UNIVERSIDADE FEDERAL FLUMINENSE
CENTRO TECNOLÓGICO
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

FABIO TEIXEIRA GUERRA

PROTOCOLOS DE TRANSPORTE PARA REDES DE ALTA VELOCIDADE: UM ESTUDO COMPARATIVO

NITERÓI
2006

FABIO TEIXEIRA GUERRA

PROTOCOLOS DE TRANSPORTE PARA REDES DE ALTA VELOCIDADE: UM ESTUDO COMPARATIVO

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Comunicação de Dados e Multimídia.

Orientador: Prof^{o} Dr.^{o} LUIZ CLAUDIO SCHARA MAGALHÃES

Niterói

2006

FABIO TEIXEIRA GUERRA

PROTOCOLOS DE TRANSPORTE PARA REDES DE ALTA VELOCIDADE: UM ESTUDO COMPARATIVO

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Redes de Dados e Multimídia.

Aprovada em 27 de Outubro 2006

BANCA EXAMINADORA

_____

_____

_____

Niterói

2006

Para Roberta e Luana, amores da minha vida.

AGRADECIMENTOS

Ao professor Luiz Claudio Schara Magalhães do Departamento de Engenharia de Telecomunicações da UFF pela confiança que teve em mim.

Em especial aos amigos Douglas Vidal Teixeira e Felipe Maya pela ajuda dispensada para a elaboração desta tese.

Aos funcionários do Departamento de Engenharia de Telecomunicações.

SUMÁRIO

RESUMO

Este trabalho tem como objetivo estudar o comportamento de protocolos de transporte baseado em taxa em um ambiente de alta velocidade. Para isso foram feitas simulações envolvendo tanto o RMTP, um protocolo baseado em taxa, quanto protocolos de transporte para redes de alta velocidade baseados em acks (confirmações) derivados do TCP.

Devido ao fato dos protocolos baseados em taxa enviarem os dados com um espaçamento constante entre os pacotes, eles conseguem atingir o valor de banda disponível na rede e se manter em torno dele. Já os protocolos baseados em acks geram rajadas que em alguns momentos superam a capacidade das filas dos roteadores ocasionando perdas e a conseqüente diminuição do tráfego por parte do protocolo da camada de transporte.

Palavras chaves: Protocolos de transporte, redes de alta velocidade, baseado em acks, baseado em taxa.

ABSTRACT

The main objective of this work is to examine the behavior of rate-based transport protocols in high speed networks. To this end, simulations were made using RMTP, a rate-based protocol, and other ack-clocked protocols derived from TCP.

Because rate-based protocols send data with regular spacing they are able to use all bandwidth available in the network, while ack-clocked protocols generate packet bursts which cause losses by overwhelming buffer space at the routers. Those losses cause a back-off that diminishes the amount of data being sent.

Key-words: transport protocols, high speed networks, ack-clocked, rate-based

1 A CAMADA DE TRANSPORTE NAS REDES DE ALTA VELOCIDADE

1.1 INTRODUÇÃO

Com o aumento da disponibilidade de banda na Internet e melhoria dos acessos através do uso de fibras ópticas nas cidades, as aplicações que utilizam esta rede como meio de comunicação, passaram a ter um novo gargalo entre elas: o protocolo TCP (Transmission Control Protocol [PO081]). Este protocolo de transporte foi publicado por Jon Postel em 1981 e vem recebendo várias melhorias durante todos esses anos tais como: TCP Tahoe [JA088], TCP Reno [JA090], TCP New Reno [FL099], TCP Vegas [BR095] e TCP SACK [MA096]. Estas atualizações e melhorias possibilitaram a sua grande utilização até os dias de hoje.

Apesar disso, atualmente é possível perceber que, devido a características inerentes do algoritmo de controle de congestionamento do TCP, apenas um fluxo entre transmissor e receptor não consegue enviar dados à taxa suficiente para utilizar de forma satisfatória um enlace de uma rede gigabit. Essas características que hoje dificultam a utilização do protocolo TCP em redes de alta velocidade são as seguintes: tamanho máximo de sua janela de transmissão pequeno para um ambiente de alta velocidade, o algoritmo de partida lenta que no caso de uma perda imediatamente após o início de uma transmissão faz com que o aumento de uso da banda seja muito lento, e o crescimento do tamanho da janela de transmissão de forma lenta no momento de prevenção de congestionamento. Estes assuntos serão aprofundados no próximo capítulo.

Assim sendo, durante os últimos anos alguns pesquisadores [FL003] [NA005] [LI004] [IN005] tentaram e continuam tentando de várias maneiras adaptar o algoritmo de controle de congestionamento do protocolo TCP para um novo paradigma de redes de alta velocidade.

Concomitantemente a esse grupo de pesquisadores surge um outro grupo [MA005], [MA001], [ME002] que verificando estas características limitadas do TCP passa a defender uma mudança radical na estrutura dos protocolos de transporte.

Surgem então duas linhas de pesquisa:

1- A primeira que desenvolve alterações no TCP criando novos "sabores" do TCP para redes de alta velocidade, mas mantendo uma de suas características principais que é a sua forma de transmissão baseada em acks (ack-clocked). Desta vertente os protocolos escolhidos para estudo neste trabalho são: o HSTCP (High Speed Transmission Control Protocol [FL003]), o MulTCP (Multiple TCP [NA005]), o BIC TCP (Binary Increase TCP [LI004]) e uma variação do BIC TCP conhecida como CUBIC TCP [IN005]. A escolha destes protocolos está baseada no fato de cada um destes protocolos apresentar uma concepção diferente dos seus controles de congestionamento[1]. Estes controles de congestionamentos serão abordados no próximo capítulo.

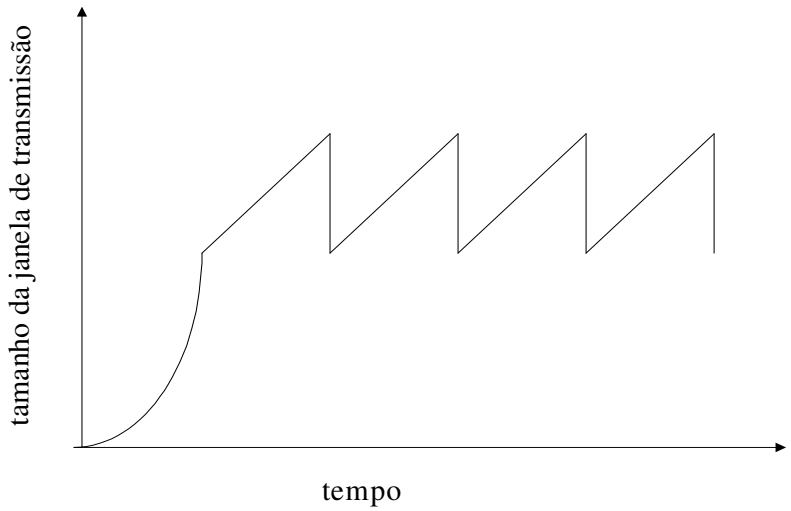
2- A segunda que defende uma forte mudança nos conceitos de protocolo de transporte em relação ao TCP, criando assim o que chamamos de protocolos baseados em taxa (rate-based). Destes protocolos o escolhido foi o RMTP (Reliable Multiplexing Transport Protocol [MA005]). Apesar do RMTP ser um protocolo desenvolvido para redes sem fio, ele possui características, vistas ao longo do trabalho, que se encaixam perfeitamente às necessidades das redes de alta velocidade. Além disso, é um protocolo bem conhecido deste grupo de pesquisa e pode ainda ser melhorado para um ambiente de alta velocidade com algumas adaptações dos seus parâmetros. Este protocolo será descrito no capítulo 3.

1.2 COMO ATINGIR ALTA VELOCIDADE FIM A FIM

As rajadas no tráfego de um fluxo TCP podem encher, em alguns momentos, as filas dos roteadores ocasionando perdas nestes. Estas perdas fazem com que o gráfico tempo x tamanho da janela de transmissão de um fluxo TCP tenha característica de dente de serra [Figura 1]. Na fase de prevenção de congestionamento, a inclinação da curva de crescimento do tamanho da janela de transmissão é pequena para um ambiente de alta velocidade, pois neste tipo de ambiente como existe uma grande oferta de banda pela camada de rede, mesmo acontecendo perdas durante a comunicação, a taxa de vazão deve retornar rapidamente para um valor próximo ao disponível na camada de rede.

[1] Exceto o par BIC/CUBIC onde o segundo é uma evolução do primeiro

Figura 1: Gráfico tempo x tamanho da janela de transmissão do TCP



No caso dos protocolos baseados em taxa o problema acima citado não existe, pois sendo o envio dos pacotes feito de forma uniforme no tempo, não existem rajadas. Isto evita o transbordo de pacotes nas filas dos roteadores e a ocorrência de perdas, possibilitando um maior aproveitamento da banda oferecida pela camada de rede.

O objetivo deste trabalho é verificar a viabilidade do uso do protocolo RMTP em redes de alta velocidade e também mostrar, através de simulações, que protocolos baseados em taxa podem ter desempenho melhor que protocolos com rajadas em redes de alta velocidade. O principal parâmetro de comparação entre os protocolos será a vazão. Outra característica interessante a ser analisada, e que está vinculada à vazão, é a estabilidade dos protocolos sob determinadas condições de tráfego.

Para a realização dos experimentos foi utilizado o simulador de rede NS-2 (versão 2.26) em uma máquina com processador Intel Pentium 4, freqüência de 2.80 GHZ, 512MB de memória RAM e sistema operacional Linux (distribuição Slackware 10.1 e kernel versão 2.4.26) no laboratório Midiacom da Universidade Federal Fluminense. Ademais, como parte da programação do NS-2 foram gerados scripts OTcl que encontram-se no anexo I e II.

Este trabalho analisa também a capacidade dos protocolos de alta velocidade compartilhar banda com o TCP, ou seja, se estes protocolos são TCP Friendly [HA003] [DO000] [PA099]. Esta informação é bem relevante nos dias de hoje, pois como a Internet é

dominada pelo TCP no nível de transporte, toda e qualquer proposta de protocolo para este nível deve possuir esta característica para não prejudicar outros fluxos que utilizam a rede.

Vale a pena ressaltar que, atualmente, existe uma tendência maior em se desenvolver protocolos de transportes derivados do TCP para redes de alta velocidade. Portanto, este estudo também tem como objetivo a divulgação de uma outra corrente de pesquisa que possui heurísticas interessantes e demonstra ser promissora. Os protocolos baseados em taxa possuem uma tendência em evitar perdas durante a comunicação, que é uma característica interessante num ambiente de redes de alta velocidade.

Esta tese é constituída de cinco capítulos. Neste capítulo é feita uma introdução ao problema e aos resultados esperados. No capítulo 2, é apresentado um embasamento teórico do protocolo TCP. Como este protocolo apresenta algumas deficiências quando utilizado em redes de alta velocidade, a primeira parte deste capítulo será a explanação destes problemas. Visto que o maior problema do TCP para ambientes de alta velocidade é o seu controle de congestionamento, serão apresentadas algumas propostas de novos controles de congestionamento para um desempenho melhor que o TCP consegue obter atualmente em redes de alta velocidade.

No capítulo 3 será apresentado o RMTP, mostrando a sua arquitetura e seu controle de congestionamento, que é conhecido como HCC (Controle de Congestionamento Homeostático).

No capítulo 4 serão apresentados os testes comparativos dos protocolos feitos com o simulador NS-2. Estes testes foram divididos em três partes: Na primeira parte, cada protocolo é testado de forma isolada para mostrar a sua capacidade de vazão em uma rede de alta velocidade. Já na segunda parte, todos os protocolos compartilham banda em um ambiente de alto tráfego para verificar qual deles possui uma maior estabilidade nesta condição de rede. Por último, foram realizados testes com o TCP compartilhando banda com os protocolos de alta velocidade. Ou seja, foram feitos cinco experimentos: TCP x HSTCP, TCP x MulTCP, TCP x BIC TCP, TCP x CUBIC TCP e TCP x RMTP.

Finalizando, no capítulo 5 será feita uma análise sobre as contribuições deste trabalho na área de protocolos de transporte para redes de alta velocidade e indicar futuros tópicos de pesquisa nesta área.

2 PROTOCOLOS DE TRANSPORTE PARA REDES DE ALTA VELOCIDADE

2.1 APRESENTAÇÃO

Neste capítulo será feita uma explanação das dificuldades de um único fluxo TCP atingir uma vazão de gigabit por segundo. Em decorrência disto estão sendo propostos novos protocolos de transporte para que toda a vazão oferecida pela camada de rede consiga ser aproveitada pela camada de transporte.

Na segunda parte deste capítulo será feita uma introdução teórica dos controles de congestionamento dos protocolos de transporte baseados em acks para redes de alta velocidade abordados nesta tese

2.2 O TCP EM ENLACES DE ALTA VELOCIDADE

O TCP é um protocolo de transporte orientado à conexão e que oferece um serviço de entrega confiável de dados. Por ser um protocolo da camada de transporte, este também realiza a função de estabelecer conexões entre aplicações residentes em computadores conectados através de redes locais ou através da Internet.

Quando o TCP foi projetado, as redes atingiam a vazão de 10Mbps. Hoje em dia 1Gbps ou até 10Gbps são valores disponibilizados pela camada de rede para um único fluxo TCP. No entanto este protocolo possui algumas características de projeto que impedem a completa utilização da banda disponível em redes com estas taxas, listadas a seguir:

- Números de seqüência dos segmentos TCP reduzido
- Tamanho dos pacotes limitado
- Tamanho da janela de transmissão limitado

• Controle de congestionamento conservador

2.2.1 Números de seqüência dos segmentos TCP reduzido

Números de seqüência são usados para identificar os segmentos TCP, pois no caso de atraso ou perda destes, o receptor usa estes números para reordená-los ou solicitar uma retransmissão.

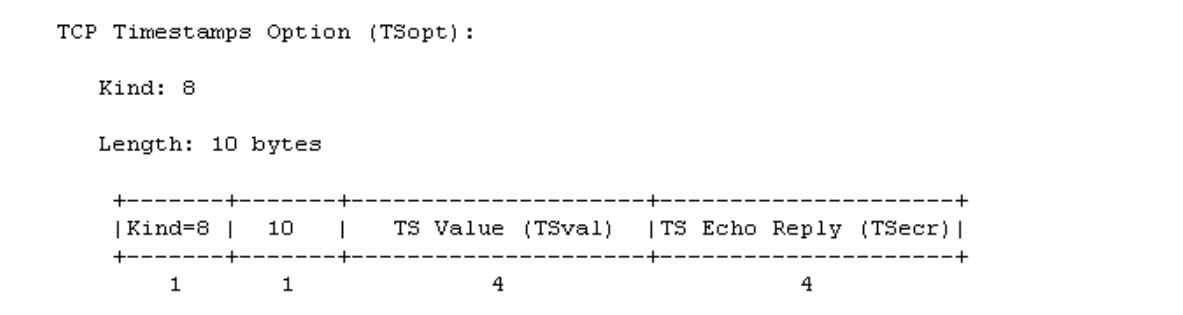
Em [PA098] observa-se que uma rede de 10Mbps necessita de 1700 segundos para o contador do número de seqüência do protocolo TCP retornar o valor inicial. Como o tempo de vida de um pacote em uma rede IP fica em torno de 120 segundos, não existe a possibilidade de dois segmentos TCP com o mesmo número de seqüência estarem na rede ao mesmo tempo. Porém, em velocidades de gigabit por segundo, os 1700 segundos citados acima diminuem para 17 segundos em redes de 1Gbps (1 Gbps é 100 vezes maior que 10 Mbps) e para 1,7 segundo em redes de 10Gbps.

Portanto, em redes de alta velocidade, existe a possibilidade de vários segmentos com o mesmo número de seqüência estarem em trânsito em um determinado instante, e no caso de reordenação ou retransmissão, o receptor e/ou o transmissor podem tratar o pacote de forma equivocada. Este problema pode ser solucionado utilizando a técnica conhecida como PAWS (Protect Against Wrapped Sequence Number) criada por Jacobson em [JA092] e descrita a seguir.

2.2.1.1 PAWS

Sabendo-se que o TCP é um protocolo simétrico (full duplex), ou seja, dados podem ser enviados a qualquer momento em ambas as direções, pode-se dentro do campo opções, usar 10 bytes para a opção TCP Timestamps Option (TSopt).

Figura 2: TCP Timestamp Option (RFC 1323)



O TSopt possui dois campos de 4 bytes chamados de Timestamps Value (TSval) e Timestamps Echo Reply (TSecr).

TSval → Contém o valor do relógio do TCP

TSecr → Contém o eco do valor de TSval recebido pelo computador. Só é válido se o bit ack estiver ligado. Quando TSecr não é válido o seu valor será zero.

Utilizando o TSopt, a técnica PAWS possui um algoritmo que rejeita segmentos antigos com o mesmo número de seqüência de segmentos novos. O PAWS assume que todo segmento TCP recebido, incluindo dados e acks, contém um timestamp SEG.TSval que possui valores únicos e crescentes. Com isso a idéia básica é: o segmento pode ser descartado como antigo se o seu timestamp SEG.TSval é menor que os timestamps recentemente recebidos numa conexão.

O PAWS seria perfeito se não fosse um problema: Se a granularidade do relógio for da ordem de micro segundo (granularidade compatível com redes de alta velocidade) o contador do timestamp voltará ao valor inicial cerca de 1 hora após o início da conexão. Este tempo é pequeno demais, inviabilizando esta técnica para redes de alta velocidade.

2.2.2 Tamanho dos pacotes limitado

Outro importante fator que limita o uso do TCP em redes de alta velocidade é o tamanho dos segmentos a serem transmitidos. Se a intenção é atingir uma alta taxa de transmissão de pacotes, quanto maior for o segmento de dados no nível de transporte menor será o overhead do cabeçalho neste nível, ocasionando assim uma maior vazão atingida.

Sabendo que o tamanho do pacote TCP influencia diretamente no desempenho de uma comunicação fim a fim, a melhora da vazão de uma conexão no nível de transporte pode ser alcançada apenas aumentando a quantidade de dados em um pacote. Em [MA097] verifica-se que a vazão de uma conexão TCP está limitada pela seguinte expressão:

Vazão ≈ 0,7 * MSS / (RTT * (Perda_Pacote)1/2)

onde:

MSS = tamanho máximo do segmento (Maximum Segmente Size)

RTT = tempo de ida e volta do pacote (Round Trip Time)

Porém, [DY099] verifica que quanto maior for o tamanho do pacote, maior será a probabilidade de perda do pacote (Perda_Pacote), se a taxa de erro de bit (bit error rate) for mantida. A partir desta constatação, não se pode afirmar plenamente que apenas aumentando o tamanho do pacote atinge-se uma maior vazão em um fluxo TCP.

2.2.3 Tamanho da janela de transmissão limitado

A respeito do tamanho da janela para protocolos de alta velocidade, existem algumas considerações importantes feitas em [PA098]:

1- Em um enlace de 1Gbps com distância de alguns milhares de quilômetros, apesar da solicitação de diminuição do tamanho da janela de transmissão, feita pelo receptor, chegar ao transmissor após alguns milisegundos, a quantidade de dados enviada fora da taxa ideal será muito pequena não afetando a conexão como um todo.

2- O algoritmo original do tamanho da janela do TCP utiliza a partida lenta (slow-start) até acontecer uma perda. Após a primeira perda, ele volta o tamanho da janela a 1 segmento e passa a ter um limiar que é a metade do tamanho da janela que teve a perda. Esse limiar serve para que, nas próximas tentativas, o TCP passe a crescer dobrando o tamanho de sua janela de transmissão até esse ponto. Após isso o TCP passa a controlar o tamanho de sua janela através do AIMD (Additive Increase Multplicative Decrease). Esse algoritmo seria muito bom se não fosse um problema: se a primeira perda acontecer logo nas primeiras transmissões teremos um limiar baixo o que causará o aumento do tamanho da janela de forma logarítmica durante um pequeno espaço de tempo. Essa situação é extremamente desinteressante para redes em geral e, principalmente, para redes de alta velocidade.

3- A terceira e última consideração surge da necessidade de, em redes de alta velocidade, a curva de crescimento da janela de transmissão ser maior e a curva de decrescimento menor do que em redes de mais baixa vazão, pois subutilizar uma rede gigabit, por exemplo, além de desinteressante é dispendioso. As novas propostas de protocolos de transporte para redes gigabit, [FL003] [NA005] [LI004] [IN005] alteram a variação do tamanho da janela de transmissão atuando assim no controle de congestionamento.

2.2.4 Controle de congestionamento conservador

Apesar de hoje o TCP ser o protocolo de transporte mais utilizado na Internet, ele possui uma limitação. Em redes de alta velocidade, o TCP não consegue atingir a taxa de gigabit por segundo oferecida pela camada de rede, pois antes de chegar à vazão disponível, ocorrem eventos que fazem com que o algoritmo de prevenção de congestionamento do TCP diminua a taxa de envio de pacotes.

