido, evento de perda de pacotes, não necessariamente indica um "forte" congestionamento [3] na rede de computadores. Um evento de perda de pacote pode ocorrer por um momentâneo congestionamento, este localizado em algum ponto do caminho entre o computador de origem e destino da informação, ou, também, pela perda da integridade dos dados contidos no pacote, este caso notadamente em *links* sem fio.
- A reação do protocolo a estes eventos é muito brusca, resultando em uma baixa utilização dos recursos disponíveis para a comunicação de dados.

Tomemos, como exemplo, a estratégia adotada pelo TCP Reno, que quando percebe um evento de perda imediatamente reduz sua janela de congestionamento à metade, ou até mesmo à 1 MSS (dependendo do evento), e conseqüentemente a sua taxa de envio de pacotes cai na mesma proporção. Após esta queda, a janela de congestionamento cresce lentamente de forma aditiva. Esta estratégia é conhecida por **AIMD** (***A****dditive* ***I****ncrease* ***M****ultiplicative* ***D****ecrease*).

Para ilustrar esta questão apresentamos, na Figura 1, apenas como exemplo desta estratégia em funcionamento, um gráfico do comportamento da janela de congestionamento do TCP Reno durante a transmissão de um fluxo de dados. Na mesma figura destacamos, em verde, três pontos que marcam eventos de perda, neste caso três ACKs recebidos em duplicata. Nos pontos destacados observamos que a janela de congestionamento é reduzida à metade e depois volta a crescer lentamente. Este comportamento é típico do TCP Reno devido a utilização do AIMD para controle da janela de congestionamento.

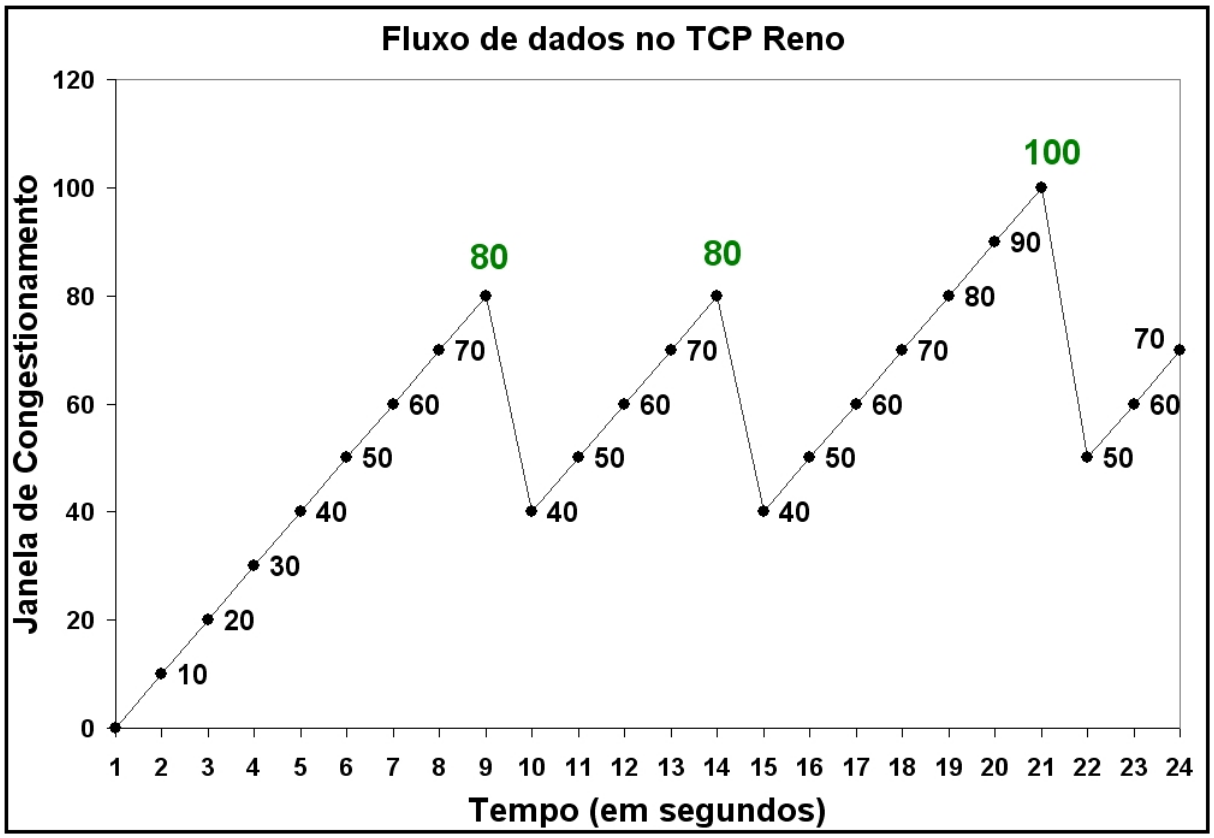

Figura 1: Exemplo do comportamento da janela de congestionamento no TCP Reno

Observando atentamente a Figura 1, logo surgem algumas questões sobre seu comportamento:

- Se a redução utilizada fosse menos agressiva poderíamos ocupar melhor a banda passante, pois, no exemplo, alcançamos uma janela de congestionamento máxima de 100 e sua média aritmética no período de tempo observado foi de apenas 55 ?

- O crescimento aditivo somente é limitado pela janela oferecida pelo receptor, através do controle de fluxo[1]. Se esta oferta for "suficientemente grande" não estaríamos sempre forçando eventos de perda de pacotes, por não nos ajustarmos a capacidade de transmissão de dados possível em um dado momento, que reflete o estado de congestionamento da rede?

- Não existiria algum outro "sinal" de congestionamento na rede para utilizarmos ou temos apenas o evento de perda de pacotes?

- Digamos que a nossa janela de congestionamento média, em um determinado momento, estivesse em torno de 90, ponto intermediário da janela de congestionamento para os eventos de perda destacados na Figura 1, não poderíamos alcançar uma performance como no exemplo da Figura 2, que exemplifica um fluxo de dados onde a taxa de transmissão não é reduzida drasticamente e, que portanto, utiliza melhor os recursos de comunicação disponíveis?

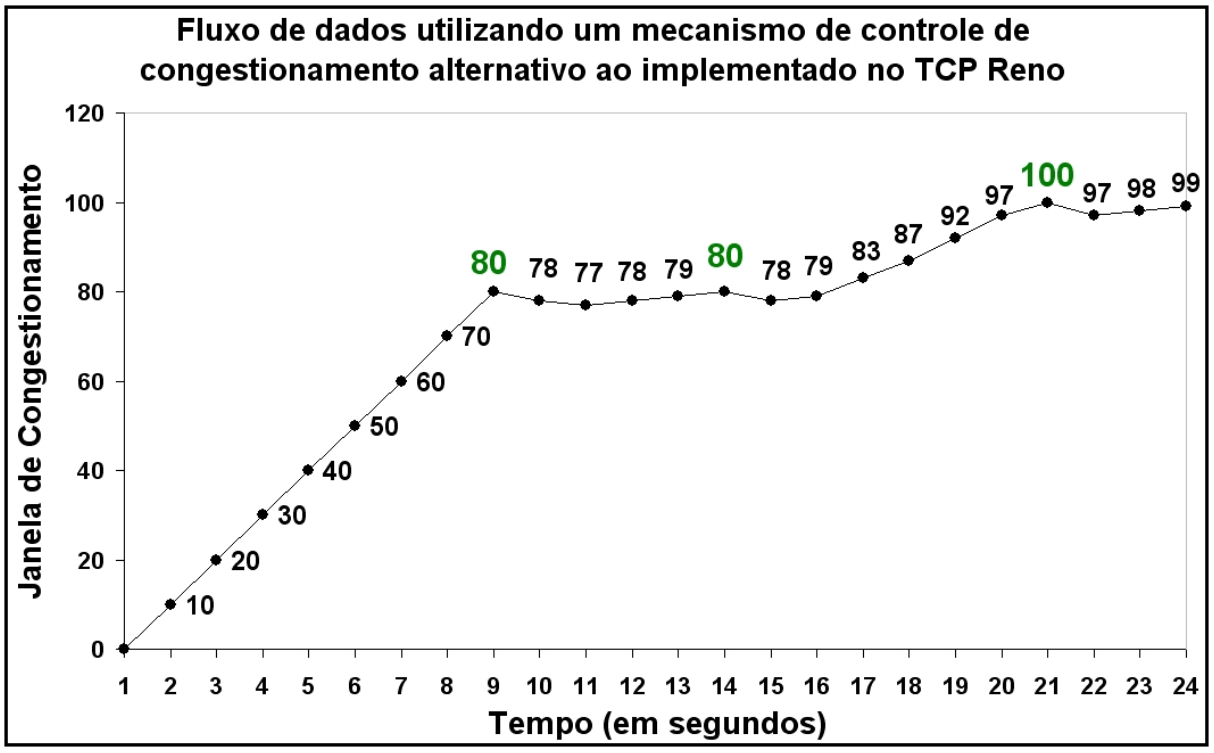

Figura 2: Exemplo do comportamento da janela de congestionamento

Estas questões motivaram pesquisadores à desenvolver novas alternativas ao AIMD

[1]O controle de fluxo é o mecanismo do TCP que ajusta a janela de congestionamento do lado transmissor da comunicação à janela de recepção do lado receptor, evitando assim que um transmissor de maior capacidade "inunde" de pacotes um receptor com capacidade inferior.

do protocolo TCP, que atuam principalmente nos pontos levantados acima, ou seja, nos mecanismos de crescimento e redução da janela de congestionamento, nos "sinais" colhidos da rede e na reação a estes "sinais".

Algumas destas estratégias apenas modificam a forma como aumentam ou diminuem a janela de congestionamento, em sua maioria, realizam esta tarefa de forma bem menos agressiva que a efetuada pelo TCP Reno. Outras, tratam o "ponto de equilíbrio" da janela de congestionamento, no caso dos exemplos entre 80 e 100 (pontos onde foram detectadas perdas de pacotes nas Figuras 1 e 2), como um problema de busca, e são mais agressivas no crescimento e diminuição da mesma, quando distantes deste "ponto de equilíbrio", e mais suaves quando em sua proximidade [5]. Algumas das alternativas ao AIMD do TCP Reno tem como principal característica introduzir mecanismos que capturam novos "sinais" de congestionamento, tanto para o decréscimo quanto para o acréscimo da janela de congestionamento. Dentre estes mecanismos, podemos destacar: (i) a observação das variações do RTT (*Round Trip Time*) no tempo; e (ii) a análise da flutuação na taxa de envio de pacotes.

Nos próximos tópicos, abordaremos as principais alternativas ao AIMD do protocolo TCP Reno e suas estratégias para responder às questões acima.

2.1 Seleção das variações do TCP

O primeiro passo, nesta avaliação, foi o de definir quais variações do TCP devem ser analisadas, em nosso trabalho. Esta seleção teve os seguintes critérios:

- O desenvolvimento específico para redes de alta performance;
- Implementações já construídas;
- Testes de avaliação já efetuados;
- Pesquisadores e instituições envolvidos no projeto; e
- Como estratégia, para balizar as avaliações, também foram selecionadas as variações do protocolo TCP mais populares atualmente: RENO e VEGAS. Evitando assim distorções na comparação entre a implementação deste trabalho e as disponíveis nos sistemas operacionais. Estas distorções ocorrem devido, principalmente, à dois fatores:

– Possíveis vantagens de performance obtidas pela implementação do TCP disponível no sistema operacional, principalmente pela utilização de funções de nível mais baixo, ou seja, mais próximas ao *kernel* do sistema operacional.

– Outra distorção, é que não temos como garantir que a implementação do TCP Reno, por exemplo no caso do MS Windows, é exatamente a definida por suas RFCs.

Tendo em vista os critérios definidos, os TCPs[2] selecionados estão relacionados na Tabela 1:

Tabela 1: Variações do TCP analisadas

Variação do TCP	**Principais Referências**
TCP Reno	[6]
TCP Vegas	[7]
High Speed TCP (HSTCP)	[8, 9]
Scalable TCP (STCP)	[10, 11]
BIC-CUBIC TCP	[5]
Westwood TCP (TCPW)	[12, 13, 14]
FAST TCP	[15, 16, 17, 18, 19]

Para um melhor entendimento dos TCPs, implementados neste trabalho, serão brevemente descritos quatro algoritmos, presentes pela primeira vez quando o TCP Reno foi descrito [6], sobre estes algoritmos recaem as modificações que conferem um alto desempenho aos TCPs analisados. Estes algoritmos foram padronizados na RFC 793 [6] para o "bom" funcionamento da Internet, evitando assim um congestionamento "global", e para um melhor aproveitamento dos recursos de comunicação disponíveis. Abaixo temos a descrição dos mesmos:

- ***slow start* (SS)** : Os TCP, anteriores ao Reno, iniciavam a transmissão injetando múltiplos pacotes a mais na rede até alcançarem o tamanho da janela de transmissão informada pelo receptor do fluxo. Esta estratégia causava problemas devido aos retardos inseridos na comunicação pelos elementos intermediários, notadamente roteadores. Desta forma, este algoritmo tem por objetivo ajustar a taxa de transmissão de pacotes à taxa de pacotes ACK recebidos pelo transmissor de forma mais suave no início da transmissão;

[2]Quando utilizarmos o termo TCPs estaremos nos referindo, sempre, às variações do TCP implementadas neste trabalho e relacionadas na Tabela 1.

- ***congestion avoidance* (CA)**: Este algoritmo, em linhas gerais, determina a resposta do TCP aos eventos de perda de pacotes, que são *timeouts* e três ACKs em duplicata. O principal objetivo deste algoritmo é o de evitar um congestionamento "global";

- ***fast retransmit***: Durante uma transmissão, alguns pacotes podem ser recebidos fora de ordem, os ACKs destes pacotes são enviados, informando também a seqüência esperada pelo receptor, pois esta falta de ordem pode ter sido ocasionada no percurso entre o *host* origem, da transmissão, e o *host* destino. Desta forma, para evitar o acionamento dos mecanismos **CA** e **SS**, é aguardado um tempo para que tenhamos a certeza de que o pacote foi perdido, esta certeza é obtida pela recepção do terceiro ACK duplicado ou um *timeout* no transmissor; e

- ***fast recovery***: Após o *fast retransmit* verificamos a perda de pacotes mas temos a certeza de que pacotes estão chegando ao receptor, no caso da recepção de três ACKs em duplicata. Desta forma, o *fast recovery* aciona o mecanismo de *congestion aviodance* e não o *slow start* para que não percamos uma melhor utilização da banda passante ao acionarmos o *slow start*.

Todos os TCPs estudados e implementados neste trabalho utilizam as estratégias de *fast retransmit* e *fast recovery* acima descritas. Portanto, no restante do texto, nos ateremos apenas nas estratégias de *slow start* e *congestion avoidance*.

2.2 Descrição das variações do TCP selecionadas

Nesta seção, temos as descrições, de cada um dos sete TCPs, que seguem o formato dos ítens a seguir descritos:

- **Contato**: Principais pesquisadores e instituições envolvidos no projeto;
- **Padrão**: Se há, ou não, algum padrão definido, e caso positivo qual é a referência a ele;
- **Cenário alvo da utilização**: Indica as principais propostas de cenários para a aplicação da variação do TCP;
- **Justiça**: Se há algum indicativo de justiça inter-protocolo e/ou intra-protocolo. Este tópico aborda o quanto *TCP-friendly* é uma implementação, ou seja, se há uma preocupação com outros fluxos que compartilham os mesmos recursos de comunicação;
- **Referências adicionais**: São referências, quando presentes, que não são artigos, mas sim sítios na Internet com informações relevantes ao TCP exposto;
- **Implementação / API**: Quando existem implementações disponíveis, indicam onde encontrar *patchs* ou implementações completas; e
- **Descrição**: Em linhas gerais, quais foram os motivos de sua implementação e suas principais características;
- **Mapa de eventos**: O mapa de eventos é apresentado através de uma tabela que contém o pseudo-código das estratégias de controle do congestionamento. Esta tabela é composta pelas seguintes colunas:
  - **Estado** - estado em que se encontra a transmissão TCP;
  - **Evento** - evento ocorrido estando o transmissor TCP em determinado estado;
  - **Ação** - ação do algoritmo acionado dado um estado e um evento; e
  - **Comentário** - algum comentário adicional considerado importante para a compreensão do mecanismo de controle de congestionamento utilizado.

Nos próximos ítens temos as descrições sucintas dos TCPs selecionados, que utilizam as abreviações da Tabela 2.

Tabela 2: Abreviações utilizadas nas variações do TCP

Parâmetro	**Abreviação**	**Descrição**
Estado do TCP	**SS**	Fase do *slow start*
	CA	*Congestion avoidance*
Eventos Capturados	**ACK**	*Acknowledge* dos dados
	TOUT	Evento de *timeout*
	DACK	Ack em duplicata recebido
	TACK	Evento de recepção do terceiro ACK duplicado
Dados e Cálculos	**CongWin**	Janela de congestionamento
	MSS	*Maximum Segment Size*
	Threshold	Tamanho máximo da janela de transmissão durante a fase de *slow start*, após este limite o protocolo TCP entra na fase de *congestion avoidance*.
	EstimatedRTT	*Round trip time* estimado
	SampleRTT	Valor do último RTT capturado
	DevRTT	Desvio do RTT
	BaseRTT	Mínimo RTT medido
	RTO	Cálculo do valor do *timeout*
	RXBW	Estimativa da taxa de transmissão
	TXBW	Estimativa da taxa de recepção

2.2.1 TCP RENO

Contato:

O TCP Reno foi descrito por V. Jacobson, Professor e pesquisador na University of California, Berkeley.

Padrão:

O padrão está disponível na RFC 2001 [6].

Cenário alvo da utilização:

Esta estratégia é para uso geral na comunicação de dados orientada à conexão.

Justiça:

Esta estratégia se demonstrou justa, tanto para com fluxos que utilizam o mesmo protocolo quanto para com outros protocolos. Tende a ser excessivamente conservadora quanto à utilização da banda passante disponível [20].

Referências adicionais:

Não foram necessárias devido a ampla sua utilização.

Implementação / API:

Esta estratégia é utilizada pela maioria dos sistemas operacionais.

Descrição:

O TCP Reno surgiu inicialmente baseado no artigo de V. Jacobson em 1990 [21]. O TCP Reno teve por objetivo, na sua criação, evitar um congestionamento global na Internet. No TCP Reno foram introduzidos os algoritmos de *slow start*, *congestion avoidance*, *fast retransmit* e *fast recovery*, descritos anteriormente.

O algoritmo para evitar o congestionamento no TCP Reno é o AIMD (***A**dditive **I**ncrease **M**ultiplicative **D**ecrease*), que funciona da seguinte forma:

- Para cada ACK recebido, contendo o número de seqüência esperado, por RTT:
  - Durante a fase de *congestion avoidence* (**CA**) é aplicada, sobre a janela de congestionamento, a Fórmula 2.1, que resulta no acréscimo de um MSS a cada RTT.

$$CongWin = CongWin + (1 \times MSS) \times \frac{(1 \times MSS)}{CongWin} \tag{2.1}$$

– Na fase de *slow start* **SS** a Fórmula 2.2 resulta na duplicação da janela de congestionamento a cada RTT. Esta fase está limitada ao limite de crescimento **Threshold**, após este limite o TCP Reno entra na fase de *congestion avoidance* (**CA**).

$$CongWin = CongWin + 1 \times MSS \tag{2.2}$$

- Na detecção de congestionamento em um dado RTT a ação do TCP Reno em resposta ao evento depende exclusivamente de qual dos dois eventos foi percebido:

  – Na recepção do terceiro ACK em duplicata a janela de congestionamento e o **Threshold** são reduzidas à metade, o que reduz o fluxo de envio dos pacotes na mesma proporção, e coloca o TCP Reno na fase de *congestion avoidance*. Estas ações são definidas pelas Fórmulas 2.3 e 2.4, que são executadas na seqüência apresentada.

$$Threshold = \frac{CongWin}{2} \tag{2.3}$$

$$CongWin = Threshold \tag{2.4}$$

  – Quando o temporizador expira, sem que o ACK com o número de seqüência aguardado tenha sido recebido, temos o evento de *timeout*, que no TCP Reno provoca a redução do limite **Threshold** a metade da janela de congestionamento e a janela de congestionamento é reduzida a apenas um MSS. As Fórmulas 2.5 e 2.6 definem estas ações.

$$Threshold = \frac{CongWin}{2} \tag{2.5}$$

$$CongWin = 1 \times MSS \tag{2.6}$$

O TCP Reno introduziu um aspecto de segurança contra o congestionamento, inédito até então, mas como a estratégia implementada é muito conservadora levou a alguns efeitos indesejados, principalmente para redes de alta performance e nas novas tecnologias de redes sem fio. Vejamos o caso da rede sem fio, a perda de um pacote nem sempre é por congestionamento. Esta também pode ser causada por alguma dificuldade momentânea na

recepção ou transmissão em algum dos *links* sem fio ocasionando a perda de integridade de pacotes, mas no TCP Reno tem como efeito a redução sua taxa de transmissão à metade, o que induz a uma subutilização da banda passante disponível.

Mapa de eventos:

Na Tabela 3, são apresentados os eventos aos quais o TCP Reno responde, dependendo do seu estado no momento, e os pseudo-códigos utilizados para efetuar esta reação.

Tabela 3: Controle do Congestionamento no TCP RENO

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin=CongWin+MSS; if (CongWin > Threshold) –>Estado <- CA;	Aumento aditivo, resultando no incremento de 1 MSS a cada RTT.
CA	**ACK**	CongWin=CongWin + MSS x (MSS/CongWin);	Resulta na duplicação da CongWIN a cada RTT.
SS ou CA	**TACK**	Threshold = CongWin/2; CongWin = Threshold; Estado <- CA;	*Fast recovery*, implementa o decréscimo multiplicativo. CongWin não cai abaixo de 1 MSS.
SS ou CA	**TOUT**	Threshold=CongWin/2; CongWin = 1 MSS; Estado <- SS;	Entra no *slow Start*.
SS ou CA	**DACK**	Incrementa o contador de ACK em duplicata para o segmento que está recebendo o ACK.	CongWin e Threshold não são alterados.

2.2.2 TCP VEGAS

Contato:

Em 1994, L. Brakmo, S. O'Malley e L. Peterson, pertencentes ao Department of Computer Science. University of Arizona,Tucson, AZ, descreveram o TCP Vegas [7].

Padrão:

O artigo [7] define os mecanismos introduzidos e alterados pelo TCP Vegas.

Cenário alvo da utilização:

Esta estratégia é para uso geral na comunicação de dados.

Justiça:

Esta estratégia se demonstrou justa, tanto para com fluxos do mesmo protocolo quanto para com outros protocolos. Tende a perder espaço quando utilizada em conjunto com fluxos que atendem a estratégias mais agressivas [20].

Referências adicionais:

Não foram necessárias devido a sua ampla utilização.

Implementação / API:

Esta estratégia é utilizada por uma grande quantidade de sistemas operacionais.

Descrição:

O principal motivo para a definição do TCP Vegas foi que as tomadas de decisão referentes ao fluxo são apenas reativas, no caso do TCP Reno, ou seja, dado um evento de perda de pacote o TCP Reno apenas reage a este, o que causa instabilidade na taxa de transmissão devido a suas estratégias de controle de congestionamento conservadoras. O TCP Vegas leva em conta as variações do RTT e da taxa de transmissão de pacotes para aumentar ou reduzir sua janela de congestionamento. Quando um evento ocorre o TCP Vegas aumenta ou decai a sua janela de congestionamento de apenas um MSS por evento detectado, durante o evento são levados em conta a taxa de transmissão esperada e a efetivamente alcançada. Desta forma, a idéia é de que através dos "sinais" que a rede envia possamos ajustar a taxa de transmissão de forma a evitar os eventos de perda de pacotes.

O mecanismo de controle do congestionamento do TCP Vegas utiliza uma única função na recepção de um ACK, contendo o número de seqüência esperado, tanto na fase de

slow start quanto na fase de *congestion avoidance*. Esta função (a qual chamamos de **FJ**, cujo pseudo-código é apresentado na Figura 3), inicialmente, define a taxa de transmissão esperada (**ExpectedRATE**), que é calculada através da janela de congestionamento corrente (**CongWin**) e o menor RTT medido até o momento (**BaseRTT**), como apresentado na Fórmula 2.7. Outro dado calculado é a taxa de transmissão atual (**ActualRATE**) que utiliza o último (**RTT**) medido e também a quantidade de *bytes* enviados desde o início da contagem de tempo deste RTT até a sua medição, esta quantidade de *bytes* designamos **dwin**, como representado na Fórmula 2.8.

$$ExpectedRATE = \frac{CongWin}{BaseRTT} \tag{2.7}$$

$$ActualRATE = \frac{dwin}{RTT} \tag{2.8}$$

A diferença entre os dois valores (**diff**), obtidos pela aplicação das Fórmulas 2.7 e 2.8, é comparado com os valores, definidos através de parâmetros, **alfa** e **beta** e desta forma temos três situações: (i) o valor de **diff** está entre **alfa** e **beta**, então o TCP Vegas considera que a taxa de transmissão está dentro do esperado e não a altera; (ii) **diff** é inferior de **alfa**, neste caso a janela de congestionamento é acrescida de um RTT; e (iii) **diff** é superior a **beta**, então a janela de congestionamento é reduzida de um MSS. Os valores, respectivamente, para **alfa** e **beta** sugeridos em [7] são de 1 MSS e 3 MSS.

```
Considerando: ALFA = 1 e BETA = 3

FJ:
   INÍCIO-FJ
      ExpectedRATE = CongWin / BaseRTT
      ActualRATE   = dwin    / dRTT
      diff         = ExpectedRATE - ActualRATE
      SE (alfa < diff) E (beta > diff) ENTÃO
         Não altera a CongWin ou o Threshold
         FIM-FJ
      FIM-SE
      SE (alfa > diff) ENTÃO
         Threshold = CongWin + 1MSS
         CongWin   = Threshold
         FIM-FJ
      FIM-SE
      Threshold = CongWin - 1MSS
      CongWin  = Threshold
   FIM-FJ
```

Figura 3: Pseudo-código da função de controle da janela de congestionamento no TCP Vegas

Quando um evento de perda de pacotes, recepção do terceiro ACK em duplicata ou um *timeout*, é percebido o TCP Vegas reduz sua janela de congestionamento de um MSS.

Mapa de eventos:

Na Tabela 4, são apresentados os eventos detectados, durante as atividade do TCP Vegas, bem como os pseudo-códigos utilizados para responder a eles.

Tabela 4: Controle do Congestionamento no TCP VEGAS

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	Função da janela (FJ).	Aumento aditivo, resultando no incremento ou decremento de 1 MSS a cada RTT.
CA	**ACK**	Função da janela (FJ).	Igual acima.
SS ou CA	**TACK**	Threshold = CongWin-1MSS; CongWin = Threshold;	*Fast recovery*, implementa o decréscimo aditivo. CongWin não cai abaixo de 1MSS. Na verdade, a quantidade de ACKs em duplicata cai de três apenas uma ocorrência de ACK em duplicata para disparar o evento TACK.
SS ou CA	**TOUT**	Threshold=CongWin-1MSS; CongWin=Threshold;	Entra no *slow start*.

2.2.3 TCP High Speed (HSTCP)

Contato:

O HSTCP foi introduzido por Sally Floyd, que é pesquisadora no ICSI (International Computer Science Institute).

Padrão:

A definição do HSTC pode ser encontrada na RFC 3649 [8], esta se encontra em estado experimental.

Cenário alvo da utilização:

Os experimentos iniciais [9] demonstram que o HSTCP é muito melhor que o TCP padrão. Sua utilização na Internet pode afetar os fluxos TCP padrão, pois para baixas perdas de pacotes ele tende a ocupar de forma agressiva a banda passante.

Justiça:

Esta característica ainda não foi explorada de forma mais conclusiva nesta variação do TCP.

Referências adicionais:

http://icir.org/floyd/hstcp.html - Neste sítio encontramos uma série de informações adicionais desta estratégia além de alguns experimentos.

Implementação / API:

Implementação para o *kernel* do Linux 2.4.19 e alguns resultados iniciais de experimentos podem ser obtidos em *http://www.hep.man.ac.uk/u/garethf/hstcp/*.