Devido a isso, pesquisadores vêm tentando resolver este problema através de novas propostas de protocolos de transporte baseados no TCP, tais como: HSTCP, MulTCP, BIC TCP e CUBIC TCP. Todos esses protocolos atuam basicamente no grande problema do TCP para redes de alta velocidade: o seu controle de congestionamento.

Uma outra corrente de pesquisa defende que através da utilização de protocolos de transporte baseados em taxa em vez de protocolos baseados em acks, conseguirá naturalmente obter um melhor comportamento em redes de alta velocidade. Uma vez que estes protocolos espaçam seus segmentos de forma regular na unidade de tempo, existe uma estabilidade maior na rede, o que facilita de sobremaneira o controle de congestionamento.

Portanto, serão abordados o controle de congestionamento do TCP e a seguir os controles de congestionamento dos protocolos, HSTCP, MulTCP, BIC TCP e CUBIC TCP. Finalmente, no próximo capítulo, será apresentado o protocolo RMTP que possui um controle de congestionamento baseado em taxa conhecido como HCC (Homeostatic Congestion Control) [MA005].

2.2.4.1 Controle de congestionamento do TCP

O controle de congestionamento do TCP é constituído de duas fases distintas: partida lenta e prevenção de congestionamento (congestion-avoidance). Na partida lenta, que é usada no início de uma conexão TCP, o tamanho da janela de transmissão começa em um segmento e vai aumentando em um segmento a cada ack recebido. Essa idéia faz com que o tamanho da janela de transmissão tenha uma curva exponencial, por exemplo, ao final do 1º RTT a janela será dois, no final do 2º RTT será quatro (pois recebeu-se 2 acks), no final do 3º RTT será oito (pois recebeu-se 4 acks) e assim sucessivamente.

A fase de partida lenta termina quando o tamanho da janela ultrapassa o valor de um patamar[2], representado por uma variável chamada threshold (limiar). Quando a janela de congestionamento fica maior que o valor atual deste limiar, o TCP entra na fase de prevenção de congestionamento onde se utiliza o algoritmo AIMD que tende a ser extremamente conservador, pois a cada perda o tamanho da janela cai à metade e volta a crescer em apenas uma unidade a cada RTT.

Resumindo temos as seguintes expressões:

Partida Lenta: $w = w + c$

onde, $c = 1$

Prevenção de congestionamento (incremento): $w = w + a / w$

onde, $a = 1$

Prevenção de congestionamento (decremento): $w = w – b * w$

onde, $b = 0,5$

O algoritmo acima descrito é frequentemente chamado de TCP Tahoe. Um problema com o algoritmo Tahoe é que, quando um segmento é perdido, o transmissor da aplicação pode ter de esperar pelo esgotamento de temporização por um longo período de tempo. Por essa razão, uma variante do Tahoe chamada Reno, é também implementada.

Tal como o Tahoe, o Reno ajusta sua janela de congestionamento para um segmento quando um temporizador expira. Contudo, o Reno também inclui o mecanismo de retransmissão rápida (fast retransmit) que dispara a transmissão de um segmento descartado ou perdido se forem recebidos três acks para este segmento antes da temporização do mesmo.

Um outro mecanismo também utilizado no TCP Reno e conhecido como recuperação rápida (fast recovery) tem a função de cancelar a fase de partida lenta após uma retransmissão rápida. Em vez da janela cair para 1 e se reiniciar a partida lenta, no caso de perdas descobertas por meio de acks duplicados, a janela de congestionamento cai à metade, e continua-se no modo AIMD.

Uma outra análise importante a ser colocada é o tamanho médio da janela do TCP no seu estado estacionário e a sua relação com a vazão. Em [PA098] observa-se que para uma conexão TCP que transmite w segmentos de tamanho MSS bytes a cada RTT segundos a vazão é dada por:

vazão = $(w * MSS) / RTT$

E que em estado estacionário, o tamanho médio da janela de transmissão é dado por:

[2] Quando houver um esgotamento de temporização, o patamar é ajustado para a metade da janela de congestionamento corrente.

$w = 1,2 / p^{1/2}$

onde: p = perda de segmentos no estado estacionário

Substituindo o valor de w da segunda equação na primeira, temos que para o TCP alcançar taxas em torno de 10Gbps a perda não deve ultrapassar a cerca de 10-10, situação impossível nas redes atuais.

2.3 HSTCP, MulTCP, BIC TCP, CUBIC TCP E SEUS CONTROLES DE CONGESTIONAMENTO

2.3.1 HSTCP

O HSTCP tem como principal objetivo se comportar como o TCP quando o caminho fim-a-fim não é formado de enlaces de alta velocidade, mas também ser capaz de crescer rapidamente a vazão do fluxo quando este encontra uma grande oferta de banda no caminho fim-a-fim.

A idéia de Sally Floyd foi de alterar o controle de congestionamento do TCP, fazendo com que este ganhe desempenho em enlaces de alta velocidade. Para isso foi criada a seguinte função resposta para o HSTCP, que controla o tamanho da janela de transmissão (w):

$w = (p / L\_P)^S L\_W$

onde

$S = (\log H\_W - \log L\_W) / (\log H\_P - \log L\_P)$

p = perda de segmentos no estado estacionário

L_W = Low_Window = limite inferior (tamanho de janela) da atuação da função resposta do HSTCP

H_W = High_Window = limite superior (tamanho de janela) da atuação da função resposta do HSTCP

L_P = Low_P = taxa de perda de segmentos para Low_window

H_P = High_P = taxa de perda de segmentos para High_window

Para a fase prevenção de congestionamento foi feita também a seguinte alteração:

$w = w + a(w) / w$

$w = w - b(w) * w$

onde

$a(w) = (w^2 * 2 * b(w)) / ((2 - b(w)) * w^{1,2} * 12,8$

b(w) = ((H_D – 0,5) (log w – log L_W)) / (log H_W – log L_W) + 0,5

H_D = High Decrease = determina o valor de b(w) para w = H_W

Analisando os experimentos de Sally Floyd em [FL003], verifica-se que para valores pequenos do tamanho da janela de transmissão o HSTCP se comporta como o TCP. Isto é de grande importância, pois um dos princípios básicos de qualquer protocolo proposto para a Internet é não degradar o desempenho dos outros protocolos que compartilhem a rede, e se o HSTCP crescesse mais que o TCP na presença de erros, o HSTCP roubaria banda do TCP.

Outra característica do HSTCP é o seu desempenho para grandes valores da janela de transmissão. Quando isto acontece a janela de transmissão do HSTCP cresce mais rápido e diminui mais devagar do que a janela de transmissão do TCP. Esta característica permite que o HSTCP use mais banda em redes com maior disponibilidade de banda (como as redes de alta velocidade).

2.3.2 MulTCP

Em [NA005] é proposto como forma de aumento da capacidade de conexões TCP, a criação de N fluxos em paralelo para poder aumentar a vazão entre um transmissor e um receptor.

2.3.2.1 MulTCP 1

Na primeira tentativa em [NA005], Nabeshima verificou que no caso de N fluxos o valor médio do tamanho da janela em condições estacionárias é dado por:

$W_{mul1} = (2N * (N - ¼))^{1/2} / P^{1/2}$

Na expressão acima, Se N = 1, obtém-se o mesmo valor de w para um único fluxo TCP, porém para N > 1, o valor da expressão acima é maior que o valor de N vezes um único fluxo TCP. Tornando assim o MulTCP1 agressivo demais quando utilizado com outros fluxos TCP.

2.3.2.2 MulTCP 2

Sendo assim, em [NA005] Nabeshima propôs uma outra versão na qual o tamanho da janela em condições estacionárias do MulTCP com N fluxos será exatamente:

$W_{mul2} = N * w = N * (1,2 / p^{1/2})$

Para a expressão acima ser verdadeira é necessária uma relação entre a e b (constantes utilizadas na fase de prevenção do congestionamento do TCP) para que a função resposta do MulTCP seja a esperada para qualquer valor de N:

$b = 2\ a / (a + 3N^2)$

A idéia de Nabeshima é interessante, no entanto é difícil tornar este protocolo amigável ao TCP. Agregando N fluxos TCP a uma determinada aplicação, se existir uma outra aplicação dividindo banda em um determinado enlace que utilize um único fluxo TCP esta aplicação será prejudicada (a primeira ficará com N/N+1 da banda, enquanto a segunda ficará com 1/N+1).

Como o controle de congestionamento do MulTCP é o mesmo do TCP, surge então outra característica indesejável do MulTCP: apesar deste utilizar N fluxos TCP para aumentar a vazão de uma determinada aplicação, o controle de congestionamento dos N fluxos continua com as mesmas deficiências do TCP quando utilizado em redes de alta velocidade. Assim, como as perdas nos protocolos baseados em acks ocorrem geralmente em rajadas em um ambiente de alto tráfego, este protocolo pode oferecer um desempenho muito aquém do desejado.

2.3.3 BIC TCP

O BIC TCP tem, na essência do seu projeto, três importantes critérios para um protocolo de alta velocidade:

• Justiça em Termos de RTT: evitar que fluxos BIC TCP com RTT pequenos ocupem toda a banda de um enlace quando este está sendo utilizado por outros fluxos.

• Amigável ao TCP: compartilhar recursos igualmente com fluxos TCP quando estes disputam banda com fluxos BIC TCP.

• Escalabilidade: capacidade de crescer rapidamente a vazão do fluxo quando a existe banda em grande quantidade.

O BIC TCP possui um algoritmo para o seu controle de congestionamento dividido em 2 partes:

1- Incremento via Busca Binária (Binary Search Increase)

2- Incremento Aditivo (Additive Increase)

1ª parte: Incremento via Busca Binária

W_{Min} = Janela mínima corrente.

W_{Max} = Janela máxima corrente

Target Window = Janela Alvo

Para o primeiro valor de W_{Min} o protocolo escolhe um valor no qual o fluxo não tenha nenhuma perda e o primeiro valor de W_{Max} é aleatório e muito grande. A cada RTT ocorre o seguinte:

Janela Alvo = $(W_{Max} - W_{Min}) / 2$

O processo acima é repetido até a diferença entre W_{Min} e W_{Max} ser menor que um limite anteriormente denominado chamado de mínimo incremento (S_{min})

No caso de perdas durante o incremento via busca binária a janela corrente passa a ser WMax e a nova janela após o decremento passa a ser o novo W_{Min}.

2ª parte: Incremento Aditivo

Quando a distância entre W_{Min} e W_{Max} é muito grande e o aumento para o ponto médio é maior que o máximo incremento (S_{max}), aumenta-se a janela sempre de S_{max} até a distância entre W_{Min} e W_{Max} ser menor que S_{max}

Ocorrendo perdas nos momentos de incremento aditivo, o BIC TCP utiliza a estratégia de decrementos múltiplos (Multiplicative Decrease) igual a do TCP.

OBSERVAÇÕES:

1- Partida Lenta: Quando a janela corrente chega até W_{Max}, o incremento via busca binária escolhe um novo valor de W_{Max} aleatoriamente e a janela corrente passa a ser W_{Min}. Se $(W_{Min} + W_{Max})/2 > S_{max}$, em vez de utilizar o incremento aditivo, o BIC TCP roda um algoritmo chamado de partida lenta onde o incremento será $C_{wnd} + 1$, $C_{wnd} + 2$, ..., $C_{wnd} + S_{max}$.

onde

C_{wnd} = Janela Corrente

Após a partida lenta ser rodada o BIC TCP passa para o modo de incremento via busca binária.

2- Convergência Rápida (Fast Convergence): no incremento via busca binária, após uma redução da janela de transmissão, novos W_{Max} e W_{Min} são definidos. Sendo o novo W_{Max} menor que o anterior, esta janela teve uma tendência descendente e conseqüentemente podem existir outros fluxos competindo com o fluxo BIC TCP. Com isso para garantir uma maior justiça (fairness[3]), deve-se reajustar o novo W_{Max} como sendo o primeiro valor da janela alvo, ou seja:

$W_{Max} = (W_{Max} - W_{Min}) / 2.$

2.3.4 CUBIC TCP

Em [IN005], Injong Rhee e Lisong Xu descrevem uma nova variante para o BIC TCP chamada de CUBIC TCP, que possui uma função cúbica de crescimento da janela. Este protocolo é muito parecido com o BIC TCP, porém é mais amigável ao TCP e mais justo em termos de RTT do que o BIC TCP. O CUBIC TCP funciona sob a seguinte função:

$W_{CUBIC} = C(t-K)^3 + W_{Max}$

Onde:

C – Fator de escala

t – Tempo decorrido desde a última redução da janela

$K = ((\beta W_{Max} / C))$

β – Constante utilizada no decremento do tamanho da janela no momento de uma perda

[3] Fairness – tenta evitar que fluxos com menores RTT capturem uma maior banda

3 O RMTP

3.1 APRESENTAÇÃO

Após a introdução dos controles de congestionamento dos protocolos de transporte baseados em acks para redes de alta velocidade, neste capítulo será feita a apresentação detalhada do protocolo de transporte baseado em taxa conhecido como RMTP.

Neste capítulo serão abordados os seguintes aspectos do RMTP: a sua arquitetura, a confiabilidade, o controle de fluxo e o seu controle de congestionamento.

3.2 INTRODUÇÃO

O RMTP é um protocolo de transporte desenvolvido em [MA005] como parte de um conjunto de protocolos que permite mobilidade através da independência do nível de transporte da camada de rede, provendo transmissão confiável para transferência de objetos (como usados no ftp e http). O RMTP foi projetado com as seguintes características:

- Múltiplos canais

- Transmissão dos dados utilizando a técnica de envio baseada em taxa e com acks seletivos (sacks) para garantir confiabilidade

- Estimativa de banda através de medição de banda disponível

3.3 ARQUITETURA RMTP

Em [MA005] a arquitetura RMTP é definida como uma arquitetura de camada de transporte, que possibilita a agregação dos recursos derivados de múltiplos canais da camada de rede. Assim, a camada de transporte provê a multiplexação inversa (ou agregação) destes canais de rede em apenas um único canal virtual para a camada de aplicação.

É importante citar que nesta tese a quantidade de canais utilizada nas simulações foi sempre um. Como no momento de criação deste protocolo a intenção maior foi prover mobilidade, Magalhães utilizou a técnica de múltiplos canais para conseguir êxito no seu projeto, assim, este capítulo estará fazendo alguns comentários vinculados a esta questão.

Através da técnica de par de pacotes (packet-pair) [KE092], utilizada pelo RMTP, tem-se como estimar o menor intervalo de tempo entre os pacotes, para se tentar minimizar problemas na comunicação entre transmissor e receptor dentre os quais se destacam: atraso nas filas dos roteadores, atraso da rede e grande variação do RTT. O valor obtido através desta técnica é usado para definir importantes parâmetros do RMTP e conseqüentemente possibilitar um bom uso deste protocolo.

Para manter a confiabilidade e o sequenciamento, o RMTP utiliza um mecanismo baseado em janela com acks seletivos. O tamanho da janela de confiabilidade (window size) define o espaço disponível no receptor para mensagens que devem ser aceitas e armazenadas.

A implementação dos acks seletivos é feita através de um mapa de bits (bit map) de tamanho fixo, portanto o tamanho da janela de confiabilidade também deverá ser fixo durante a conexão o que acarreta também num tamanho fixo do cabeçalho do protocolo e simplifica o processamento.

Como o tamanho da janela de confiabilidade deve acomodar uma quantidade de segmentos para que o protocolo não pare se acontecer uma perda, o RMTP define o tamanho da janela como sendo um múltiplo de 32 que seja maior que duas vezes a soma do produto atraso x banda de cada canal dividida pelo tamanho do segmento. O valor 32 decorre do tamanho da palavra de memória usada para o mapa de bits. A cada inteiro sem sinal que se junta ao mapa de bits, o tamanho deste cresce em 32.

A variável responsável pelo controle do espaço disponível na fila do receptor é chamada de espaço livre da fila (free buffer space). A quantidade de espaço livre nesta variável é que permite o avanço da janela de transmissão, ademais, se esta variável for igual a

zero o transmissor continuará a transmitir apenas o que estiver na janela de confiabilidade, só podendo passar disso quando este receber novamente a informação que a variável não é mais zero. Existe um campo no cabeçalho do RMTP que informa ao transmissor o quanto de espaço livre existe disponível na fila do receptor.

A variável taxa de recebimento máxima (maximum receiver rate), como o nome já diz, é responsável em informar qual a taxa máxima que o receptor deve processar os segmentos RMTP. Esta taxa pode ser limitada pelo transmissor ou pelo receptor, dependendo das características das máquinas envolvidas na conexão.

A taxa máxima (maximum rate) é a variável que indica a quantidade máxima de dados que o RMTP pode enviar através de um único canal. O RMTP utiliza continuamente mensagens de teste (probes) nos canais ativos, para atingir a banda acessível. Quando o valor da taxa máxima se iguala ao valor da banda máxima do canal (banda esta negociada no início da conexão) o RMTP pára de testar o canal com objetivo de evitar uma sobrecarga e conseqüentes perdas.

A última variável importante é chamada de taxa inicial (intial rate). Como o seu valor é obtido através da técnica de par de pacotes, e sabendo que esta técnica pode super estimar a banda na maioria de suas estimativas, o valor da taxa inicial será a metade do valor obtido no primeiro teste do par de pacotes.

3.4 CONFIABILIDADE

O RMTP implementa uma transmissão confiável baseada em retransmissão e detecção de descontinuidades nos números de seqüência para identificar perdas. Os segmentos recebidos pelo receptor são notificados para o transmissor através do envio de acks. Existem dois tipos de acks: acumulativo e seletivo. O ack acumulativo carrega o número do próximo segmento esperado (último recebimento + 1). O ack seletivo é um mapa de bits do estado da fila do receptor, sendo o bit zero a posição do próximo segmento esperado. O transmissor mantém um mapa de bits individual por canal para identificar quais segmentos já foram enviados.

Para detectar uma descontinuidade no número de seqüência, o RMTP verifica espaços no mapa de bits dos acks seletivos em cada canal. Quando o transmissor envia um determinado segmento, o bit correspondente a este segmento no mapa de bits do canal é marcado. Desta forma, ao ser recebido, o ack seletivo é comparado com o mapa de bits de

cada canal e se for encontrado descontinuidade ou descontinuidades no número de seqüência, os segmentos perdidos são retransmitidos. É necessário usar a comparação dos mapas de bits porque quadros podem ser transmitidos por canais de diferentes atrasos, o que gerará fatalmente recepção fora de ordem. Um segmento só pode ser considerado perdido se o segmento enviado depois dele naquele canal já tiver sido recebido.

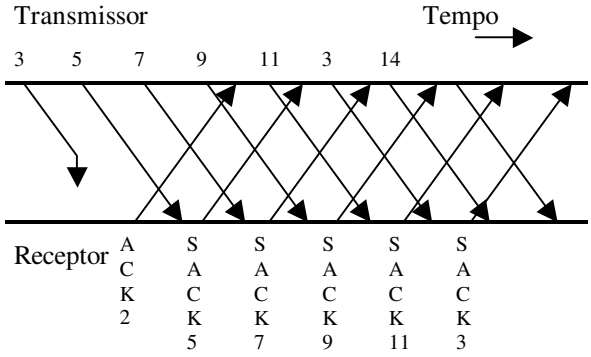
Uma vez retransmitido o segmento, o protocolo não pode mais continuar a seqüência de detecção das descontinuidades, pois o segmento retransmitido está fora de ordem. Devido a este fato torna-se necessário utilizar dois mapas de bits para controlar as retransmissões.

O primeiro mapa de bits é chamado de segmentos retransmitidos (retransmitted frames), que controla todos os segmentos retransmitidos e evita múltiplas retransmissões do mesmo segmento.

Para verificar se a retransmissão foi perdida, o RMTP possui um segundo mapa de bits chamado de segmentos marcadores (marker frames), usado para indicar o segmento que foi enviado imediatamente após a retransmissão. Se um ack de segmentos marcadores chegar antes do ack de segmentos retransmitidos o segmento retransmitido foi perdido e com isso deve-se enviá-lo de novo.

O mapa de bits conhecido como segmentos marcadores, possui uma função interessante a saber: mesmo depois de uma descontinuidade ser detectada e um segmento perdido ser retransmitido, todos os acks gerados antes da detecção do segmento perdido que chegam ao receptor, irão conter a mesma descontinuidade. Assim, várias retransmissões desnecessárias vão ocorrer. Para evitar estas retransmissões, um segmento no mapa de bits segmentos retransmitidos só pode ser retransmitido novamente após a sua marcação no mapa de bits segmentos marcadores ser apagada. A posição no mapa de bits segmentos marcadores é apagada quando um ack ou uma descontinuidade daquela posição é detectado. Essa técnica permite ao protocolo fazer todas as suas retransmissões sem usar temporizações explícitas.