Descrição:

O HSTCP tem como principal característica a redução do tempo de recuperação de uma perda de pacote através da modificação do algoritmo AIMD do TCP Reno. A modificação do algoritmo somente entra em ação em "grandes" janelas de congestionamento, ou seja, se a janela for inferior a um determinado valor (exemplo: menor que 16 segmentos) o AIMD padrão é utilizado. O HSTCP mantém o início lento, *slow start*, do TCP Reno.

O HSTCP utiliza duas funções, de nome "**a**" e "**b**", que retornam valores lidos da Tabela 5 definida pelos projetistas do método e capturada no *patch* para o *kernel* 2.4.20 do Linux. Esta tabela relaciona o tamanho janela de congestionamento corrente à dimensão do crescimento ou da redução aplicados a **CongWin**. Estas funções são acionadas na

ocorrência de um evento de perda de pacote ou ACK recebido com o número de seqüência esperado.

Tabela 5: Tabela que relaciona o tamanho da janela de congestionamento aos valores "**a**" e "**b**" para o High Speed TCP

CongWin	**a**	**b**	**CongWin**	**a**	**b**	**CongWin**	**a**	**b**	**CongWin**	**a**	**b**
41	1	126	5308	21	62	19078	41	45	48075	61	32
116	2	113	5771	22	61	20067	42	44	50261	62	32
217	3	104	6254	23	60	21092	43	43	52559	63	31
340	4	98	6758	24	59	22155	44	43	54977	64	31
483	5	94	7284	25	58	23257	45	42	57528	65	30
647	6	90	7831	26	57	24399	46	41	60225	66	29
829	7	86	8401	27	56	25584	47	41	63086	67	29
1030	8	84	8993	28	55	26813	48	40	66130	68	28
1249	9	81	9609	29	54	28088	49	40	69383	69	27
1486	10	79	10249	30	53	29412	50	39	72873	70	27
1741	11	77	10913	31	52	30786	51	38	76641	71	26
2015	12	75	11603	32	51	32215	52	38	80738	72	25
2306	13	73	12318	33	51	33700	53	37	85231	73	25
2616	14	71	13061	34	50	35244	54	37	90219	74	24
2944	15	70	13830	35	49	36852	55	36	95845	75	23
3290	16	68	14628	36	48	38527	56	35	102342	76	22
3656	17	67	15456	37	47	40273	57	35	110131	77	21
4040	18	65	16313	38	47	42095	58	34	110131	77	21
4443	19	64	17202	39	46	43999	59	34	120146	78	20
4866	20	63	18123	40	45	45990	60	33	135785	79	19

O mecanismo de controle do congestionamento, para janelas inferiores a 16 MSS, é igual ao do TCP Reno, mas para janelas superiores a este valor funciona da seguinte forma:

- Quando da recepção de um ACK, com o número de seqüência esperado, é aplicada a Fórmula 2.9, tanto na fase de *slow start* quanto na fase de *congestion avoidance*.

$$CongWin = CongWin + \frac{a(CongWin)}{CongWin} \tag{2.9}$$

- Em resposta a um evento de congestionamento, recepção do terceiro ACK em duplicata ou *timeout*, o HSTCP utiliza a Fórmula 2.10 para efetuar a redução da janela de congestionamento.

$$CongWin = CongWin - \frac{b(CongWin)}{256} \times CongWin \tag{2.10}$$

Mapa de eventos:

Abaixo, na Tabela 6, são apresentados os eventos, detectados pelo HSTCP e, também, os pseudo-códigos utilizados para responder a eles.

Tabela 6: Controle do Congestionamento no High Speed TCP

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin=CongWin+ a(CongWin)/CongWin; if (CongWin > Threshold) –> Estado <- CA;	A função "**a**" retorna dados de uma tabela em memória. (de acordo com o *patch* para *kernel* 2.4.20 do Linux)
CA	**ACK**	CongWin=CongWin+ a(CongWin)/CongWin;	A função "**a**" retorna dados de uma tabela em memória. (de acordo com o *patch* para *kernel* 2.4.20 do Linux).
SS ou CA	**TACK**	Threshold=CongWin - (b(CongWin)/256) x CongWin; CongWin=Threshold; Estado <- CA;	*fast recovery*, implementa o decréscimo multiplicativo. CongWin não cai abaixo de 1MSS. A função "**b**" retorna dados de uma tabela em memória. (de acordo com o *patch* para *kernel* 2.4.20 do Linux).
SS ou CA	**TOUT**	Threshold=CongWin - (b(CongWin)/256) x CongWin; CongWin=Threshold; Estado <- SS;	Entra no *Slow start*. A função "**b**" retorna dados de uma tabela em memória. (de acordo com o *patch* para *kernel* 2.4.20 do Linux).
SS ou CA	**DACK**	Incrementa o contador de ACK em duplicata para o segmento que está recebendo o ACK.	CongWin e Threshold não são alterados.

2.2.4 Scalable TCP (STCP)

Contato:

O Scalable TCP foi lançado em Abril de 2003 no artigo [11] por Tom Kelly, Cambridge University, UK. Tom Kelly é pesquisador da divisão de Tecnologia da Informação do CERN.

Padrão:

A definição do STCP pode ser encontrada junto com o HSTCP na RFC 3649 [10], que se encontra em estado experimental.

Cenário alvo da utilização:

Os experimentos [11] iniciais demonstram que o STCP é muito melhor que o TCP Reno. Sua utilização e o impacto desta performance geral da Internet ainda não foram mensurados.

Justiça:

Esta característica ainda não foi explorada.

Referências adicionais:

Neste sítio, *http://www-lce.eng.cam.ac.uk/ ctk21/scalable/*, encontramos além do detalhamento desta variação do TCP alguns resultados de experimentos.

Implementação / API:

Existe uma implementação para Linux na forma de um *patch* para o *kernel*, que pode ser encontrada em *http://www-lce.eng.cam.ac.uk/ ctk21/scalable/*.

Descrição:

A meta principal do STCP é melhorar o tempo de recuperação durante a perda de pacotes em relação ao TCP Reno.

Os tempos de recuperação de pacotes perdidos para o TCP Reno, bem como no HSTCP, são proporcionais ao tamanho da janela de congestionamento e ao MSS, já no STCP este tempo é proporcional apenas ao MSS. Como no HSTCP existe um tamanho de janela no qual este mecanismo é acionado, enquanto a janela de congestionamento for inferior a esta dimensão o STCP funciona de forma idêntica ao TCP Reno. A janela de congestionamento para a ativação do mecanismo deve superar o tamanho de 16 MSS.

Veja, na seqüência, o algoritmo para evitar o congestionamento, comparado ao mesmo do TCP Reno:

- Para cada ACK recebido, com o número de seqüência esperado, por RTT, o TCP Reno aplica as Fórmulas 2.11 e 2.12 nas fases de slow start e congestion avoidance, respectivamente, já o STCP aplica a Fórmula 2.13 em qualquer uma das fases. Podemos verificar através das fórmulas apresentadas que o STCP é uma versão menos "agressiva" quanto ao crescimento da janela de congestionamento que o TCP Reno.

$$CongWin = CongWin + 1 \times MSS \tag{2.11}$$

$$CongWin = CongWin + (1 \times MSS) \times \frac{(1 \times MSS)}{CongWin} \tag{2.12}$$

$$CongWin = CongWin + 0,01 \times MSS \tag{2.13}$$

- Quando um congestionamento é detectado em um dado RTT o TCP Reno aplica as Fórmulas STCP4 e STCP5, como uma resposta a recepção do terceiro ACK em duplicata e no estouro do temporizador (*timeout*), já o STCP utiliza as Fórmulas STCP6 e STCP7 para tratar os mesmos eventos, respectivamente. Podemos verificar através das fórmulas apresentadas que o STCP é uma versão menos "conservadora" quanto à redução da janela de congestionamento que o TCP Reno.

$$Threshold = 0.5 \times CongWin \longrightarrow CongWin = Threshold \tag{2.14}$$

$$Threshold = 0.5 \times CongWin \longrightarrow CongWin = 1 \times MSS \tag{2.15}$$

$$Threshold = 0.875 \times CongWin \longrightarrow CongWin = Threshold \tag{2.16}$$

$$Threshold = 0.875 \times CongWin \longrightarrow CongWin = 1 \times MSS \tag{2.17}$$

Os valores de 0.01 e 0.875 são sugeridos, mas na implementação Linux estes podem ser configurados no */proc* por um super usuário.

Mapa de eventos:

Na Tabela 7, são apresentados os eventos que o SPTC detecta e, também, os pseudo-códigos utilizados para responder a eles.

Tabela 7: Controle do Congestionamento no Scalable TCP

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin=CongWin+0,01MSS; if (CongWin>Threshold) –>Estado <- CA;	Resulta em um acréscimo de 1% de um MSS na CongWIN a cada RTT.
CA	**ACK**	CongWin=CongWin+0,01MSS;	Resulta em um acréscimo de 1% de um MSS na CongWIN a cada RTT.
SS ou CA	**TACK**	Threshold=CongWin x 0.875; CongWin=Threshold; Estado <- CA;	*Fast recovery*, implementa o decréscimo multiplicativo. CongWin não cai abaixo de 1 MSS.
SS ou CA	**TOUT**	Threshold=CongWin x 0.875; CongWin=1 MSS; Estado <- SS;	Entra no *slow start*.
SS ou CA	**DACK**	Incrementa o contador de ACK em duplicata para o segmento que está recebendo o ACK.	CongWin e Threshold não são alterados.

2.2.5 FAST TCP

Contato:

O FAST TCP foi, inicialmente, introduzido por Steven Low. Professor of Computer Science and Electrical Engineering. B.S. of the California Institute of Technology, Computer Science.

Padrão:

A principal referência ao FAST TCP pode ser encontrada no IETF Draft [19].

Cenário alvo da utilização:

Os experimentos iniciais demonstram que o FAST TCP [18] é muito melhor que os TCP Reno e Vegas em ambientes com uma grande banda passante e um grande RTT.

Justiça:

Consegue uma justiça proporcional sem penalizar os grandes fluxos [20, 22] (para os casos estudados até o momento).

Referências adicionais:

O endereço a seguir é mantido pelos criadores desta variação e contém muitos artigos e dados sobre o FAST TCP. (*http://netlab.caltech.edu/FAST/*).

Implementação / APIs:

Encontramos o FAST TCP como um *patch* para o *kernel* 2.4.22 e 2.4.24 do Linux.

Descrição:

O Fast TCP, baseado no padrão TCP Vegas, tem por principal característica prover várias propriedades de fluxo, tais como: equilíbrio estável, justiça bem definida, altas taxas de *throughput* e de utilização de banda passante.

Esta estratégia requer modificações apenas no lado emissor do transporte e nenhuma cooperação do lado dos roteadores ou receptores. O Fast TCP utiliza o retardo de fila e a perda de pacotes como medida de congestionamento enquanto o TCP Reno utiliza apenas a perda de pacotes. Para estabilizar a taxa de envio de pacotes (reduzindo a probabilidade de perdas) utiliza uma equação ao invés do AIMD, adotado pelo TCP Reno, para controlar a taxa de envio de pacotes da fonte. Desta forma, a oscilação ao nível de pacotes é reduzida atingindo os objetivos de alta performance, estabilidade e justiça desta estratégia.

O mecanismo de controle do congestionamento do FAST TCP é composto por quatro principais componentes. Estes são funcionalmente independentes e, por esta razão, podem ser projetados e implementados separadamente, são eles:

- **Estimativa**;
- **Controle de janela**;
- **Controle de dados**; e
- **Burstiness Control**.

A seguir temos a descrição de cada um deles:

- **Estimativa**:

Este componente computa duas informações de *feedback* do estado da rede:

Retardo de fila - Calcula RTT mínimo (BaseRTT) e o RTT médio. O RTT médio considera os dez últimos RTTs medidos, mas na implementação deste trabalho este valor pode ser alterado através de parâmetro.

Indicação de perda - Sinaliza para o controle dos dados que ocorreu um evento de perda.

- **Controle de janela**:

Utiliza o retardo de fila como sua principal medida para o ajuste da janela de congestionamento.

A dinâmica do retardo tem a escala correta com respeito à capacidade do *link* e ajuda a manter a estabilidade à medida que a rede cresce em capacidade.

O FAST TCP periodicamente ajusta a sua janela de congestionamento (CongWin), através da Fórmula 2.18:

$$min(2 \times CongWin, (1-g) \times CongWin + g \times (\frac{BaseRTT}{RTT}) \times CongWin + a)) \quad (2.18)$$

Onde: $\begin{cases} \mathbf{g}, & \text{é uma constante entre 0 e 1 (cujo padrão é 0,5);} \\ \mathbf{a}, & \text{é parâmetro de justiça do protocolo (cujo padrão é 100); e} \\ \textbf{em azul}, & \text{expressão que mede a quantidade de pacotes no } \textit{path}. \end{cases}$

Podemos observar que o ajuste da janela depende da flutuação da capacidade do *path*. Em outras palavras, a magnitude do ajuste para cima ou para baixo depende basicamente da distância do ponto de equilíbrio da transmissão.

- **Controle de dados**:

O controle de dados seleciona o próximo pacote a ser transmitido entre três candidatos:

1. Novos pacotes;
2. Pacotes perdidos; e
3. Pacotes transmitidos e que ainda não receberam ACKs.

Quando não se têm perdas os novos pacotes são transmitidos à medida que os ACKs chegam. Durante uma recuperação de perda a decisão é enviar uma mistura entre os três potenciais candidatos, listados anteriormente. Os autores, em [19], afirmam que esta decisão se torna mais importante quando a banda passante e o *delay* são grandes. De fato, no *patch* implementado para Linux, a decisão recai sobre o mais antigo pacote sem ACK, como nos demais TCPs.

- **Burstiness Control**:

Este é o componente responsável pelo ajuste no fluxo dos pacotes, a serem transmitidos, à largura de banda disponível. A CPU pode ficar ocupada por um longo período servindo às interrupções dos pacotes recebidos, gerando um acúmulo de pacotes a serem transmitidos e aumentando a probabilidade de perdas. O controle é feito na decisão de quantos pacotes serão enviados e no aumento da janela de congestionamento, aproveitando-se deste período ocioso, para o ponto de equilíbrio da taxa de transmissão pelo componente de controle da janela de congestionamento. De fato, no *patch* implementado para Linux, este mecanismo não é implementado.

Mapa de eventos:

A Tabela 8, apresenta os eventos que o FAST TCP detecta e, também, os pseudo-códigos utilizados para responder a eles.

Tabela 8: Controle do Congestionamento no FAST TCP

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin = FC();	Atualiza a janela de congestionamento.
CA	**ACK**	CongWin = FC();	Atualiza a janela de congestionamento.
SS ou CA	**TACK**	Threshold = FC(); CongWin = Threshold; Estado <- CA;	*Fast recovery.* O Três ACKs em duplicata no FAST é sinalizado no primeiro ACK duplicado recebido.
SS ou CA	**TOUT**	Threshold = FC(); CongWin = Threshold; Estado <- SS;	Entra no *slow start.*

O pseudo-algoritmo da função de ajuste (**FC**) da janela de congestionamento do FAST TCP é apresentado na Figura 4. Os parâmetros indicados pelos criadores do FAST TCP, em [19], são: (i) g = 0,5; (ii) a = 100; e (iii) b = 100;

```
Considerando: g = 0.5 , a = 100 e b = 100.
              OldCongWin é a janela de congestionamento no RTT anterior.
FC:

  INÍCIO-FC

    TagCongWin=oldCongWinx(avgRTT-baseRTT)

    SE (TagCongWin < axavgRTT) OU (TagCongWin >= bxavgRTT) ENTÃO

      CongWIN=min{2xCongWIN,(1-g)xoldCongWIN+gx(BaseRTT/AvgRTT)xCongWIN+a}

    FIM-SE

  FIM-FJ
```

Figura 4: Pseudo-código da função de controle da janela de congestionamento no FAST TCP

2.2.6 BIC-CUBIC TCP (CUBIC)

Contato:

A variação do TCP BIC-CUBIC surgiu em 2005 e foi introduzida pelo Associate Professor Injong Rhee, Department of Computer Science, North Carolina State University.

Padrão:

A principal referência ao TCP BIC-CUBIC é o artigo [5].

Cenário alvo da utilização:

Os experimentos iniciais [5] demonstram que o BIC-CUBIC TCP é muito melhor que o TCP Reno em redes de alta velocidade. Sua utilização na Internet e o impacto de sua utilização ainda não foram mensurados, mas acredita-se que não ocasionará maiores problemas, pois em testes em redes heterogêneas, e relativamente grandes, não apresentaram problemas.

Justiça:

A justiça intra-protocolo é provida pela função CUBIC [5], ou seja, por fluxos do mesmo protocolo.

Implementação / APIs:

O código para a simulação da CUBIC pode ser encontrado em:

http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.htm

O CUBIC é utilizado como o TCP padrão do *kernel* do LINUX desde a versão 2.6.7.

Referências adicionais:

O sítio a seguir contém vasto material sobre esta variação do TCP e é mantido pelo projetista do BIC-CUBIC.

http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/index_files/Page815.htm

Descrição:

O BIC é uma variante do TCP Reno para melhorar a performance em redes de alta velocidade. O principal ponto do BIC é o de ter apenas uma única função para o crescimento da janela de congestionamento, o que proporciona uma melhoria na estabilidade do protocolo, na utilização dos recursos da rede e, também, é mais justa em relação à outros fluxos de alta velocidade, com o mesmo ou diferentes RTT médios.

No BIC, o controle de congestionamento é visto como um problema de busca, para o qual o mecanismo de controle do congestionamento precisa descobrir uma taxa de transmissão justa e eficiente para o *PATH* fim a fim da conexão corrente. A técnica utilizada, que é a da busca binária, permite que a taxa cresça de forma logarítmica, ou seja, mais rapidamente quando distante do ponto de equilíbrio e mais lentamente a medida que se aproxima deste. A técnica de busca binária é combinada com um acréscimo aditivo (maior que no caso do TCP Reno). Este acréscimo aditivo também permite ao protocolo aumentar a janela mais rapidamente quando há muita banda disponível, minimizando a questão do *slow start*.

A função CUBIC melhora a funcionalidade proposta no BIC de forma a ter apenas uma função para o controle de janela, não apenas para o seu crescimento como no caso do BIC (uma função cúbica - como o nome sugere).

A CUBIC, após uma redução de janela, cresce rapidamente até se aproximar de **Wmax** (janela máxima - calculada dinamicamente - inicialmente contém um valor considerado infinito) e começa a reduzir a velocidade de crescimento quando se aproxima de **Wmax** até um crescimento zero quando atinge a **Wmax**. Após este ponto, ela cresce lentamente nas proximidades de **Wmax** e mais rapidamente quando mais distante. Este crescimento ou diminuição mais lento nas proximidades de **Wmax** confere estabilidade ao modelo e a velocidade de crescimento ou diminuição da janela quando distante de **Wmax** provê escala a esta implementação.

A janela de congestionamento da CUBIC, tanto para o seu crescimento quanto para a sua redução, é calculada através das Fórmulas 2.19 e 2.20.

$$CongWin = min(CongWin + Smax, C \times (t - k) \times 3 + Wmax) \tag{2.19}$$

$$k = \sqrt[3]{Wmax \times \frac{B}{C}} \tag{2.20}$$

Onde:
- $Smax$, é um valor que limita os "saltos" de crescimento;
- C, é um fator de escala;
- t, é o tempo decorrido desde a última redução de janela;
- $Wmax$, é o tamanho da janela desde a última redução;
- k, é calculado pela Fórmula 2.20 ; e
- B, é um fator de decréscimo após o evento de perda de pacote.

Para aumentar a justiça e estabilidade, o incremento da janela não pode ser superior a **Smax** por segundo. Empiricamente [5] temos que **C** é **0.4**, **Smax** é **160** e **B** é **0.2**. Baseado nas análises dos autores, estes valores fornecem uma razoável justiça e velocidade de convergência ao TCP CUBIC.

Mapa de eventos:

A Tabela 9, apresenta os eventos detectados pelo BIC-CUBIC TCP, bem como os pseudo-códigos utilizados para responder a eles.

Tabela 9: Controle do Congestionamento no BIC-CUBIC TCP

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin=min{Smax+CongWin, (C(t-K)3+Wmax)}; if (CongWin > Threshold) –> Estado <- CA;	K é raiz-cúbica(Wmaxx B/C). Empiricamente C é 0.4, Smax é 160 e B é 0.2. t - é o tempo contado desde a última redução de janela.
CA	**ACK**	CongWin=min{Smax+CongWin, (C(t-K)3+Wmax)};	Igual acima.
SS ou CA	**TACK**	Captura o novo t. Wmax = CongWin; Threshold=min{Smax+CongWin, (C(t-K)3+Wmax)}; CongWin=Threshold; Estado <- CA;	*Fast recovery.*
SS ou CA	**TOUT**	Captura o novo t. Wmax=CongWin; Threshold=min{Smax+CongWin, (C(t-K)3+Wmax)}; CongWin=Threshold; Estado <- SS;	Entra no *slow start.*
SS ou CA	**DACK**	Incrementa o contador de ACK em duplicata para o segmento que está recebendo o ACK.	CongWin e Threshold não são alterados.

2.2.7 TCP Westwood (TCPW)

Contato:

O TCPW foi descrito, inicialmente, em 2002 no artigo [12] pelos pesquisadores L. A. Grieco e S. Mascolo. Saverio Mascolo, Associate Professor at Politecnico di Bari. Mascolo at Poliba, IT.

Padrão:

No artigo [12], encontramos as definições do TCP Westwood.

Cenário alvo da utilização:

Os experimentos [14] iniciais demonstram que o TCPW é muito melhor que o TCP Reno em redes sem fio e também obtém bons resultados em redes convencionais e de alta velocidade.

Justiça:

Alguns estudos [14] indicam que esta estratégia é justa em relação ao TCP RENO e apresenta uma melhor performance.

Implementação / API:

O código do TCP Westwood+ pode ser encontrado em:

http://c3lab.poliba.it/index.php/Westwood:Linux

TCP Westwood+ está disponível como *patch* no *Kernel* 2.4 e 2.6 do Linux.

Referências adicionais:

Abaixo temos o sítio oficial deste desenvolvimento que contém uma série de dados sobre os estudos realizados pelos projetistas desta estratégia.

http://c3lab.poliba.it/index.php/Westwood

Descrição:

O TCPW é uma modificação do TCP RENO do lado de quem envia a informação.

O algoritmo de controle do congestionamento do TCPW é baseado na estimativa da banda passante fim-a-fim. Esta variação, continuamente estima a taxa de transmissão de pacotes através do monitoramento da taxa de recepção de pacotes ACK. Quando o congestionamento é experimentado, esta se adapta alterando a janela de congestionamento (**CongWin**) e o *slow start threshold* (**Threshold**). A janela de congestionamento é

incrementada de forma aditiva, e quando um evento de perda é detectado, tanto a janela de congestionamento quanto o **Threshold** recebem o valor da banda passante estimada (**BWE**) multiplicada pelo mínimo RTT medido. Desta forma, é evitada a redução da janela à metade, como no caso do AIMD do TCP Reno. Este mecanismo adaptativo tem outro efeito interessante, durante um congestionamento pesado a janela decai mais rapidamente e quando o congestionamento é pontual a queda é menor.

O algoritmo para o controle do congestionamento quando na recepção de um ACK, com o número de seqüência esperado, aumenta a janela de congestionamento (**CongWin**) exatamente como no TCP Reno, como podemos observar pelas Fórmulas 2.21 e 2.22, para as fases de *slow start* e *congestion avoidance*, respectivamente. Outra ação tomada neste momento e o cálculo da banda passante estimada (**BWE**), que leva em conta o mínimo RTT medido (BaseRTT) e a quantidade de *bytes* enviados no últimos RTTs dez medidos, no *patch* do *Kernel* 2.4 e 2.6 do Linux, nesta implementação podemos modificar o padrão de dez para outro valor através de parâmetros, este cálculo é apresentado na Fórmula 2.23.

$$CongWin = CongWin + 1 \times MSS \tag{2.21}$$

$$CongWin = CongWin + (1 \times MSS) \times \frac{(1 \times MSS)}{CongWin} \tag{2.22}$$

$$BWE = \frac{\sum_{i=1}^{N} \frac{dWin_i}{RTT_i}}{N} \tag{2.23}$$

Onde:
- N, é a quantidade de RTTs e dWins armazenados para a estimativa;
- $dWin$, é a quantidade de **bytes** transmitida no RTT; e
- RTT, é o RTT medido no dWin.

Nos eventos de perda de pacotes, recepção do terceiro ACK em duplicata ou um *timeout*, o TCPW utiliza o **BWE**, calculado na Fórmula 2.23, para efetuar a redução da janela de congestionamento, através das Fórmulas 2.24 e 2.25 , respectivamente em relação ao evento detectado. Note que o limite **Threshold** da janela de congestionamento nunca cai abaixo do valor de dois MSSs.

$$Threshold = max(2 \times CongWin, \frac{BWE \times BaseRTT}{MSS}) \longrightarrow CongWin = Threshold \tag{2.24}$$

$$Threshold = max(2 \times CongWin, \frac{BWE \times BaseRTT}{MSS}) \longrightarrow CongWin = 1 \qquad (2.25)$$

Mapa de eventos:

Na Tabela 10, temos os principais eventos detectados pelo TCP Westwood, e, também, os pseudo-códigos utilizados para responder a eles.

Tabela 10: Controle do Congestionamento no TCP Westwood

Estado	**Evento**	**Ação**	**Comentário**
SS	**ACK**	CongWin=CongWin+MSS; if (CongWin>Threshold) –> Estado <- CA;	Resulta na duplicação da CongWIN a cada RTT.
CA	**ACK**	CongWin=CongWin+MSSx (MSS /CongWin);	Aumento aditivo, resultando no incremento de 1 MSS a cada RTT.
SS ou CA	**TACK**	Threshold=max(2,(TXBWx BaseRTT)/MSS); CongWin=Threshold; Estado <- CA;	*Fast recovery*, implementa o decréscimo multiplicativo. CongWin não cai abaixo de 1 MSS.
SS ou CA	**TOUT**	Threshold=max(2,(TXBWx BaseRTT)/MSS); CongWin=1MSS; Estado <- SS;	Entra no *slow start*.
SS ou CA	**DACK**	Incrementa o contador de ACK em duplicata para o segmento que está recebendo o ACK.	CongWin e Threshold não são alterados.

3 Implementação das variações do TCP

O objetivo desta implementação é desenvolver uma biblioteca de funções (APIs), que comporte as mesmas funcionalidades *socket* disponibilizadas pelos ambientes UNIX e MS Windows, dado que estas foram utilizadas no desenvolvimento do Servidor RIO. Todos os TCPs foram implementados nesta biblioteca para que realizássemos a avaliação de performance dos mesmos.

Como base, para a implementação das APIs *socket*, foram desenvolvidas duas das camadas da pilha de protocolos de comunicação da Internet [23], que se encontram relacionadas na Tabela 11:

Tabela 11: Camadas da biblioteca implementada

Camada	**Implementação**
Transporte	UDP
	TCP Reno
	TCP Vegas
	TCP Fast
	TCP WestWood
	TCP Bic-Cubic
	TCP Scalable
	TCP High Speed
Rede	IP

Na implementação utilizamos a linguagem "C" padrão, para que obtivéssemos portabilidade entre os ambientes UNIX e MS Windows.

Uma questão, que devemos esclarecer, é o porque decidimos implementar as variações do TCP e não utilizar as versões já implementadas. As razões que nos motivaram a esta implementação são as seguintes:

1. Não encontramos implementações disponíveis para todas as variações do TCP, selecionadas neste trabalho, para os sistemas operacionais alvo da avaliação, UNIX e

MS Windows.

2. Para que fossem evitadas injustiças nas avaliações devido a possíveis otimizações de código de algumas das implementações do protocolo TCP, pois devemos avaliar apenas a estratégia definida para o controle de congestionamento.

3. Pela oportunidade de implementar um arcabouço do protocolo TCP que facilitasse a construção no futuro de novas variações do TCP.