Figura 3: Detecção e Retransmissão de um Frame Perdido

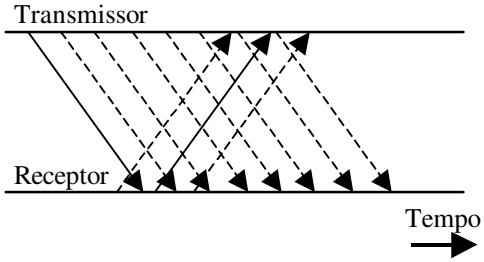


Para ilustrar o problema observe o exemplo da Figura 3, usado por Magalhães em [MA005]. Um canal que transmite os segmentos 3 e 5, e recebe um ack do 5, teve o seu segmento 3 perdido. Assim, o segmento 3 será retransmitido pelo primeiro canal disponível, sendo retirado (apagado) do mapa de bits do canal original e ligado no mapa de bits do canal responsável pela retransmissão. Será também ligado no mapa de bits segmentos retransmitidos e o segmento seguinte deste canal será ligado no mapa de bits segmentos marcadores. Uma vez que o segmento é retransmitido, ele só será retransmitido de novo, se o segmento posterior à ele receber o ack. Se o segmento 14 segue a retransmissão do segmento 3 neste exemplo, o segmento 3 só pode ser retransmitido após o ack do segmento 14 ser recebido. Como o segmento 3 precede o 14 neste canal, se o ack do 14 chegar antes do 3 significa que o 3 perdeu-se novamente.

O RMTP não possui proteção contra segmentos fora de ordem nos canais. Segmentos que chegarem fora de ordem no mesmo canal causarão retransmissões desnecessárias. Com a utilização de dois acks em um, para economizar banda em um sentido, diminui-se o número de retransmissões desnecessárias à metade, mas não resolve o problema.

A detecção de descontinuidade pode falhar se todos os segmentos da janela de confiabilidade forem perdidos. Para resolver este problema, em vez de usar um temporizador explícito, faz-se o seguinte: se a janela de confiabilidade estiver cheia, e nenhum segmento estiver acessível para retransmissão na fila do canal, então, as taxas caem à metade em todos os canais e os segmentos são retransmitidos, começando do último segmento sem ack. Isto funciona como um temporizador natural. As taxas são reduzidas à metade para evitar um repentino descompasso entre o tamanho da janela e o produto atraso x banda, o que pode causar retransmissões desnecessárias.

Figura 4: Produto atraso x banda em um ambiente baseado em taxa



Se apenas um canal está sendo usado, o preenchimento da janela de confiabilidade pode ser causado por um tamanho da janela de confiabilidade subestimado. Para que isso não aconteça, o tamanho mínimo da janela de confiabilidade deve ser maior que o dobro do atraso

da propagação da banda do canal mais um tempo aleatório extra para combater a sincronização.

Na Figura 4 existe um exemplo, também citado por Magalhães em [MA005], de um canal capaz de enviar sete segmentos antes de receber um ack, neste caso, o tamanho mínimo da janela de confiabilidade é oito. Se um tamanho menor for escolhido, todo segmento vai ser retransmitido duas vezes no caso de alguma perda. A janela de confiabilidade deve poder armazenar segmentos suficientes para permitir que o ack de um segmento recebido a tempo evite a sua retransmissão. Se mais de um canal está sendo usado, então o tamanho de janela mínimo (minimum window size) é dado pela razão das bandas e do maior atraso de propagação. O tamanho de janela mínimo é definido como o tamanho que o espaço de armazenamento temporário precisa ter para acomodar todos os segmentos que são transmitidos entre o tempo que o segmento é enviado no canal de maior atraso e o tempo que o seu ack é recebido.

3.5 CONTROLE DE FLUXO

O controle de fluxo é uma parte importante de um protocolo de transporte, pois caso a rede consiga entregar mais pacotes do que o receptor possa processar em certo intervalo de tempo, ele deve atuar evitando a perda de pacotes no receptor.

No RMTP existem dois mecanismos usados para o controle de fluxo, um é a taxa máxima (maximum rate) negociada no início da transmissão. A taxa máxima define um limite no número de segmentos enviados por segundo, mesmo se o protocolo medir uma banda acessível maior do que a que está sendo usada no momento, esta não ultrapassará a taxa máxima.

O segundo mecanismo é a fila do receptor. Esta é uma fila auxiliar que suaviza o efeito gerado pelo descompasso de banda e existência de atrasos, e está implementada como parte da janela de recepção.

3.6 CONTROLE DE CONGESTIONAMENTO

Esta seção contém a descrição de um algoritmo de controle de congestionamento chamado: Controle de Congestionamento Homeostático (Homeostatic Congestion Control –

HCC) [MA005], desenvolvido para protocolos de transporte em redes sem fio e baseados em taxa.

Apesar do HCC ter sido desenvolvido para redes sem fio, ele possui várias características que vão diretamente ao encontro das necessidades de uma rede de alta velocidade tais como:

- Manutenção da taxa de transmissão abaixo do ponto de congestionamento da rede
- Convergência para a banda disponível
- Estabilidade quando utilizado sozinho e na presença de outro tráfego
- Tentativa de ser um protocolo justo (Fairness)

O TCP superestima a banda do caminho a ser percorrido pelo segmento, causando perdas mesmo em condições estáveis. Esta característica encorajou Magalhães em [MA005] a aceitar o desafio de tentar medir a banda disponível da rede sem causar problemas à mesma.

3.6.1 POR QUE MEDIR A BANDA?

Com o aumento da confiabilidade dos enlaces da internet, pode-se afirmar que quando um pacote é perdido, uma ou mais filas dos roteadores entre transmissor e receptor estão acima de sua capacidade. Com isso, no controle de congestionamento do TCP e de suas variações para rede de alta velocidade, quando um transmissor recebe a notificação de perda de um pacote é sinal que ele deve diminuir a taxa, pois algum roteador está recebendo pacotes acima da sua capacidade de encaminhá-los.

Outra consideração interessante é o tempo entre a existência de um congestionamento e a ação para extingui-lo. Como dito no parágrafo anterior, o TCP e seus diferentes “sabores” dependem que a informação de perda chegue ao transmissor para aí sim tomar uma atitude de diminuição de banda. À priori este tempo será de no mínimo ½ RTT, apesar de o desenvolvimento do ECN (Explicit Congestion Notification) [MA003] [MA103] [RA001] nos roteadores diminuir este tempo, sempre existirá um □t entre o congestionamento e a ação para término deste.

Verifica-se então que o mundo ideal seria ter a informação da banda disponível no momento de envio dos pacotes, porém, isso só é possível em redes com reserva de banda.

Então, pode-se afirmar que o TCP e suas derivações estão sempre testando a rede, ou seja, enviando segmentos de maneira a forçar o congestionamento e aí atuar de forma reativa.

Já o HCC tenta atuar de forma preventiva medindo com a maior precisão possível a capacidade da rede para aí sim enviar dados.

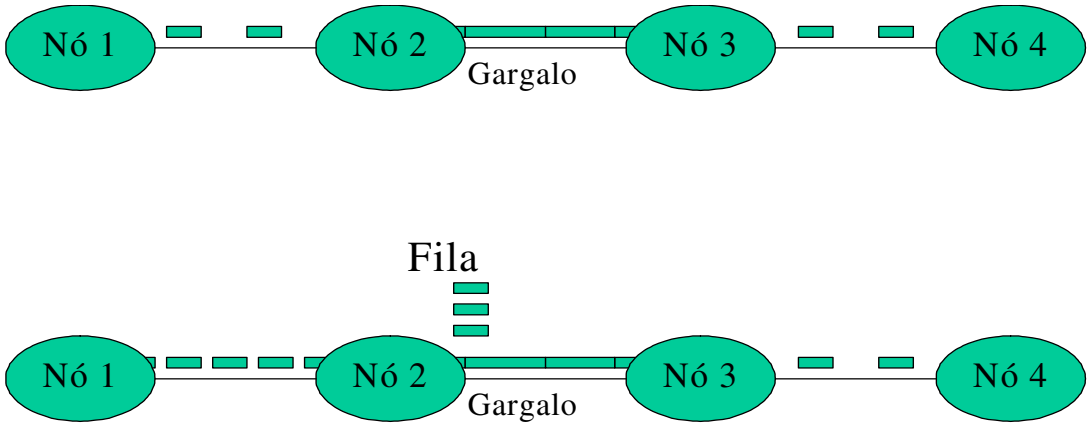
3.6.2 A TÉCNICA DO PAR DE PACOTES

O espaçamento de pacotes adjacentes causado pelo enlace de gargalo entre o transmissor e o receptor [Figura 5], é um fenômeno muito comum na Internet. A partir destes espaçamentos gerados pelos enlaces de menor capacidade criou-se um mecanismo de controle de fluxo que através da análise da variação do espaçamento destes pacotes no receptor ajusta a taxa de transmissão do fluxo [KE092].

Para o HCC Magalhães utilizou esta idéia conhecida como par de pacotes para tentar inferir um valor de taxa de transmissão amigável ao TCP e também justa com os outros fluxos existentes.

É importante citar que a técnica de par de pacotes obtém resultados mais precisos quando utilizada em redes de roteadores que possuem filas do tipo fair-queueing. Como na Internet a maioria dos roteadores utiliza filas do tipo FIFO (First In First Out) estas medições podem ser subestimadas ou superestimadas.

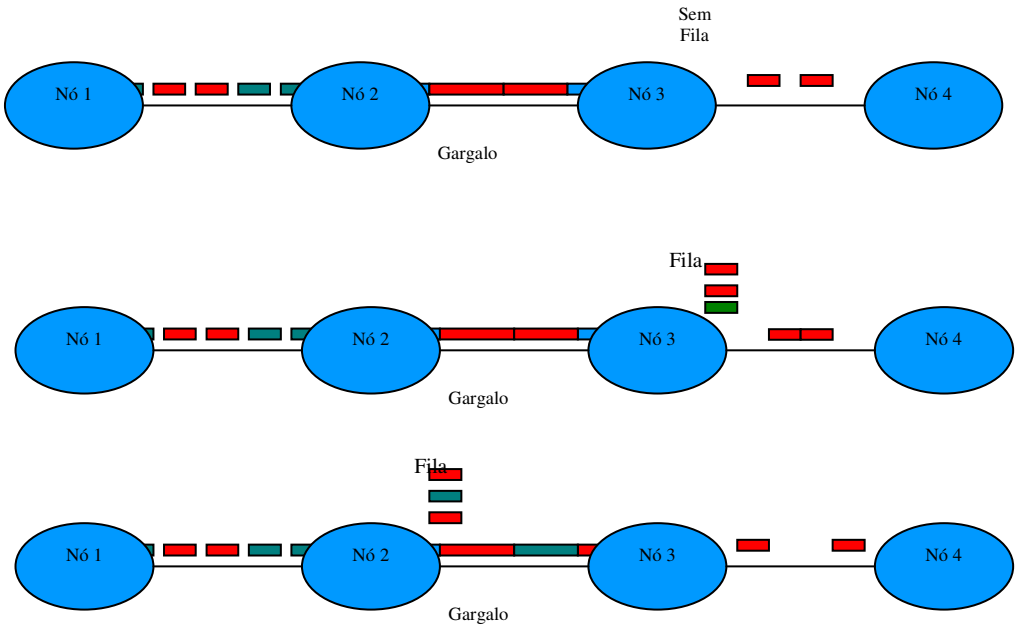
Figura 5: Se a taxa de envio está abaixo da taxa de serviço no enlace de gargalo, o tempo dos pacotes é mantido, se não uma fila é formada e os pacotes atrasados



Isto acontece devido à existência de tráfego concorrente. Se a fila do tipo fair-queueing for usada, então cada fluxo terá uma parte da banda de forma justa, pois o fair-queueing efetivamente isola cada fluxo do efeito das rajadas dos outros tráfegos. Logo, o par de pacotes irá medir a banda alocada para o fluxo. Entretanto, se a política de fila for FIFO

(First In First Out), um comportamento mais complexo ocorre. Como pode ser observado na [Figura 6], dois efeitos aparecem: compressão do tempo, quando existe uma fila no roteador e o primeiro pacote fica atrasado e expansão do tempo, quando um ou mais pacotes ficam entre o primeiro e segundo pacotes do par de pacotes. Os dois efeitos podem afetar os mesmos pacotes em roteadores diferentes ao longo do caminho, o que gera uma grande dispersão nos valores medidos no receptor [MA005].

Figura 6: O tempo de chegada de dois pacotes consecutivos pode mudar de pendendo das condições da rede



3.6.3 CONTROLE HOMEOTÁTICO (HCC)

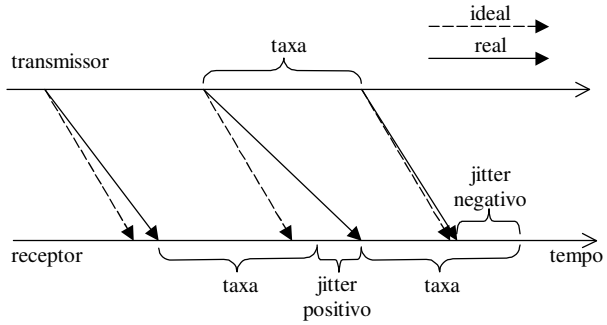
O conceito de controle homeostático surge com base em dois mecanismos - par de pacotes e monitoramento do jitter - utilizados para alcançar o equilíbrio (homeostase). Enquanto o método de par de pacotes superestima a banda, o monitoramento do jitter verifica a sobrecarga da rede e diminui a banda usando a média dos jitters.

Em virtude da existência das filas dos roteadores, estes conseguem durante um certo intervalo de tempo receber uma quantidade de pacotes maior do que a capacidade que estes têm de encaminhá-los. Assim, a taxa máxima de serviço pode ser violada durante um tempo limitado.

Como o TCP tem sua transmissão baseada em acks, ele, naturalmente, gera rajadas que são enfileiradas nas filas antes de serem transmitidas. Porém, em caso de muitas rajadas ao mesmo tempo ou uma rajada muito longa haverá perdas.

Sendo o jitter a diferença entre o intervalo de envio dos pacotes e o intervalo no qual estes pacotes são recebidos [Figura 7], existe a possibilidade de jitters positivos e jitters negativos (isso vai depender de como estará a rede no momento).

Figura 7: Exemplo de jitter positivo e jitter negativo



O HCC usa o monitoramento de jitter para verificar se a taxa está sendo violada, pois jitters positivos mostram que os pacotes estão ficando atrasados em relação aos pacotes anteriores e, conseqüentemente, que a rede está congestionada. Este congestionamento pode estar sendo gerado inclusive por tráfego concorrente.

Para o HCC, dois jitters positivos é sinal de violação da taxa, assim uma nova taxa deve ser calculada (diminuída), sendo esse cálculo feito através das informações dos dois pacotes citados anteriormente.

Vale ressaltar, porém, que o monitoramento do jitter impede que a rede entre no estado de saturação, mas não impede uma possível perda de pacote.

Já para técnica do par de pacotes, o HCC funciona enviando trens de cinco pacotes onde os dois últimos são chamados de pacotes de teste (probe packets) e são responsáveis em medir a banda acessível. A medição do tempo de chegada entre os dois pacotes de teste indica o tempo mínimo de separação entre pacotes que a rede pode atingir.

Como o par de pacotes mede a capacidade do enlace e não a banda acessível, para impedir uma mudança abrupta, o novo valor é misturado com o valor corrente. Se for dada preponderância a valores antigos, a curva será mais suave, e mais lentamente o algoritmo vai convergir para o novo valor. É importante citar que mudanças rápidas podem levar a oscilações e ultimamente, a congestionamentos.

Em caso de perdas, o HCC se comporta igual ao TCP, ou seja, utiliza o mecanismo de decremento multiplicativo para reduzir a taxa à metade sempre que houver uma perda.

3.6.4 ALGORITMO

O algoritmo do HCC, mostrado pela primeira vez em [MA005], é constituído das seguintes fases.

1- Aumento exponencial

2- Prevenção de congestionamento

3- Controle de congestionamento

Na fase de aumento exponencial, como o próprio nome já diz, busca-se a chegada até a banda disponível o mais rápido possível. Com isso os pares de pacotes são enviados uma vez a cada cinco pacotes. O período em que cada pacote é enviado acontece de acordo com a equação (1) a seguir. O erro ou a diferença entre o período ótimo e o período corrente é dado pela equação (2).

$$P_{n+1} = (1 - \alpha) * P_n + \alpha * P_{Medido} \quad (1)$$

$$\alpha \in [0,1]$$

$$Erro = ((1 - \alpha)^n) * (P_0 - P_{Ótimo}) \quad (2)$$

A fase prevenção de congestionamento acontece quando uma seqüência de jitters positivos é detectada, mostrando assim que a rede está ficando carregada, e as filas nos roteadores estão aumentando. Portanto, nesta fase, o período é corrigido pelo erro entre o período corrente e o período ótimo. O tamanho do erro é dado pela soma dos jitters, pois o jitter é causado pela diferença entre o que foi medido e as variações que os pacotes sofrem na rede. Então o novo período é calculado de acordo com a equação (3).

n – número de medições

$$P_{n+1} = P_n + (jitter_1 + ... + jitter_n) / n \quad (3)$$

onde n é igual a 2 ou 3

Na terceira fase, controle de congestionamento, o protocolo notificou que a rede está congestionada, através de relatos de perdas. Então, o protocolo inicia o decremento multiplicativo, dobrando o período de envio a cada RTT de acordo com a expressão abaixo:

Se (tempo_corrente > tempo_última_perda + RTT + 2 * P_n)

$P_{n+1} = P_n * 2$

Sendo P_C o período corrente, P_{Medido} o período medido e $P_{Ótimo}$ o período ótimo, temos três casos.

1- Muito pequeno: $P_{Medido} < (P_{Ótimo} - (1 - \alpha) * P_C) / \alpha$

2- Lugar Ideal: $P_{Ótimo} > P_{Medido} > (P_{Ótimo} - (1 - \alpha) * P_C) / \alpha$

3- Muito Grande: $P_{Medido} > P_{Ótimo}$

Se o período é muito pequeno (caso 1), a taxa será grande (alta) e pode causar congestionamento. Se o período é muito grande (caso3) o protocolo não atingiu a vazão máxima. Se o período cair na área ótima (caso2), o novo período será parecido com o corrente, o que resultará no uso da banda acessível do enlace. A natureza homeostática do controle congestionamento atua na correção dos erros nas medições. Como existe uma tendência de super estimar a banda acessível, normalmente as medições irão cair nos casos 1 e 2. O caso 2 é ótimo. O caso 1 ocasionará jitters positivos e a correção de jitters entrará em ação, diminuindo a taxa. O caso 3 é mais raro e apenas temporariamente diminui a banda, até a próxima medição.

3.7 RESUMO

Neste capítulo foi feita uma explanação teórica do protocolo de transporte baseado em taxa conhecido como RMTP. Após a apresentação teórica dos protocolos de transporte abordados nesta tese (capítulo 2 e capítulo 3) serão feitas a seguir (capítulo 4) simulações com estes protocolos utilizando o simulador de rede conhecido como NS-2. Estas simulações têm o intuito de observar o comportamento destes protocolos em ambientes de alta velocidade e principalmente, verificar se o protocolo RMTP, desenvolvido inicialmente para redes sem fio, tem condições de ser utilizado em redes de alta velocidade.

4 EXPERIMENTOS REALIZADOS

4.1 APRESENTAÇÃO

Nos capítulos anteriores foi vista uma introdução teórica sobre os protocolos de transporte abordados nesta tese. Após isso, através de simulações, este capítulo terá como objetivo confirmar algumas suposições feitas durante os três primeiros capítulos e revelar novos conhecimentos para a área de protocolos de transporte em redes de alta velocidade.

4.2 INTRODUÇÃO

Neste capítulo serão apresentados experimentos para verificar três questões importantes na análise de protocolos de transporte. Primeiramente, será mostrado o desempenho de cada protocolo operando sozinho, gerando assim uma linha de base ou referência (baseline). Para isso foram realizados testes com cada protocolo separadamente e sob diversas condições de rede, mantendo os parâmetros dos enlaces de acesso constantes e variando os parâmetros do enlace de backbone.

O segundo experimento tem por objetivo analisar a estabilidade dos protocolos. A topologia definida para este teste é constituída de seis enlaces de acesso (um para cada protocolo) convergindo para um backbone que opera com sobrecarga.

Por último, serão feitas simulações com os protocolos para redes de alta velocidade e o RMTP, verificando a característica de ser amigável ao TCP de cada um destes protocolos sob uma determinada condição de rede.

4.3 AMBIENTE DE TESTE

Sendo o ambiente de teste utilizado neste trabalho o simulador NS-2, será feito a seguir um histórico de todas as fases de montagem deste ambiente. A máquina utilizada para a instalação do NS-2 foi um Pentium 4 com 512M de memória RAM, com o sistema operacional Linux (distribuição Slackware) e versão 2.4.26 no Kernel. Esta máquina é de propriedade do Laboratório Midiacom da Universidade Federal Fluminense.

4.3.1 INSTALAÇÃO DO NS-2 (NETWORK SIMULATOR)

O NS-2 (Network Simulator) é um simulador de rede que oferece suporte para o estudo de redes de computadores, tendo sido usado em várias pesquisas. Atualmente, o NS-2 possui uma variedade de módulos tais como protocolos de transporte, protocolos de multicast sobre redes com fio e redes sem fio (locais ou satélites), e protocolos de roteamento entre outros. É interessante informar também que uma nova versão NS-3 já está em desenvolvimento.

O NS-2 foi criado como uma variante do REAL Network Simulator em 1989, e tem sido desenvolvido continua e substancialmente nestes anos. Em 1995, o NS-2 passou a ser desenvolvido pela DARPA (Defense Advanced Research Projects Agency) através do projeto VINT (Virtual InterNetwork Testbed) na LBNL (Lawrence Berkeley National Laboratory), XEROX Parc (Palo Alto Research Center), UCB (University of California Berkeley) e na USC/ISI (University Southern California / Information Sciences Institute).

Atualmente o NS-2 é desenvolvido principalmente através da DARPA com a SAMAN (Simulation Augmented by Measurement and Analysis for Networks) e através da NSF (National Science Foundation) com a CONSER (Collaborative Simulation for Education and Research), todos em colaboração com a ICIR (ICSI Center for Internet Research), apesar da UCB Daeddelus, CMU Monarch e Sun Microsystems estarem também realizando grandes contribuições na área de redes sem fio.

O código do NS-2 é constituído de duas partes: a primeira parte é escrita em C++ e conseqüentemente orientada a objeto, já a segunda parte é escrita em OTcl que é uma linguagem de script também orientada a objeto.

A versão do NS-2 utilizada neste trabalho foi a 2.26. Para a instalação desta foi acessado o sítio http://www.isi.edu/nsnam/dist/ns-allinone-2.26.tar.gz e feito o download do arquivo ns-allinone-2.26.tar.gz com o código fonte do simulador. Vale a pena citar também que o sítio http://www.isi.edu/nsnam/ns/ possui todas as informações para a instalação do NS-2 além de outros tópicos também interessantes para o uso do simulador.

Após a instalação do NS-2 verificou-se a necessidade de instalação do TCP/SACK-TS (Selective Ack com a opção Time Stamp) para um melhor desempenho da simulação em redes de alta velocidade. Esta característica de transmissão do TCP encontra-se no sítio http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/sack-ts-ns/sack-ts-code.tar.gz .

Feito o download do arquivo sack-ts-code.tar.gz foi feita a descompactação deste e copiou-se os arquivos originados da descompactação no diretório ns-allinone-2.26/ns-2.26/tcp dentro do NS-2. O arquivo makefile, que se encontra em ns-allinone-2.26/ns-2.26, foi editado e na lista dos arquivos objetos “OBJ_CC” inseriu-se a seguinte linha: tcp/sacklist.o tcp/intdlist.o tcp/hashtable.o tcp/tp-sack-ts.o tcp/scoreboard-ts.o. Esta linha fez com que o NS-2 gerasse os arquivos objetos dos arquivos de extensão .cc e .h que foram originados da descompactação do arquivo sack-ts-code.tar.gz. Para que a modificação acima tivesse efeito no momento de execução do simulador, foi feita a recompilação do NS-2. Para isso em ns-allinone-2.26/ns-2.26 digitou-se “make clean” <enter> e em seguida “make” <enter>.

Deve-se citar também que o controle de congestionamento do agente TCP utilizado em todas as simulações desta tese foi o do TCP Reno com a característica de acks seletivos.

4.3.2 INSTALAÇÃO DOS PROTOCOLOS HSTCP, BIC TCP E CUBIC TCP

Para a instalação dos controles de congestionamento dos protocolos HSTCP, BIC TCP e CUBIC TCP no NS-2, foi feito o download do arquivo tcp.cc no sítio http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/tcp.cc e do arquivo tcp.h no sítio http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/tcp.h . Após isso

substituiu-se os arquivos tcp.cc e tcp.h existentes, pelos arquivos baixados e recompilou-se o NS-2, fazendo com que, todo e qualquer agente TCP criado no NS-2 a partir deste momento estivesse submetido às regras destes novos arquivos.

Estes novos arquivos possuem não só o controle de congestionamento do TCP como também o controle de congestionamento dos protocolos de alta velocidade acima citados, ou seja, o NS-2 passa a aceitar não só os agentes já existentes antes da instalação destes novos arquivos como os novos agentes para redes de alta velocidade.

4.3.3 INSTALAÇÃO DO PROTOCOLO MULTCP

Foi solicitado ao Sr. Nabeshima, projetista do protocolo MulTCP, o algoritmo do controle de congestionamento do protocolo MulTCP para o NS-2. Após receber por correio eletrônico a resposta da solicitação, introduziu-se este novo controle de congestionamento nos arquivos tcp.cc e tcp.h citados no item 4.3.2 criando assim uma terceira versão destes arquivos com os controles de congestionamento dos quatro protocolos de alta velocidade estudados durante este trabalho.

Sabendo que o MulTCP utiliza N fluxos TCP em paralelo aumentando assim a capacidade de vazão de uma determinada aplicação, é importante citar que o valor de N em todas as simulações feitas nesta tese é quatro.

4.3.4 INSTALAÇÃO DO PROTOCOLO RMTP

Em [MA005], Magalhães descreve um controle de congestionamento para o NS-2 igual ao controle de congestionamento do RMTP. Além do RMTP, este controle de congestionamento também é utilizado em um outro protocolo de transporte também desenvolvido por Magalhães conhecido como MMTP (Multimedia Multiplexing Transport Protocol) [MA005].

Após a obtenção dos arquivos do RMTP (rmtp.cc e rmtp.h) foi preciso realizar várias configurações no NS-2 para que este conseguisse executar o RMTP. Para isto foi utilizado um tutorial conhecido como - Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns" - que é encontrado no sítio http://www.isi.edu/nsnam/ns/tutorial. Analisando o tutorial acima citado verificou-se a necessidade das seguintes alterações:

1- Como um novo tipo de agente foi criado, foi necessário que no arquivo packet.h do NS-2 fosse inserido este novo tipo de pacote relativo ao novo agente

2- Definiu-se uma entrada para os novos pacotes do RMTP no arquivo ns-packet.tcl do NS-2

3- No arquivo makefile incluiu-se na lista de arquivos objetos a serem gerados no momento de compilação o arquivo rmtp.o

4- Após estas alterações foi feita uma recompilação do código do NS-2 para que este conseguisse gerar simulações com o novo protocolo inserido.

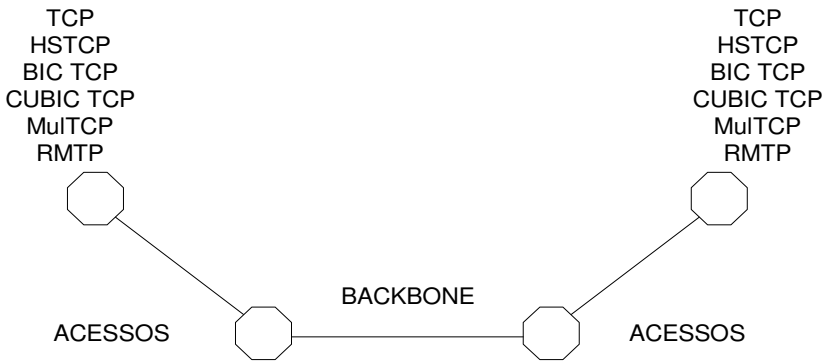
É importante citar que estas alterações foram necessárias apenas para o RMTP e não para os protocolos de alta velocidade derivados do TCP, pois no caso do TCP foram inseridos apenas novos controles de congestionamento (todos utilizando o mesmo agente do TCP) enquanto que para o RMTP foi inserido um novo protocolo e conseqüentemente um novo agente para o NS-2.

4.4 TESTE DE VAZÃO COM TCP, HSTCP, MulTCP, BIC TCP, CUBIC TCP e RMTP

Como primeira bateria de testes, foram feitas simulações com todos os seis protocolos um de cada vez em uma rede gigabit. Estes testes têm como principal objetivo verificar qual dos protocolos possui maior vazão de pacotes.

No capítulo 2, foi dito que, como o TCP não possui capacidade de encher enlaces da ordem de gigabit por segundo surgiram novos protocolos fim a fim para atender a esta necessidade. Devido a esta afirmação o teste desta seção foi considerado o mais importante, pois mostrou qual dos protocolos de transporte foi capaz de através de uma única conexão de transporte, melhor utilizar a banda disponível pela camada de rede.

Figura 8: Topologia para os testes de vazão



Nesta primeira fase de testes foi utilizada uma topologia simples [Figura 8], com apenas um acesso que será utilizado pelos seis protocolos um de cada vez. Além disso, foram feitos testes em várias condições de rede através de um script Otcl [Anexo I], portanto, apesar de manter constante a velocidade, o atraso e o tamanho da fila dos acessos, no backbone foram realizadas variações no atraso e no tamanho da fila com a intenção de simular um ambiente real de uma rede de dados. A Tabela 1 mostra como foi montado o teste desta seção.

Tabela 1: Situações de rede para teste de vazão

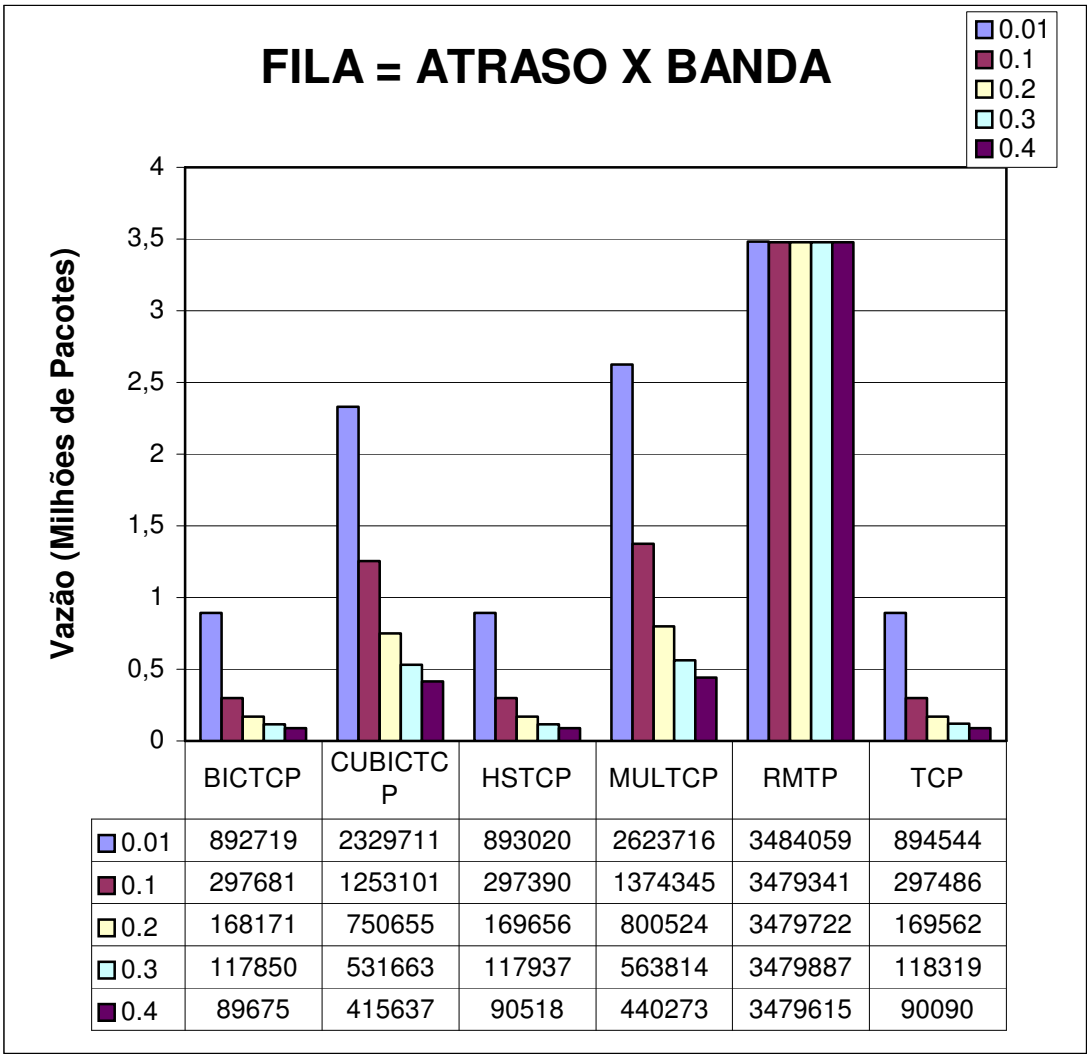
ACESSO			BACKBONE		
BANDA	ATRASO	BUFFER	BANDA	ATRASO	BUFFER
1Gbps	0,01 ms	produto atraso x banda	1Gbps	0,01 ms	produto atraso x banda
				0,1 ms	
				0,2 ms	
				0,3 ms	
				0,4 ms	
1Gbps	0,01 ms	produto 5 x atraso x banda	1Gbps	0,01 ms	produto 5 x atraso x banda
				0,1 ms	
				0,2 ms	
				0,3 ms	
				0,4 ms	

O valor 0,01 ms de atraso nos enlaces de acesso e no primeiro caso no enlace de backbone é um valor de atraso aceitável para redes LAN e WAN. Como existe na Internet

uma grande variação no atraso, foram realizados experimentos com este variando em 10, 20 30 e 40 vezes o valor de 0,01 ms. A partir da tabela acima chega-se aos seguintes gráficos [Figura 9] e [Figura 10]. Estes experimentos tiveram a duração de 30 segundos para cada situação descrita na tabela.

Duas observações importantes: a primeira é o tamanho do segmento TCP. Não só nesta seção como em toda a tese, o valor do segmento TCP foi de 1000 bytes. A segunda observação é que o algoritmo do controle de congestionamento utilizado nos agentes TCP de todas as simulações desta tese foi o TCP Reno com acks seletivos.

Figura 9: Gráfico com vazão dos protocolos utilizando buffer = atraso x banda

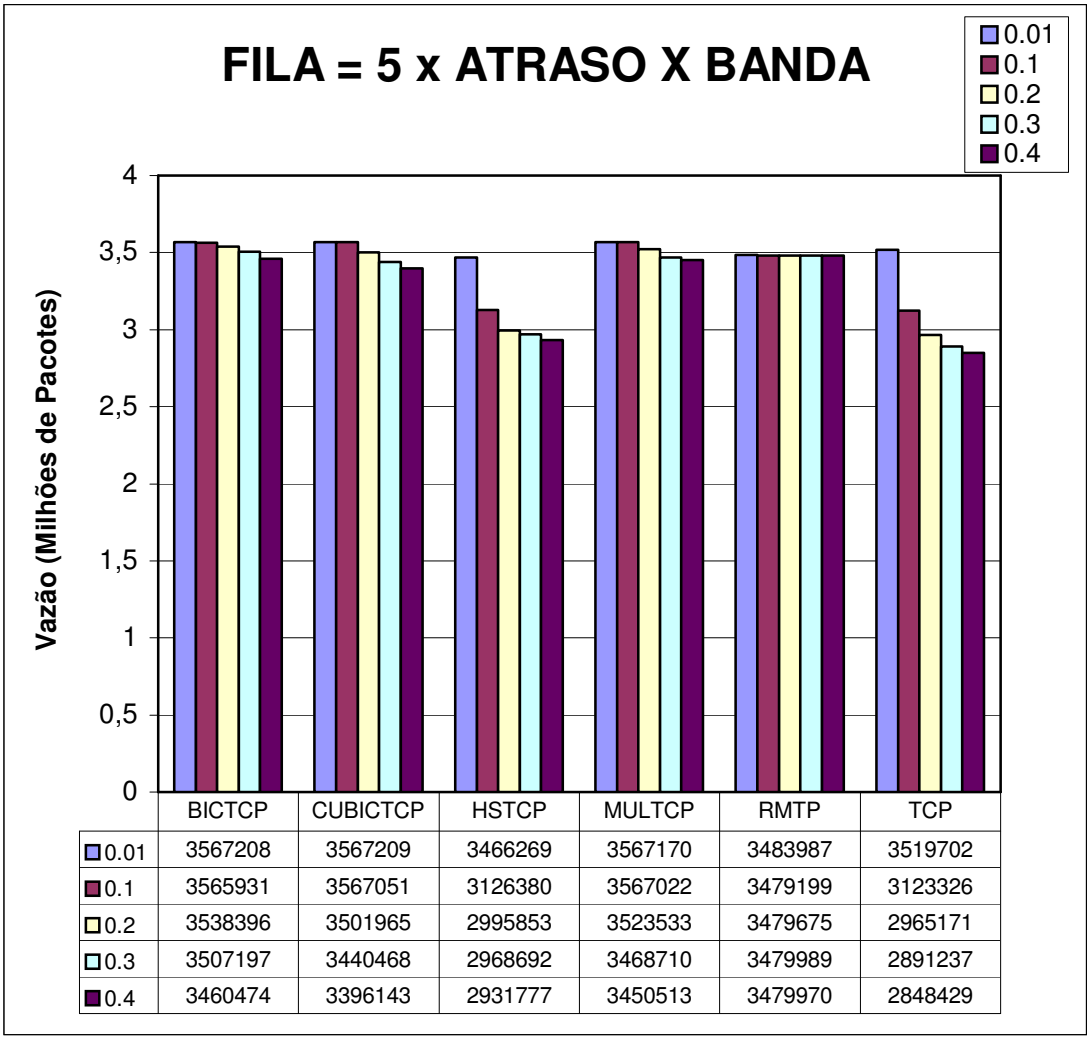


	BICTCP	CUBICTCP	HSTCP	MULTCP	RMTP	TCP
0.01	892719	2329711	893020	2623716	3484059	894544
0.1	297681	1253101	297390	1374345	3479341	297486
0.2	168171	750655	169656	800524	3479722	169562
0.3	117850	531663	117937	563814	3479887	118319
0.4	89675	415637	90518	440273	3479615	90090

Os dois gráficos desta seção mostram uma estabilidade do RMTP, pois mesmo na presença de uma variação de 40 vezes o atraso no enlace de backbone os fluxos RMTP conseguem se manter com a mesma capacidade de vazão.

Além disso, é verificado no segundo gráfico [Figura 10], que com o aumento do tamanho das filas nos enlaces de acesso e de backbone, os protocolos de alta velocidade conseguem atingir uma vazão semelhante à do RMTP, ou seja, com um valor de tamanho da fila sendo cinco vezes o produto atraso x banda consegue-se absorver as rajadas geradas pelos protocolos baseados em acks, o que ocasiona a diminuição de perdas nos fluxos destes protocolos e conseqüentemente uma melhor vazão atingida.

Figura 10: Gráfico com vazão dos protocolos utilizando buffer = 5 x atraso x banda



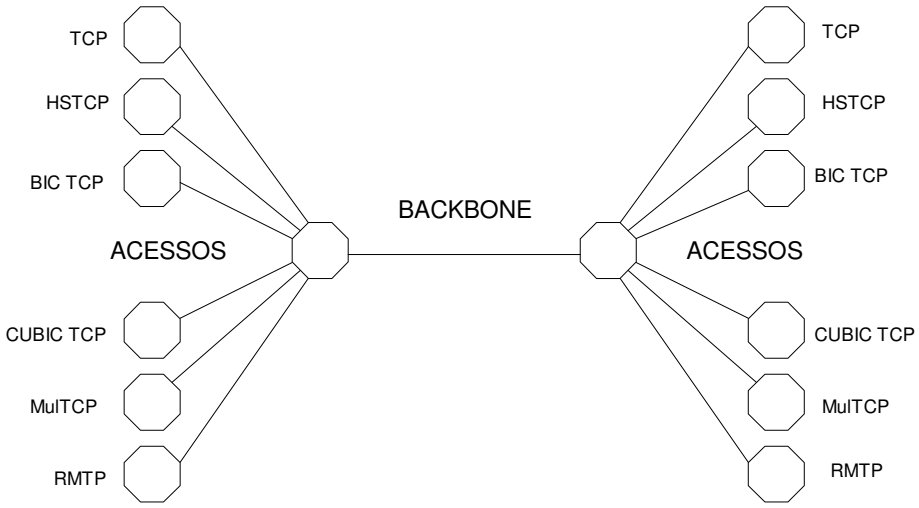
	BICTCP	CUBICTCP	HSTCP	MULTCP	RMTP	TCP
0.01	3567208	3567209	3466269	3567170	3483987	3519702
0.1	3565931	3567051	3126380	3567022	3479199	3123326
0.2	3538396	3501965	2995853	3523533	3479675	2965171
0.3	3507197	3440468	2968692	3468710	3479989	2891237
0.4	3460474	3396143	2931777	3450513	3479970	2848429

Ainda na análise do segundo gráfico observamos que o TCP e o HSTCP com os atrasos na ordem de 0.1 ms a 0.4 ms, não conseguem atingir uma vazão ótima mesmo com o tamanho da fila sendo cinco vezes o tamanho do produto atraso banda. Sendo o TCP o protocolo mais utilizado na Internet, esta é sem dúvida uma motivação para estudos na área de protocolos de transporte para redes de alta velocidade.