De forma a apresentar a implementação, efetuada neste trabalho, utilizaremos uma ilustração da arquitetura da biblioteca, Figura 5, na qual as seguintes representações são empregadas:

- **Retângulos desenhados através de linhas contínuas**: representam os componentes de *software* da implementação;
- **Linhas e setas**: expõem a interação entre os componentes;
- **Retângulos tracejados**: definem os componentes "externos", ou seja, os que interagem com a biblioteca implementada mas não fazem parte da mesma;
- **Cilindros**: representam arquivos em disco; e
- **Cores utilizadas**: são utilizadas para separar o que foi implementado neste trabalho dos componentes disponibilizados pelo sistema operacional e do Servidor RIO, como apresentado a seguir:
  - Em **azul** temos o Servidor RIO;
  - Em **verde** estão os componentes implementados neste trabalho;
  - Em **vermelho** temos os componentes disponibilizados pelo sistema operacional; e
  - Em **preto** destacaremos os componentes a medida que estes forem detalhados na seqüência do texto.

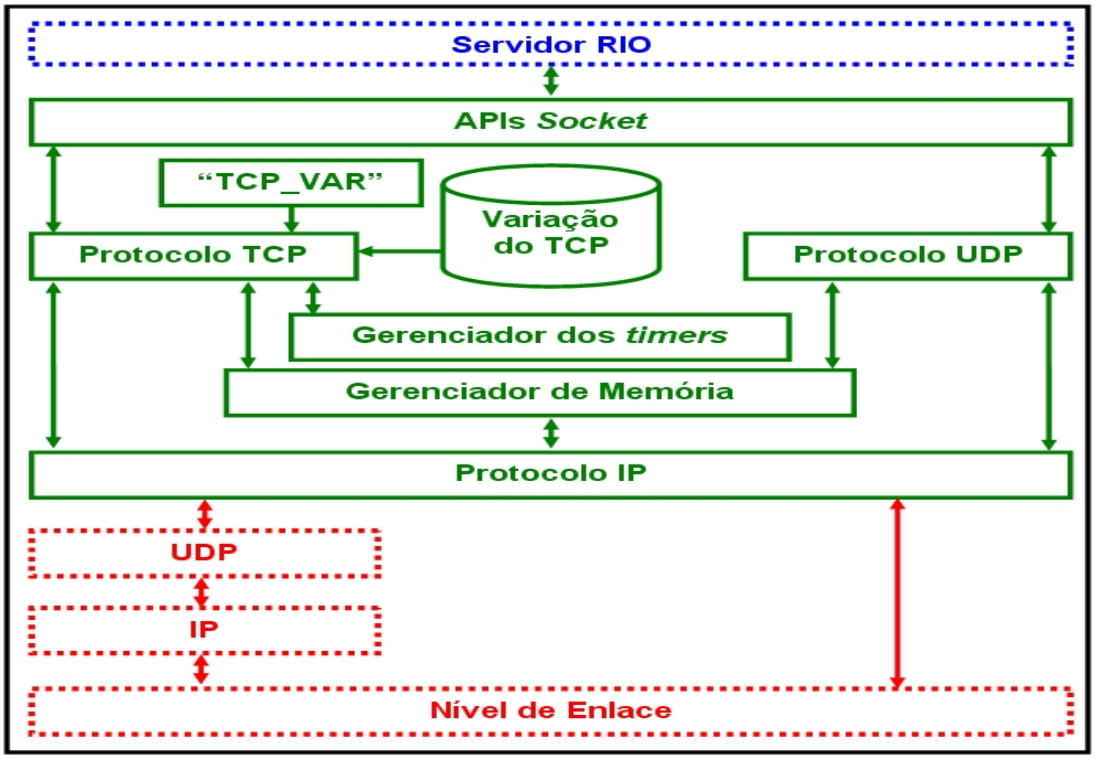

Figura 5: Arquitetura da implementação da biblioteca deste trabalho

Na Figura 5, destacadas em vermelho, temos as interfaces entre a biblioteca desenvolvida neste trabalho e a pilha de protocolos da Internet. Podemos notar que a biblioteca implementada pode utilizar como interface para a pilha de protocolos da Internet tanto a camada de enlace quanto pode fazê-lo através do protocolo UDP. Esta estratégia, que subverte a ordem da pilha de protocolos da Internet, teve razões práticas para esta forma de implementação, que estão descritas na seção 3.3, na qual apresentamos a camada de rede da biblioteca implementada e sua interação com a camada de enlace.

Nas próximas seções apresentaremos cada um dos grupos de componentes implementados, que serão destacados em preto, tendo como base a Figura 5.

3.1 APIs *Socket* e sua interação com o Servidor RIO

As APIs *socket* são o ponto de interface entre o Servidor RIO e a biblioteca implementada, através delas é que os dados são enviados ou recebidos da rede de computadores, como visto na Figura 6.

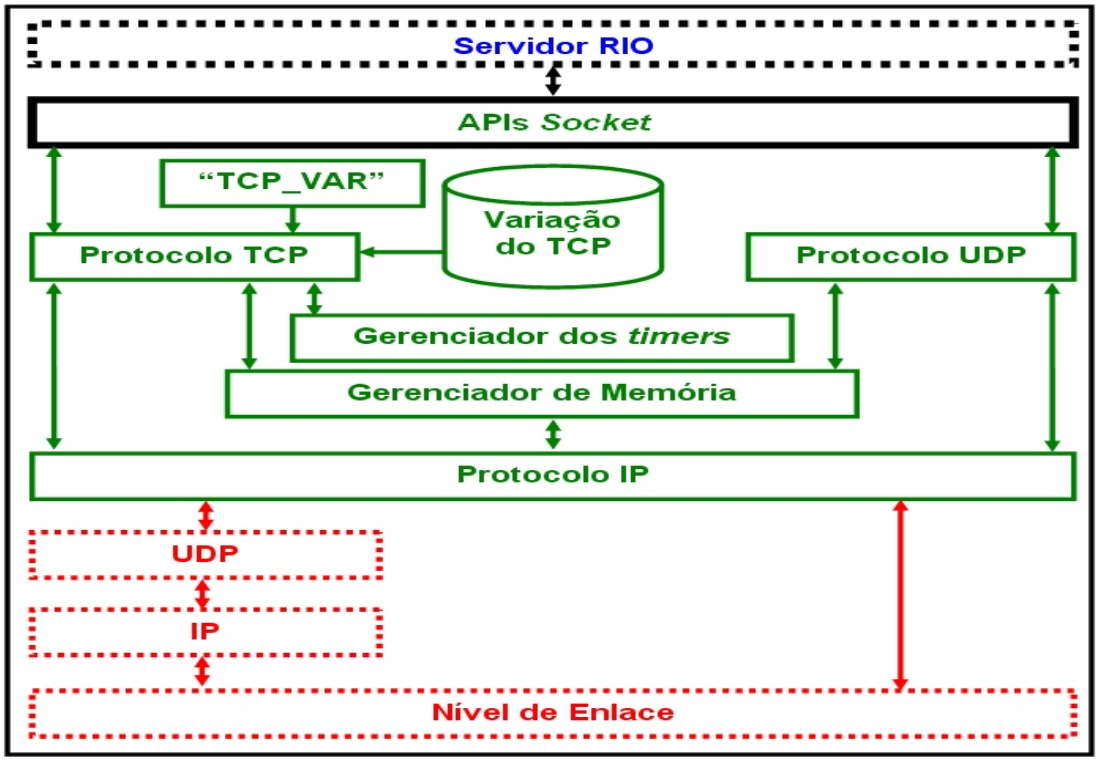

Figura 6: APIs *Socket* implementadas e sua interação com o Servidor RIO

A implementação das APIs *socket* foi baseada nas funções disponíveis para Linux e MS Windows, com o objetivo de minimizar as alterações nos aplicativos desenvolvidos para o Servidor RIO, dado que estas foram utilizadas na construção do mesmo.

As funções construídas, para este trabalho, têm os mesmos: (i) nomes; (ii) estruturas de dados; (iii) definições; e (iv) parâmetros, das disponibilizadas pelos sistemas operacionais. Por este motivo, foi adicionado um prefixo ao nome das funções, para que não ocorressem problemas durante a compilação, no caso de um aplicativo utilizar as duas implementações simultaneamente, ou seja, as APIs *socket* deste trabalho e as disponibilizadas pelo sistema operacional. O prefixo escolhido foi "**F_**", onde a letra "**F**" significa "**função**". Vejamos, como um exemplo, o caso da criação de um *socket*:

- Biblioteca do sistema operacional:
  - sock = socket(AF_INET, SOCK_DGRAM, 0);
- Biblioteca desenvolvida para este trabalho:
  - sock = F_socket(AF_INET, SOCK_DGRAM, 0);

A seguir, temos uma listagem das funções da camada de *socket*, desenvolvidas neste trabalho:

F_socket: Cria um *socket*.

F_listen: Define a "escuta" de um *socket*.

F_connect: Efetua a conexão de um *socket*.

F_accept: Aceita a conexão.

F_bind: Liga um conjunto de endereço e porta de comunicação à um *socket*.

F_close_s: Fecha um *socket*.

F_shutdown: Encerra a estrutura de comunicação da biblioteca implementada.

F_sockinit: Inicializa a estrutura de comunicação.

F_sockkick: Força a retransmissão de pacotes para um determinado *socket*.

F_recv: Recebe dados através de um *socket*.

F_recvfrom: Recebe dados através de um *socket* e retorna o endereço de onde vieram os dados recebidos.

F_send: Envia dados do usuário através de um *socket* (apenas para *sockets* orientados à conexão).

F_sendto: Envia dados do usuário para um *socket* destino específico.

F_socklen: Retorna o tamanho da fila de pacotes do *socket* tanto para envio quanto para recepção.

F_eolseq: Retorna a convenção de "fim-de-linha" para o *socket* requisitado.

F_getpeername: Recupera o *peername* do *socket* previamente alocado no *accept*.

F_getsockname: Retorna o nome local, passado em uma chamada ao F_bind() anterior.

F_settos: Define o tipo de serviço da Internet a ser usado.

F_htons: Conversão do endereço de *host* para *short int*.

F_htonl: Conversão do endereço de *host* para *long*.

F_inet_addr: Converte um endereço IP do formato de rede para o formato separado por pontos.

3.2 Camada de transporte

Na camada de transporte, da pilha de protocolos, foram implementados os protocolos UDP e TCP.

O UDP foi baseado na RFC 768 [24], que define este protocolo. Decidimos implementar o UPD por ser um protocolo simples e também poder ser utilizado, futuramente, no Servidor RIO.

O TCP padrão para o desenvolvimento foi o TCP Reno, e sua construção foi baseada nas RFCs 793 [25], 1180 [26], 2525 [27], 2582 [28], 2861 [29], 3390 [30], 4413 [31] e 4614 [32].

Antes de prosseguirmos, nas implementações do TCP e do UDP, devemos destacar dois módulos, que ofereceram suporte para as tarefas de gerenciamento de memória e controle dos *timers*, apresentados na Figura 7, e suas interações com os demais módulos implementados. Estes módulos ficam ativos, através de *threads*[1] (uma para cada módulo), durante todo o tempo no qual a biblioteca está sendo utilizada.

[1]Os *threads* são sub-processos, em nível de usuário, que podem ser chamados e programados dentro do mesmo espaço de processo.

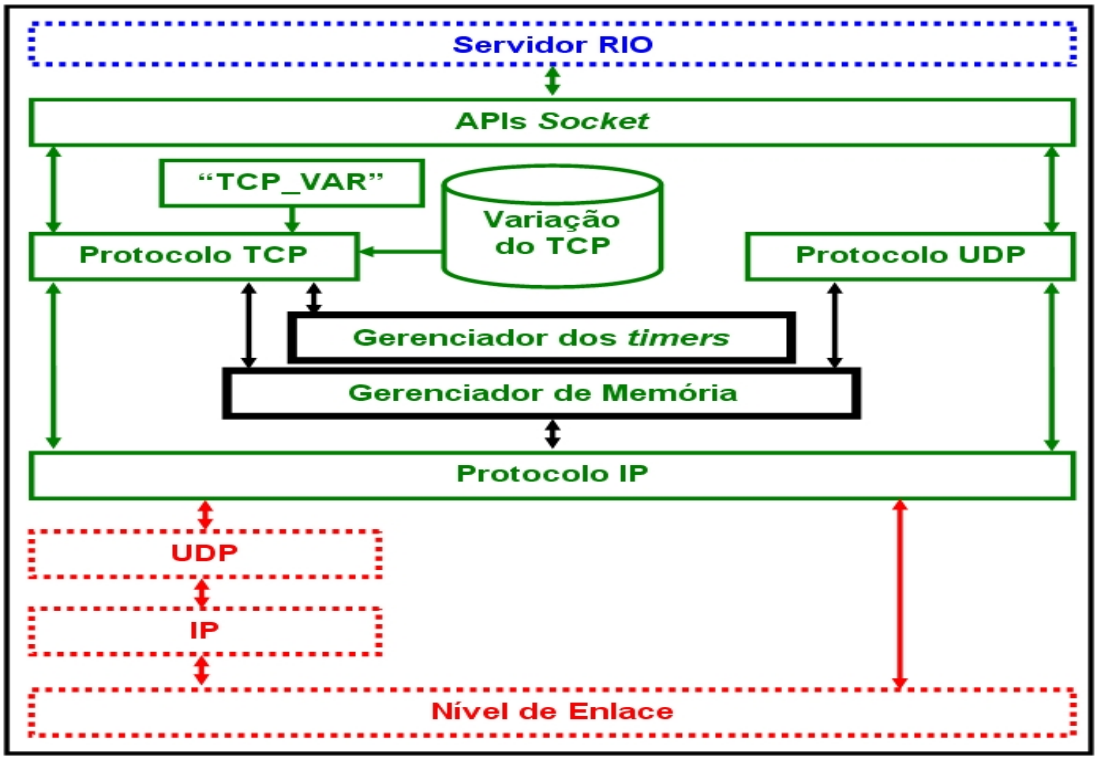

Figura 7: Gerenciadores de memória e dos *timers*

O gerenciador de memória foi criado para atender às necessidades dos seguintes módulos:

- **Protocolo TCP**: na alocação e desalocação de memória para as janelas de transmissão e recepção, na criação e eliminação dos segmentos;
- **Protocolo UDP**: na alocação e desalocação de memória para os segmentos; e
- **Protocolo IP**: na criação e eliminação dos datagramas.

Este módulo foi desenvolvido porque, no Linux, quando uma fatia de memória é alocada por uma *thread* esta não pode ser desalocada por outra, caso isto ocorra erros de execução acontecem. Portanto, a *thread* do módulo gerenciador de memória é a única responsável pela alocação e liberação de memória para a biblioteca desenvolvida. Esta tarefa é efetuada pelas seguintes funções:

mallocw: Aloca a quantidade de *bytes*, que lhe é requisitada através de parâmetro; e

freew: Libera a memória previamente alocada.

A memória, necessária para a execução deste módulo, é alocada em blocos de $32 Kbytes$ e estes blocos são organizados através de uma lista encadeada. Na Figura 8, temos um exemplo desta lista, na qual as fatias em azul representam porções de memória alocadas pelos módulos do sistema.

Outro ponto, visível na Figura 8, são os "espaços vazios" nos blocos de memória, isto ocorre devido ao processo constante de alocação e desalocação de memória, que busca a primeira porção de memória contígua que atenda à requisição ***mallocw***. Devido a esta forma de atuação, periodicamente, o gerenciador de memória reorganiza os espaços de memória visando o melhor aproveitamento dos mesmos, e quando um bloco está completamente vazio, a memória, alocada para ele, é liberada para o sistema operacional, exceto quando este for o bloco "zero", como apresentado na Figura 8, pois ao menos um bloco sempre permanece alocado. A alocação de novos blocos é efetuada quando uma requisição ***mallocw*** não puder ser atendida por nenhum dos blocos correntemente alocados.

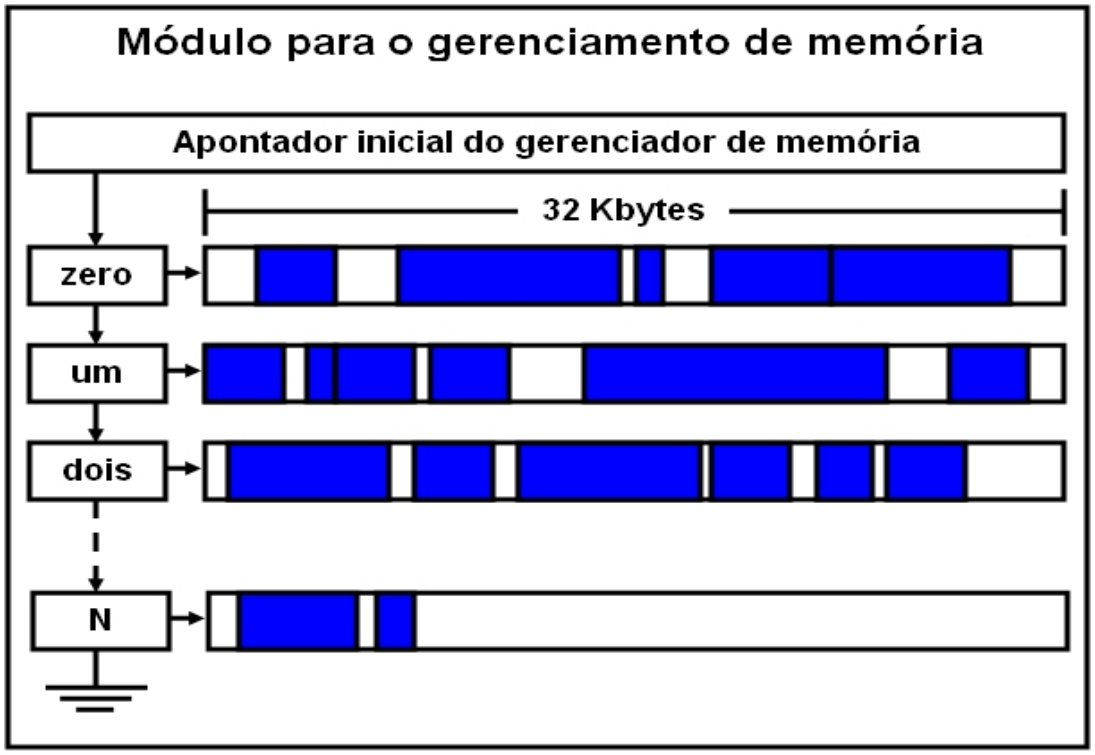

Figura 8: Gerenciador de memória

O outro módulo, que merece um destaque como base para a implementação, é gerenciador de *timers*. O *timer* é uma estrutura de dados, que é composta por um tempo de expiração, que estabelece a duração do *timer*, e um apontador para o *socket* TCP que o criou. O módulo, que implementa a utilização desta estrutura, funciona através de uma *thread* e controla uma lista ordenada de *timers*. A ordem desta lista é definida pela expiração do *timer*, ou seja, o mais próximo de expirar compõe a cabeça da lista, e se dois, ou mais, *timers* tem o mesmo tempo de expiração a ordem é fornecida pela inserção, desta forma, o elemento mais recentemente inserido fica mais distante da cabeça da lista. A inserção ou remoção de elementos na lista é efetuada pelo TCP, de acordo com suas necessidades para a implementação do protocolo, a única exceção é no caso da expiração de um *timer*. Quando ocorre a expiração de um *timer*, uma função, responsável pelo atendimento à esta exceção, é ativada para o fluxo TCP (*socket*) que criou o *timer*, agora expirado. O TCP faz uso das funções relacionadas a seguir para a manutenção dos *timers*:

start_timer: Inicia um *timer* e o insere na lista seguindo a ordem definida anteriormente.

set_timer: Atribui o tempo de expiração para um determinado *timer*, passado como parâmetro.

read_timer: Retorna quantos mili-segundos faltam para a expiração.

stop_timer: Finaliza um *timer*, antes de sua expiração, e o retira da lista.

Agora, que definimos os módulos, para o gerenciamento de memória e controle dos *timers*, voltaremos ao TPC para nos aprofundarmos em seus componentes e na interação dos mesmos com o restante da biblioteca implementada, como apresentado na Figura 9.

Sabemos que, neste trabalho, sete variações do TCP foram implementadas, e que os algoritmos que correspondem ao *slow start*, à resposta ao *timeout*, ao tratamento do terceiro ACK em duplicata e ação decorrente da recepção de um ACK esperado, diferem para cada um dos TCPs, como conseqüência estes foram alterados de forma a serem executados de acordo com o TCP utilizado para o fluxo corrente. Quando algum dos eventos ocorre, obviamente que requisite algum destes algoritmos, uma única função, para cada um dos eventos, é ativada e nela é feita a seleção da rotina específica da variação do TCP responsável pelo fluxo corrente.

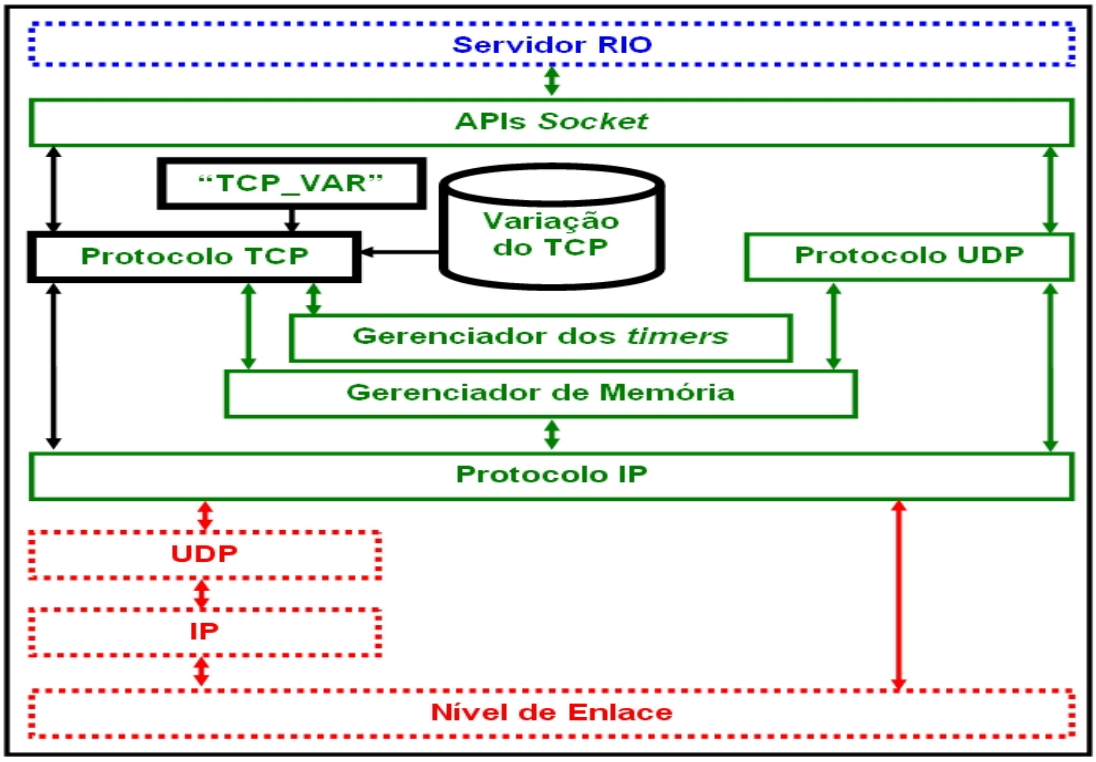

Figura 9: Protocolo TCP

Outro ponto relevante, é a escolha de qual dos sete TCPs será utilizado por um determinado fluxo. Para isto, foi criada a variável de ambiente **TCP_VAR**, destacada na Figura 9, que tem como seu conteúdo o caminho completo e o nome do arquivo de configuração do TCP a ser utilizado, este arquivo é representado pelo cilindro "**Variação do TCP**". Isto é feito no momento em que o processo executa a instrução que cria um *socket*, através dos seguintes passos:

- **Passo 1**: Capturar o conteúdo da variável de ambiente "**TCP_VAR**", que deve conter o caminho completo do arquivo de configuração do TCP desejado. Caso esta variável não esteja definida ou o arquivo apontado por ela não possa ser lido, utiliza-se o TCP Reno.
- **Passo 2**: Leitura do conteúdo do arquivo de configuração do TCP. Caso este conteúdo seja inválido, utiliza-se o TCP Reno.
- **Passo 3**: Instanciação dos dados de configuração. Neste passo, são iniciadas todas as variáveis referentes ao TCP escolhido, com os valores lidos durante o **passo 2**. Como exemplo, apresentamos o conteúdo do arquivo de configuração do TCP Reno:
  - **TCPTYPE=RENO**: Nome do TCP.

- **TDUPACKS=3**: Quantidade de ACKs duplicados que constituem o evento de "três ACKS recebidos em duplicata".
- **AIMD_AI=1.0**: Fator para o aumento da janela de congestionamento, relacionado ao MSS.
- **AIMD_MD=0.5**: Fator para o decréscimo da janela de congestionamento.
- **MSG=/tmp/tcp_logfile.txt**: Esta é uma definição opcional, na qual é informado o nome do arquivo onde serão gravadas as mensagens de exceções na execução do TCP. Como exemplo de exceção, temos o que ocorre nos passos 1 e 2, descritos anteriormente, quando não é possível configurar a variação do TCP desejada e o TCP Reno é utilizado. Neste caso, através deste arquivo, o usuário pode saber que a variação do TCP configurada não é a que, inicialmente, foi definida.

O relacionamento entre o protocolo TCP e as APIs *socket*, representado por uma seta de dupla direção na Figura 9, é composto por um conjunto de funções que mapeiam as funcionalidades disponibilizadas nas APIs *socket* para as funções, que as implementam, do protocolo TCP. A seguir, temos a lista das funções da camada TCP relacionadas às APIs *socket*, desenvolvidas neste trabalho:

so_tcp_listen: Rotina de *listen* do protocolo TCP.

so_tcp_conn: Efetua a conexão.

so_tcp_recv: Executa a recepção dos pacotes.

so_tcp_send: Envia pacotes no TCP.

so_tcp_qlen: Retorna o tamanho da fila do protocolo TCP tanto para envio quanto para recepção.

so_tcp_kick: Força a retransmissão.

so_tcp_shut: Fecha um *socket* de três formas distintas.

so_tcp_close: Fecha um *socket*, de forma normal, limpando suas estruturas para reutilização.

autobind: Liga o endereço e porta local ao *socket*. Se não for chamado de forma explícita, antes de um *connect* ou *listen* através da função *bind* das APIs *socket*, será feito de forma automática quando algum comando necessitar desta ligação. O *bind* foi implementado desta forma para seguir as implementações Linux e MS Windows.

A implementação do TCP, deste trabalho, também contempla as tarefas de multiplexação e demultiplexação, *checksum* do segmento TCP e gerência das filas de recepção e transmissão, de acordo com as RFCs que definem estas tarefas [25, 26, 27, 28, 29, 30, 31, 32].

O protocolo UDP realiza, basicamente, as tarefas de multiplexação e demultiplexação, ou seja, recebem as mensagens da aplicação, através das APIs *socket*, que são encapsuladas em segmentos UDP e repassadas ao protocolo IP, e no sentido contrário, recebem os segmentos UDP, extraem o cabeçalho do segmento e encaminham a mensagem ao *socket* apropriado, como apresentado na Figura 10. Além destas funções, o UDP também calcula e verifica o *checksum* do segmento.