4.5 TESTE DE DESEMPENHO COM TCP, HSTCP, MulTCP, BIC TCP, CUBIC TCP e RMTP

Esta segunda fase de testes consistiu na verificação do protocolo de maior estabilidade e melhor desempenho em condições de alto tráfego de rede.

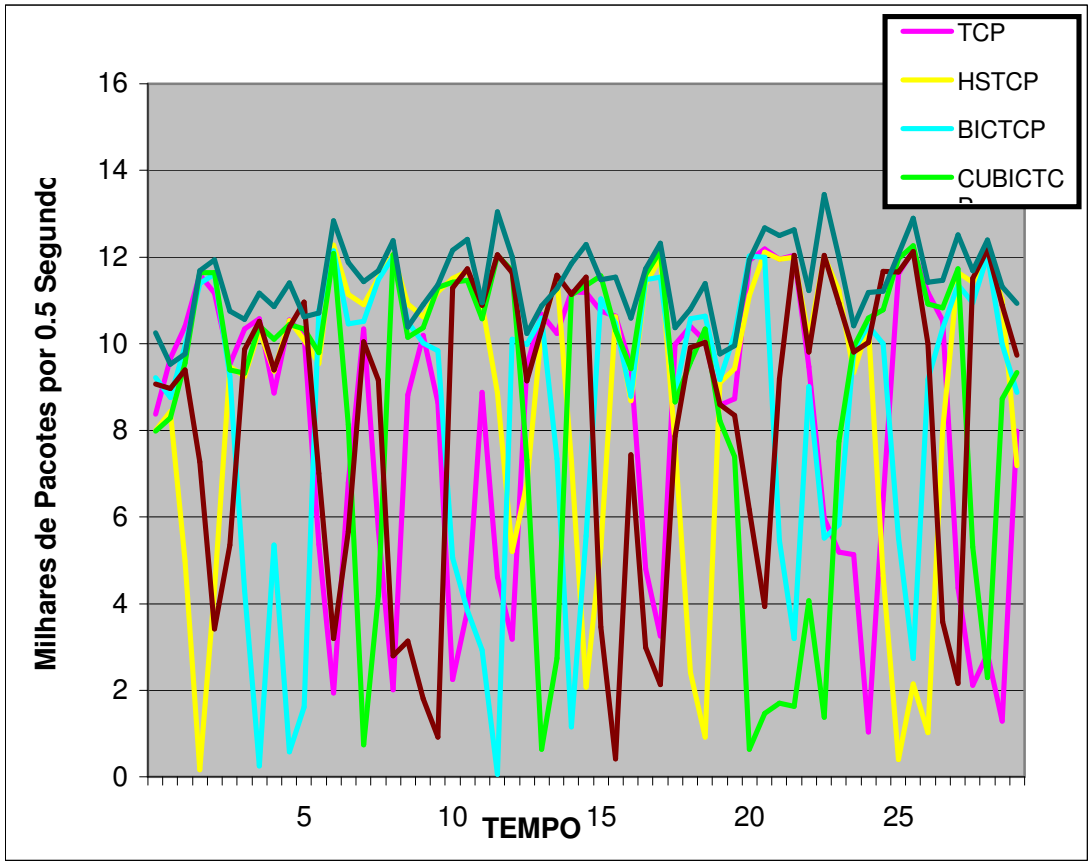
Figura 11: Topologia dos testes de desempenho



Assim, utilizando a topologia da Figura 11, criou-se uma condição de rede para uma avaliação de desempenho dos controles de congestionamento, quando estes disputam recursos da rede entre si. Para estes testes foi utilizado o script feito em linguagem OTcl [Anexo I] gerando seis fluxos de acesso sendo um fluxo de cada protocolo estudado com velocidade de 250 Mbps, passando por um backbone com uma velocidade de 1 Gbps. O atraso para os links de acesso foi de 0,2 ms enquanto que no backbone o atraso foi de 0,1 ms.

Para o tamanho das filas tanto nos enlaces de acesso como nos enlaces de backbone foi utilizado o produto atraso x banda. O tipo de fila também nos dois tipos de enlaces foi o drop tail. O tempo de duração deste teste assim como os testes do item anterior foi de 30 segundos.

Figura 12: Gráfico de vazão com todos os protocolos disputando banda



Analisando o gráfico da Figura 12, observa-se que o protocolo RMTP mantém uma banda maior que os outros (linha verde escura), isto se deve às características do HCC que é o controle de congestionamento do RMTP e também ao fato do RMTP ser baseado em taxa em vez de acks. Outra observação interessante é que além da banda utilizada do RMTP se manter maior que a dos outros protocolos esta também possui uma estabilidade maior que a dos outros protocolos.

Uma conseqüência do que foi dito acima é que o RMTP neste caso terá uma vazão maior que a dos outros protocolos. A Tabela 2 mostra a quantidade total de pacotes recebidos ao final dos 30 segundos de teste. Para uma melhor análise a tabela inteira encontra-se no Anexo III.

Tabela 2: Vazão dos protocolos no teste de desempenho

TCP	HSTCP	BIC TCP	CUBIC TCP	MulTCP	RMTP
478510	518364	493509	510688	490147	674833

Os testes das seções 4.4 e 4.5 tiveram como principal objetivo a comparação entre duas linhas de pesquisa de protocolos de transporte já citadas anteriormente, os protocolos baseados em acks (TCP, HSTCP, MulTCP, BIC TCP e CUBIC TCP) versus os protocolos baseados em taxa (RMTP).

Assim, pode-se observar que nos dois tipos de simulações feitas nas seções 4.4 e 4.5 o protocolo RMTP demonstrou uma capacidade em atender as necessidades de uma rede de alta velocidade. Pode-se dizer também que como o RMTP foi um protocolo inicialmente desenvolvido para redes sem fio, com um ajuste nos seus parâmetros para um ambiente de alta velocidade, pode-se obter resultados ainda melhores.

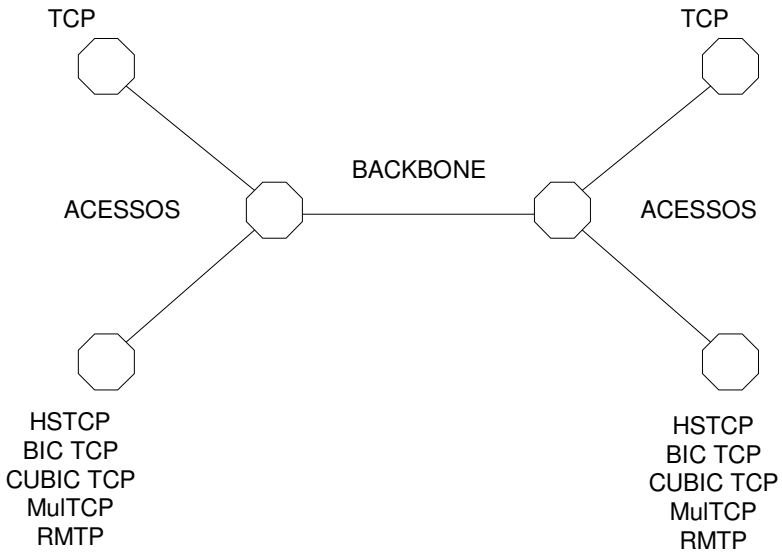
4.6 TESTES DE VERIFICAÇÃO DA CARACTERÍSTICA "TCP FRIENDLY" COM OS PROTOCOLOS HSTCP, MulTCP, BIC TCP, CUBIC TCP e RMTP

Nesta última seção de experimentos verificou-se o quanto os protocolos de alta velocidade são amigáveis ao TCP (TCP Friendly), ou seja, se em certas condições de rede estes protocolos conseguem compartilhar a banda oferecida com fluxos TCP sem deteriorá-los.

A priori, todos os protocolos utilizados são considerados amigáveis ao TCP, pois uma das maiores preocupações na implementação de um protocolo novo é esta. Porém a idéia aqui é fazer uma comparação entre os protocolos sob uma determinada condição de rede, ou seja, dos protocolos escolhidos (HSTCP, MulTCP, BIC TCP, CUBIC TCP e RMTP), descobrir qual possui a característica TCP Friendliness na situação de rede proposta.

A topologia utilizada para esta seção de testes [Figura 13], corresponde ao tipo mais simples, ou seja, duas conexões (uma TCP e uma de alta velocidade) passando por um backbone.

Figura 13: Topologia de testes de TCP Friendliness



Portanto, esses protocolos foram testados um a um com o TCP em duas condições de rede conforme a Tabela 3. Este ambiente de rede foi simulado através do script OTcl que se encontra no anexo II desta tese.

Tabela 3: Condições de rede utilizadas para testes TCP Friendliness

Condições de Rede	Banda Backbone	Atraso Enlace Backbone	Banda Protocolo TCP (Acesso)	Banda Protocolo A.V. (Acesso)	Atraso Links de acesso
1	1Gbps	0.01 ms	750Mbps	750Mbps	0.015 ms
2	1Gbps	0.01 ms	1Gbps	1Gbps	0.01 ms

Apesar de o valor da banda do backbone se manter constante, as bandas dos fluxos TCP e A.V. (Alta Velocidade) variam em dois valores (750Mbps e 1Gbps), assim, existem duas situações distintas de redes mostradas na Tabela 3. Na condição 1 existe um congestionamento razoável pois os dois fluxos de 750Mbps geram um excesso de banda de 50% no backbone. Já na condição 2 existe um alto grau de congestionamento, com a capacidade do backbone sendo ultrapassada em 100%.

Para o atraso de pacote na rede, os enlaces de acesso foram mantidos com os valores de 0,015 ms para a banda de 750Mbps e 0,01 ms para a banda de 1Gbps. O tamanho da fila escolhido para os seguintes experimentos, foi o produto atraso x banda dos enlaces multiplicado por cinco. Já o tipo de fila escolhido foi o drop tail. Este tipo de

comportamento da fila está presente na grande maioria dos roteadores da Internet. Além disso, foram feitos alguns testes com o tipo de fila RED (Random Early Detect) e não foram obtidas grandes diferenças em relação ao tipo drop tail.

Não de fez simulações com dois fluxos TCP pois estes já compartilham banda amigavelmente se todos os parâmetros da conexão forem iguais. Se os caminhos tiverrem RTTs diferentes, no entanto, o fluxo com menor RTT tende a capturar mais banda, pois reage mais rapidamente depois de um congestionamento.

Todas as simulações desta seção de testes tiveram 30 segundos de duração.

TCP X BICTCP

Figura 14: TCP x BICTCP sob Condição 1

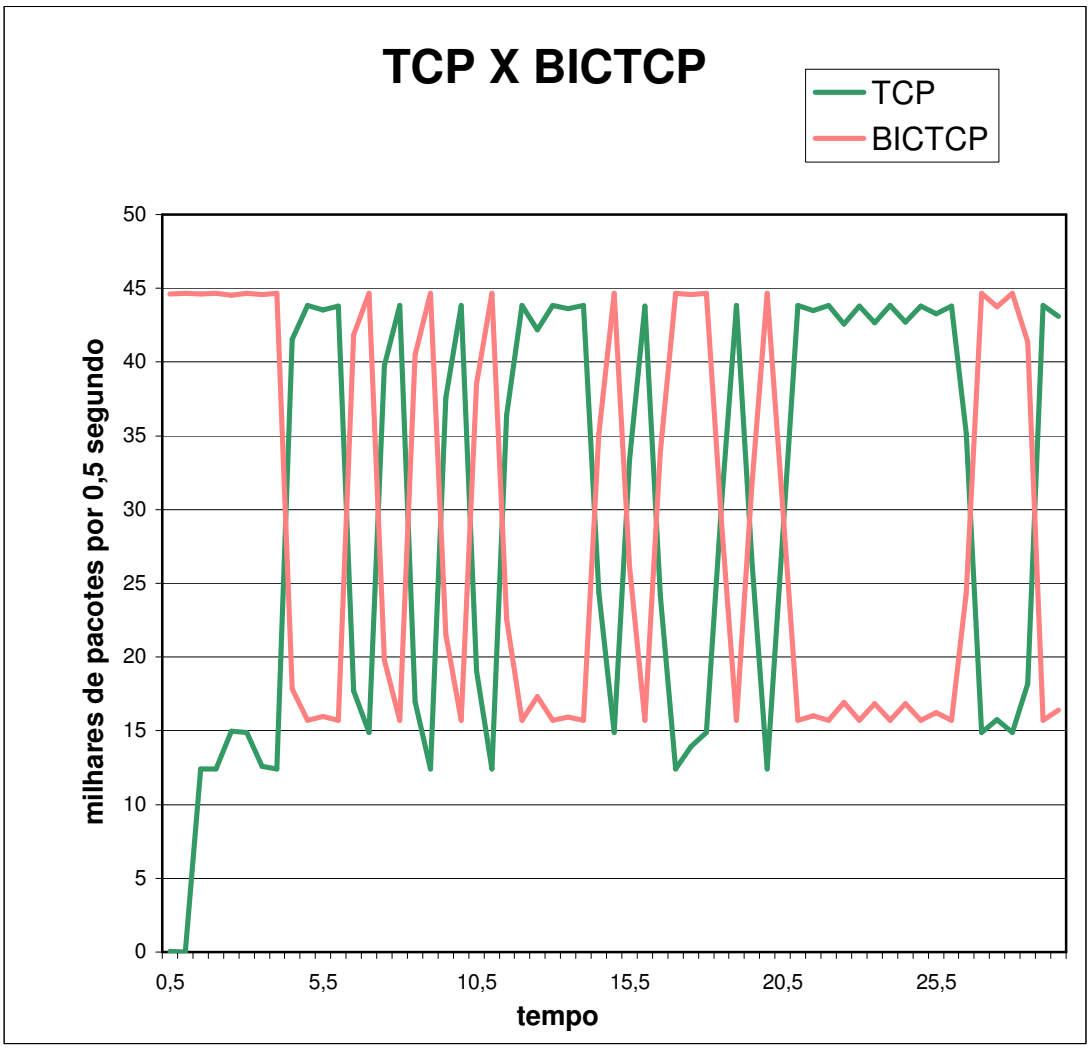
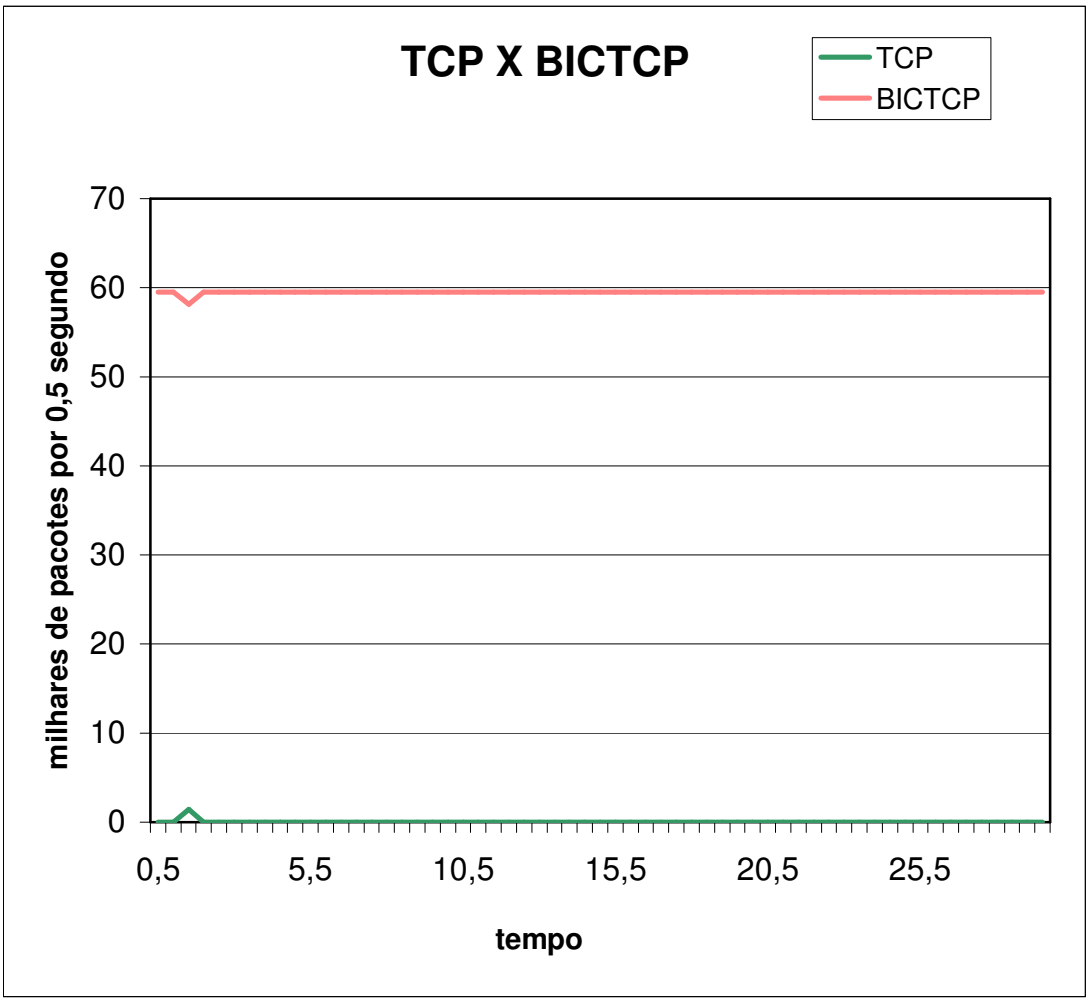


Figura 15: TCP x BICTCP sob Condição 2



Nesta seção pode-se verificar que apesar do BIC TCP ser amigável ao TCP na condição 1 [Figura 14], na condição 2 [Figura15], que possui alto grau de congestionamento, ele impediu a transmissão do fluxo TCP. Esta agressividade do BIC TCP foi um dos motivos que levou ao grupo de estudos deste protocolo, desenvolver uma nova versão do BIC TCP conhecida como CUBIC TCP que será analisada a seguir.

TCP X CUBICTCP

Figura 16: TCP x CUBICTCP sob Condição 1

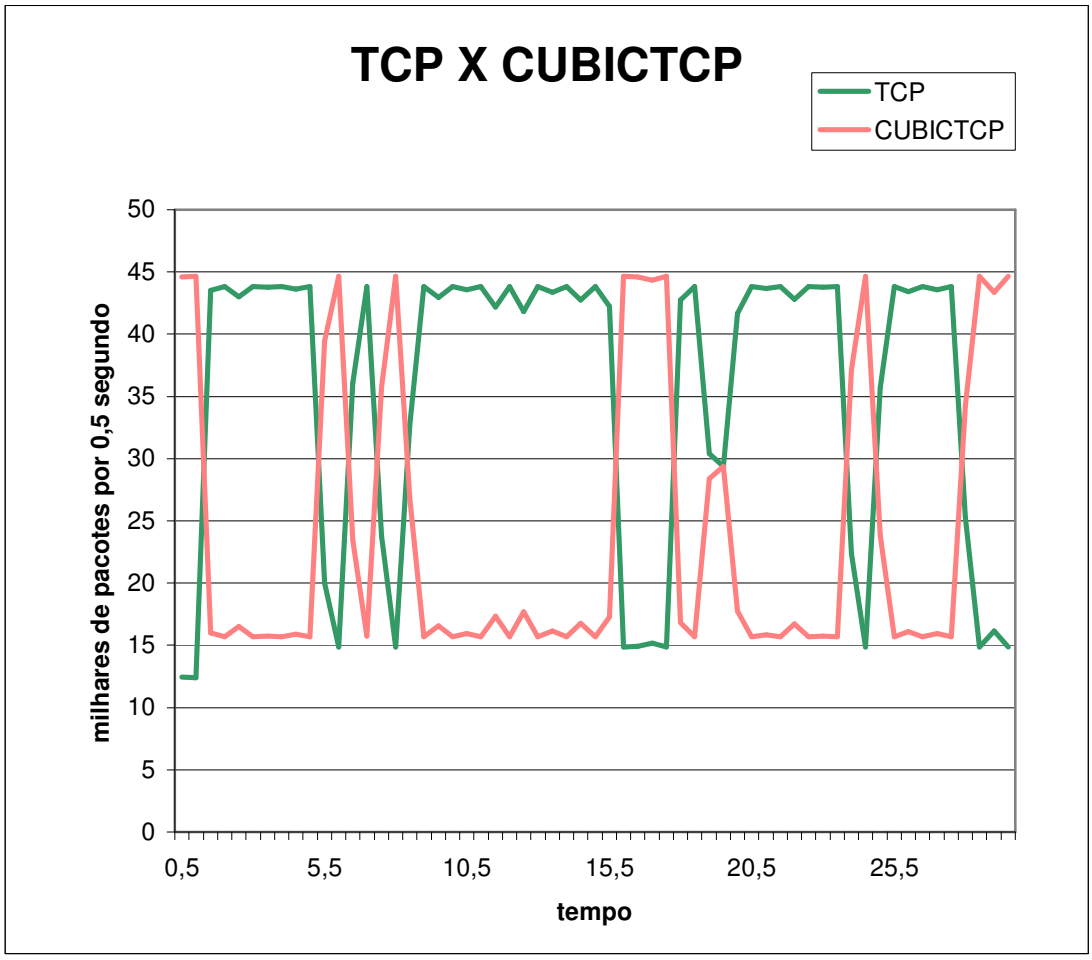
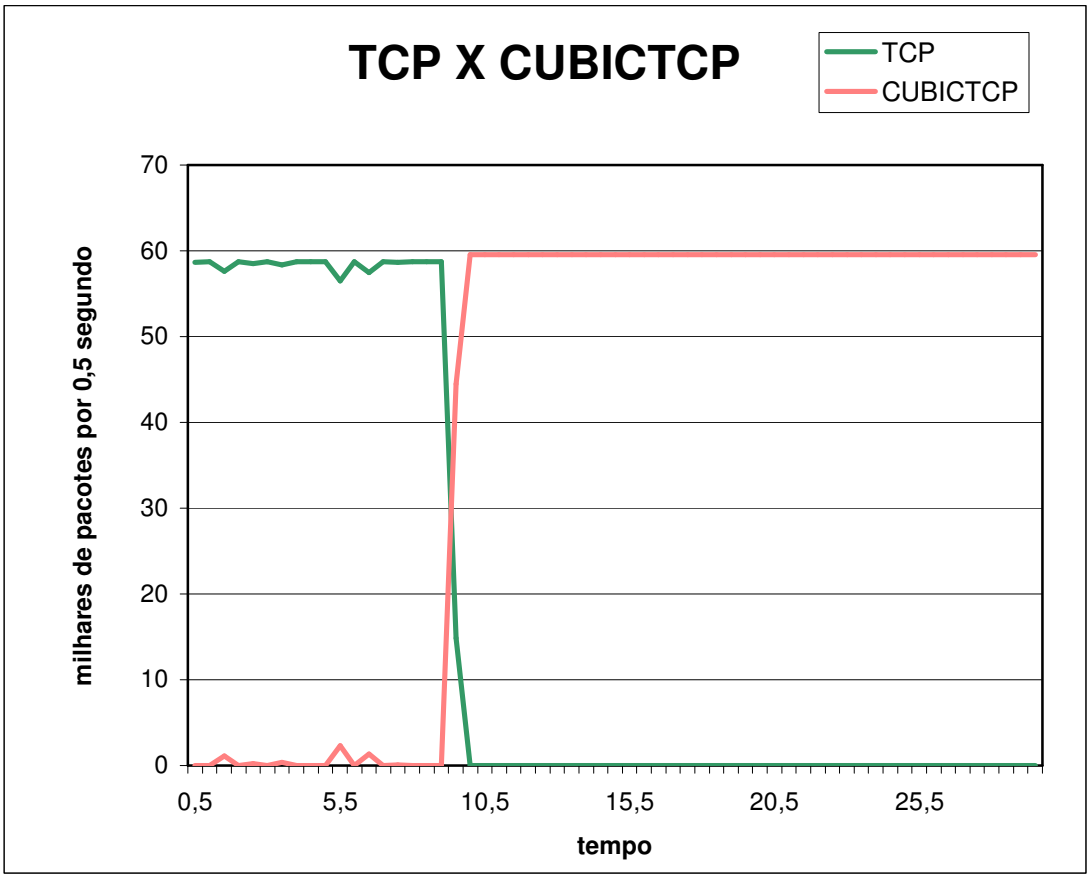


Figura 17: TCP x CUBICTCP sob Condição 2



Apesar do projeto do CUBIC TCP tentar ser mais amigável ao TCP que o BIC TCP, nas condições de alto grau de congestionamento, o fluxo TCP também é impedido de ser transmitido com cerca de 13 segundos de duração do teste.

Como os fluxos de protocolos baseados em acks em ambientes reais trabalham com rajadas, este valor de treze segundos passa a ser interessante, pois neste intervalo de tempo um dos dois ou até mesmo os dois fluxos podem diminuir a sua vazão e assim manter a rede capaz de escoar os dois fluxos ao mesmo tempo.

TCP X HSTCP

Figura 18: TCP x HSTCP sob Condição 1

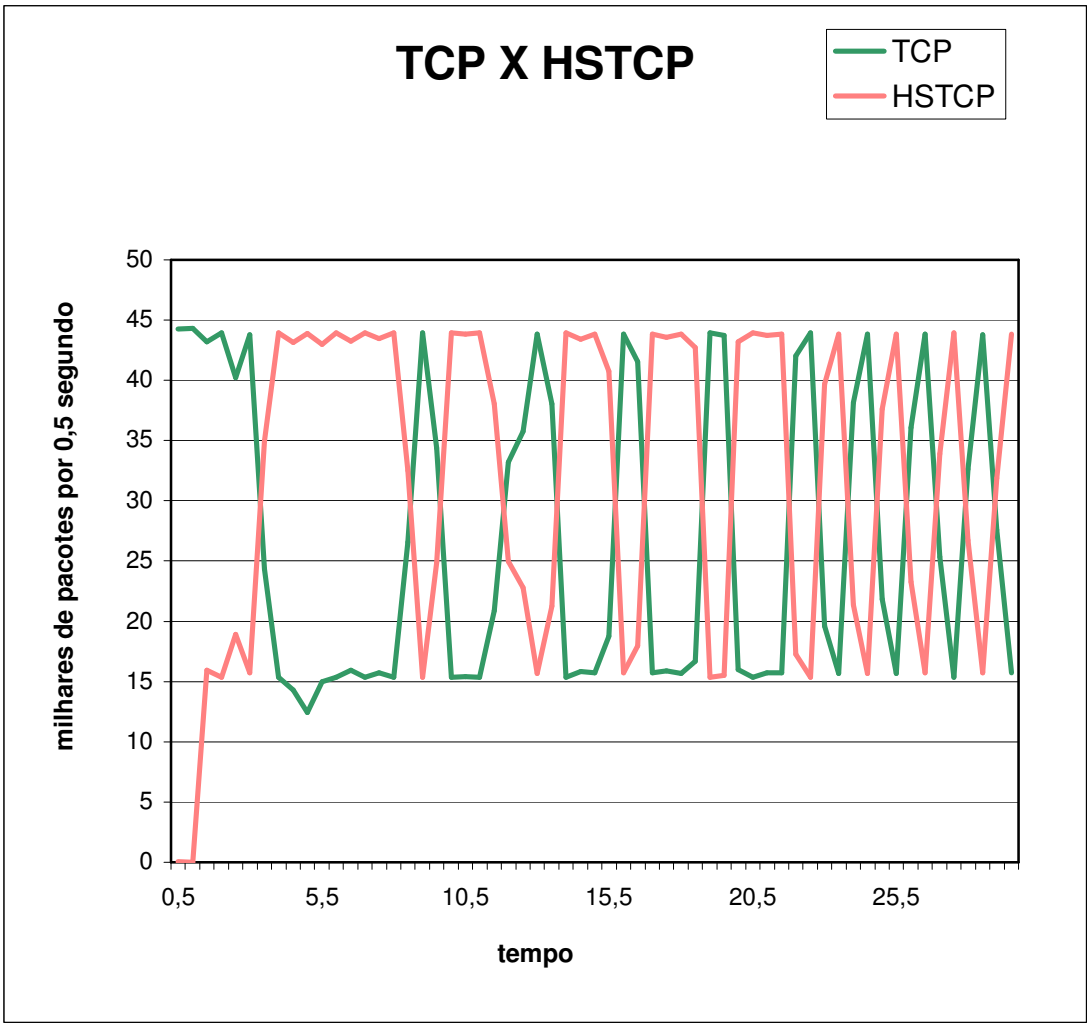
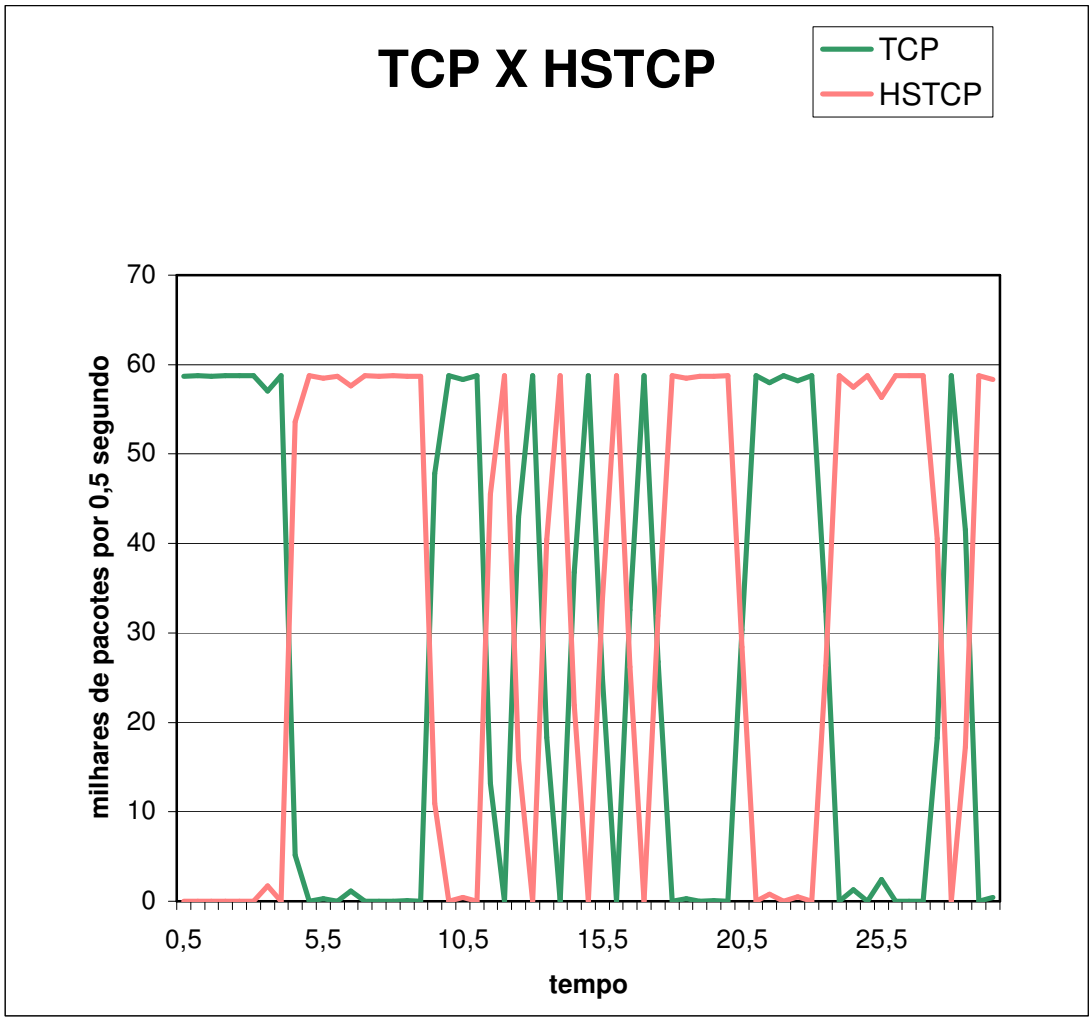


Figura 19: TCP x HSTCP sob Condição 2



Os gráficos acima sugerem que o HSTCP é o protocolo mais amigável ao TCP entre os protocolos abordados neste trabalho. Nas duas condições este protocolo se mostra extremamente adaptado às necessidades e limites da rede. Na condição 2 ele foi o único que manteve os dois fluxos transmitindo até o final do teste, ou seja não inviabilizou nenhum do dois fluxos existentes.

Esta conclusão já era esperada, pois o controle de congestionamento do HSTCP é quase igual ao do TCP. Outra questão interessante citada em [MA005] é a seguinte: “Para um protocolo ser completamente amigável ao TCP ele deve ser igual ao TCP”. No caso acima isto é quase uma verdade o que também justifica o bom rendimento do HSTCP nesta análise.

TCP X MulTCP

Figura 20: TCP x MulTCP sob Condição 1

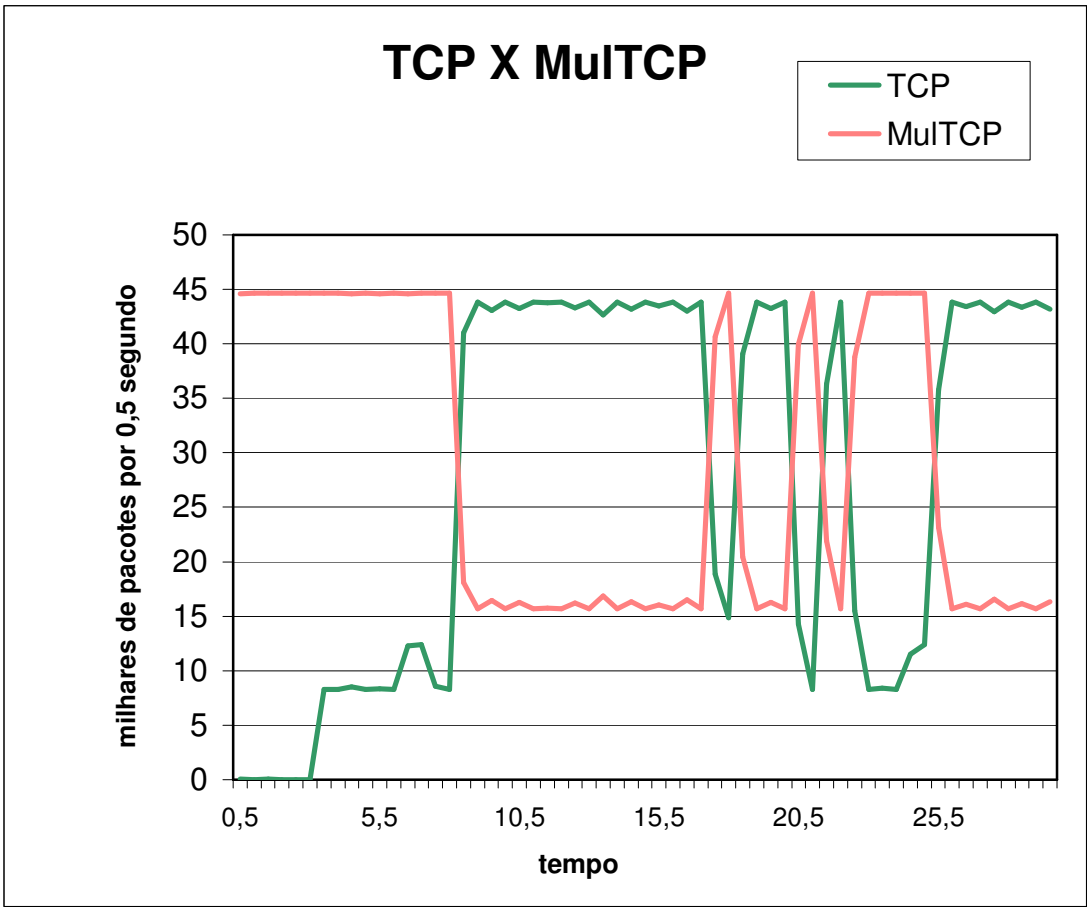
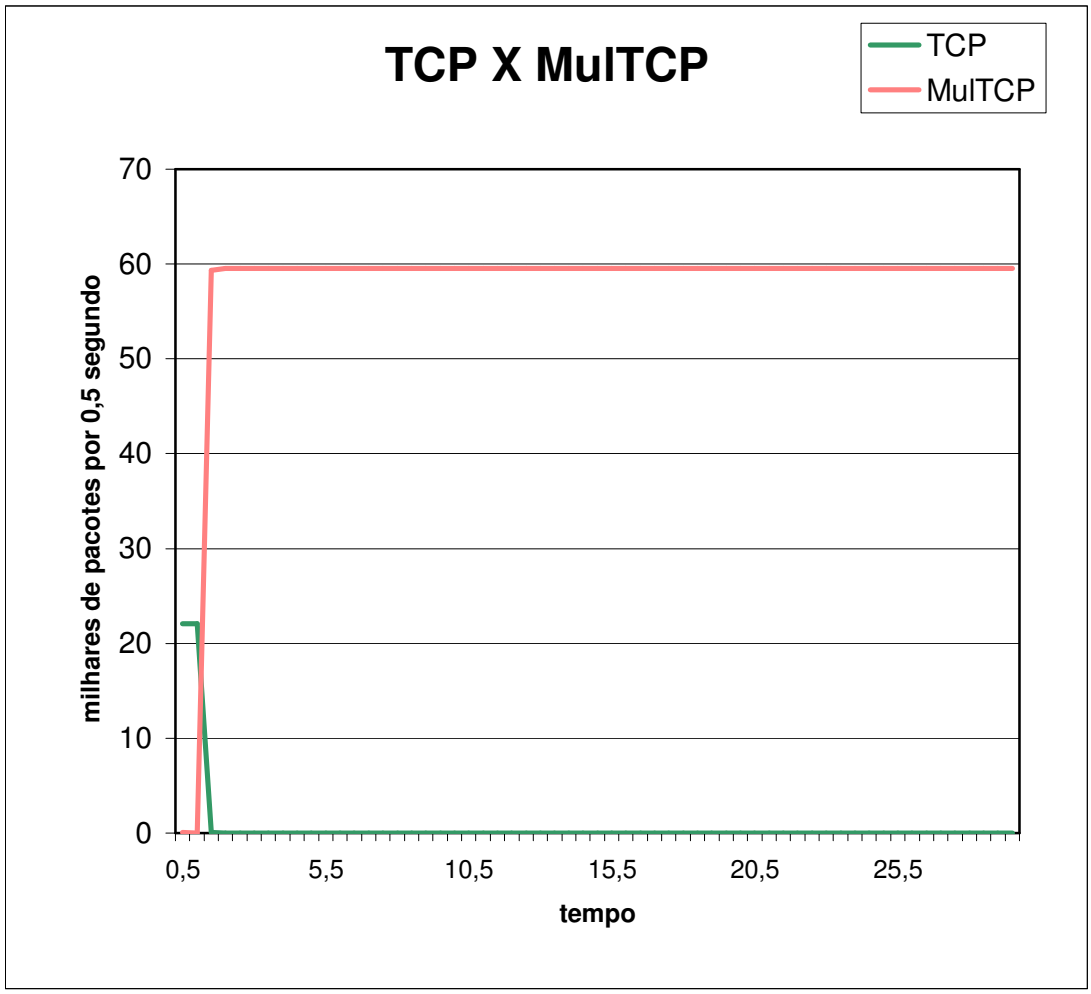


Figura 21: TCP x MulTCP sob Condição 2



O MulTCP, junto com o BICTCP, demonstrou ser o protocolo com o pior desempenho na questão amigável ao TCP. Isto já era esperado, pois como o MulTCP agrega fluxos TCP para uma única aplicação, a tendência será que este agregado de fluxos TCP tome sempre a banda com maior intensidade que um fluxo simples. Apesar de na condição 1 o MulTCP ter um desempenho razoável, a sua deficiência aparece na condição de alto congestionamento (condição 2), inviabilizando a transmissão do fluxo TCP durante os 30 segundos de simulação.