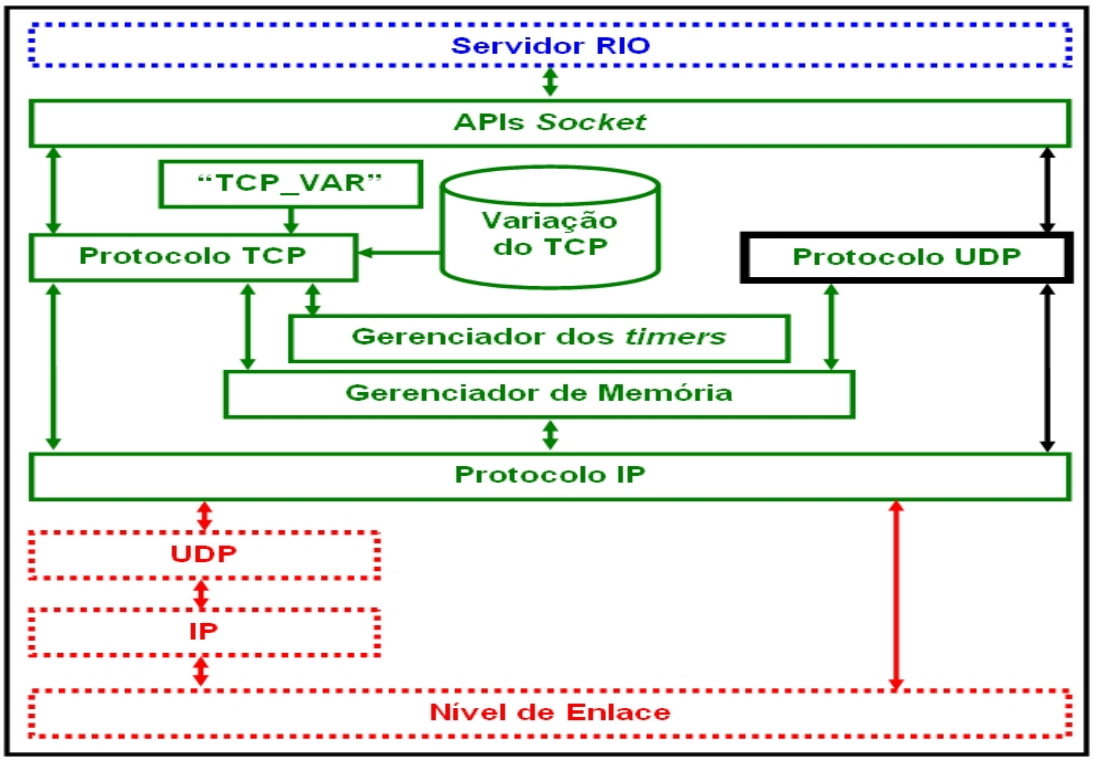

Figura 10: Protocolo UDP

O relacionamento entre o protocolo UDP e as APIs *socket*, representado por uma seta de dupla direção na Figura 10, é composto por um conjunto de funções que mapeiam as funcionalidades disponibilizadas nas APIs *socket* para as funções, que as implementam, do protocolo UDP. A seguir, temos a relação das funções da camada UDP relacionadas às APIs *socket*, desenvolvidas neste trabalho:

so_udp_bind: Liga um conjunto de endereço e porta de comunicação à um *socket*.

so_udp_conn: *Connect* do *socket* UDP. Apenas faz um *bind* se este ainda não tiver sido feito.

so_udp_recv: Efetua a recepção de pacotes.

so_udp_send: Responsável pelo envio de pacotes UDP.

so_udp_qlen: Retorna o tamanho da fila do *socket* tanto para envio quanto para recepção de pacotes.

so_udp_shut: Fecha um *socket* de três formas distintas.

so_udp_close: Fecha um *socket*, limpando suas estruturas para a reutilização.

3.3 Camada de rede

Nesta seção, como exposto na Figura 11, apresentaremos a implementação do protocolo IP, bem como a abordagem, deste trabalho, para o envio e recepção de datagramas.

Como vimos, na camada de rede foi implementado o protocolo IP. Este foi baseado nas RFCs 791 [33], 894 [34] e 1191 [35], que o definem.

Do protocolo IP, implementamos as seguintes funções:

- O encapsulamento dos segmentos, TPC ou UDP, em datagramas IP, incluindo a rotina de *checksum*;
- A extração dos segmentos, TPC ou UDP, dos datagramas IP;
- Interface com a camada de transporte, na recepção e no encaminhamento dos segmentos;
- A comunicação com a camada inferior da pilha de protocolos, no encaminhamento e na recepção dos datagramas; e

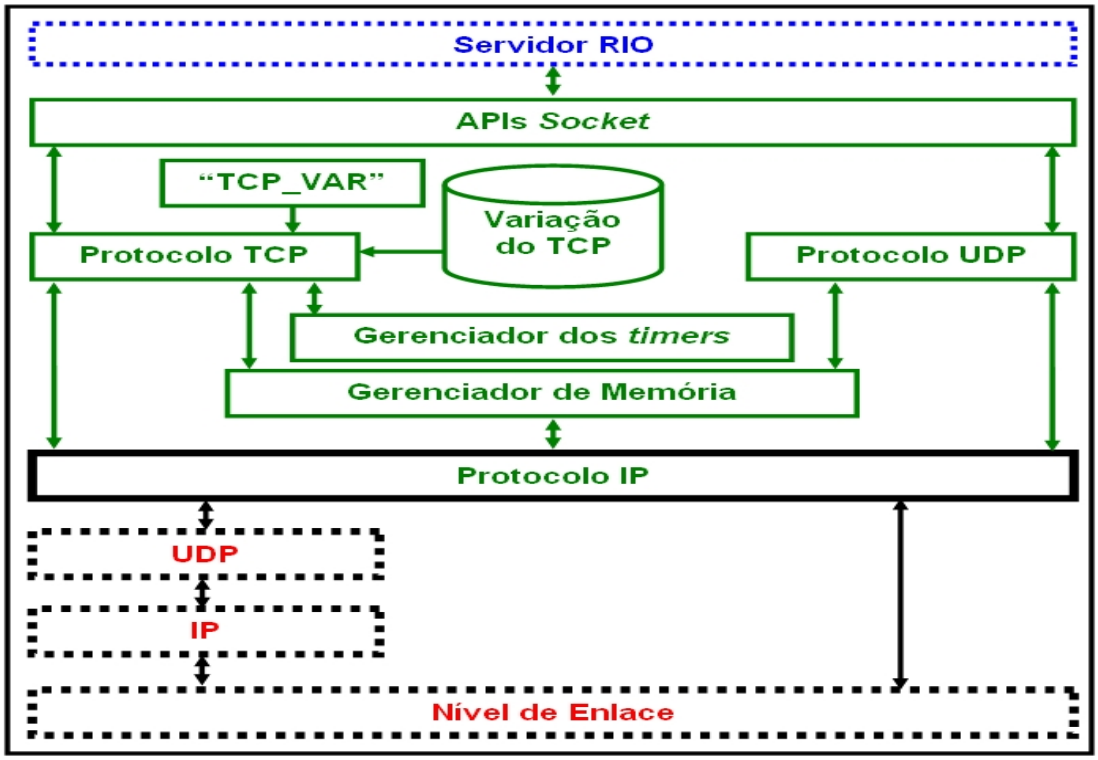

Figura 11: Protocolo IP

- Devido ao funcionamento desta biblioteca estar restrito aos *hosts*, não foram implementados os algoritmos de roteamento do protocolo IP, mas a rotina que realizaria esta tarefa foi construída e as chamadas à ela efetuadas, com o objetivo de uma futura evolução da biblioteca implementada, neste trabalho.

Outro aspecto relevante da camada de rede, é que o envio ou recepção dos datagramas pode ser feito de duas formas, como apresentado na Figura 11, descritas a seguir:

- Pelo próprio IP implementado nesta biblioteca, assumindo o papel do protocolo IP do *host* onde está em execução, e realizando uma interface direta com a camada de enlace; ou
- Através do protocolo UDP disponibilizado pelo sistema operacional.

Esta estratégia, que subverte a pilha de protocolos da Internet, foi definida por alguns motivos práticos, listados a seguir:

- A utilização direta da camada de enlace requer alguns privilégios do sistema operacional e é bloqueada através do *FireWall* nos sistemas operacionais, nos quais esta

implementação é executada. Desta forma, a configuração de segurança de comunicação dos servidores, onde é executado o Servidor RIO, ficaria prejudicada, pois atributos de segurança teriam que ser liberados para os usuários que executassem a biblioteca implementada neste trabalho. Com a opção de utilizar o UPD, como camada de enlace, a biblioteca tornou-se mais flexível à utilização, de forma a atender plenamente as necessidades do Servidor RIO, sem que para isto tivéssemos que alterar aspectos de segurança de acesso no *FireWall* do sistema operacional ou atribuir privilégios aos usuários do sistema;

- Com a utilização do protocolo UDP não necessitamos de um número grande de *sockets* para comunicação, estes sendo em pequeno número e conhecidos, o que facilita e reduz os riscos na configuração da segurança na comunicação; e
- A interface direta com a camada de enlace pressupõe a construção de drives específicos para cada uma das placas a serem utilizadas, o que reduz a portabilidade da biblioteca implementada, quando utilizamos esta estratégia.

A seguir, temos uma lista das funções do IP implementadas:

ip_send: Responsável pelo encapsulamento e envio dos datagramas.

ip_recv: Responsável pela recepção, validação e encaminhamento, quando for o caso, para a camada superior da pilha de protocolos.

ip_route: O roteamento não foi implementado aqui mas algumas funções desta rotina permaneceram apenas para o encaminhamento correto dos datagramas às outras camadas da pilha de protocolos. Outro motivo, para a sua construção foi para que no futuro seja facilitada a implementação do roteamento IP.

net_route: Encaminha o datagrama IP para a interface de saída apropriada.

A Figura 12, ilustra o esquema de funcionamento do protocolo IP implementado, podemos observar que as funções "ip_send" e "ip_recv" são as únicas responsáveis pela comunicação entre a camada de transporte e de rede.

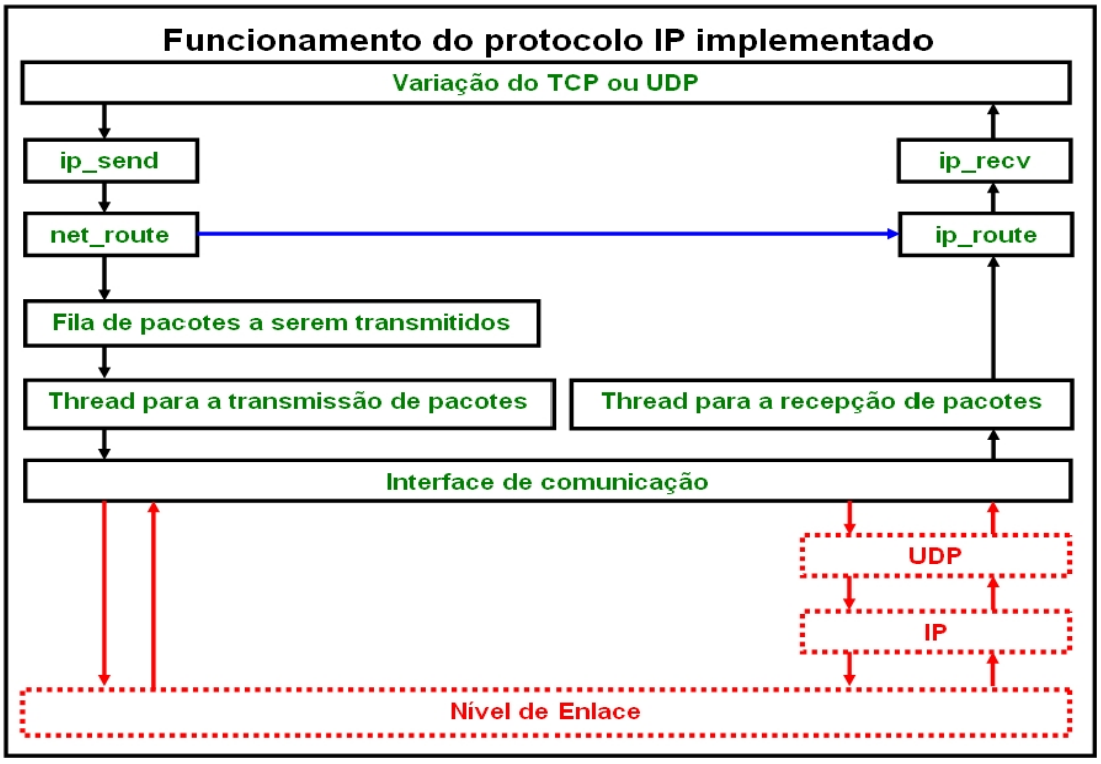

Figura 12: Protocolo IP

No exemplo, apresentado na Figura 12, utilizamos um *host* que contém apenas uma interface de comunicação, ou seja, um único endereço IP, mas esta biblioteca contempla o caso de mais de uma interface de comunicação por *host*. Neste caso, teremos uma *thread* de recepção de pacotes e uma *thread* de transmissão de pacotes, com sua própria fila de envio, para cada uma das interfaces de comunicação do *host*. Desta forma, a função "net_route" é responsável pelo encaminhamento do datagrama para a fila de transmissão de pacotes correta, de acordo com seu endereço IP. Um caso especial, no encaminhamento para a transmissão de datagramas, é quando o endereço de origem é igual ao endereço de destino, este é conhecido por *loopback*, o procedimento para atender a esta situação está destacado em azul na Figura 12, ou seja, o datagrama é diretamente encaminhado da função "net_route" para a função "ip_route".

4 Experimentos

O processo de avaliação, para determinar qual dos TCPs examinados é o que melhor atende a tarefa de dar o suporte necessário ao armazenamento de objetos no Servidor RIO, foi definido com base nos artigos, das variações do TCP, que efetuam avaliações e comparações de performance [17, 20, 36, 5, 7, 30]. Desta forma, visando a descrição, de forma clara, de todos os elementos que compõem a avaliação realizada, decidimos por subdividi-los nos seguintes ítens:

- **Ambiente de execução**: Definição dos equipamentos, sistemas operacionais e recursos utilizados para os experimentos.
- **Descrição dos experimentos**: Expõe a condução dos experimentos, através do programa construído para tal, definição dos dados capturados e dos critérios utilizados na avaliação.
- **Resultados obtidos**: Apresenta os resultados dos experimentos, para cada um dos TCPs, e as conclusões finais da avaliação.

4.1 Ambiente de execução

Neste ponto, vamos definir o ambiente, utilizado nos experimentos, através dos equipamentos, dos sistemas operacionais e das condições de utilização dos mesmos, estes dados estão listados a seguir:

- **Equipamentos**: Computadores Intel mono processados de 3.2 GHz, com 1 Gbyte de memória principal, 80 Gbytes de disco SATA e placa *Ethernet* 10/100 Mbits. Interconectados através de um *switch* 10/100 Mbits.
- **Sistemas operacionais**: Mandriva Linux 10.2 e o MS Windows XP Professional Edition.

- **Utilização dos recursos**: Os experimentos foram realizados sob situações controladas. Esta idéia, de "controlar" o ambiente de avaliação, é justa, no que tange a observação do comportamento dos fluxos TCP sob os exatos mesmos parâmetros. Para isto, inicialmente, devemos definir o que significa "controlar o ambiente de avaliação", segundo [10, 5, 9, 13, 37] isto significa que os recursos dos computadores e da rede utilizados devem obedecer as seguintes condições:

  - Os computadores devem estar executando, durante o experimento, apenas o programa responsável pela transmissão dos dados e o programa de coleta dos dados para posterior análise, desde que estas tarefas não sejam efetuadas pelo mesmo programa. Além destes, apenas as tarefas que o sistema operacional ativa, durante sua inicialização, podem estar em execução.

  - No caso dos recursos de rede, apenas os participantes do experimento deverão trafegar dados.

4.2 Descrição dos experimentos

As avaliações foram efetuadas através de um programa construído especificamente para este fim, utilizando a biblioteca implementada neste trabalho, com as variações do TCP a serem analisadas. No programa, que executa os experimentos, estão contidas: (i) uma parte **"cliente"**; e (ii) uma parte **"servidora"**. Cada uma delas foi executada em uma máquina diferente, sendo que ambas estão interconectadas através da rede definida na seção 4.1. A parte servidora aguarda a conexão de um cliente, este requisita um arquivo, que lhe é transmitido, e quando o arquivo for completamente recebido e gravado, o experimento é considerado encerrado. Durante a sua execução, os dados de performance são capturados e gravados em um arquivo, para que estes possam ser posteriormente tabulados. A Figura 13 apresenta uma visão geral da execução do experimento:

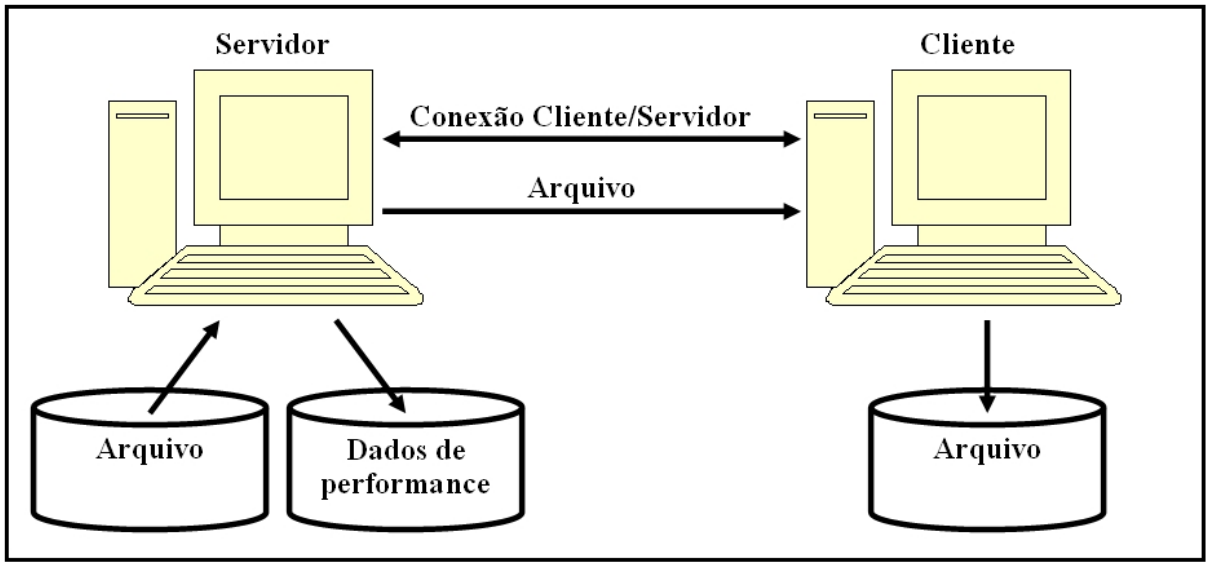

Figura 13: Visão geral da execução do experimento

Dos textos lidos, como referência para a avaliação de performance [10, 5, 9, 13, 37], foram selecionados os parâmetros de performance e qualidade presentes em todos. Estes parâmetros, são calculados ou, também, apenas resultados diretos dos dados capturados durante a execução dos experimentos, que são:

- **Tempo total do experimento**: é o período de tempo medido desde a chamada ao programa do experimento até a sua conclusão. Esta medida é expressa em segundos e com uma precisão de mili-segundos. No restante do texto, quando nos referirmos a "tempo" utilizamos, sempre, estas mesmas medida e precisão; e
- **Quantidade de *bytes* enviados à cada segundo durante todo o experimento**: é a taxa de envio de *bytes* que é capturada, segundo a segundo, durante a transmissão, esta fornece a quantidade de *bytes* enviados do segundo anterior até o segundo corrente, a qual chamaremos de "ponto capturado". Portanto, os pontos capturados são expressos em *bytes* por segundo. Antes dos pontos capturados serem utilizados, nos cálculos dos parâmetros de avaliação dos TCP, estes são convertidos para MBits por segundo, com uma precisão de três casas decimais, através da Fórmula 4.1 aplicada a cada um dos pontos capturados pelo programa do experimento. Cada um dos dados, resultantes da aplicação da Fórmula 4.1, na seqüência do texto será chamado de "ponto medido".

$$Ponto\_Medido = \frac{Ponto\_Capturado \times 8 \times 100}{100.000.000} \tag{4.1}$$

Onde:
- **Ponto_Capturado**, é o valor de cada um dos dados capturados;
- **8**, para a conversão de *bytes* para *bits*;
- **100**, para a obtenção do percentual;
- **100.000.000**, $100 MBits$ por segundo.

A seguir, temos as descrições dos parâmetros utilizados como medida para comparação entre os TCPs, estes utilizam os dados colhidos durante os experimentos, aos quais nos referimos anteriormente:

- **Desvio padrão da taxa de envio de pacotes (DVP - para a tabulação dos resultados)**: Este dado fornece a estabilidade da transmissão, ou seja, quanto mais estável for o fluxo TCP menor será a flutuação da taxa de envio de *bytes*. Para o cálculo, desta medida de dispersão, utilizamos o desvio padrão, que é uma medida do grau de dispersão dos valores em relação ao valor médio (a média), sendo assim o que mais se aproximou do parâmetro a ser colhido. O desvio padrão é obtido pela seguinte fórmula, que é empregada nos pontos medidos que contém a quantidade de *Mbits* transmitidos à cada segundo durante toda a transmissão do arquivo do experimento:

$$\mathbf{DVP} = \sqrt{\frac{\sum_{j=1}^{N}(Ponto\_Medido_j - M)^2}{N}} \tag{4.2}$$

Onde:
- **N**, é o número total de pontos medidos;
- **M**, é a média aritmética dos pontos medidos;
- **Ponto_Medido**, são os pontos medidos, indexados por j.

- **Ocupação da banda passante (BP - para a tabulação dos resultados)**: Este parâmetro, expresso em *MBits* por segundo, é a média de ocupação da banda passante durante toda a transmissão, como o valor máximo a ser alcançado é de 100 *MBits* por segundo podemos utilizar este dado como o percentual médio de ocupação da banda passante. O objetivo desta medição é demonstrar o quão efetivo é o TCP, em avaliação, quanto à ocupação dos recursos de comunicação disponíveis. Este dado é calculado de acordo com a seguinte fórmula:

$$\mathbf{BP} = \frac{\sum_{j=1}^{N} Ponto\_Medido_j}{N} \tag{4.3}$$

$$\text{Onde:} \begin{cases} \textbf{N}, & \text{é o número total de pontos de medição;} \\ \textbf{Ponto\_Medido}, & \text{são os pontos medidos, indexados por j.} \end{cases}$$

- **Tempo total de transmissão do arquivo (Tempo - para a tabulação dos resultados)**: Este dado fornece a performance obtida na tarefa executada. A obtenção deste dado é direta, pois é capturado durante o experimento. Como definido, anteriormente, este dado está expresso em segundos.

Visando um critério único e mensurável, para a justiça na escolha do TCP mais adequado, que leve em conta a estabilidade, ocupação dos recursos e performance, utilizamos um sistema de pontuação para os dados capturados [18]. A quantificação dos dados foi igual para cada um dos parâmetros definidos, por considerarmos que todos tem a mesma relevância, ou seja, o mesmo "peso" na avaliação. Desta forma, foram atribuídas pontuações, que são números reais no intervalo fechado entre zero e um, para cada um dos parâmetros colhidos nos experimentos. Isto representa, que para os parâmetros descritos, a pontuação dos resultados é efetuada da forma contida na Tabela 12, e os cálculos efetuados para tal serão apresentados na seqüência do texto.

Tabela 12: Esquema para a pontuação dos experimentos

Parâmetro	**Cálculos**	**Pontuação**
Desvio padrão da taxa de envio de pacotes (estabilidade)	Cálculo do desvio padrão da taxa média de envio de pacotes (segundo a segundo).	1 - para o menor desvio. 0 - para o maior desvio. Proporcional - para os demais.
Ocupação da banda passante (ocupação dos recursos disponíveis)	Cálculo efetuado através da Fórmula 4.3	1 - para o maior percentual ocupado. 0 - para o menor percentual ocupado. Proporcional - para os demais.
Tempo total de transmissão do arquivo (performance)	Resultado direto	1 - para o menor tempo de transmissão. 0 - para o maior tempo de transmissão. Proporcional - para os demais.

Na apresentação dos cálculos efetuados na pontuação dos experimentos, para cada um dos parâmetros definidos anteriormente, utilizaremos a Tabela 13, nela estão listados alguns exemplos dos valores obtidos.

Utilizando os dados da Tabela 13, são efetuados os seguintes cálculos na obtenção dos pontos, para cada um dos parâmetros definidos:

Tabela 13: Exemplo dos dados capturados através de um experimento.

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	18,594	18,141	17,360	17,187	16,188	17,844	15,719
BP	67,317	68,491	71,348	73,740	77,204	69,811	79,076
DVP	5,196	6,494	18,860	3,158	6,932	6,237	6,410

- **Tempo total de transmissão do arquivo (Tempo)**: Inicialmente encontramos os valores máximo (**MÁX**) e mínimo (**MÍN**), que, no exemplo são 18,594 e 15,719 respectivamente. Utilizando os valores **MÁX** e **MÍN**, aplicamos a seguinte fórmula:

$$PontosTempo = 1 - \frac{\textbf{Tempo}(i) - \textbf{MÍN}}{\textbf{MÁX} - \textbf{MÍN}} \tag{4.4}$$

  Onde: **Tempo**(i) - é o tempo capturado na execução do experimento, para cada um dos TCPs.

- **Ocupação da banda passante (BP)**: Antes dos cálculos devemos encontrar os valores máximo (**MÁX**) e mínimo (**MÍN**), que, no exemplo são 79,076 e 67,317 respectivamente. Utilizando os valores **MÁX** e **MÍN**, aplicamos a seguinte fórmula:

$$PontosBP = \frac{\textbf{BP}(i) - \textbf{MÍN}}{\textbf{MÁX} - \textbf{MÍN}} \tag{4.5}$$

  Onde: **BP**(i) - é o percentual de ocupação média da banda passante na execução do experimento, para cada um dos TCPs, resultado da Fórmula 4.1.

- **Desvio padrão da taxa de envio de pacotes (DVP)**: Primeiramente, definimos os valores máximo (**MÁX**) e mínimo (**MÍN**), que, no exemplo são 18,860 e 3,158 respectivamente. Utilizando os valores **MÁX** e **MÍN**, aplicamos a seguinte fórmula:

$$PontosDVP = 1 - \frac{\textbf{DVP}(i) - \textbf{MÍN}}{\textbf{MÁX} - \textbf{MÍN}} \tag{4.6}$$

  Onde: **DVP**(i) - é o desvio padrão da taxa de envio de pacotes obtida na execução do experimento, para cada um dos TCPs, resultado da Fórmula 4.2.

Como resultado dos cálculos de pontuação, construímos a Tabela 14, na qual destacamos, em negrito, as melhores pontuações. A linha "**Total**", da Tabela 14, é a soma dos pontos de cada uma das variações do TCP, e através dela avaliamos o resultado de todo o experimento.

Tabela 14: Exemplo da pontuação obtida através de um experimento.

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,158	0,429	0,489	0,837	0,261	**1,000**
BP	0,000	0,100	0,343	0,546	0,841	0,212	**1,000**
DVP	0,870	0,788	0,000	**1,000**	0,760	0,804	0,793
Total	0,870	1,045	0,772	2,036	2,437	1,277	**2,793**

Definidos os cálculos dos parâmetros e das pontuações para um único experimento, agora vamos apresentar a totalização dos resultados, calculados através das seguintes fórmulas:

- **Tempo**: define a pontuação do parâmetro "tempo", para cada um dos TCPs, considerando todos os experimentos realizados.

$$TotalPontosTempo(i) = \frac{\sum_{j=1}^{N} PontosTempo(j)}{N} \tag{4.7}$$

Onde:
- **i**, índice para as variações do TCP;
- **j**, índice para os experimentos efetuados;
- **N**, quantidade de experimentos efetuados;
- **PontosTempo**, pontuação obtida através da Fórmula 4.4.

- **BP**: calcula a pontuação do parâmetro "banda passante", para cada um dos TCPs, considerando todos os experimentos realizados.

$$TotalPontosBP(i) = \frac{\sum_{j=1}^{N} PontosBP(j)}{N} \tag{4.8}$$

Onde:
- **i**, índice para as variações do TCP;
- **j**, índice para os experimentos efetuados;
- **N**, quantidade de experimentos efetuados;
- **PontosBP**, pontuação obtida através da Fórmula 4.5.

- **DVP**: efetua o cálculo da pontuação do parâmetro "desvio padrão", para cada um dos TCPs, considerando todos os experimentos realizados.

$$TotalPontosDVP(i) = \frac{\sum_{j=1}^{N} PontosDVP(j)}{N} \tag{4.9}$$

Onde:
- **i**, índice para as variações do TCP;
- **j**, índice para os experimentos efetuados;
- **N**, quantidade de experimentos efetuados;
- **PontosDVP**, pontuação obtida através da Fórmula 4.6.

- **Total**: calcula a pontuação "total", para cada um dos TCPs, considerando todos os experimentos realizados.

$$TotalPontos(i) = \frac{\sum_{j=1}^{N} \frac{PontosTempo(j)+PontosBP(j)+PontosDVP(j)}{3}}{N} \tag{4.10}$$

Onde:
- **i**, índice para as variações do TCP;
- **j**, índice para os experimentos efetuados;
- **N**, quantidade de experimentos efetuados.