TCP X RMTP

Figura 22: TCP x RMTP sob Condição 1

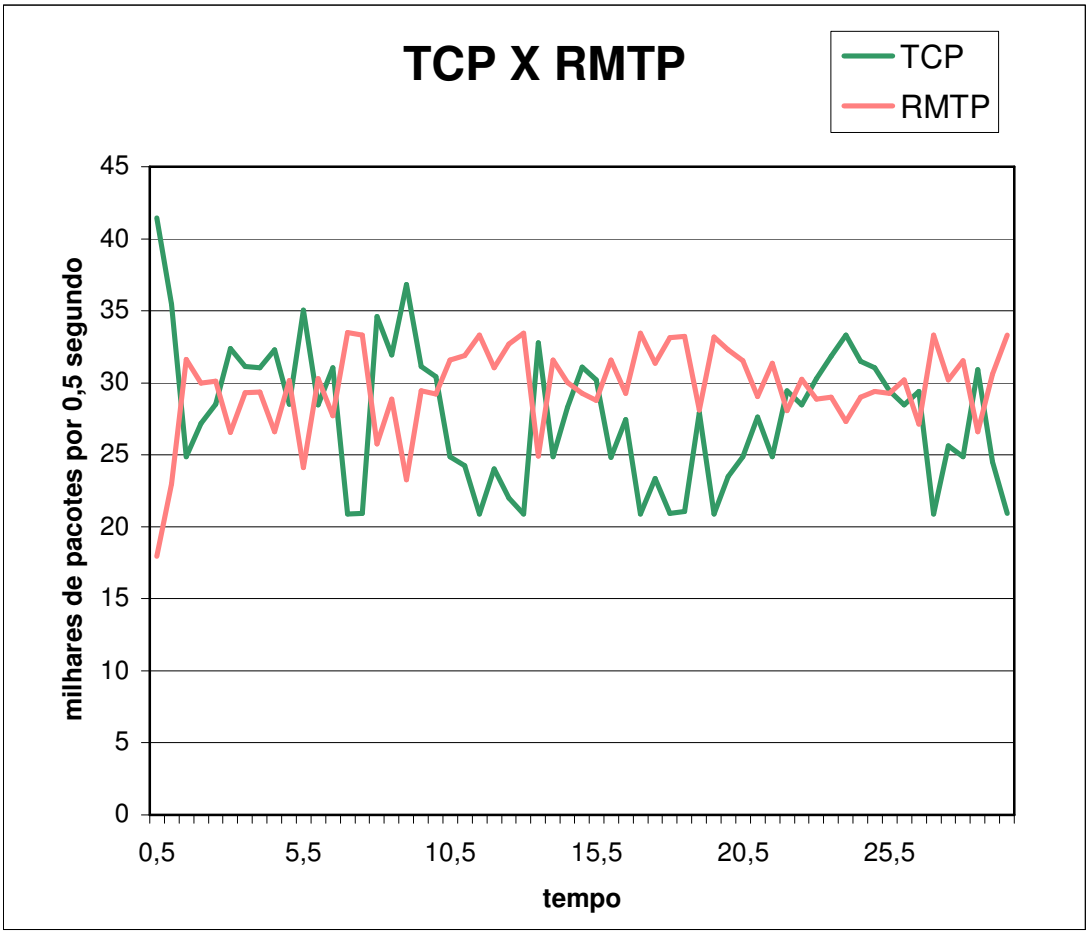
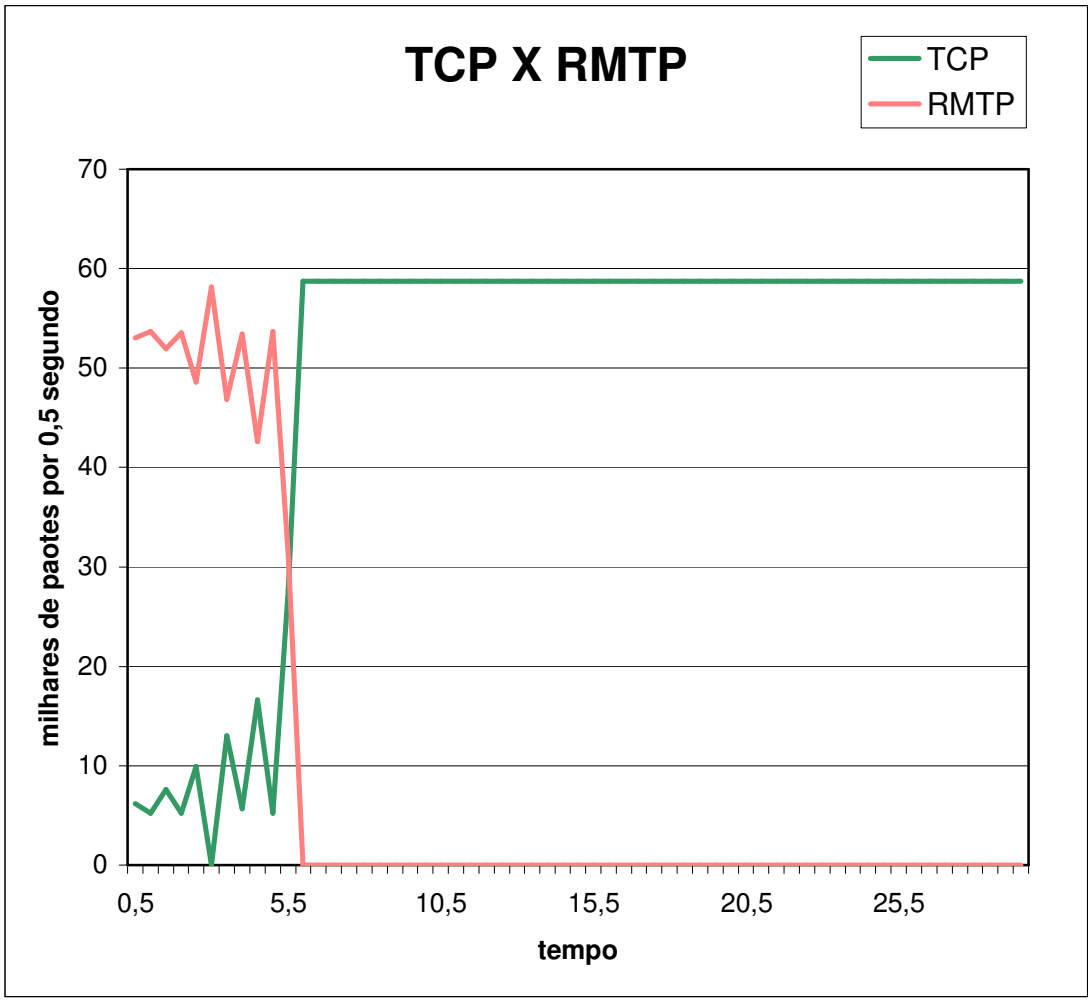


Figura 23: TCP x RMTP sob Condição 2



Apesar de na condição 2 o RMTP apresentar uma característica indesejada, pois em torno de 7 segundos o TCP impede a transmissão do fluxo RMTP. Na condição 1 o gráfico RMTP x TCP apresenta uma característica única em relação aos protocolos analisados. Tanto o fluxo RMTP como o fluxo TCP, possuem uma grande estabilidade, pois, a vazão dos dois fluxos fica em torno do valor de 30000 pacotes a cada 0,5 segundo. Pode-se dizer, neste caso, que o fato do fluxo RMTP ser baseado em taxa, e por isso possuir grande estabilidade faz com que o TCP também atue de forma mais estável do que quando comparado com os protocolos baseados em acks.

5 CONCLUSÃO

A principal contribuição deste trabalho foi analisar as características de um protocolo baseado em taxa, conhecido como RMTP, e compará-lo com outros protocolos desenvolvidos para redes de alta velocidade.

Esta tese é constituída basicamente de duas partes: a primeira parte faz uma introdução teórica dos protocolos abordados (TCP, HSTCP, MulTCP, BIC TCP, CUBIC TCP e RMTP) enquanto que a segunda realiza um estudo comparativo, através de simulações no NS-2, entre estes protocolos.

Após a bateria de experimentos do capítulo 4, acredita-se ter atingido o objetivo principal que era mostrar que apesar do protocolo RMTP ter sido projetado para um ambiente de redes sem fio e baixa velocidade, ele consegue, devido às características do seu controle de congestionamento e do seu modo de enviar pacotes para a rede, atender às necessidades de um protocolo de transporte para rede de alta velocidade.

Entende-se também que este modo de envio de pacotes do RMTP é essencial para o bom desempenho deste protocolo nos testes realizados, pois dado que o espaçamento regular dos pacotes gera uma carga menor nas filas dos roteadores, protocolos baseados em taxa são menos agressivos à rede, e isto resulta em uma característica de menor perda e maior estabilidade. Esta característica se mostrou tão interessante e marcante que no caso dos experimentos da seção 4.6 o RMTP chega a estabilizar o fluxo TCP (naturalmente um fluxo de rajadas).

Um ponto negativo do RMTP nestes testes é justamente a granularidade das variáveis no NS-2. Para hoje produzir-se estas simulações do capítulo 4 em uma rede real, é necessária uma granularidade do sistema operacional da ordem de micro segundos. Isto porém hoje em dia é conseguido apenas em sistema operacionais de tempo real (no caso do

Linux conhecidos como RT Linux) que são sistemas para situações específicas e conseqüentemente de difícil utilização em larga escala, limitando assim o uso do RMTP.

Uma solução para este problema seria introduzir um cartão de interface de rede (Network Interface Card – NIC) com capacidade de realizar o envio dos pacotes para rede com um intervalo de tempo entre pacotes capaz de gerar na rede velocidades na faixa de gigabit por segundo. Esta atividade deverá ser realizada em breve utilizando placas de rede da empresa conhecida como Big Foot Networks (www.bigfootnetworks.com) e como infra-estrutura a rede GIGA, uma rede experimental da RNP e do CPqD.

Outra questão interessante de análise é a vulnerabilidade do TCP e dos protocolos baseados em ack de não saberem lidar com as perdas, pois a queda de desempenho destes protocolos está ligada diretamente a uma característica intrínseca deles que é transmitir de forma sempre crescente os dados e reduzindo de forma agressiva a sua taxa quando da ocorrência de congestionamento.

Assim, os protocolos baseados em taxa podem ser uma alternativa a este problema, pois no caso do RMTP ele utiliza o HCC para verificar o congestionamento da rede. O HCC possui uma diferença básica em relação aos protocolos baseados em acks: antes de enviar os pacotes o HCC verifica qual a capacidade da rede para evitar o congestionamento, assim o HCC evita perdas e também a sua redução da taxa.

Pensando também nos novos serviços que estão sendo viabilizados na Internet tais como a TV Digital, pode-se considerar que fluxos com as características dos fluxos baseados em taxa tendem a ser mais interessantes para estes tipos de serviço, pois oferecem um espaçamento constante na chegada dos pacotes no receptor em contraste com o alto jitter dos fluxos baseados em acks.

Apesar do trabalho apresentado nesta dissertação ter sido realizado com o uso do NS-2, espera-se ter oferecido uma boa visão do confronto das duas linhas de pesquisa abordadas, e também deixando como possibilidades de futuros trabalhos não só o teste do RMTP em uma rede de alta velocidade, já citado anteriormente, como também pesquisas na área de sistemas operacionais para que através da utilização de distribuições Linux seja viável a utilização do RMTP em redes de alta velocidade.

Referências Bibliográficas:

[BR095] Lawrence Brakmo, e Larry Peterson “Vegas: End to End Congestion Avoidance on a Global Internet” IEEE Journal, 1995

[DO000] Dorgham Sisalem, e Henning Schulzrinne, "The Direct Adjustment Algorithm: A TCP-Friendly Adaptation Scheme", in Quality of Future Internet Services, Berlin, Germany, Setembro 2000.

[DY099] Phil Dykstra, “Gigabit Ethernet Jumbo Frames and Why You Should Care”, WareOnEarth Communication, Dezembro 1999.

[FL003] Sally Floyd, “High Speed TCP for Large Congestion Windows”, RFC 3649, Dezembro 2003.

[FL099] Sally Floyd, e Tom Henderson, “The NewReno Modification to TCP´s Fast Recovery Algorithm”, RFC 2582, Abril 1999.

[HA003] Mark Handley, Sally Floyd, Jitendra Padhye, e Joerg Widmer “TCP Friendly Rate Control (TFRC)”, RFC3448, Janeiro 2003.

[IN005] Injong Rhee, e Lisong Xu, “CUBIC: A New TCP-Friendly High Speed TCP Variant”, PFLDNET 2005.

[JA088] Van Jacobson, e Michael Karels, “Cogestion Avoidance and Control”, SIGCOMM´ 88

[JA090] Van Jacobson, “Modified TCP Congestion Control and Avoidance Algorithms”, Abril 1990.

[JA092] Van Jacobson, Bob Braden, e Dave Borman, “TCP Extensions for High Performance”, RFC 1323, Maio 1992.

[KE092] Srinivasan Keshav, "A Control-Theoretic Approach to Flow Control", SIGCOMM 1992.

[LI004] Lisong Xu, Khaled Harfoush, e Injong Rhee, “Binary Increase Congestion Control for Fast, Long Distance Networks”, INFOCOM 2004.

[MA001] Marco Mazzucco, Harinath Sivakumar, Yin Pan, e Qing Zhang, “Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks”, Dezembro 2001.

[MA003] Marek Malowidzki, "Simulation-based Study of ECN Performance in RED

Networks", SPECTS, 2003.

[MA005] Luiz Claudio Magalhães, “A Transport Layer Approach to Host Mobility”, University of Illinois – Setembro 2005.

[MA096] Matt Mathis, Jamshid Mahdavi, Sally Floyd, e Allyn Romanov, “TCP Selective Acknowledgment Options”, RFC 2018, Outubro 1996

[MA097] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, e Teunis Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm", in Computer Communication Review, Julho 1997.

[MA103] Marek Malowidzki, "ECN is Fine - But Will It Be Used?", PDCN, 2003.

[ME002] Mark Meiss, “A High-Speed Rate-Controlled Protocol for File Transfer”, iGrid 2002.

[NA005] Masayoshi Nabeshima, “Performance Evaluation of MulTCP in High-Speed Wide Area Networks”, IEICE TRANS. COMMUN. VOL. E88-B, Nº 1, Janeiro 2005.

[PA098] Craig Partridge, “Gigabit Networking – Addison Wesley Longman – 6ª Edição”, 1998.

[PA099] Jitendra Padhye, Jim Kurose, Don Towsley, e Rajeev Koodli, "A Model Based TCP-friendly Rate Control Protocol", in Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Basking Ridge, NJ, Junho 1999.

[PO081] Jon Postel, “Transmission Control Protocol”, RFC793, Setembro 1981.

[RA001] Ramakrishnan, Sally Floyd, e David Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC3168, Setembro 2001.

Anexo I – script utilizado nos testes das seções 4.4 e 4.5

```
# testes com protocolos TCP, HSTCP, BICTCP, CUBICTCP, MulTCP e RMTP #


# criando o objeto #
set ns [new Simulator]
ns-random 0


# abrindo o arquivo de saida #
set tf stdout
$ns trace-all $tf


# definindo a procedure finish #
proc finish {} {
        global ns tf nf
        $ns flush-trace
        exit 0
}


# parametros do backbone: bandwidth, delay, RED/DropTail, BufferSize #
set BB_bandwidth [lindex $argv 0]
set BB_delay [lindex $argv 1]
set BB_queue [lindex $argv 2]
set BB_buffer [lindex $argv 3]


# parametros dos links de acesso: bandwidth, delay, RED/DropTail,
BufferSize #
set AC_bandwidth [lindex $argv 4]
set AC_delay [lindex $argv 5]
set AC_buffer [lindex $argv 6]


# numero de fluxos #
set tcp_flow [lindex $argv 7]
set hstcp_flow [lindex $argv 8]
set bic_flow [lindex $argv 9]
set cubic_flow [lindex $argv 10]
set multcp_flow [lindex $argv 11]
set mmtp_flow [lindex $argv 12]


      # parametros para o TCP #
      Agent/TCP set minrto_ 1
      Agent/TCP set timestamps_ 1
      Agent/TCP set ecn_ 1
      Agent/TCP set window_ 100000
      Agent/TCP set packetSize_ 1000
      Agent/TCP set overhead_ 0.000008
      Agent/TCP set max_ssthresh_ 100
      Agent/TCP set maxburst_ 2

      # parametros para o BIC #
      Agent/TCP set bic_beta_ 0.8
      Agent/TCP set bic_B_ 4
      Agent/TCP set bic_max_increment_ 32
```

```
      Agent/TCP set bic_min_increment_ 0.01
      Agent/TCP set bic_fast_convergence_ 1
      Agent/TCP set bic_low_utilization_threshold_ 0
      Agent/TCP set bic_low_utilization_checking_period_ 2
      Agent/TCP set bic_delay_min_ 0
      Agent/TCP set bic_delay_avg_ 0
      Agent/TCP set bic_delay_max_ 0
      Agent/TCP set bic_low_utilization_indication_ 0

      # parametros para o CUBIC #
      Agent/TCP set cubic_beta_ 0.8
      Agent/TCP set cubic_max_increment_ 16
      Agent/TCP set cubic_fast_convergence_ 1
      Agent/TCP set cubic_scale_ 0.4
      Agent/TCP set cubic_tcp_friendliness_ 1
      Agent/TCP set cubic_low_utilization_threshold_ 0
      Agent/TCP set cubic_low_utilization_checking_period_ 2
      Agent/TCP set cubic_delay_min_ 0
      Agent/TCP set cubic_delay_avg_ 0
      Agent/TCP set cubic_delay_max_ 0
      Agent/TCP set cubic_low_utilization_indication_ 0

      # parametros para o MULTCP #
      Agent/TCP set multcp_n_ 4


      # parametros para o HSTCP #
      set high_window 83000
      set high_p 0.0000001
      set high_decrease 0.1
      set hstcp_fix 1

      # criando os nós do backbone #
      set n1 [$ns node]
      set n2 [$ns node]


      # criando o link do backbone #

      if {$BB_queue == 0} {
      $ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms RED
      } else {
      $ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms
DropTail
      }

      $ns queue-limit $n1 $n2 $BB_buffer
      $ns queue-limit $n2 $n1 $BB_buffer

      # parametros para a fila com RED #
      Queue/RED set bottom_ 0
      Queue/RED set thresh_ 0
      Queue/RED set maxthresh_ 0
      Queue/RED set q_weight_ 0
      Queue/RED set adaptive_ 1
      Queue/RED set bytes_ false
      Queue/RED set queue_in_bytes_ false


      # criando os nós e os links de acesso para o protocolo TCP #
      for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
```

```
        set tcpnode(l$i) [$ns node]
        set tcpnode(r$i) [$ns node]

        $ns duplex-link $tcpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $tcpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $tcpnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $tcpnode(l$i) $AC_buffer

        $ns queue-limit $tcpnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $tcpnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para o protocolo HSTCP #
      for {set i 0} {$i < [expr $hstcp_flow]} {incr i} {

        set hstcpnode(l$i) [$ns node]
        set hstcpnode(r$i) [$ns node]

        $ns duplex-link $hstcpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $hstcpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $hstcpnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $hstcpnode(l$i) $AC_buffer

        $ns queue-limit $hstcpnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $hstcpnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para o protocolo BIC #
      for {set i 0} {$i < [expr $bic_flow]} {incr i} {

        set bicnode(l$i) [$ns node]
        set bicnode(r$i) [$ns node]

        $ns duplex-link $bicnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $bicnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $bicnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $bicnode(l$i) $AC_buffer

        $ns queue-limit $bicnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $bicnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para o protocolo CUBIC #
      for {set i 0} {$i < [expr $cubic_flow]} {incr i} {

        set cubicnode(l$i) [$ns node]
        set cubicnode(r$i) [$ns node]
```

```
        $ns duplex-link $cubicnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $cubicnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $cubicnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $cubicnode(l$i) $AC_buffer

        $ns queue-limit $cubicnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $cubicnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para o protocolo MULTCP #
      for {set i 0} {$i < [expr $multcp_flow]} {incr i} {

        set multcpnode(l$i) [$ns node]
        set multcpnode(r$i) [$ns node]

        $ns duplex-link $multcpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $multcpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $multcpnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $multcpnode(l$i) $AC_buffer

        $ns queue-limit $multcpnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $multcpnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para o protocolo MMTP #
      for {set i 0} {$i < [expr $mmtp_flow]} {incr i} {

        set mmtpnode(l$i) [$ns node]
        set mmtpnode(r$i) [$ns node]

        $ns duplex-link $mmtpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $mmtpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $mmtpnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $mmtpnode(l$i) $AC_buffer

        $ns queue-limit $mmtpnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $mmtpnode(r$i) $AC_buffer
      }


      # criando conexao TCP #
      for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
      set tcp($i) [new Agent/TCP/Sack1]
      $ns attach-agent $tcpnode(l$i) $tcp($i)
      set sink($i) [new Agent/TCPSink/Sack1]
      $tcp($i) set fid_ $i
      $sink($i) set fid_ $i
      $ns attach-agent $tcpnode(r$i) $sink($i)
      $ns connect $tcp($i) $sink($i)
      }
```

```
# criando uma aplicacao FTP sobre a conexao TCP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
set ftp($i) [new Application/FTP]
$ftp($i) attach-agent $tcp($i)
}



# criando conexao HSTCP #
for {set i 0} {$i < [expr $hstcp_flow]} {incr i} {
set hstcp($i) [new Agent/TCP/Sack1]
$ns attach-agent $hstcpnode(l$i) $hstcp($i)
set sink($i) [new Agent/TCPSink/Sack1]
$ns attach-agent $hstcpnode(r$i) $sink($i)
$ns connect $hstcp($i) $sink($i)

[set hstcp($i)] set windowOption 8
[set hstcp($i)] set low_window 31
[set hstcp($i)] set high_window 83000
[set hstcp($i)] set high_p 0.0000001
[set hstcp($i)] set high_decrease 0.1
[set hstcp($i)] set hstcp_fix 1
}


# criando uma aplicacao FTP sobre a conexao HSTCP #
for {set i 0} {$i < [expr $hstcp_flow]} {incr i} {
set hstcpftp($i) [new Application/FTP]
$hstcpftp($i) attach-agent $hstcp($i)
}


# criando conexao BIC #
for {set i 0} {$i < [expr $bic_flow]} {incr i} {
set bic($i) [new Agent/TCP/Sack1]
$ns attach-agent $bicnode(l$i) $bic($i)
set sink($i) [new Agent/TCPSink/Sack1]
$ns attach-agent $bicnode(r$i) $sink($i)
$ns connect $bic($i) $sink($i)

[set hstcp($i)] set windowOption 12
[set hstcp($i)] set low_window 14
}


# criando uma aplicacao FTP sobre a conexao BIC #
for {set i 0} {$i < [expr $bic_flow]} {incr i} {
set bicftp($i) [new Application/FTP]
$bicftp($i) attach-agent $bic($i)
}


# criando conexao CUBIC #
for {set i 0} {$i < [expr $cubic_flow]} {incr i} {
set cubic($i) [new Agent/TCP/Sack1]
$ns attach-agent $cubicnode(l$i) $cubic($i)
set sink($i) [new Agent/TCPSink/Sack1]
$ns attach-agent $cubicnode(r$i) $sink($i)
$ns connect $cubic($i) $sink($i)
```

```
[set hstcp($i)] set windowOption 13
[set hstcp($i)] set low_window 14
}


# criando uma aplicacao FTP sobre a conexao CUBIC #
for {set i 0} {$i < [expr $cubic_flow]} {incr i} {
set cubicftp($i) [new Application/FTP]
$cubicftp($i) attach-agent $cubic($i)
}


# criando conexao MULTCP #
for {set i 0} {$i < [expr $multcp_flow]} {incr i} {
set multcp($i) [new Agent/TCP/Sack1]
$ns attach-agent $multcpnode(l$i) $multcp($i)
set sink($i) [new Agent/TCPSink/Sack1]
$ns attach-agent $multcpnode(r$i) $sink($i)
$ns connect $multcp($i) $sink($i)

[set hstcp($i)] set windowOption 14
[set hstcp($i)] set low_window 0
}


# criando uma aplicacao FTP sobre a conexao MULTCP #
for {set i 0} {$i < [expr $multcp_flow]} {incr i} {
set multcpftp($i) [new Application/FTP]
$multcpftp($i) attach-agent $multcp($i)
}


# criando conexao MMTP #
for {set i 0} {$i < [expr $mmtp_flow]} {incr i} {
set mmtp($i) [new Agent/mmtp]
$ns attach-agent $mmtpnode(l$i) $mmtp($i)
set mmtps($i) [new Agent/mmtp]
$ns attach-agent $mmtpnode(r$i) $mmtps($i)
$ns connect $mmtp($i) $mmtps($i)
}


# inicializando eventos FTP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
$ns at 0 "$ftp($i) start"
$ns at 30 "$ftp($i) stop"
}

for {set i 0} {$i < [expr $hstcp_flow]} {incr i} {
$ns at 0 "$hstcpftp($i) start"
$ns at 30 "$hstcpftp($i) stop"
}

for {set i 0} {$i < [expr $bic_flow]} {incr i} {
$ns at 0 "$bicftp($i) start"
$ns at 30 "$bicftp($i) stop"
}

for {set i 0} {$i < [expr $cubic_flow]} {incr i} {
$ns at 0 "$cubicftp($i) start"
```

```
$ns at 30 "$cubicftp($i) stop"
}
for {set i 0} {$i < [expr $multcp_flow]} {incr i} {
$ns at 0 "$multcpftp($i) start"
$ns at 30 "$multcpftp($i) stop"
}

for {set i 0} {$i < [expr $mmtp_flow]} {incr i} {
$ns at 0 "$mmtps($i) receive"
$ns at 0 "$mmtp($i) start"
#$ns at 20 "$mmtp($i) stop"
}


# finalizando após 30 segundos #
$ns at 30 "finish"


# roda ns #
$ns run
```