O TCP que alcançar a maior pontuação total, obtida através da Fórmula 4.10 será o escolhido.

Agora, que já definimos o programa utilizado, os dados capturados e o critério de avaliação, podemos apresentar os elementos que compõem os experimentos efetuados. Cada um dos experimentos utiliza variações entre os três elementos descritos a seguir.

4.2.1 Elemento 1: Tamanho do arquivo transmitido

Nos experimentos, foram utilizados arquivos de 150 e 500 MBytes, para a transmissão. Visto que, nos artigos estudados, não estão definidos precisamente os tamanhos dos arquivos a serem transmitidos para a coleta dos dados, optamos por selecionar os dois tamanhos de arquivos, acima citados, pelos seguintes motivos:

- **150 MBytes**: As medições dos artigos [10, 5, 18] utilizam uma comunicação de curto tempo, para avaliar a velocidade na qual o TCP converge para a estabilidade na taxa de envio de pacotes. Através dos tempos medidos e das taxas de transmissão alcançadas foi possível estimar entre 100 e 200 MBytes o tamanho destes arquivos, desta forma, optamos por um valor intermediário.

- **500 MBytes**: Nos artigos [10, 5, 13, 37] são utilizadas transmissões de longo curso para verificar a convergência e permanência do TCP nas proximidades do ponto de

equilíbrio da transmissão. Este ponto de equilíbrio é caracterizado pela taxa média de transmissão de pacotes obtida, após a etapa inicial do *slow start*.

Outro motivo, para a escolha destes tamanhos para os arquivos, é que eles cobrem a grande maioria dos objetos multimídia, referentes às aulas armazenadas em vídeo, do Servidor RIO.

4.2.2 Elemento 2: Sistemas Operacionais envolvidos no experimento

Outro aspecto, também importante na avaliação, é o da utilização da biblioteca desenvolvida em diferentes sistemas operacionais. Devido à implementação deste trabalho contemplar os sistemas operacionais UNIX e MS Windows, decidimos por realizar os experimentos nestes ambientes, de forma a termos certeza de que escolhemos a melhor variação do TCP para todas as plataformas nas quais o Servidor RIO é executado, como apresentado na Tabela 15:

Tabela 15: Sistemas operacionais envolvidos nos experimentos

Servidor	**Cliente**
Linux	MS Windows
Linux	Linux
MS Windows	MS Windows

4.2.3 Elemento 3: Perda de pacotes durante o experimento

Um ponto fundamental na avaliação, do protocolo TCP, é a forma de reação aos eventos de perda de pacotes. Neste trabalho, implementamos o protocolo IP, o que nos forneceu o total controle da transmissão, tornando possível interferir no fluxo de comunicação para avaliar este aspecto. A interferência no fluxo, para contemplar esta questão, foi à perda controlada de dez pacotes em *burst* [18], ou seja, um conjunto de pacotes em seqüência foi perdido propositalmente em determinado momento da comunicação. O momento, definido pela quantidade de *bytes* transmitidos até o ponto que marca o início do evento, e a quantidade de pacotes perdidos foram idênticos para todos os experimentos dos TCPs avaliados. Portanto, temos dois tipos de experimentos quanto aos eventos de perda de pacotes, como apresentados na Tabela 16.

Tabela 16: Variações dos experimentos quanto a perda de pacotes

Abreviação	**Variação**	**Descrição**
Sem Perdas	Sem perda forçada de pacotes.	Neste experimento, a perda de pacotes pode existir apenas devido às características do TCP em avaliação. Alguns dos TCPs sempre forçam o crescimento da janela de congestionamento, o que leva à perdas eventuais de pacotes.
Com Perdas	Com perda forçada de pacotes.	Nesta variação dos experimentos utilizamos a perda controlada de dez pacotes [18] em *burst*, para avaliarmos a forma de reação do TCP dado um evento de perda. Neste caso, também podem ocorrer outras perdas, não forçadas devido ao comportamento da variação do TCP em avaliação.

4.3 Resultados dos experimentos

Definimos, até este ponto, do que consistem os experimentos efetuados e quais foram os critérios de avaliação, portanto podemos agora listar, na Tabela 17, toda a bateria de experimentos realizados. Devemos notar, que para cada uma das linhas da Tabela 17 temos um experimento que foi realizado sobre cada um dos sete do TCPs desenvolvidos neste trabalho, desta maneira foram realizados um total de 84 experimentos.

Tabela 17: Listagem de todas as variações dos experimentos efetuados

Experimento	**Servidor - Cliente**	**Arquivo**	**Evento de Perda**
1	Linux - Linux	150 MB	Sem Perdas
2	Linux - Linux	150 MB	Com Perdas
3	Linux - Linux	500 MB	Sem Perdas
4	Linux - Linux	500 MB	Com Perdas
5	Linux - MS Windows	150 MB	Sem Perdas
6	Linux - MS Windows	150 MB	Com Perdas
7	Linux - MS Windows	500 MB	Sem Perdas
8	Linux - MS Windows	500 MB	Com Perdas
9	MS Windows - MS Windows	150 MB	Sem Perdas
10	MS Windows - MS Windows	150 MB	Com Perdas
11	MS Windows - MS Windows	500 MB	Sem Perdas
12	MS Windows - MS Windows	500 MB	Com Perdas

Os resultados obtidos nos experimentos, listados na Tabela 17, estão expostos de 4.3.1 à 4.3.12, sendo que, cada um deles contém:

- Um gráfico: que apresenta os pontos medidos e que fornece a taxa de transmissão, capturada segundo à segundo;
- A tabela dos resultados obtidos[1], para:
  - **Tempo**: Tempo total da transmissão;
  - **BP**: Percentual médio de ocupação da banda passante; e
  - **DVP**: Desvio padrão da taxa de transmissão de pacotes.
- Na segunda tabela, temos a pontuação obtida no experimento[2], por cada um dos TCPs.

4.3.1 Experimento 1: Linux - Linux, 150 MB e Sem Perdas

Neste experimento, apresentado na Figura 14, podemos destacar a rápida convergência do FAST TCP para o seu ponto de equilíbrio na taxa de transmissão, e a manutenção deste durante toda a execução do experimento. Como resultado, o FAST TCP foi o que concluiu a tarefa mais rapidamente e que melhor ocupou a banda passante disponível, como apresentado na Tabela 18. Um ponto interessante é o *slow start* do HSTCP, como este obedece à uma função que retorna valores fixos para o seu crescimento, de acordo com o tamanho da janela de transmissão corrente, isto resultou em uma convergência bastante lenta para o seu ponto de equilíbrio, o que está refletido na Tabela 18, na qual obteve o pior resultado no parâmetro "desvio padrão".

[1] Os melhores resultados foram destacados em negrito.
[2] As melhores pontuações obtidas estão destacadas em negrito.

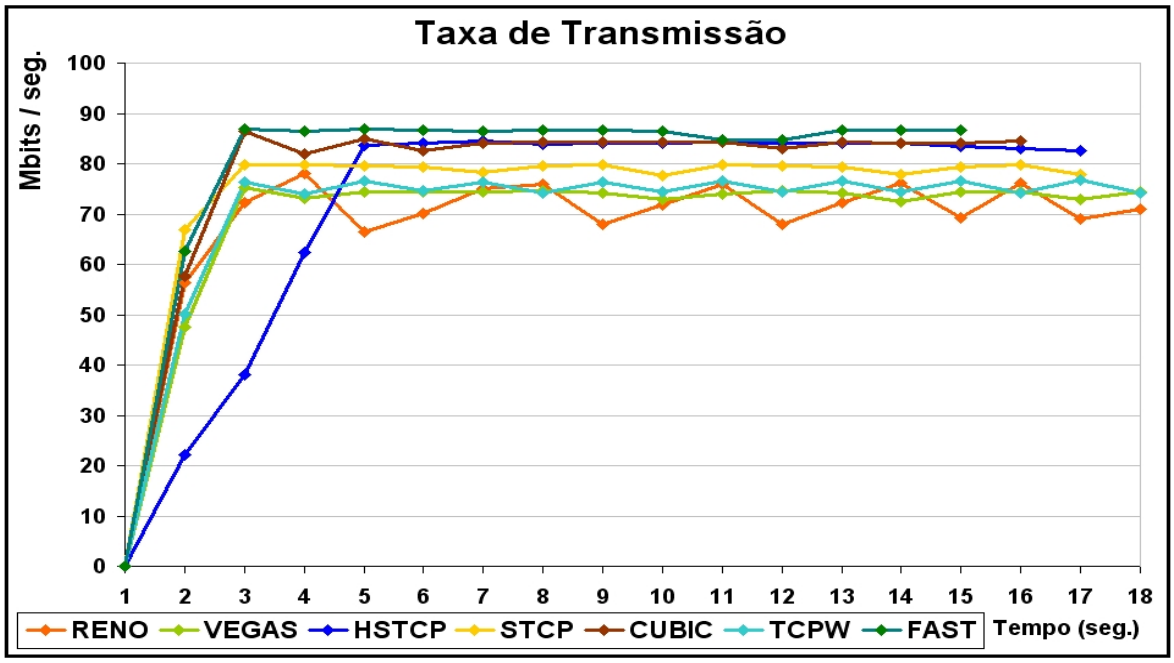

Figura 14: Experimento 1

Tabela 18: Dados capturados no experimento 1

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	18,594	18,141	17,360	17,187	16,188	17,844	**15,719**
BP	67,317	68,491	71,348	73,740	77,204	69,811	**79,076**
DVP	5,196	6,494	18,860	**3,158**	6,932	6,237	6,410

Tabela 19: Pontuação no experimento 1

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,158	0,429	0,489	0,837	0,261	**1,000**
BP	0,000	0,100	0,343	0,546	0,841	0,212	**1,000**
DVP	0,870	0,788	0,000	**1,000**	0,760	0,804	0,793
Total	0,870	1,045	0,772	2,036	2,437	1,277	**2,793**

4.3.2 Experimento 2: Linux - Linux, 150 MB e Com Perdas

Na Figura 15, notamos a rápida convergência do BIC-CUBIC TCP ao seu ponto de equilíbrio, o que o levou ao melhor resultado neste experimento, como apresentado na Tabela 21, apesar de seu ponto de equilíbrio ser inferior ao alcançado pelo FAST TCP. Ainda na Figura 15, podemos observar os efeitos indesejados do mecanismo de controle de congestionamento do TCP Reno, que o faz perder muito espaço da banda passante disponível, na Tabela 20 podemos comparar seu desempenho em relação aos outros TCPs, quando ocorrem eventos de perda de pacotes. Neste caso, o TCP Reno caiu de 76,162 para 11,450 na ocupação da banda passante.

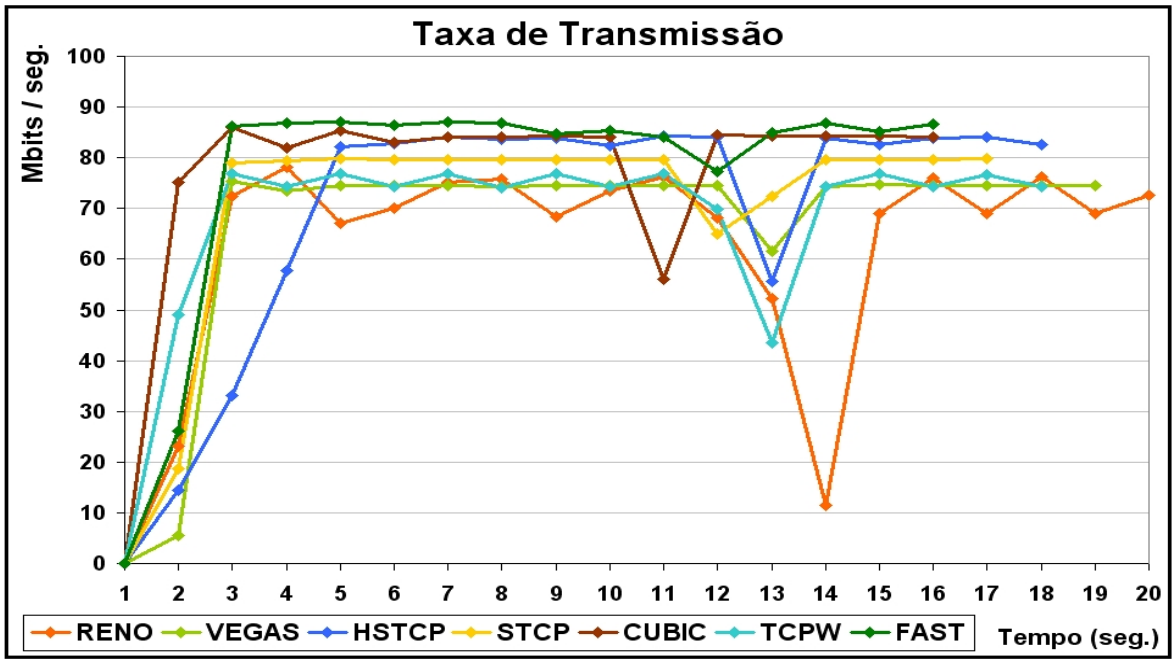

Figura 15: Experimento 2

Tabela 20: Dados capturados no experimento 2

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	19,766	18,235	17,750	17,438	16,500	18,344	**15,891**
BP	62,196	66,232	69,196	70,105	**76,625**	67,798	76,310
DVP	18,000	16,378	20,860	15,390	**7,563**	9,786	15,468

Tabela 21: Pontuação no experimento 2

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,395	0,520	0,601	0,843	0,367	**1,000**
BP	0,000	0,280	0,485	0,548	**1,000**	0,388	0,978
DVP	0,215	0,337	0,000	0,411	**1,000**	0,833	0,406
Total	0,215	1,012	1,005	1,560	**2,843**	1,588	2,384

4.3.3 Experimento 3: Linux - Linux, 500 MB e Sem Perdas

No terceiro experimento, o primeiro a ser realizado com o arquivo de 500*MBytes*, foi possível observar (Figura 16) a grande vantagem de se manter o fluxo em um ponto de equilíbrio mais elevado. Isto pode ser observado no tempo de execução da tarefa, por exemplo, comparando os resultados obtidos pelo FAST TCP ou BIC-CUBIC TCP em relação ao TCP Reno, na Tabela 20. Outro ponto, interessante, é que para transmissões de longo curso o *slow start* do HSTCP já não é tão prejudicial, o que pode ser observado pela pontuação obtida na ocupação da banda passante, apresentada na Tabela 21.

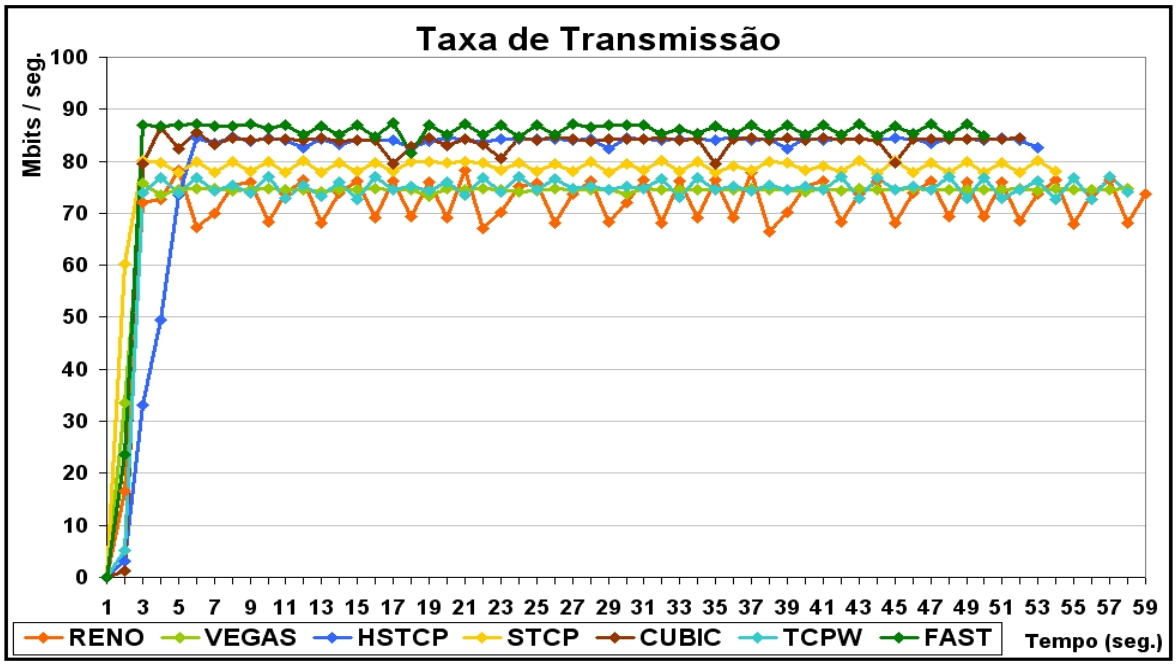

Figura 16: Experimento 3

Tabela 22: Dados capturados no experimento 3

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	59,172	57,750	52,468	54,593	51,454	57,297	**50,032**
BP	70,438	72,471	79,122	77,206	80,597	72,541	**83,185**
DVP	8,199	5,420	13,912	**2,743**	11,659	9,344	9,014

Tabela 23: Pontuação no experimento 3

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,156	0,733	0,501	0,844	0,205	1,000
BP	0,000	0,160	0,681	0,531	0,797	0,165	1,000
DVP	0,511	0,760	0,000	1,000	0,202	0,409	0,439
Total	0,511	1,075	1,415	2,032	1,843	0,779	**2,439**

4.3.4 Experimento 4: Linux - Linux, 500 MB e Com Perdas

O Experimento 4, apresentado na Figura 17, é muito interessante para observar que apesar de não obter a melhor pontuação no desvio padrão, que mede a estabilidade do TCP em relação ao seu ponto de equilíbrio, o FAST TCP foi o que mais ocupou a banda passante e mais rapidamente encerrou a tarefa, apresentado na Tabela 24. Na Figura 17, podemos observar que ele flutua muito próximo ao seu ponto de equilíbrio, isto ocorre porque a função de crescimento e redução da janela de congestionamento do FAST TCP, ao mesmo tempo em que tenta evitar perdas de pacotes, ela procura ocupar ao máximo a banda passante disponível, como pode ser visto na Tabela 25.

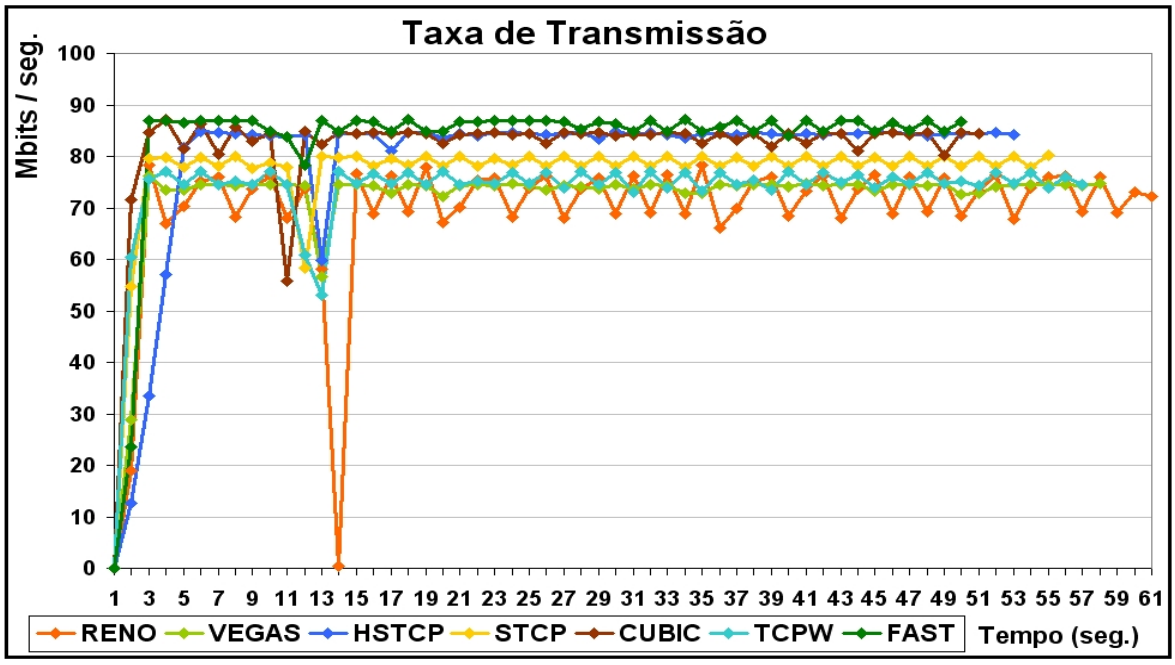

Figura 17: Experimento 4

Tabela 24: Dados capturados no experimento 4

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	60,359	58,265	52,656	54,859	51,765	57,469	**50,156**
BP	69,195	71,791	79,282	76,743	81,459	73,220	**83,012**
DVP	12,171	6,439	12,875	4,380	4,501	**4,198**	9,045

Tabela 25: Pontuação no experimento 4

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,205	0,755	0,539	0,842	0,283	**1,000**
BP	0,000	0,188	0,730	0,546	0,888	0,291	**1,000**
DVP	0,081	0,742	0,000	0,979	0,965	**1,000**	0,441
Total	0,081	1,135	1,485	2,064	**2,695**	1,575	2,441

4.3.5 Experimento 5: Linux - MS Windows, 150 MB e Sem Perdas

Neste experimento, visto na Figura 18, observamos o quão prejudicial para pequenos fluxos é o *slow start* do HSTCP. O ponto de equilíbrio do HSTCP e do BIC-CUBIC, após o "início lento", é muito semelhante, mas os resultados obtidos para o tempo total de transmissão diferem de forma significativa, como podemos observar na Tabela 26. Como conseqüência, a pontuação final do BIC-CUBIC TCP, apresentada na Tabela 27, foi bastante superior.

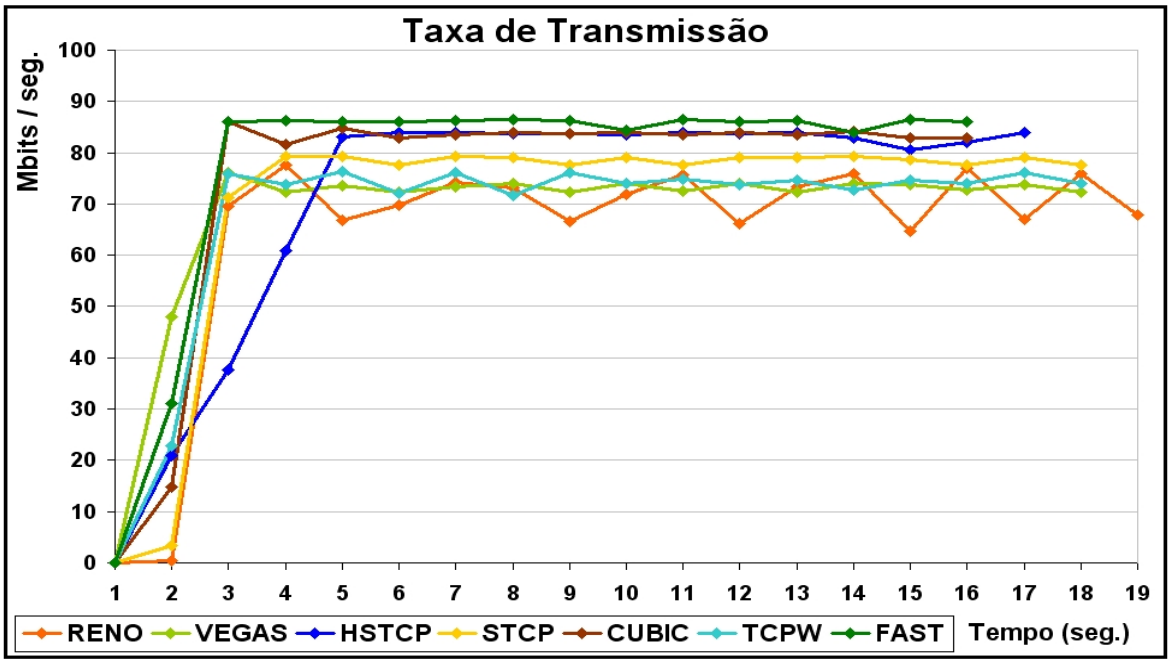

Figura 18: Experimento 5

Tabela 26: Dados capturados no experimento 5

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	18,766	18,359	17,453	17,281	16,281	18,078	**15,797**
BP	63,880	67,865	70,732	69,654	74,147	67,429	**77,178**
DVP	4,309	1,067	12,805	1,995	1,014	1,460	**0,810**

Tabela 27: Pontuação no experimento 5

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,137	0,442	0,500	0,837	0,232	**1,000**
BP	0,000	0,300	0,515	0,434	0,772	0,267	**1,000**
DVP	0,708	0,979	0,000	0,901	0,983	0,946	**1,000**
Total	0,708	1,415	0,958	1,836	2,592	1,444	**3,000**

4.3.6 Experimento 6: Linux - MS Windows, 150 MB e Com Perdas

Aproveitaremos os resultados obtidos no sexto experimento, apresentados na Figura 19, para destacar a estabilidade na taxa de envio de pacotes alcançada pelo TCP Vegas, o que pode ser observado pela sua pontuação na Tabela 29. O TCP Vegas, assim como o FAST TCP, observa a variação do RTT e da taxa de envio de pacotes para tomar decisões quanto ao crescimento ou redução de sua janela de congestionamento. Esta estratégia é muito eficiente na estabilização do fluxo, mas como é aplicada de forma tímida, ou seja, em pequenas porções, tende a convergir para um ponto de equilíbrio não muito acima do TCP Reno, o que reflete nos seus resultados quanto ao tempo total de transmissão, como apresentado na Tabela 28.

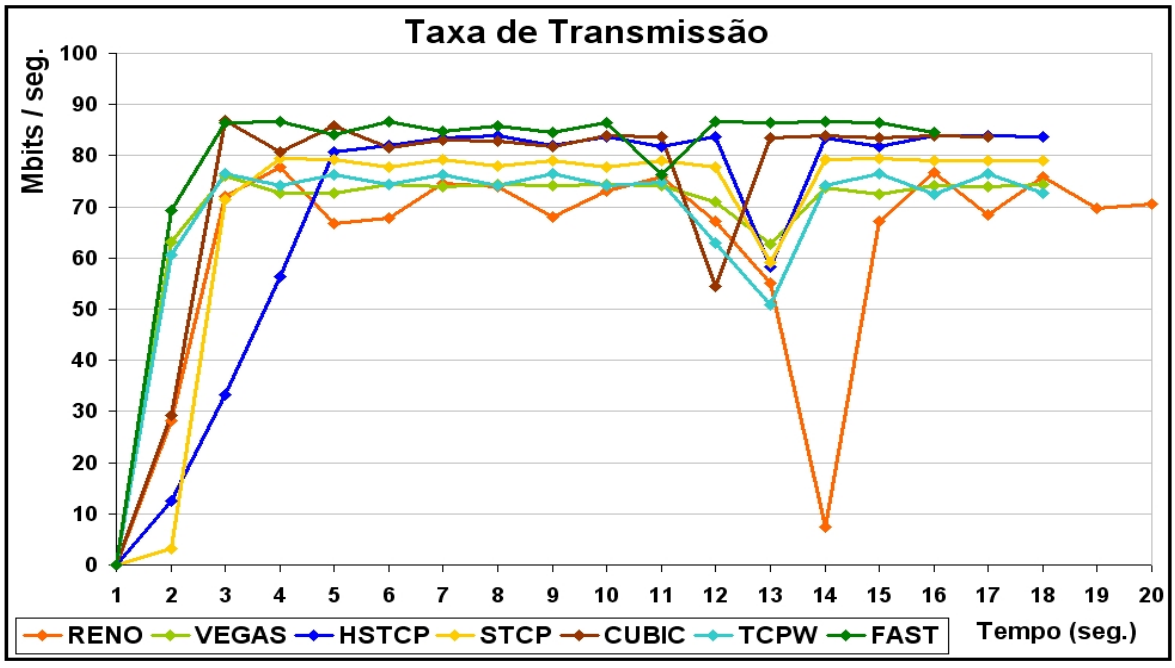

Figura 19: Experimento 6

Tabela 28: Dados capturados no experimento 6

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	19,890	18,407	17,765	17,500	16,672	18,500	**15,922**
BP	61,803	68,413	68,793	68,737	73,680	67,954	**78,814**
DVP	17,772	**3,712**	20,975	18,611	15,024	7,106	4,860

Tabela 29: Pontuação no experimento 6

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,374	0,536	0,602	0,811	0,350	**1,000**
BP	0,000	0,389	0,411	0,408	0,698	0,362	**1,000**
DVP	0,186	**1,000**	0,000	0,137	0,345	0,803	0,933
Total	0,186	1,762	0,946	1,147	1,854	1,515	**2,933**

4.3.7 Experimento 7: Linux - MS Windows, 500 MB e Sem Perdas

No sétimo experimento, apresentado na Figura 20, notamos a estabilidade no fluxo de transmissão do STCP, através da pontuação obtida no parâmetro "desvio padrão" (Tabela 31). A estabilidade do STCP é explicada pelo seu incremento tímido da janela de congestionamento (de 0,01 MSS à cada RTT) e por sua redução de 0,875 quando temos a ocorrência de um evento de perda. Esta estratégia, não garantiu bons resultados no tempo de execução da tarefa, apresentado na Tabela 30, isto se deve ao fato de que quando uma redução na janela de congestionamento ocorre, sua queda não é muito grande, mas a recuperação desta queda é muito lenta, o que tende a estabilizar o fluxo em um patamar não muito elevado.

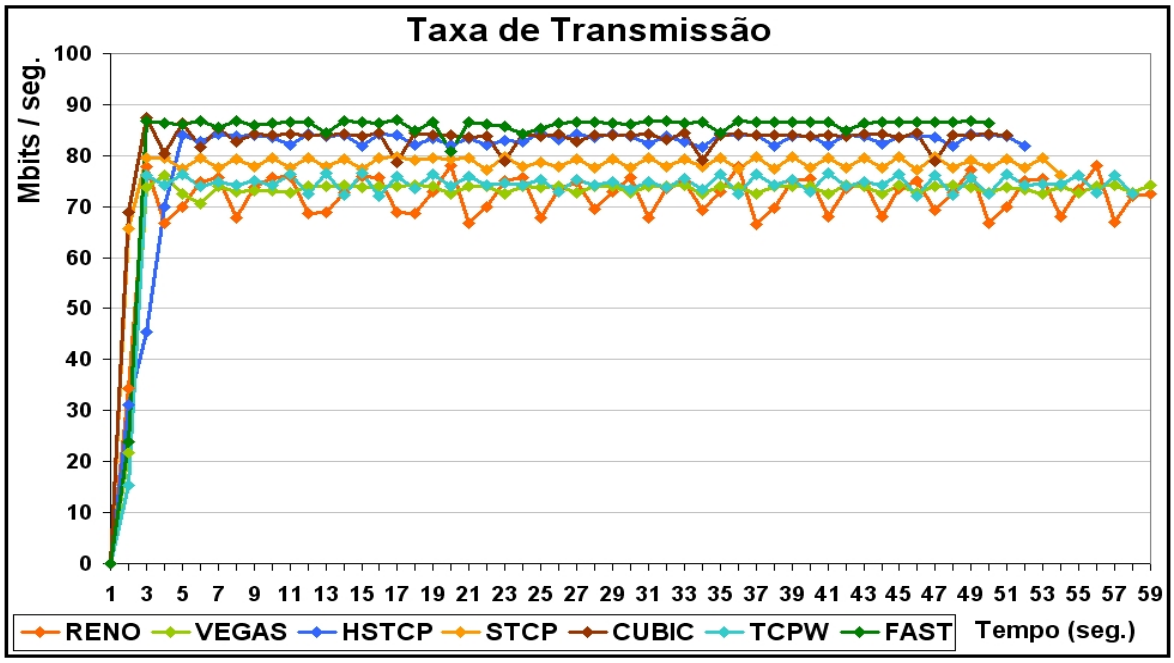

Figura 20: Experimento 7

Tabela 30: Dados capturados no experimento 7

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	59,390	58,407	52,891	54,828	51,610	57,640	**50,047**
BP	70,472	71,454	79,707	76,934	81,601	72,284	**83,173**
DVP	6,065	6,861	9,107	**2,028**	2,663	7,975	8,939

Tabela 31: Pontuação no experimento 7

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,105	0,696	0,488	0,833	0,187	**1,000**
BP	0,000	0,077	0,727	0,509	0,876	0,143	**1,000**
DVP	0,430	0,317	0,000	**1,000**	0,910	0,160	0,024
Total	0,430	0,500	1,423	1,997	**2,619**	0,490	2,024

4.3.8 Experimento 8: Linux - MS Windows, 500 MB e Com Perdas

O Experimento 8, apresentado na Figura 21, é um bom exemplo para observarmos estabilidade no fluxo de transmissão do BIC-CUBIC TCP, através da pontuação obtida no parâmetro "desvio padrão" (Tabela 33), que é, quase sempre, superior ao FAST TCP. A estabilidade do BIC-CUBIC TCP é conferida pela sua função cúbica, que estabiliza a sua taxa de transmissão nas proximidades de seu ponto de equilíbrio. Esta estratégia, não superou o FAST TCP, no tempo de transmissão e na ocupação da banda passante, o que pode ser constatado na Tabela 32, porque como o BIC-CUBIC TCP não observa os "sinais" de congestionamento que a rede oferece, portanto, ele tende a ocasionar um número de eventos de perda de pacotes superior ao FAST TCP, e como conseqüência, acaba por convergir para um ponto de equilíbrio levemente inferior ao alcançado pelo FAST TCP.

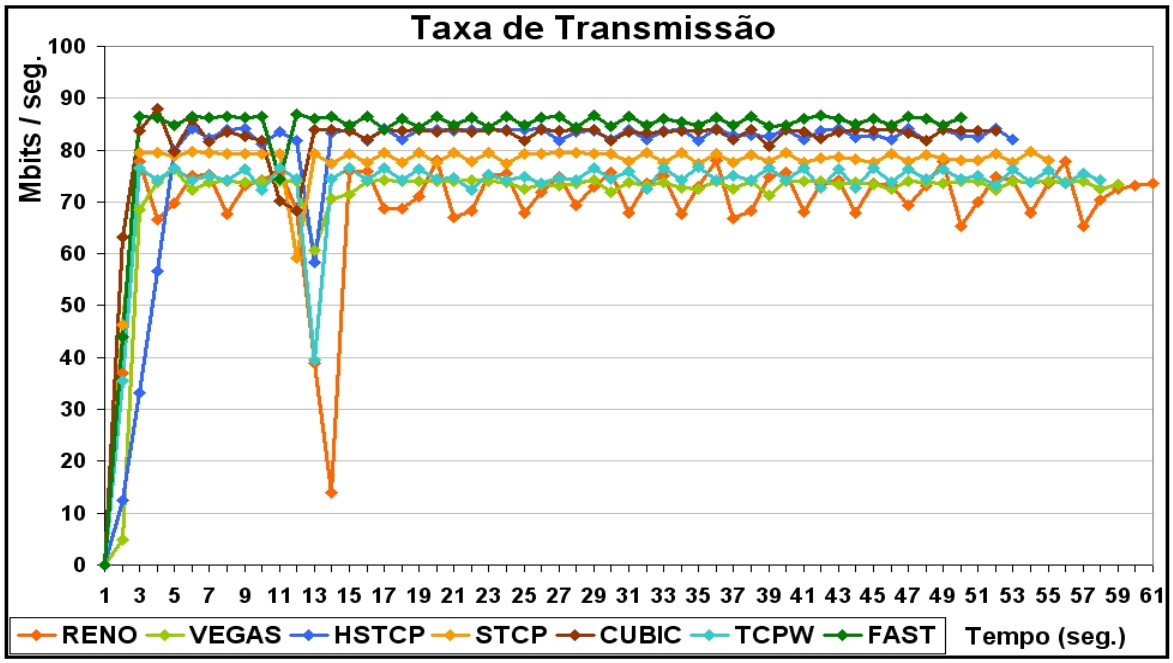

Figura 21: Experimento 8

Tabela 32: Dados capturados no experimento 8

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	60,890	58,688	53,250	55,031	52,062	57,859	**50,422**
BP	68,851	70,827	78,321	76,351	80,888	72,289	**82,968**
DVP	10,237	9,208	12,737	5,175	**4,096**	7,075	6,197

Tabela 33: Pontuação no experimento 8

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,210	0,730	0,560	0,843	0,290	**1,000**
BP	0,000	0,140	0,671	0,531	0,853	0,244	**1,000**
DVP	0,289	0,408	0,000	0,875	**1,000**	0,655	0,757
Total	0,289	0,759	1,401	1,966	2,696	1,188	**2,757**

4.3.9 Experimento 9: MS Windows - MS Windows, 150 MB e Sem Perdas

O nono experimento é um bom exemplo dos resultados alcançados pelo FAST TCP. Neste experimento, o FAST TCP foi vitorioso em todos os parâmetros, como exposto na Tabela 35. Um ponto, que deve ser notado, é o *slow start*, que obteve uma convergência muito eficiente para atingir o ponto de equilíbrio da taxa de transmissão, como podemos observar na Figura 22. Como conseqüência, os dados para o tempo de transmissão e ocupação da banda passante foram muito melhores que os obtidos pelos demais TCPs, estes dados estão na Tabela 34.

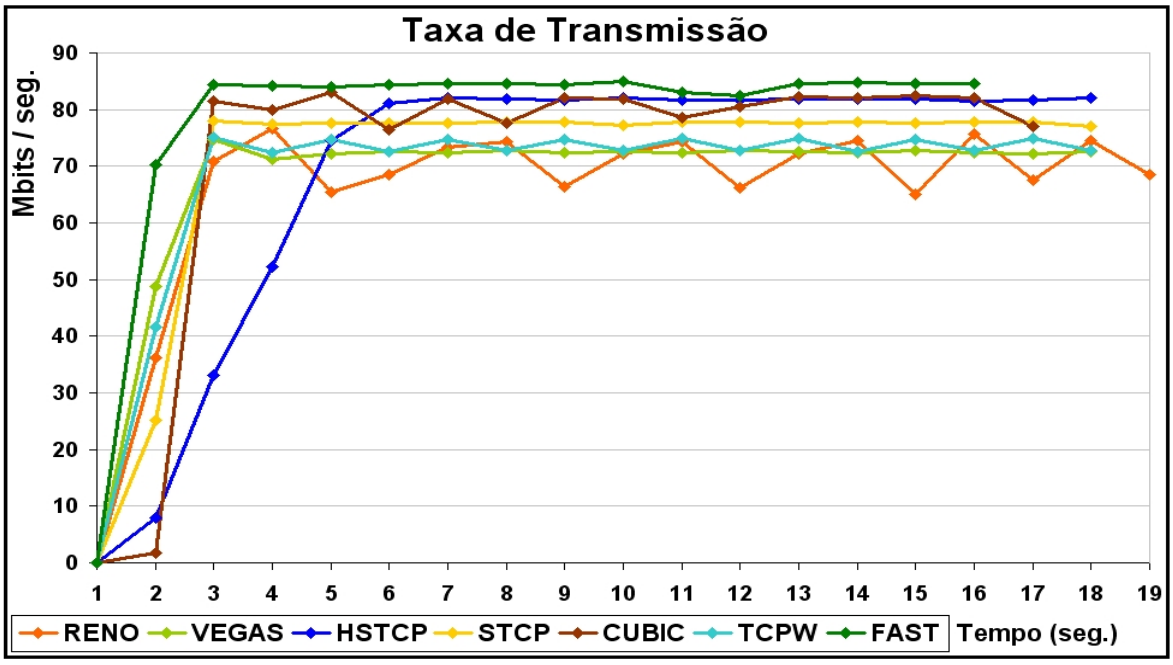

Figura 22: Experimento 9

Tabela 34: Dados capturados no experimento 9

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	18,922	18,515	17,719	17,469	16,797	18,219	**16,111**
BP	65,374	67,250	68,402	70,491	71,265	67,899	**78,084**
DVP	9,010	5,834	21,328	12,753	19,821	7,868	**3,662**

Tabela 35: Pontuação no experimento 9

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,145	0,428	0,517	0,756	0,250	**1,000**
BP	0,000	0,148	0,238	0,403	0,464	0,199	**1,000**
DVP	0,697	0,877	0,000	0,485	0,085	0,762	**1,000**
Total	0,697	1,169	0,666	1,405	1,305	1,211	**3,000**

4.3.10 Experimento 10: MS Windows - MS Windows, 150 MB e Com Perdas

Neste experimento, podemos observar um dado interessante, como apresentado na Figura 23, o STCP foi o mais estável na pontuação do parâmetro "desvio padrão" e obteve o melhor resultado na pontuação total, como visto na Tabela 37, apesar do FAST TCP ter alcançado uma melhor performance nos outros parâmetros analisados (Tabela 36). Isto se deve ao fato de que o FAST TCP perdeu pacotes durante seu *slow start*, e apesar da sua rápida recuperação não conseguiu alcançar o STCP, devido ao tamanho do fluxo (150 *Mbytes*). Dado que, o STCP também foi melhor que o BIC-CUBIC e o HSTCP, este experimento indica a importância da estabilidade na taxa de transmissão para performance do protocolo TCP e para a melhor utilização dos recursos de rede disponíveis.

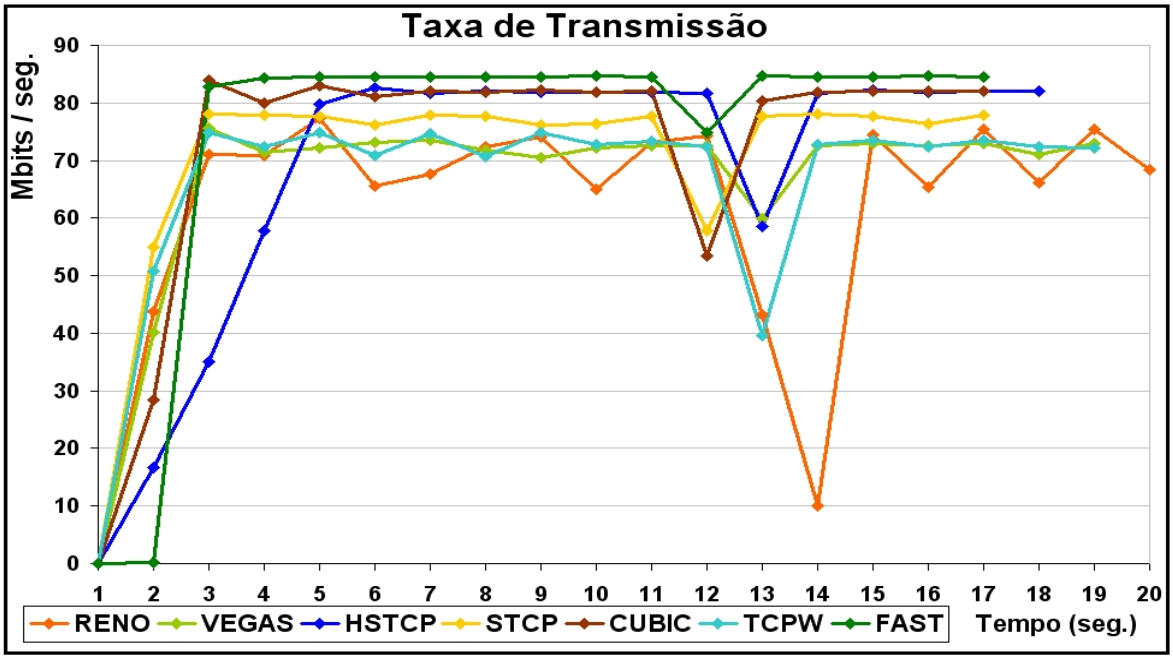

Figura 23: Experimento 10

Tabela 36: Dados capturados no experimento 10

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	20,157	18,687	17,984	17,796	17,002	18,860	**16,156**
BP	61,672	66,359	68,435	70,342	72,283	66,253	**73,953**
DVP	16,341	8,091	19,470	**7,230**	14,781	9,284	21,040

Tabela 37: Pontuação no experimento 10

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,367	0,543	0,590	0,789	0,324	**1,000**
BP	0,000	0,382	0,551	0,706	0,864	0,373	**1,000**
DVP	0,340	0,938	0,114	**1,000**	0,453	0,851	0,000
Total	0,340	1,687	1,208	**2,296**	2,106	1,548	2,000

4.3.11 Experimento 11: MS Windows - MS Windows, 500 MB e Sem Perdas

No Experimento 11, apresentado na Figura 24, podemos observar a rápida convergência do FAST TCP para o seu ponto de equilíbrio e, principalmente, a manutenção deste durante toda a transmissão do arquivo. Este fato, teve uma conseqüência direta no excelente resultado obtido, apresentado na Tabela 38, nos parâmetros "tempo" e "BP", que foram significativamente superiores aos alcançados pelos demais TCPs. No parâmetro "desvio padrão", o resultado foi apenas inferior ao do TCP Vegas, como visto na Tabela 39, o que garantiu ao FAST TCP o melhor desempenho neste experimento.

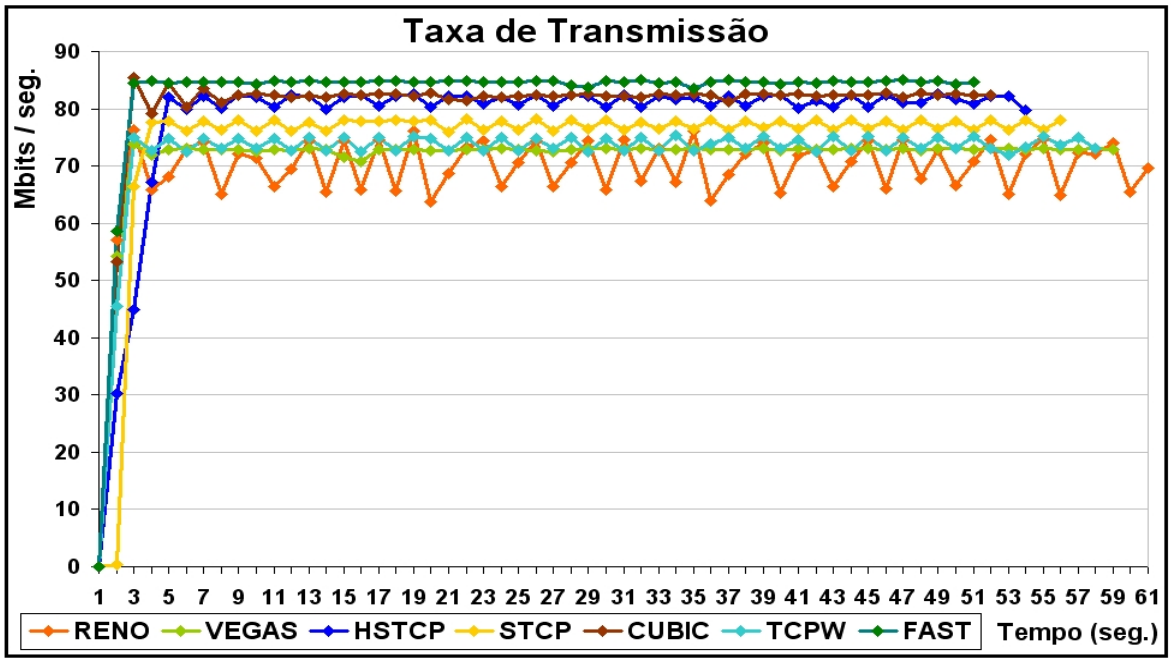

Figura 24: Experimento 11

Tabela 38: Dados capturados no experimento 11

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	61,047	59,015	54,047	55,812	52,437	58,218	**50,875**
BP	68,970	71,280	78,048	74,312	80,133	72,107	**82,545**
DVP	4,164	**2,487**	8,780	10,473	4,138	3,912	3,710

Tabela 39: Pontuação no experimento 11

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,200	0,688	0,515	0,846	0,278	**1,000**
BP	0,000	0,170	0,669	0,394	0,822	0,231	**1,000**
DVP	0,790	**1,000**	0,212	0,000	0,793	0,822	0,847
Total	0,790	1,370	1,569	0,908	2,462	1,331	**2,847**

4.3.12 Experimento 12: MS Windows - MS Windows, 500 MB e Com Perdas

No décimo segundo experimento, visto através da Figura 25, observamos a performance do FAST TCP, vencedor em todos os parâmetros avaliados (apresentados na Tabela 41). Como, neste experimento, foram efetuadas perdas controladas de pacotes, os resultados alcançados no parâmetro "desvio padrão", expostos na Tabela 40, indicam uma forte estabilidade na estratégia de controle de congestionamento implementada pelo FAST TCP.

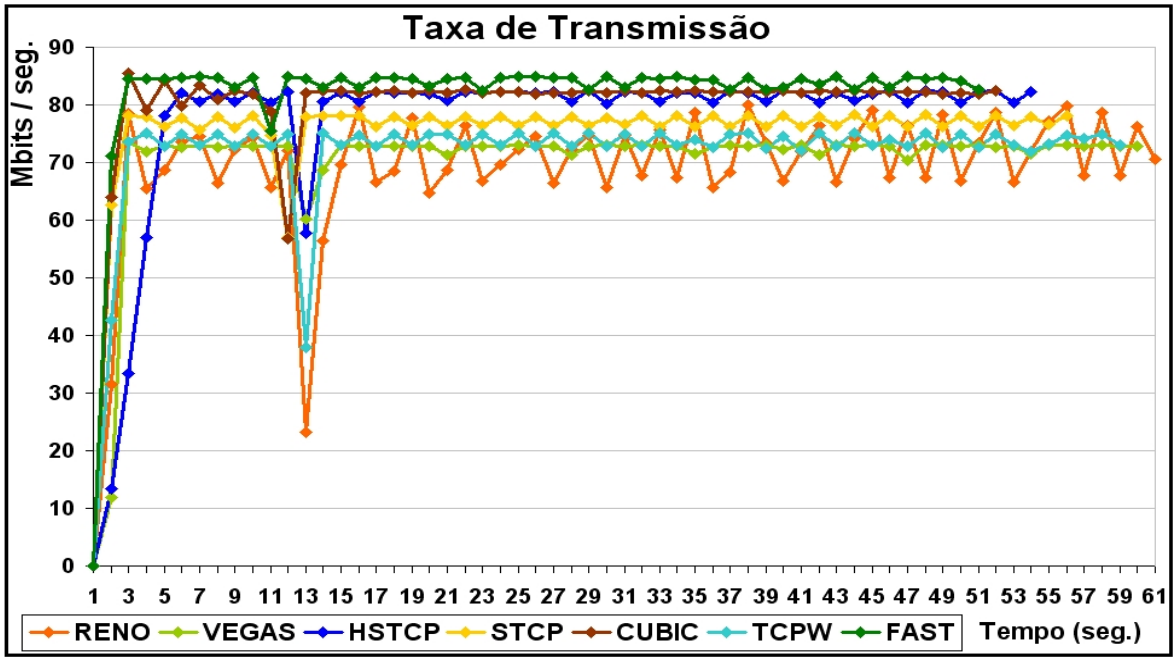

Figura 25: Experimento 12

Tabela 40: Dados capturados no experimento 12

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	61,687	59,329	54,219	56,079	52,781	58,671	**51,297**
BP	69,003	70,210	77,008	75,225	79,796	71,499	**82,087**
DVP	9,492	8,086	12,139	3,458	4,446	6,293	**2,357**

Tabela 41: Pontuação no experimento 12

	RENO	VEGAS	HSTCP	STCP	CUBIC	TCPW	FAST
Tempo	0,000	0,227	0,719	0,540	0,857	0,290	**1,000**
BP	0,000	0,092	0,612	0,476	0,825	0,191	**1,000**
DVP	0,271	0,414	0,000	0,887	0,786	0,598	**1,000**
Total	0,271	0,734	1,331	1,903	2,469	1,079	**3,000**

4.4 Pontuação obtida nos experimentos

Esta seção, tem o objetivo de apresentar a totalização dos pontos, obtidos pelos TCPs nos experimentos, para dada um dos parâmetros avaliados e, também, a pontuação total alcançada.

Na tabela 42, apresentamos a pontuação obtida, nos experimentos, em relação ao tempo total de transmissão dos arquivos, e, com o objetivo de facilitar a visualização dos resultados, na Figura 26 temos um gráfico com a pontuação média obtida por cada um dos TCPs neste parâmetro de avaliação:

Tabela 42: Pontuação sobre o tempo total em todos os experimentos

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
1	0,000	0,158	0,429	0,489	0,837	0,261	1,000
2	0,000	0,395	0,520	0,601	0,843	0,367	1,000
3	0,000	0,156	0,733	0,501	0,844	0,205	1,000
4	0,000	0,205	0,755	0,539	0,842	0,283	1,000
5	0,000	0,137	0,442	0,500	0,837	0,232	1,000
6	0,000	0,374	0,536	0,602	0,811	0,350	1,000
7	0,000	0,105	0,696	0,488	0,833	0,187	1,000
8	0,000	0,210	0,730	0,560	0,843	0,290	1,000
9	0,000	0,145	0,428	0,517	0,756	0,250	1,000
10	0,000	0,367	0,543	0,590	0,789	0,324	1,000
11	0,000	0,200	0,688	0,515	0,846	0,278	1,000
12	0,000	0,227	0,719	0,540	0,857	0,290	1,000
Soma	0,000	2,679	7,219	6,442	9,939	3,318	12,000
Média	0,000	0,223	0,602	0,537	0,828	0,276	**1,000**

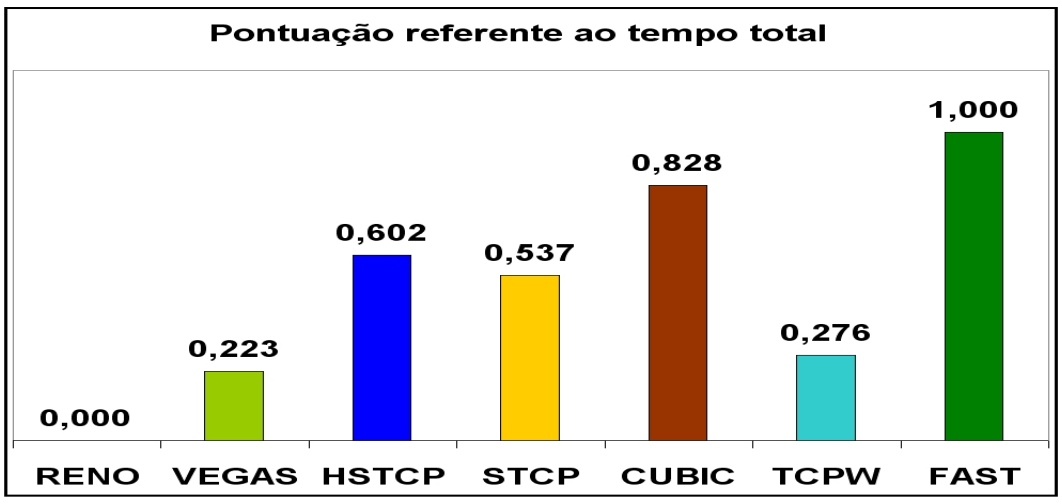

Figura 26: Pontuação obtida quanto aos tempos totais nos experimentos

Na tabela 43, apresentamos a pontuação obtida, nos experimentos, em relação à ocupação da banda passante disponível, e, com o objetivo de facilitar a visualização dos resultados, na Figura 27 temos um gráfico com a pontuação média obtida por cada um dos TCPs neste parâmetro de avaliação:

Tabela 43: Pontuação sobre a ocupação da banda passante em todos os experimentos

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
1	0,000	0,100	0,343	0,546	0,841	0,212	1,000
2	0,000	0,280	0,485	0,548	1,000	0,388	0,978
3	0,000	0,160	0,681	0,531	0,797	0,165	1,000
4	0,000	0,188	0,730	0,546	0,888	0,291	1,000
5	0,000	0,300	0,515	0,434	0,772	0,267	1,000
6	0,000	0,389	0,411	0,408	0,698	0,362	1,000
7	0,000	0,077	0,727	0,509	0,876	0,143	1,000
8	0,000	0,140	0,671	0,531	0,853	0,244	1,000
9	0,000	0,148	0,238	0,403	0,464	0,199	1,000
10	0,000	0,382	0,551	0,706	0,864	0,373	1,000
11	0,000	0,170	0,669	0,394	0,822	0,231	1,000
12	0,000	0,092	0,612	0,476	0,825	0,191	1,000
Soma	0,000	2,424	6,633	6,031	9,699	3,065	11,978
Média	0,000	0,202	0,553	0,503	0,808	0,255	**0,998**