Anexo II – script utilizado nos testes da seção 4.6

```
# testes com protocolos de alta velocidade 1 a 1 com o protocolo TCP #


# criando o objeto #
set ns [new Simulator]
ns-random 0


# abrindo o arquivo de saida #
set tf stdout
$ns trace-all $tf


# definindo a procedure finish #
proc finish {} {
        global ns tf nf
        $ns flush-trace
        exit 0
}


# parametros do backbone: bandwidth, delay, RED/DropTail, BufferSize #
set BB_bandwidth [lindex $argv 0]
set BB_delay [lindex $argv 1]
set BB_queue [lindex $argv 2]
set BB_buffer [lindex $argv 3]


# parametros dos links de acesso: bandwidth, delay, BufferSize #
set AC_bandwidth [lindex $argv 4]
set AC_delay [lindex $argv 5]
set AC_buffer [lindex $argv 6]


# quantidade de fluxos TCP e de alta velocidade #
set tcp_flow [lindex $argv 7]
set hs_flow [lindex $argv 8]


# tipo de protocolo de alta velocidade #
# HSTCP == 8
# BIC TCP == 12
# CUBIC TCP == 13
# MulTCP == 14
# MMTP == 15
set flow_type [lindex $argv 9]


# parametros para os protocolos de alta velocidade #
if { $flow_type == 8 } {
# HSTCP
  set low_window 31
}
if { $flow_type == 12 } {
# BIC TCP
  set low_window 14
}
if { $flow_type == 13 } {
```

```
# CUBIC TCP
  set low_window 14
}
if { $flow_type == 14 } {
# MulTCP
  set low_window 0
}
if { $flow_type == 15 } {
# MMTP

      # parametros para o TCP #
      Agent/TCP set minrto_ 1
      Agent/TCP set timestamps_ 1
      Agent/TCP set ecn_ 1
      Agent/TCP set window_ 100000
      Agent/TCP set packetSize_ 1000
      Agent/TCP set overhead_ 0.000008
      Agent/TCP set max_ssthresh_ 100
      Agent/TCP set maxburst_ 2

      # parametros para o BIC TCP #
      Agent/TCP set bic_beta_ 0.8
      Agent/TCP set bic_B_ 4
      Agent/TCP set bic_max_increment_ 32
      Agent/TCP set bic_min_increment_ 0.01
      Agent/TCP set bic_fast_convergence_ 1
      Agent/TCP set bic_low_utilization_threshold_ 0
      Agent/TCP set bic_low_utilization_checking_period_ 2
      Agent/TCP set bic_delay_min_ 0
      Agent/TCP set bic_delay_avg_ 0
      Agent/TCP set bic_delay_max_ 0
      Agent/TCP set bic_low_utilization_indication_ 0

      # parametros para o CUBIC #
      Agent/TCP set cubic_beta_ 0.8
      Agent/TCP set cubic_max_increment_ 16
      Agent/TCP set cubic_fast_convergence_ 1
      Agent/TCP set cubic_scale_ 0.4
      Agent/TCP set cubic_tcp_friendliness_ 1
      Agent/TCP set cubic_low_utilization_threshold_ 0
      Agent/TCP set cubic_low_utilization_checking_period_ 2
      Agent/TCP set cubic_delay_min_ 0
      Agent/TCP set cubic_delay_avg_ 0
      Agent/TCP set cubic_delay_max_ 0
      Agent/TCP set cubic_low_utilization_indication_ 0


      # parametros para o MulTCP #
      Agent/TCP set multcp_n_ 4


      # criando os nós do backbone #
      set n1 [$ns node]
      set n2 [$ns node]


      # criando o link do backbone #

      if {$BB_queue == 0} {
      $ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms RED
      } else {
```

```
      $ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms
DropTail
      }

      $ns queue-limit $n1 $n2 $BB_buffer
      $ns queue-limit $n2 $n1 $BB_buffer


      # parametros para a fila com RED #
      Queue/RED set bottom_ 0
      Queue/RED set thresh_ 0
      Queue/RED set maxthresh_ 0
      Queue/RED set q_weight_ 0
      Queue/RED set adaptive_ 1
      Queue/RED set bytes_ false
      Queue/RED set queue_in_bytes_ false


      # criando os nós e os links de acesso para o protocolo TCP #
      for {set i 0} {$i < [expr $tcp_flow]} {incr i} {

        set tcpnode(l$i) [$ns node]
        set tcpnode(r$i) [$ns node]

        $ns duplex-link $tcpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $tcpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $tcpnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $tcpnode(l$i) $AC_buffer

        $ns queue-limit $tcpnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $tcpnode(r$i) $AC_buffer
      }


      # criando os nós e os links de acesso para os protocolos de alta-
velocidade #
      for {set i 0} {$i < [expr $hs_flow]} {incr i} {

        set hsnode(l$i) [$ns node]
        set hsnode(r$i) [$ns node]

        $ns duplex-link $hsnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
        $ns duplex-link $hsnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

        $ns queue-limit $hsnode(l$i) $n1 $AC_buffer
        $ns queue-limit $n1 $hsnode(l$i) $AC_buffer

        $ns queue-limit $hsnode(r$i) $n2 $AC_buffer
        $ns queue-limit $n2 $hsnode(r$i) $AC_buffer
      }


      # criando conexao TCP #
      for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
      set tcp($i) [new Agent/TCP/Sack1]
      $ns attach-agent $tcpnode(l$i) $tcp($i)
```

```
        set sink($i) [new Agent/TCPSink/Sack1]
        $tcp($i) set fid_ $i
        $sink($i) set fid_ $i
        $ns attach-agent $tcpnode(r$i) $sink($i)
        $ns connect $tcp($i) $sink($i)
        }


        # criando uma aplicacao FTP sobre a conexao TCP #
        for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
        set ftp($i) [new Application/FTP]
        $ftp($i) attach-agent $tcp($i)
        }


        # criando conexao MMTP #
        for {set i 0} {$i < [expr $hs_flow]} {incr i} {
        set mmtp($i) [new Agent/mmtp]
        $ns attach-agent $hsnode(l$i) $mmtp($i)
        set mmtps($i) [new Agent/mmtp]
        $mmtp($i) set fid_ [expr $i + $tcp_flow]
        $mmtps($i) set fid_ [expr $i + $tcp_flow]
        $ns attach-agent $hsnode(r$i) $mmtps($i)
        $ns connect $mmtp($i) $mmtps($i)
        }


        # inicializando eventos FTP #
        for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
        $ns at 0 "$ftp($i) start"
        $ns at 30 "$ftp($i) stop"
        }
        for {set i 0} {$i < [expr $hs_flow]} {incr i} {
        $ns at 0 "$mmtps($i) receive"
        $ns at 0 "$mmtp($i) start"
        }


        # finalizando após 30 segundos #
        $ns at 30 "finish"


        # roda ns #
        $ns run


}


# parametros para o TCP #
Agent/TCP set minrto_ 1
Agent/TCP set timestamps_ 1
Agent/TCP set ecn_ 1
Agent/TCP set window_ 100000
Agent/TCP set packetSize_ 1000
Agent/TCP set overhead_ 0.000008
Agent/TCP set max_ssthresh_ 100
Agent/TCP set maxburst_ 2


# parametros para o BIC TCP #
```

```
Agent/TCP set bic_beta_ 0.8
Agent/TCP set bic_B_ 4
Agent/TCP set bic_max_increment_ 32
Agent/TCP set bic_min_increment_ 0.01
Agent/TCP set bic_fast_convergence_ 1
Agent/TCP set bic_low_utilization_threshold_ 0
Agent/TCP set bic_low_utilization_checking_period_ 2
Agent/TCP set bic_delay_min_ 0
Agent/TCP set bic_delay_avg_ 0
Agent/TCP set bic_delay_max_ 0
Agent/TCP set bic_low_utilization_indication_ 0


# parametros para o CUBIC #
Agent/TCP set cubic_beta_ 0.8
Agent/TCP set cubic_max_increment_ 16
Agent/TCP set cubic_fast_convergence_ 1
Agent/TCP set cubic_scale_ 0.4
Agent/TCP set cubic_tcp_friendliness_ 1
Agent/TCP set cubic_low_utilization_threshold_ 0
Agent/TCP set cubic_low_utilization_checking_period_ 2
Agent/TCP set cubic_delay_min_ 0
Agent/TCP set cubic_delay_avg_ 0
Agent/TCP set cubic_delay_max_ 0
Agent/TCP set cubic_low_utilization_indication_ 0


# parametros para o MulTCP #
Agent/TCP set multcp_n_ 4


# parametros para o HSTCP #
set high_window 83000
set high_p 0.0000001
set high_decrease 0.1
set hstcp_fix 1


# criando os nós do backbone #
set n1 [$ns node]
set n2 [$ns node]


# criando o link do backbone #
if {$BB_queue == 0} {
$ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms RED
} else {
$ns duplex-link $n1 $n2 [expr $BB_bandwidth]Mb [expr $BB_delay]ms DropTail
}

$ns queue-limit $n1 $n2 $BB_buffer
$ns queue-limit $n2 $n1 $BB_buffer


# parametros para a fila com RED #
Queue/RED set bottom_ 0
Queue/RED set thresh_ 0
Queue/RED set maxthresh_ 0
Queue/RED set q_weight_ 0
Queue/RED set adaptive_ 1
Queue/RED set bytes_ false
```

```
Queue/RED set queue_in_bytes_ false


# criando os nós e os links de acesso para o protocolo TCP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {

  set tcpnode(l$i) [$ns node]
  set tcpnode(r$i) [$ns node]

  $ns duplex-link $tcpnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
  $ns duplex-link $tcpnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

  $ns queue-limit $tcpnode(l$i) $n1 $AC_buffer
  $ns queue-limit $n1 $tcpnode(l$i) $AC_buffer

  $ns queue-limit $tcpnode(r$i) $n2 $AC_buffer
  $ns queue-limit $n2 $tcpnode(r$i) $AC_buffer
}


# criando os nós e os links de acesso para os protocolos de alta-velocidade
#
for {set i 0} {$i < [expr $hs_flow]} {incr i} {

  set hsnode(l$i) [$ns node]
  set hsnode(r$i) [$ns node]

  $ns duplex-link $hsnode(l$i) $n1 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail
  $ns duplex-link $hsnode(r$i) $n2 [expr $AC_bandwidth]Mb [expr
$AC_delay]ms DropTail

  $ns queue-limit $hsnode(l$i) $n1 $AC_buffer
  $ns queue-limit $n1 $hsnode(l$i) $AC_buffer

  $ns queue-limit $hsnode(r$i) $n2 $AC_buffer
  $ns queue-limit $n2 $hsnode(r$i) $AC_buffer
}


# criando conexao TCP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
set tcp($i) [new Agent/TCP/Sack1]
$ns attach-agent $tcpnode(l$i) $tcp($i)
set sink($i) [new Agent/TCPSink/Sack1]
$tcp($i) set fid_ $i
$sink($i) set fid_ $i
$ns attach-agent $tcpnode(r$i) $sink($i)
$ns connect $tcp($i) $sink($i)
}


# criando uma aplicacao FTP sobre a conexao TCP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
set ftp($i) [new Application/FTP]
$ftp($i) attach-agent $tcp($i)
}
```

```
# criando conexao alta velocidade #
for {set i 0} {$i < [expr $hs_flow]} {incr i} {
set hstcp($i) [new Agent/TCP/Sack1]
$ns attach-agent $hsnode(l$i) $hstcp($i)
set sink($i) [new Agent/TCPSink/Sack1]
$hstcp($i) set fid_ [expr $i + $tcp_flow]
$sink($i) set fid_ [expr $i + $tcp_flow]
$ns attach-agent $hsnode(r$i) $sink($i)
$ns connect $hstcp($i) $sink($i)

[set hstcp($i)] set windowOption_ $flow_type
[set hstcp($i)] set low_window_ $low_window
[set hstcp($i)] set high_window_ $high_window
[set hstcp($i)] set high_p_ $high_p
[set hstcp($i)] set high_decrease_ $high_decrease
[set hstcp($i)] set hstcp_fix_ $hstcp_fix

}


# criando uma aplicacao FTP sobre a conexao alta velocidade #
for {set i 0} {$i < [expr $hs_flow]} {incr i} {
set hsftp($i) [new Application/FTP]
$hsftp($i) attach-agent $hstcp($i)
}


# inicializando eventos FTP #
for {set i 0} {$i < [expr $tcp_flow]} {incr i} {
$ns at 0 "$ftp($i) start"
$ns at 30 "$ftp($i) stop"
}
for {set i 0} {$i < [expr $hs_flow]} {incr i} {
$ns at 0 "$hsftp($i) start"
$ns at 30 "$hsftp($i) stop"
}


# finalizando após 30 segundos #
$ns at 30 "finish"


# roda ns #
$ns run
```

Anexo III – tabela do teste da seção 4.5

Tempo	TCP	HSTCP	BICTCP	CUBICTCP	MULTCP	RMTP
0,5	8368	8006	9212	7985	9068	10250
1	9653	8432	8755	8277	8961	9526
1,5	10392	4982	9958	9551	9393	9764
2	11586	170	11392	11636	7276	11676
2,5	11165	4332	11613	11632	3419	11927
3	9524	9469	9246	9389	5357	10745
3,5	10335	9235	4297	9310	9863	10556
4	10579	10313	256	10411	10508	11170
4,5	8854	9361	5358	10098	9396	10849
5	10549	10522	581	10439	10376	11399
5,5	10191	10080	1615	10339	10954	10623
6	5370	9542	10762	9796	7013	10700
6,5	1930	12272	12138	12085	3197	12833
7	6788	11136	10453	8140	5665	11876
7,5	10336	10893	10510	739	10040	11437
8	5708	11513	11474	4187	9158	11684
8,5	2005	12053	12003	12050	2797	12383
9	8807	10909	10500	10144	3128	10383
9,5	10195	10612	10012	10370	1808	10903
10	8633	11191	9830	11293	913	11356
10,5	2254	11504	5054	11420	11283	12151
11	3824	11667	3843	11466	11722	12408
11,5	8871	10909	2923	10576	10878	10937
12	4620	8852	62	12043	12058	13040
12,5	3180	5210	10094	11686	11634	12010
13	9494	6777	9983	7318	9135	10226
13,5	10678	10339	10588	641	10320	10875
14	10239	11551	7356	2741	11574	11275
14,5	11192	7310	1152	11115	11130	11844
15	11175	2068	5558	11349	11527	12282
15,5	10757	5275	11025	11563	3466	11476
16	10632	10615	10463	10314	417	11534
16,5	9477	8686	8778	9418	7432	10582
17	4802	11372	11471	11593	2994	11728
17,5	3255	11975	11539	12141	2125	12319
18	9986	7678	8839	8651	7863	10359
18,5	10389	2414	10578	9571	9902	10793
19	10061	916	10633	10336	10029	11387
19,5	8571	9098	9163	8217	8591	9760
20	8727	9415	10213	7398	8341	9957
20,5	11924	11099	12022	640	6169	11933
21	12192	12114	11986	1471	3939	12668
21,5	11953	11941	5458	1699	9187	12499
22	12033	11989	3194	1634	12043	12626
22,5	9464	10097	9005	4060	9806	11218
23	5971	12013	5514	1378	12041	13439

23,5	5193	11230	5824	7746	10932	11997
24	5133	9337	9848	9928	9802	10406
24,5	1034	10404	10416	10589	10026	11179
25	6385	4636	9996	10774	11670	11211
25,5	11609	392	5513	11962	11656	12070
26	12162	2146	2741	12253	12123	12891
26,5	11185	1014	9169	10918	10003	11412
27	10533	8019	10344	10825	3580	11464
27,5	4366	11643	11361	11731	2157	12517
28	2118	11412	10936	5309	11509	11678
28,5	2840	12024	12038	2285	12171	12391
29	1281	11012	9985	8727	10888	11317
29,5	7982	7188	8879	9331	9734	10934
TOTAL	478510	518364	493509	510688	490147	674833