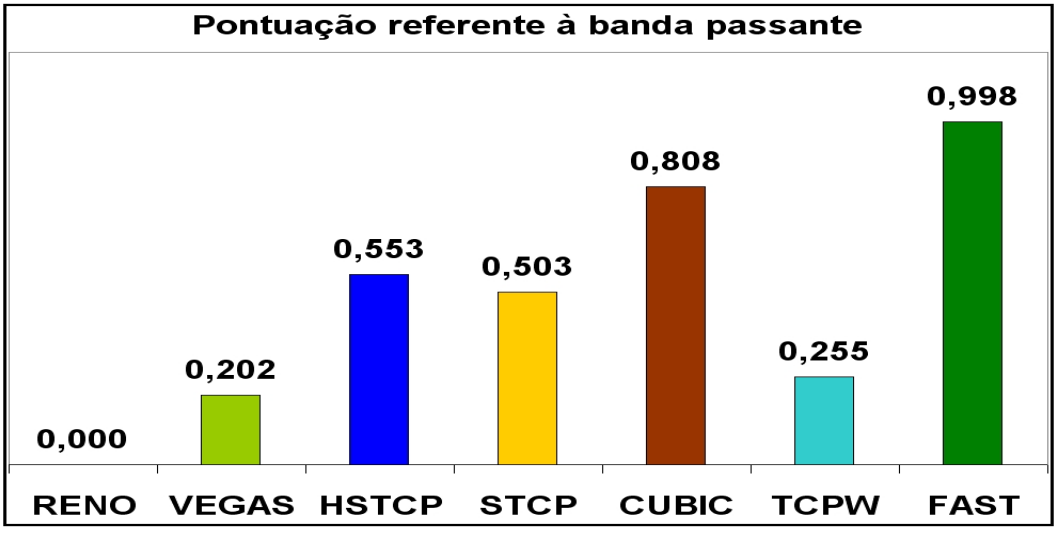

Figura 27: Pontuação obtida quanto a ocupação da banda passante nos experimentos

Na tabela 44, apresentamos a pontuação obtida, nos experimentos, em relação ao desvio padrão da taxa de transmissão de pacotes, e, com o objetivo de facilitar a visualização dos resultados, na Figura 28 temos um gráfico com a pontuação média obtida por cada um dos TCPs neste parâmetro de avaliação:

Tabela 44: Pontuação sobre o desvio padrão em todos os experimentos

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
1	0,870	0,788	0,000	1,000	0,760	0,804	0,793
2	0,215	0,337	0,000	0,411	1,000	0,833	0,406
3	0,511	0,760	0,000	1,000	0,202	0,409	0,439
4	0,081	0,742	0,000	0,979	0,965	1,000	0,441
5	0,708	0,979	0,000	0,901	0,983	0,946	1,000
6	0,186	1,000	0,000	0,137	0,345	0,803	0,933
7	0,430	0,317	0,000	1,000	0,910	0,160	0,024
8	0,289	0,408	0,000	0,875	1,000	0,655	0,757
9	0,697	0,877	0,000	0,485	0,085	0,762	1,000
10	0,340	0,938	0,114	1,000	0,453	0,851	0,000
11	0,790	1,000	0,212	0,000	0,793	0,822	0,847
12	0,271	0,414	0,000	0,887	0,786	0,598	1,000
Soma	5,389	8,560	0,326	8,677	8,283	8,643	7,639
Média	0,449	0,713	0,027	**0,723**	0,690	0,720	0,637

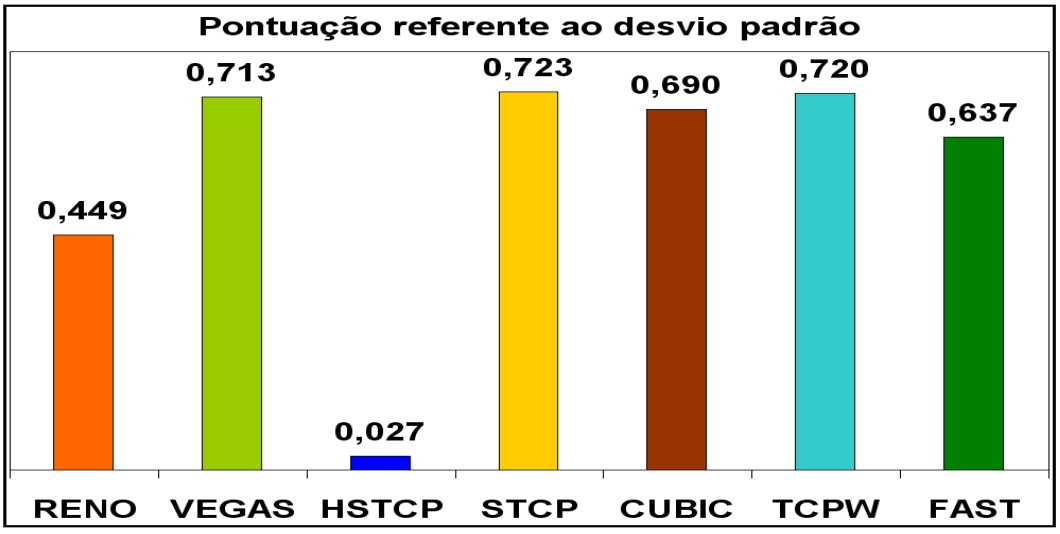

Figura 28: Pontuação obtida quanto ao desvio padrão nos experimentos

Na tabela 45, apresentamos a pontuação média obtida em cada um dos parâmetros avaliados e a média total de pontos alcançados por cada um dos TCPs.

Tabela 45: Pontuação resultante de todos os experimentos efetuados

	RENO	**VEGAS**	**HSTCP**	**STCP**	**CUBIC**	**TCPW**	**FAST**
Tempo	0,000	0,223	0,602	0,537	0,828	0,276	**1,000**
BP	0,000	0,202	0,553	0,503	0,808	0,255	**0,998**
DVP	0,449	0,713	0,027	**0,723**	0,690	0,720	0,637
Pontuação	0,150	0,380	0,394	0,587	0,776	0,417	**0,878**

Na Figura 29, temos um gráfico com a pontuação média obtida por cada um dos TCPs, considerados todos os experimentos.

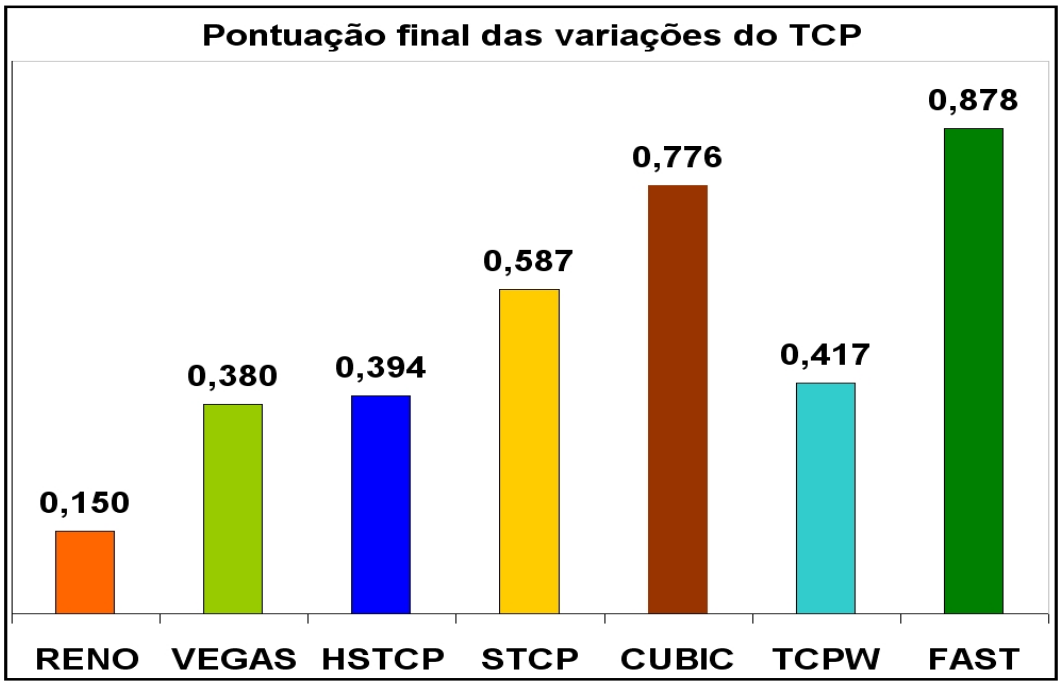

Figura 29: Pontuação final obtida nos experimentos

Dados os resultados obtidos, o TCP que contém as melhores características de performance, estabilidade e ocupação dos recursos disponíveis é o FAST TCP, como apresentado na Figura 29, portanto esta é a variação do TCP escolhida para a utilização no Servidor RIO.

4.5 Considerações sobre a avaliação efetuada

Neste ponto, vamos tecer alguns comentários à cerca da avaliação efetuada. Estes foram distribuídos nos tópicos da seguinte forma: (i) nos dois primeiros nos referimos à infra-estrutura utilizada; e (ii) nos restantes a cada um dos TCPs que participaram dos experimentos:

- **Sistemas Operacionais**: Se observarmos nos dados colhidos, em relação ao tempo total da execução, apenas nos experimentos cujos sistemas operacionais são os mesmos tanto no papel de servidor quanto no papel de cliente (4.3.1 à 4.3.4 e 4.3.9 à 4.3.12), temos que o sistema operacional Linux ofereceu resultados, em média, dois e meio porcento melhores que os obtidos pelo MS Windows. Dado que, o programa executado nos experimentos foi o mesmo, este resultado se deve ao fato de que o Linux se utiliza melhor dos recursos de *hardware* oferecidos. Para justificar esta afirmativa, o mesmo experimento também foi executado sem a utilização da biblioteca implementada para este trabalho e os resultados foram bastante semelhantes, ou seja, apresentaram uma suave vantagem no caso do sistema operacional Linux.

- **Banda passante**: Os resultados obtidos na ocupação da banda passante, que ficaram em torno dos setenta e cinco porcento, no caso médio, e levando em conta todos os experimentos, deve-se a três principais causas:

  - ***Overhead* de processamento da biblioteca implementada**: Como a biblioteca foi inteiramente desenvolvida neste trabalho, e para fins acadêmicos, esta não faz alterações no *Kernel* dos sistemas operacionais e não utiliza funções de mais baixo nível que as disponibilizadas pela linguagem "C" padrão. Desta forma, as otimizações que requisitariam funções de mais baixo nível foram deixadas de lado. Esta decisão foi tomada visando facilitar a portabilidade das bibliotecas desenvolvidas entre os diferentes sistemas operacionais, não envolvendo um número grande de funções específicas de cada um deles.

  - **Captura dos dados**: A captura dos dados envolve um certo "esforço" de processamento e acessos ao disco. Portanto, o intervalo entre as capturas, de um segundo, foi alvo de análise e sua definição levou em conta dois fatores. O primeiro é o custo da captura, pois se estreitássemos o intervalo, para meio segundo por exemplo, foi observado que os métodos eram influenciados pela captura de dados. Esta influência fica caracterizada pela suavização das curvas

da taxa de envio de pacotes, pois o tempo maior entre os *bursts* de dados reduzia as perdas não forçadas de pacotes, ou seja, perdas de pacotes devidas apenas às características de crescimento das janelas de transmissão dos TCPs avaliados. O segundo fator é que se o intervalo entre as capturas de dados fosse aumentado, notamos uma suavização nas curvas, o que não possibilitaria a demonstração, de forma mais conclusiva, da flutuação nas taxas de envio de pacotes dos TCPs. Desta forma, foi utilizado um segundo como intervalo entre as capturas, pois este foi o intervalo de tempo que sofreu o menor impacto dos dois fatores abordados acima.

- **Sistema Operacional**: Outro ponto a notar, é que nos experimentos, com o programa exemplo, sem a utilização das bibliotecas implementadas para este trabalho, a ocupação média da banda passante não foi maior que oitenta e três porcento. Portanto, a diferença na ocupação da banda passante (75% com as bibliotecas implementadas e 83% com as bibliotecas do sistema operacional) pode ser atribuída aos tópicos abordados nos ítens anteriores.

A seguir, temos os comentários relativos à cada um dos TCPs avaliados neste trabalho, dentro dos critérios estabelecidos para os experimentos, ou seja, tempo total, banda passante e desvio padrão:

- **TCP RENO**:
  - O TCP RENO obteve os piores resultados em todos os critérios avaliados, este fato não se deve apenas à forma como cresce a sua janela de congestionamento, mas também devido a sua reação aos eventos de perda. Estes dois fatores combinados reduzem de forma significativa a sua performance na transmissão de dados. O crescimento de sua janela de congestionamento não leva em conta os "sinais" de congestionamento que a rede oferece, o aumento ou a redução do RTT médio e a flutuação na taxa de envio de pacotes, desta forma, esta variação, fatalmente, induz o fluxo de dados a um evento de perda de pacotes. Durante um evento de perda o TCP RENO, de forma conservadora, reduz sua janela de congestionamento à metade o que tem como conseqüências a baixa utilização da banda passante disponível, um tempo total de transmissão maior que nos demais TCPs e uma forte flutuação na taxa de transmissão de pacotes.

- **TCP VEGAS**:

  - O TCP VEGAS tem como principal vantagem à utilização da taxa média de envio de pacotes como reguladora de sua janela de congestionamento, tanto para seu crescimento quanto para a sua redução. Esta estratégia confere uma maior estabilidade na taxa de envio de pacotes, o que pode ser observado nos seus bons resultados no quesito "desvio padrão", nos experimentos efetuados. Outro ponto positivo, é que a função acionada nos eventos é única, tanto de perda de pacotes quanto nos ACKs recebidos, e sua fórmula, que tem como alvo a ocupação da banda passante disponível, conduz a taxa de transmissão de pacotes a uma rápida convergência a seu ponto de equilíbrio, como pode ser notado nos gráficos apresentados. O principal ponto negativo desta variação do TCP é que a convergência para o ponto de equilíbrio se dá em um nível inferior ao que seria possível, visto que outras variações do TCP conseguiram convergir para níveis de ocupação da banda passante mais elevados. Esta convergência, a níveis mais baixos, ocorre pelo fato de que quando é detectado um evento de perda de pacotes a taxa tende a ser estabilizar em um patamar próximo este, porque sua estratégia de crescimento é tímida, visando evitar novas perdas de pacotes.

- **HIGH SPEED TCP**:

  - Esta variação do TCP tem como seu ponto positivo a convergência para uma taxa de transmissão de pacotes bastante alta (comparável às taxas obtidas pelo TCP BIC-CUBIC e pelo FAST TCP), sendo até superior às obtidas pelo SCALABLE TCP, que ficou à sua frente na avaliação final. O HIGH SPEED TCP peca pelo tempo que leva para convergir ao seu ponto de equilíbrio na taxa de envio de pacotes. Para transmissões de longo curso, este fato não chega a ser relevante, mas para transmissões mais curtas isto influencia tanto no tempo total da transmissão quanto na ocupação média da banda passante disponível. Outra questão, relevante para esta variação do TCP, é que sua performance é bastante afetada pelas tabelas "**a**" e "**b**", que relacionam a janela de congestionamento corrente, dentro de um determinado fluxo, a medida de seu crescimento ou redução, como descrito em [9] e apresentado na seção 2.2.3. Portanto, apenas como um exemplo ilustrativo, se uma instalação tem uma grande capacidade para a banda passante e fluxos de maior curso esta deveria

utilizar valores nestas tabelas diferentes de outra com menor banda passante e com fluxos menores. Isto não deveria ocorrer, pois uma implementação do protocolo TCP deveria se adequar às características de sua utilização de forma automática.

- **SCALABLE TCP**:
  - O SCALABLE TCP tem uma forma de atuação, tanto nos eventos de perda quando nos ACKs recebidos, similar a do TCP RENO, mas de forma bem menos agressiva. Esta estratégia conferiu ao STCP uma maior estabilidade na sua taxa de transmissão de pacotes e uma convergência mais rápida a seu ponto de equilíbrio na ocupação da banda passante. O ponto negativo é que por não observar os "sinais" de congestionamento que a rede emite, ou seja, a flutuação na taxa de envio de pacotes e no RTT médio, esta estratégia leva a eventos de perda e tende por isto a alcançar um ponto de equilíbrio inferior ao BIC-CUBIC TCP, HIGH SPEED TCP e ao FAST TCP.

- **WESTWOOD TCP**:
  - Esta variação do TCP utiliza a mesma estratégia de crescimento da janela de congestionamento do TCP RENO e, como conseqüência, sofre dos mesmos problemas, ou seja, induz a perda não forçada de pacotes, pois sua janela de congestionamento cresce atá que um evento de perda seja observado ou o crescimento seja limitado pelo controle de fluxo. Como a sua estratégia para os eventos de perda leva em conta a banda passante estimada, isto conferiu ao WESTWOOD TCP uma maior estabilidade que a oferecida pelo TCP RENO, mas não foi suficiente para evitar a flutuação na taxa de envio de pacotes, devido aos constantes eventos de perda. Desta forma, esta variação converge para um ponto de equilíbrio não muito acima do obtido pelo TCP RENO.

- **BIC-CUBIC TCP**:
  - O BIC-CUBIC TCP foi o que mais se aproximou do FAST TCP, o melhor pontuado nos experimentos realizados. Estes resultados expressivos, se devem à sua estratégia agressiva de crescimento da janela de congestionamento, ocasionando uma rápida convergência a seu ponto de equilíbrio, e a sua função cúbica, que reduz a flutuação nas proximidades de seu ponto de equilíbrio.

Esta estratégia leva em conta o WINMAX, tamanho da janela de congestionamento quando do último evento de perda registrado, que estabelece o limite para o seu crescimento em um determinado ponto do fluxo de dados. A utilização do WINMAX tem como conseqüência um maior tempo para reagir a um maior espaço para o crescimento da janela de congestionamento ou quando na proximidade de um evento de perda. Estes fatores reunidos tem como conseqüência um ponto de equilíbrio um pouco inferior ao obtido pelo FAST TCP.

- **FAST TCP**:
  - O FAST TCP, que obteve a melhor avaliação, tem muitos pontos positivos, entre eles podemos destacar: (i) a rápida convergência para o ponto de equilíbrio; (ii) a pequena flutuação no ponto de equilíbrio; (iii) sua reação aos eventos de perda com uma recuperação da taxa média de envio de pacotes de forma bastante rápida; e (iv) o fato de levar em conta os "sinais" de congestionamento emitidos pela rede em uma função única, tanto para o acréscimo quanto para o decréscimo da janela de congestionamento. Estes pontos acima são verificáveis nos gráficos, principalmente nos experimentos que forçam a perda de pacotes, pois esta variação do TCP foi a que menos perdeu espaço da banda passante durante estes eventos. Por observar os "sinais" emitidos pela rede, e com isto poder forçar o aumento da janela de congestionamento ou reduzí-la quando na proximidade de um evento de perda, levou o FAST TCP a um ponto de equilíbrio na taxa de transmissão mais alto que os demais. Como conseqüência, da pouca flutuação da taxa de transmissão de pacotes e da rápida convergência ao ponto equilíbrio esta estratégia, obteve os melhores resultados no tempo total da tarefa de transmitir os arquivos.

5 Implementação no Servidor RIO

Neste capítulo, fazemos uma breve introdução ao Servidor RIO, apresentamos a implementação e finalizamos com os experimentos e conclusões obtidas.

5.1 Introdução ao Servidor RIO

Servidores multimídia são servidores que armazenam e distribuem conteúdo multimídia e controlam as solicitações de clientes que consomem estes conteúdos. Conteúdo multimídia, também designado como objeto multimídia, é composto de vídeos, textos, fotografias, músicas entre outros. Em geral, este servidores possuem um ou mais processadores e administram vários discos.

O Servidor Multimídia RIO (*Randomized I/O*) [1, 2, 38], foi projetado e implementado inicialmente na Universidade da Califórnia - UCLA em 1998. É um servidor com um sistema de armazenamento paralelo, isto é, distribui e acessa dados de múltiplos discos simultaneamente, baseado em alocação aleatória e replicação de blocos, suportando diversos tipos de objetos multimídia, gerenciando também múltiplas aplicações concorrentes de clientes, com ou sem restrição de tempo.

O Servidor Multimídia RIO tem três componentes básicos: Nó Servidor, Nós de Armazenamento e os Nós Clientes, conforme ilustrado na Figura 30. Além disso, utiliza vários discos em paralelo para otimizar o acesso ao espaço de armazenamento e largura de banda, proporcionando assim o atendimento a um número maior de clientes, controla ainda a admissão de novos clientes para evitar comprometer a QoS dos serviços prestados aos clientes como um todo.

O Servidor Multimídia RIO é gerenciado a partir de um único Nó Servidor, que autentica e controla os acessos dos clientes, bem como processa as solicitações de consulta e manutenção dos objetos multimídia.

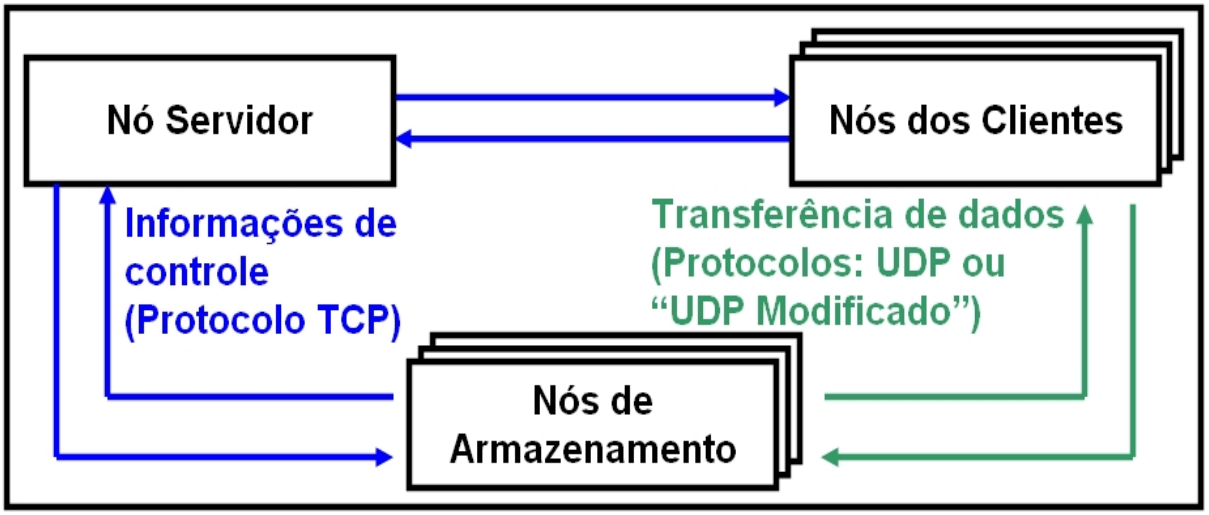

Figura 30: Interações entre o Nó Servidor, os Nós de Armazenamento e os Nós dos Clientes

Os Nós de Armazenamento são dispositivos de armazenamento conectados ao Nó Servidor ou dispositivos de armazenamento remotos conectados a outras máquinas. Os Nós de Armazenamento mantém em seus discos, o universo de objetos multimídia disponíveis para acesso dos clientes, que são atendidos, após as suas solicitações terem sido validadas pelo Nó Servidor.

Os Nós Clientes permitem que os usuários, autorizados pelo Servidor RIO, consultem e façam manutenção dos objetos multimídia nos Nós de Armazenamento. Existem dois tipos de operações efetuadas pelos usuários sobre os objetos multimídia, que foram listadas abaixo:

- **Apresentação**: O usuário solicita ao Nó Servidor a apresentação de um determinado objeto, neste momento o Nó Servidor verifica os dados do usuário através de seu nome e senha. Caso a solicitação do usuário seja aceita pelo Nó Servidor, são efetuadas trocas informações de controle entre o Nó Servidor e os Nós Cliente e de Armazenamento. Esta comunicação de controle envia aos Nós Cliente e de Armazenamento as informações necessárias para que a transmissão do objeto requisitado possa ser efetuada. Para realizar a comunicação de controle o Servidor RIO utiliza o protocolo TCP como ilustrado pela Figura 30. Finalmente, blocos de dados do objeto solicitado são transmitidos pelos Nós de Armazenamento para o Nó Cliente, que armazenando-os em *buffers* do cliente, repassando-os para um tocador de mídia e finalmente possibilitando ao usuário a sua visualização, nesta comunicação é utilizado o protocolo UDP e, no Servidor RIO, é chamada de comunicação em "tempo

real".

- **Manutenção**: Esta operação inclui as tarefas:

  – exclusão de um objeto armazenado no Servidor RIO;

  – inclusão de um novo objeto multimídia no Servidor RIO; e

  – transmissão de um objeto armazenado no Servidor RIO para sua gravação no Nó Cliente.

  Nestas tarefas, as informações de controle são trocadas, da mesma forma que na operação de apresentação, entre o Nó Servidor e o Nó Cliente bem como entre o Nó Servidor e os Nós de armazenamento. Para a transmissão dos objetos entre o Nó Cliente e os Nós de Armazenamento é utilizado um protocolo similar ao UDP ("UDP modificado"), que envia algumas confirmações periodicamente, esta comunicação, no Servidor RIO, tem o nome de comunicação em tempo "não real".

O nosso objetivo é avaliar o quanto é útil substituir o "UDP modificado" por conexões TCP, usando o FAST TCP, que mostrou melhor desempenho nos experimentos por nós realizados.

O Nó Servidor, apresentado na Figura 30, centraliza e define as informações sobre a forma de recuperar ou armazenar dados nos Nós de Armazenamento. No caso da inclusão, de um objeto multimídia, a forma de armazenar os dados é definida pelo roteador do Nó Servidor, visando uma melhor distribuição dos dados, e é efetuada através da alocação aleatória dos blocos de dados do objeto para gravação nos Nós de Armazenamento [1]. O Nó Servidor armazena por meio de estruturas de dados um mapa das informações contidas em cada um dos Nós de Armazenamento e de cada um dos objetos armazenados, estas estruturas são utilizadas na distribuição aleatória dos blocos bem como na seleção dos blocos dos objetos para a sua posterior apresentação aos clientes. Na recuperação dos objetos multimídia, o roteador define de onde serão lidos os blocos de informação nos Nós de Armazenamento, com o objetivo de alcançar a melhor performance possível na recuperação dos dados. As questões relativas a forma de armazenamento ou recuperação dos dados, que podem ser encontradas em [1], vão além escopo deste trabalho. Portanto, assumimos que localização dos dados é conhecida e está disponível, através da comunicação de dados de controle descrita anteriormente, para a transmissão de dados em tempo "não real", que ocorre entre o Nó Cliente e os Nós de armazenamento do Servidor RIO.

De forma a apresentar os principais componentes envolvidos na comunicação em "tempo não real", vamos expor os dois casos onde esta comunicação ocorre, que são: (i) a inserção de um objeto multimídia nos Nós de Armazenamento; e (ii) a recuperação de um objeto multimídia para arquivamento pelo Nó Cliente.

A inserção de um novo objeto multimídia no Servidor RIO envolve as seguintes etapas, ilustradas na Figura 31:

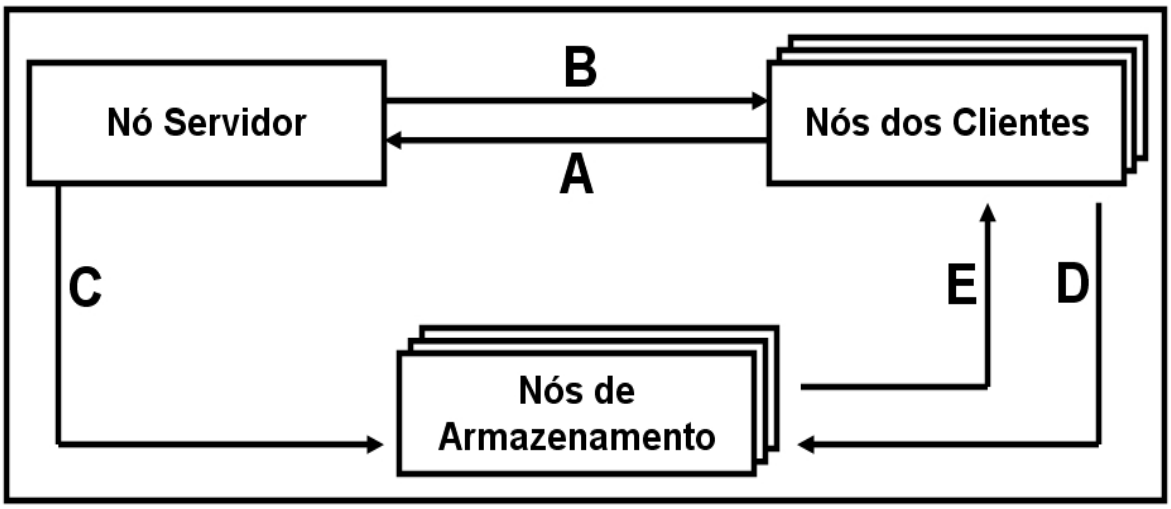

Figura 31: Etapas para a inserção um novo objeto multimídia no Servidor RIO

- **Etapa A**: O Nó Cliente requisita a inserção de um novo objeto multimídia ao Nó Servidor e repassa as informações relativas a este objeto (tamanho, dono, etc, ...).
- **Etapa B**: O Nó Servidor informa ao Nó Cliente onde serão armazenados os blocos de dados referentes ao objeto multimídia em questão, ou seja, em quais Nós de Armazenamento e em que local dos mesmos.
- **Etapa C**: O Nó Servidor informa aos Nós de Armazenamento que eles receberão blocos para armazenamento do objeto multimídia do Nó Cliente.
- **Etapa D**: O Nó Cliente envia os blocos de dados do objeto multimídia para os Nós de Armazenamento, definidos pelo Nó Servidor.
- **Etapa E**: Os Nós de Armazenamento confirmam periodicamente o recebimento dos dados enviados pelo Nó Cliente.

Neste caso, a comunicação em tempo "não real" envolve as etapas **D** e **E**.

A recuperação um objeto armazenado no Servidor RIO para arquivamento no Nó Cliente envolve as etapas descritas a seguir, ilustradas na Figura 32:

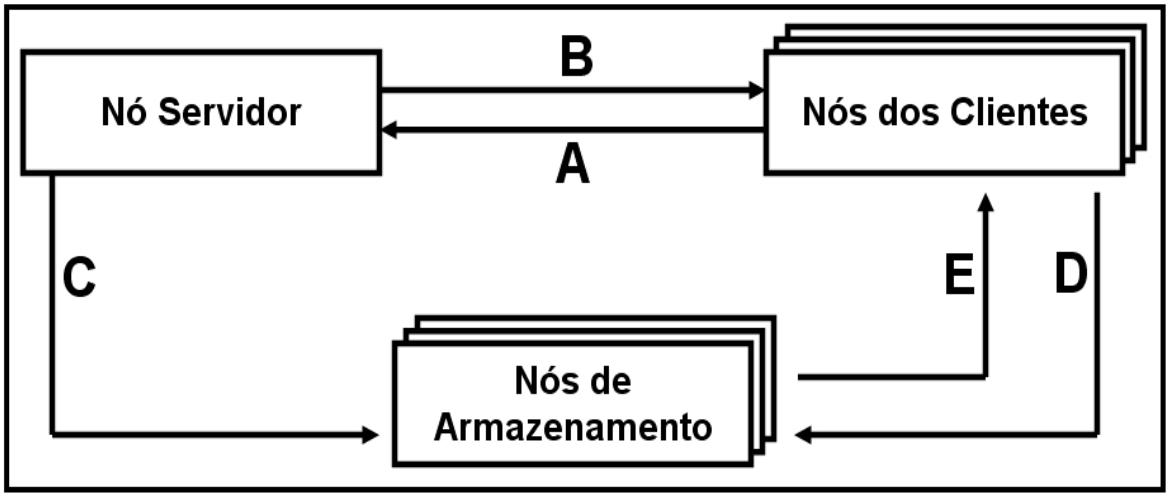

Figura 32: Etapas para a recuperação de um objeto armazenado no Servidor RIO para arquivamento

- **Etapa A**: O Nó Cliente requisita a recuperação de um objeto multimídia ao Nó Servidor.
- **Etapa B**: O Nó Servidor informa ao Nó Cliente de onde serão recebidos os blocos de dados referentes ao objeto multimídia requisitado, ou seja, de quais Nós de Armazenamento e quais os blocos dentro deles.
- **Etapa C**: O Nó Servidor informa aos Nós de Armazenamento que eles enviarão blocos do objeto multimídia para Nó Cliente e quais serão exatamente estes blocos.
- **Etapa D**: O Nó Cliente confirma periodicamente o recebimento dos dados enviados pelos Nós de Armazenamento.
- **Etapa E**: Os Nós de Armazenamento enviam os blocos de dados do objeto multimídia, definidos pelo Nó Servidor, para o Nó Cliente armazená-los.

Da mesma forma, que no caso anterior, a comunicação em tempo "não real" envolve as etapas **D** e **E**.

Finalmente, o Servidor RIO foi modificado [39, 40, 41, 42], sendo atualmente usado em um projeto de pesquisa que envolve as seguintes universidades: Universidade Federal do Rio de Janeiro (UFRJ), Universidade Federal Fluminense (UFF), Universidade Federal de Minas Gerais (UFMG). O Servidor RIO, também, é utilizado como ferramenta para os cursos de ensino à distância pelo CEDERJ [43], Centro de educação superior à distância do Estado do Rio de Janeiro.

5.2 Implementação

Esta implementação alterou a classe ***RioNeti*** do Servidor RIO, que é a responsável pela implementação, através de *sockets* UDP, tanto da comunicação em "tempo real" quanto da comunicação em "tempo não real", ou seja, nela está implementado o UDP Modificado. Esta classe é utilizada pela classe ***NetMgr***, que é responsável pelo gerenciamento da comunicação efetuada através da classe ***RioNeti***. A classe ***NetMgr*** é referenciada por diversos módulos do Servidor RIO, e com o objetivo de manter a compatibilidade da classe ***NetMgr*** com os módulos do Servidor RIO, que a utilizam, este trabalho ficou restrito a alteração da classe ***RioNeti***.

As operações de inserção ou recuperação de um objeto multimídia são compostas por um conjunto de requisições para a transferência de dados, atendidas pela classe ***RioNeti***. Desta forma, cada uma das requisições tem sua própria identidade chamada "reqid" (*request id*), ou seja, um número que a identifica. Cada uma das requisições é composta por um grupo de blocos de dados de um objeto multimídia. As requisições, também, definem o sentido da comunicação, ou seja, se este conjunto de blocos é aguardado por um nó ou deve ser enviado à um nó, lembremos que os nós envolvidos nesta comunicação tem de um lado um Nó Cliente e do outro lado um Nó de Armazenamento.

De forma a exemplificar esta comunicação, realizamos a recuperação, por parte de um Nó Cliente, de um objeto armazenado no Servidor RIO. Os dados desta comunicação estão listados na Tabela 46, onde podemos observar que para este objeto, de aproximadamente 3,5 *MBytes*, foram criadas vinte e oito requisições e que cada uma delas contém noventa blocos de dados de 1436 *Bytes*. Dado que, estas requisições são independentes, pois na classe ***RioNeti*** não temos como precisar a que objeto elas pertencem, estabelecemos conexões TCP para transmitir cada uma delas independentemente, ou seja, como se cada uma delas representasse um único objeto. No caso do exemplo, listado na Tabela 46, temos o estabelecimento e encerramento de vinte e oito conexões TCP para a completa trans-

missão do objeto, cada uma destas conexões é responsável pela transmissão de noventa blocos de informação para o Nó Cliente. Esta forma de implementação nos penalizou, pois a cada requisição transmitida temos o tempo de estabelecimento da conexão TCP, a etapa do *slow start* e o encerramento da conexão TCP, mas por outro lado herdamos as características de segurança na transmissão de dados oferecida pelo TCP. Com relação a questão do tempo de execução da tarefa, apresentada através de experimentos na seção 5.3, vamos observar que a utilização do TCP para esta comunicação em "tempo não real" foi penalizada em torno de 4,5 porcento em relação ao UDP modificado.

Tabela 46: Dados da recuperação de um objeto armazenado no Servidor RIO

Tamanho do arquivo (em *bytes*)	3619891
Quantidade de requisições	28
Quantidade de blocos de dados por requisição	90
Tamanho dos blocos de dados (em *bytes*)	1436

Outro ponto, necessário para esta implementação foi diferenciar as requisições em "tempo real" das de "tempo não real", dentro da classe ***RioNeti***, esta distinção é feita porque as de "tempo não real" requisitam o envio de pacotes ACK, ou seja, são enviadas através do UDP modificado, descrito anteriormente. A implementação deste trabalho afeta as requisições com esta característica, que ao invés de utilizar o UDP modificado serão atendidas através do protocolo TCP.

Nossa implementação afetou a classe ***RioNeti*** em três pontos:

- **Inicialização e Parada**: A implementação incluiu nos métodos *start* e *stop*, da classe ***RioNeti***, a inicialização e encerramento, respectivamente, de um servidor TCP que "escuta" em uma porta específica para cada objeto ***RioNeti*** instanciado. Não foi possível utilizar apenas uma porta, pois podemos em um mesmo servidor encontrar mais de uma instância da classe ***RioNeti*** em execução, mas estas portas estão restritas a um grupo conhecido e limitado, o que facilita a configuração de segurança dos *FireWalls* das máquinas.

- **Dados necessários ao estabelecimento da conexão TCP**: Para que esta comunicação fosse possível, foi incluída, nos pacotes transmitidos, a porta na qual o servidor TCP do ***RioNeti*** aguarda a conexão.

- **Envio dos dados através da conexão TCP**: Quando a conexão TCP é estabelecida, apenas uma requisição para a transferência de um conjunto blocos é atendida

através dela. Essa solução é necessária para manter a compatibilidade com as classes superiores do Servidor RIO. Após o término da transferência dos blocos de uma requisição a conexão TCP é encerrada.

O código desenvolvido executa tanto no MS Windows quanto no sistema operacional Linux, isto é necessário, proque o Servidor RIO faz uso destes sistemas operacionais. Nós decidimos fazer uso do Linux nos experimentos, apresentados a seguir, porque é o sistema operacional mais utilizado, pelo Servidor RIO, para a execução dos Nós de Armazenamento e do Nó Servidor.

5.3 Experimentos e resultados obtidos

Foram realizados quatro experimentos para avaliar esta implementação, que comparam a performance entre a utilização do "UDP Modificado" e do TCP. Os resultados obtidos estão expostos na Tabela 47. Nestes experimentos, realizamos a operação de recuperação para armazenamento de objetos multimídia, que foi apresentada na seção 5.1. Os objetos desta operação foram os arquivos de 150 e 500 *Mbytes*, definidos na seção 4.2.1. Para a execução dos experimentos e medição dos tempos desta tarefa utilizamos a ferramenta "riosh", que é responsável pelas manutenções nos objetos armazenados do Servidor RIO. Os equipamentos utilizados, definidos na seção 4.1, foram: dois computadores Intel mono processados de 3.2 GHz, com 1 *Gbyte* de memória principal, 80 *Gbytes* de disco SATA, placa *Ethernet* 10/100 *Mbits* e interconectados através de um *switch* 10/100 *Mbits*. O sistema operacional utilizado nos experimentos foi o Mandriva Linux 10.2.

Tabela 47: Resultados obtidos no Servidor RIO

Servidor RIO / Objetos	**UDP Modificado**	**TCP**
150 Mega Bytes	28 segundos 294 milésimos	29 segundos 708 milésimos
500 Mega Bytes	90 segundos 057 milésimos	94 segundos 380 milésimos

Através das medidas de performance, apresentadas na Tabela 47, podemos observar que os tempos do TCP não foram melhores que os obtidos pelo "UDP Modificado", algo em torno de 4.5 porcento piores, isto se deve a forma pela qual os objetos multimídia são transmitidos através da classe ***RioNeti*** do Servidor RIO.

A classe ***RioNeti*** atende à uma fila de requisições e cada uma das requisições aponta para um conjunto de blocos de dados do objeto requisitado, portanto para a transmissão

de um único objeto são criadas uma série de requisições, como exemplificado na Tabela 46. As requisições são únicas e independentes, ou seja, na classe ***RioNeti*** não temos como definir a que fluxo de dados pertence uma determinada requisição. Desta forma, para cada uma das requisições, em tempo "não real", uma conexão TCP é estabelecida para o seu atendimento, como conseqüência, esta estratégia é penalizada pelos tempos de estabelecimento da conexão TCP, fase de *slow start* e encerramento da conexão, o que não ocorre no caso do "UDP Modificado", pois o UDP não é um protocolo orientado à conexão.

Uma conclusão pode ser obtida, através dos tempos medidos, é que os tempos de execução foram muito próximos entre as duas soluções, isto nos indica que se, no nível da classe ***RioNeti***, tivéssemos apenas uma requisição por fluxo ou a identificação das requisições pertencentes um determinado fluxo, poderíamos alcançar uma performance superior a obtida pelo "UDP Modificado", pois não seríamos penalizados pelo repetido estabelecimento de conexões TCP, conseqüentemente da fase de *slow start*, a cada uma das requisições a serem processadas.

Mesmo na atual situação existente no Servidor RIO, ou seja, a questão da identificação dos fluxos, o uso do TCP, a um baixo custo, teve como vantagem a melhoria na segurança para a inserção e para a recuperação de objetos nos equipamentos remotos do Servidor RIO, pois atualmente nas conexões de baixa velocidade as transmissões seguidas vezes não alcançam sucesso, sendo esta a principal preocupação dos projetistas do Servidor RIO para este trabalho. Isto se deve diretamente às propriedades herdadas da utilização do TCP, ou seja, um protocolo que garante entrega de forma íntegra das informações requisitadas para transmissão.

6 Conclusões

O objetivo deste trabalho foi estudar o comportamento do protocolo TCP em redes de alto desempenho. Para tanto, investigamos o quanto o TCP pode se tornar vantajoso, em relação ao UDP, em redes com baixo retardo de propagação e baixo congestionamento, ou seja, modificações na implementação do TCP que maximizam a ocupação da banda passante disponível. Por outro lado, foi importante, também, averiguar se o TCP é, ou não, eficiente para a transferência de uma quantidade massiva de dados, um exemplo disto é a transferência que ocorre entre os servidores do serviço multimídia do projeto - Servidor RIO.

O Servidor RIO utiliza uma modificação do protocolo UDP para implementar um serviço de transferência de dados confiável, no entanto esta confiabilidade também é provida pelo TCP, o que tornou este um bom caso para a avaliação efetuada. Este UDP modificado realiza a comunicação em tempo "não real" do Servidor RIO, que é responsável pela criação de novos objetos multimídia e pela recuperação de objetos para armazenamento. Para a distribuição ou recuperação de um grande número de objetos entre os servidores RIO, os protocolos TCP atuais, Reno e Vegas, que adotam medidas conservadoras para o controle de congestionamento, podem trazer problemas, pois na ocorrência de um evento de perda de pacote, estas variações do TCP realizam uma forte redução da janela de transmissão induzindo a uma baixa utilização da banda passante. No caso das transmissões de grandes volumes estas medidas podem trazer limitações e instabilidade para a taxa de transmissão de dados. Desta forma, investigamos e implementamos sete variações do protocolo TCP, duas delas são as estratégias mais utilizadas atualmente na Internet, TCP Reno e TCP Vegas, e as cinco restantes modificam o algoritmo de controle de congestionamento para minimizar as limitações e instabilidade na taxa de transmissão de dados, estas estão: (i) TCP Westwood, (ii) BIC-CUBIC TCP, (iii) FAST TCP, (iv) Scalable TCP e (v) HighSpeed TCP.

Os sete TCPs foram avaliados através de uma série de experimentos, nos quais foram efetuadas transmissões de arquivos, e destes foram medidos: (i) o tempo total de

transmissão do arquivo, (ii) a estabilidade do fluxo de dados e (iii) a ocupação da banda passante. Nos experimentos, também, avaliamos as estratégias de controle de congestionamento dos TCPs, para tal implementamos o protocolo IP, da camada de rede da pilha de protocolos da Internet, desta forma interferimos no fluxo de dados forçando a perda de pacotes e, também, pudemos coletar dados mais precisos para as avaliações. Um programa foi construído especificamente para este fim, e em conseqüência, tornou-se necessária a implementação das APIs *socket*, que serviram de interface entre este programa e os TCPs.

Finalmente, substituímos a comunicação em tempo "não real" do Servidor RIO pelo FAST TCP, que obteve a melhor avaliação na bateria de experimentos realizados neste trabalho. Através da avaliação dos resultados obtidos nesta implementação, para o Servidor RIO, nos foi possível extrair conclusões e indicações para trabalhos futuros.

O estudo e implementação, que tiveram como resultado este trabalho, foram uma oportunidade de constatar, de forma prática, o quanto ainda podemos evoluir em relação aos protocolos de comunicação [36, 44]. O problema da otimização dos recursos disponíveis para as redes de computadores atuais, notadamente a Internet, que tem seus limites práticos bastante claros, mas que nos retornam dados, através das variações do RTT e da taxa de transmissão, que corretamente interpretados, podem nos levar a um novo patamar de qualidade na comunicação de dados. Esta conclusão é confirmada principalmente na performance obtida pelas variações do TCP FAST e BIC-CUBIC. O FAST TCP foi o protocolo que melhor se utilizou destes dados, capturados durante a comunicação, assim sendo foi o que atingiu os melhores resultados tanto na ocupação da banda passante disponível quanto na performance geral.

6.1 Trabalhos Futuros

A atuação dentro do Servidor RIO deu-se em um nível baixo, dentro das camadas de gerenciamento dos objetos multimídia. Desta forma, estivemos limitados ao nível das requisições de conjuntos de blocos de informações, não sendo assim possível aproveitar todos os benefícios dos TCPs de alta performance.

Como trabalho futuro, deveríamos isolar fluxos completos, ou seja, todo o atendimento a um objeto multimídia, com características de comunicação em tempo "não real", deveria estar contido em apenas uma requisição por Nó de Armazenamento ou por várias requisições que identifiquem a qual fluxo pertencem. Visto que, com um fluxo maior curso, ou seja, uma conexão TCP que atenda a uma quantidade de dados mais significativa que

os atuais 128*Kbytes*, aproveitaríamos de forma mais eficiente às variações do TCP aqui estudadas.

Como trabalhos futuros em relação à biblioteca que implementou as variações do TCP, construída para este trabalho, listamos os seguintes:

- Implementar e avaliar dos TCPs utilizando os modelos de perda de pacotes descritos na literatura.
- Avaliar esta implementação utilizando uma rede com capacidade de GigaBits por segundo.
- Implementar e avaliar os TCPs emulando o retardo fim-a-fim no *path*.
- Verificar a influência do tráfego UDP na concorrência pela banda passante disponível em relação aos TCPS da biblioteca implementada, como descrito em [45].

Referências

[1] SANTOS, J. *RIO: A Universal Multimedia Storage System Based on Random Data Allocation and Block Replication*. Tese (Doutorado).

[2] MUNTZ, R.; SANTOS, J.; BERSON, S. Rio: a real-time multimedia object server. *SIGMETRICS Perform. Eval. Rev.*, http://doi.acm.org/10.1145/262391.262398, v. 25, n. 2, p. 29–35, August. 0163-5999.

[3] MARTIN, J.; NILSSON, A.; RHEE, I. The incremental deployability of rtt-based congestion avoidance for high speed tcp internet connections. In: *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM Press, 2000. p. 134–144. ISBN 1-58113-194-1.

[4] PAGANINI, F. et al. Congestion control for high performance, stability, and fairness in general networks. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 13, n. 1, p. 43–56, 2005. ISSN 1063-6692.

[5] RHEE, I.; XU, L. Cubic: A new tcp-friendly high-speed tcp variant. Third International Workshop on Protocols for Fast Long-Distance Networks, www.csc.ncsu.edu/faculty/rhee/export/cubic-paper.pdf : LYON - FRANCE, February 2005.

[6] STEVENS, W. *RFC REVISION 793: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. Network Working Group, www.ietf.org/rfc/rfc793.txt, January 1994.

[7] BRAKMO, L.; PETERSON, L. Tcp vegas: end-to-end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, v. 13, n. 8, p. 1465, October 1995.

[8] FLOYD, S. *RFC 3649: HighSpeed TCP for Large Congestion Windows*. Network Working Group, www.ietf.org/rfc/rfc3649.txt, December 2003.

[9] SOUZA, E. de; AGARWAL, D. A highspeed tcp study: Characteristics and deployment issues. LBNL Technical, http://www-itg.lbl.gov/evandro/hstcp/, LYON - FRANCE, October 2005.

[10] XU, L.; HARFOUSH, K.; RHEE, I. Binary increase for fast, long distance networks. *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1354672, v. 4, n. 2, p. 2514–2524, January 2004. ISSN: 0743-166X,ISBN: 0-7803-8355-9,INSPEC Accession Number: 8244063.

[11] KELLY, T. Scalable tcp: improving performance in highspeed wide area networks. In: *SIGCOMM Comput. Commun.* [S.l.]: ASIGCOMM, 2003. p. 83–91.

[12] GRIECO, L. A.; MASCOLO, S. Tcp westwood and easy red to improve fairness in high-speed networks. In: *PIHSN '02: Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*. London, UK: Springer-Verlag, 2002. p. 130–146. ISBN 3-540-43658-8.

[13] GERLA, M. et al. Tcp westwood: Congestion window control using bandwidth estimation. *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=965869, v. 3, n. 1, p. 1698–1702, November 2001. ISBN: 0-7803-7206-9,INSPEC Accession Number: 7313624.

[14] GERLA, M. et al. Tcp westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs. *Computer Communications*, doi:10.1016/S0140-3664(03)00114-2, v. 27, n. 1, p. 41–58, January 2004.

[15] JIN, C. et al. Fast tcp: From theory to experiments. *Network, IEEE*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1383434, v. 19, n. 1, p. 4–11, January-February 2005. ISSN: 0890-8044,INSPEC Accession Number: 8269178.

[16] WANG, J. et al. Modelling and stability of fast tcp. *24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1498323, v. 2, n. 2, p. 938–948, March 2005. ISSN: 0743-166X,ISBN: 0-7803-8968-9,INSPEC Accession Number: 8589196.

[17] HEGDE, S. et al. Fast tcp in high-speed networks: An experimental study. *Proc. GridNets*, http://www.broadnets.org/2004/workshop-papers/Gridnets/hegde_s.pdf, October 2004.

[18] JIN, C.; WIE, D.; LOW, S. Fast tcp: Motivation, architecture, algorithims, performance. Proceedings of IEEE Infocom, Hong Kong, 2004, December 2003.

[19] JIN, C.; WEI, D.; LOW, S. Fast tcp for high-speed long-distance networks. ietf draft. Network Working Group, www.ietf.org/draft/draft-jwl-tcp-fast-01.txt, June 2003.

[20] TANG, A. et al. Equilibrium and fairness of networks shared by tcp reno and vegas/fast. *Journal Telecommunication Systems*, Springer Netherlands, v. 30, n. 4, p. 417–439, December 2005. ISSN: 1018-4864 (Print) 1572-9451 (Online).

[21] JACOBSON, V. *RFC 2001: Modified TCP Congestion Avoidance Algorithm*. Network Working Group, www.ietf.org/rfc/rfc2001.txt, April 1990.

[22] TANG, J.; LOW, S. Local stability of fast tcp. *Decision and Control, 2004. CDC. 43rd IEEE Conference*, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1428822, v. 1, n. 1, p. 1023–1028, December 2004. ISSN: 0191-2216,ISBN: 0-7803-8682-5,INSPEC Accession Number: 8435440.

[23] KUROSE, J. F.; ROSS, K. W. *Computer Networking: A top-down approach featuring the Internet*. Third. [S.l.]: Addison Wesley, 2005. ISBN 0-321-22735-2.

[24] POSTEL, J. *RFC 768: User Datagram Protocol*. Network Working Group, www.ietf.org/rfc/rfc768.txt, August 1980.

[25] "ISI". *RFC 793: TRANSMISSION CONTROL PROTOCOL*. Network Working Group, www.ietf.org/rfc/rfc793.txt, September 1981.

[26] SOCOLOFSKY, T.; KALE, C. *RFC 1180: A TCP/IP Tutorial*. Network Working Group, www.ietf.org/rfc/rfc1180.txt, January 1991.

[27] ALLMAN, M. et al. *RFC 2525:Known TCP Implementation Problems*. Network Working Group, www.ietf.org/rfc/rfc2525.txt, March 1999.

[28] FLOYD, S.; HENDERSON, T. *RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm*. Network Working Group, www.ietf.org/rfc/rfc2582.txt, April 1999.

[29] HANDLEY, M.; PADHYE, J.; FLOYD, S. *RFC 2861: TCP Congestion Window Validationm*. Network Working Group, www.ietf.org/rfc/rfc2861.txt, June 2000.

[30] ALLMAN, M.; FLOYD, S.; PARTRIDGE, C. *RFC 3390: Increasing TCPś Initial Window*. Network Working Group, www.ietf.org/rfc/rfc3390.txt, October 2002.

[31] WEST, M.; MCCANN, S. *RFC 4413: TCP/IP Field Behavior*. Network Working Group, www.ietf.org/rfc/rfc4413.txt, March 2006.

[32] DUKE, M. et al. *RFC 4614: A Roadmap for Transmission Control Protocol (TCP): Specification Documents*. Network Working Group, www.ietf.org/rfc/rfc4614.txt, September 2006.

[33] ISI. *RFC 791: INTERNET PROTOCOL*. Network Working Group, www.ietf.org/rfc/rfc791.txt, September 1981.

[34] HORNIG, C. *RFC 894: A Standard for the Transmission of IP Datagrams over Ethernet Network*. Network Working Group, www.ietf.org/rfc/rfc894.txt, April 1984.

[35] MOGUL, J. *RFC 1191: Path MTU Discovery*. Network Working Group, www.ietf.org/rfc/rfc1191.txt, November 1990.

[36] WANG, J. et al. Cross-layer optimization in tcp/ip networks. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 13, n. 3, p. 582–595, 2005. ISSN 1063-6692.

[37] MASCOLO, S.; VACIRCAY, F. The effect of reverse traffic on the performance of new tcp congestion control algorithms. PFLDNet2006, www.hpcc.jp/pfldnet2006/paper/s2_02.pdf , NARA - JAPAN, March 2006.

[38] SILVA, E. de Souza e. *Projeto Giga - Subprojeto - DIVERGE: Distribuição de Vídeo em Larga escala sobre Redes Giga, com Aplicações a Educação.* 2003. Universidade Federal do Rio de Janeiro - UFRJ.

[39] CARDOZO, A. de Q. *Mecanismo para Garantir Qualidade de Serviço de Aplicações de Vídeo sob Demanda*. Dissertação (Mestrado) — Universidade Federal Fluminense - UFF, 2002.

[40] NETTO, B. C. M. *Patching Interativo: um novo método de compartilhamento de recursos para transmissão de vídeo com alta interatividade*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro - UFRJ, 2004.

[41] GORZA, M. L. *Um Novo Mecanismo de Compartilhamento de Recursos para Transmissão de Vídeo com Alta Interatividade e Experimentos*. Dissertação (Mestrado) — Universidade Federal Fluminense - UFF, 2003.

[42] GONçALVES, S. da S. *Reorganização de Objetos Multimídia em Tempo Real quando da Remoção e Inserção de Unidades de Armazenamento do Servidor RIO*. Dissertação (Mestrado) — Universidade Federal Fluminense - UFF, 2006.

[43] CEDERJ. *CENTRO DE EDUCAÇÃO SUPERIOR A DISTÂNCIA DO ESTADO DO RIO DE JANEIRO*. Disponível em: <http://www.cederj.edu.br/cecierj/>.

[44] TANG, A.; WANG, J.; LOW, S. Counter-intuitive throughput behaviors innetworks under end-to-end control. IEEE /ACM Transactions on Networking, v. 14, n. 2, p. 355–368, April 2006.

[45] GU, Y. et al. *Congestion control for small buffer high speed networks*. Amherst, MA, 01002, February 2006.