Universidade Federal Fluminense

Departamento de Engenharia de Telecomunicações

Mestrado em Engenharia de Telecomunicações

Protocolos de Transporte Para Redes de Alta Velocidade: Um Estudo Comparativo

Fábio Teixeira Guerra

Orientador: Prof. Luiz Claudio Schara Magalhães, PhD.

Rio de Janeiro
2006

Fábio Teixeira Guerra

Protocolos de Transporte Para Redes de Alta Velocidade: Um Estudo Comparativo

Dissertação de Mestrado submetida à Banca Examinadora no Departamento de Engenharia de Telecomunicações, da Universidade Federal Fluminense, como parte dos requisitos necessários à obtenção do grau de mestre.

Rio de Janeiro, 27 de outubro de 2006.

Aprovada por:

_____
Presidente, Prof.

_____
Prof.

_____
Prof.

_____
Prof.

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, 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 dos buffers 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, ack-clocked , rate-based

Transport Protocols for High Speed Networks: A Comparative Study

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 the diminishes the amount of data being sent.

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

Capítulo 1 A Camada de Transporte nas Redes de Alta Velocidade

1.1 Introdução

Podemos afirmar hoje em dia que a Internet do século XXI é bem diferente da Internet do século XX. O advento da TV Interativa, as soluções para telefonia através da Internet, os serviços de telemedicina cada vez mais difundidos, as pesquisas na área da física que são desenvolvidas em conjunto com vários países através da Internet, são alguns exemplos de como a grande rede passou por várias transformações nos últimos anos. O grande motivo desta mudança se deve ao fato de que como a Internet possui um núcleo que tende a ser o mais simples possível, a implementação de um novo serviço se torna viável apenas realizando alterações nas pontas da rede.

Entretanto, todas essas inovações tecnológicas dependem de um fator indispensável: banda disponível. Assim, durante a década de 90, aconteceu um pesado investimento das operadoras de telecomunicações nesta área, criando anéis de fibra óptica não só entre países, mas também dentro das grandes cidades (chamados de anéis de última milha) e implantando uma grande quantidade de roteadores de alta capacidade (roteadores de *backbone*) no intuito de fazer com que o maior número possível de pessoas passasse a se conectar a Internet.

Com o aumento da disponibilidade de banda na Internet, 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, fora publicado por Jon Postel em 1981 e vem recebendo várias melhorias durante todos esses anos, o que possibilitou a sua grande utilização até os dias de hoje. Apesar disso, atualmente é possível perceber que, devido às características inerentes

do TCP, apenas um fluxo entre transmissor e receptor não consegue enviar dados a taxa suficiente para utilizar plenamente um *link* de uma rede *gigabit*.

Assim sendo, durante os últimos anos alguns pesquisadores tentaram e continuam tentando de várias maneiras adaptar o protocolo TCP para um novo paradigma de redes de alta velocidade. Estes pesquisadores têm obtido certo sucesso em alguns tópicos, todavia, esta tarefa não é de grande facilidade, pois como citado acima, sendo o TCP um protocolo com vinte e cinco anos de vida apresenta características na sua constituição que podemos considerar ultrapassadas: tamanho máximo de sua janela de transmissão pequeno para um ambiente de alta velocidade, o falho algoritmo de partida lenta no caso de uma perda imediatamente após o início de uma transmissão e o crescimento do tamanho da janela de transmissão de forma lenta no momento de prevenção de congestionamento.

Concomitantemente a esse grupo de pesquisadores surge um outro grupo que verificando estas características ultrapassadas do TCP passa a defender uma mudança radical na estrutura dos protocolos de transporte.

Surgem então duas linhas de pesquisa.

1. A corrente 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].

2. A corrente 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.

1.2 Como Atingir Alta Velocidade Fim a Fim

As rajadas no tráfego do TCP podem encher, em alguns momentos, os buffers dos roteadores e gerar um gráfico tempo x tamanho da janela de transmissão com característica de dente de serra. Como 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, temos então um problema a ser resolvido.

No caso dos protocolos baseados em taxa o problema acima citado não existe, pois sendo o envio dos pacotes baseado no tempo, não existem rajadas o que evita o transbordo de dados nas filas dos roteadores.

O objetivo deste trabalho é comparar o desempenho dos protocolos baseados em *acks*: HSTCP, MulTCP, BIC TCP e CUBIC TCP com o RMTP, baseado em taxa, em um ambiente de 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 diferentes condições de tráfego.

O critério de escolha destes protocolos se deu da seguinte forma: procurou-se levantar protocolos baseados em *acks* com características diferentes, ou seja, apesar de serem

derivações do protocolo TCP, os controles de congestionamento dos quatro protocolos citados acima possuem diferenças entre si, exceto no caso do CUBIC TCP, que é uma variante do BIC TCP e ambos foram desenvolvidos pelo mesmo grupo de estudo da Universidade do Estado da Carolina do Norte.

A escolha do RMTP, dentro do grupo de protocolos baseados em taxa, foi devido a este protocolo possuir características interessantes para redes de alta velocidade e também nunca ter sido testado neste ambiente, pois Magalhães desenvolveu este protocolo em [MA005] para redes móveis.

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 estão 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 os 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 extremamente promissora dentro do nosso ponto de vista.

Este trabalho é constituído de cinco capítulos. Neste capítulo é feita uma introdução ao problema e os 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 da comunidade científica para um desempenho melhor que o do TCP em redes de alta velocidade. Portanto, a segunda parte do capítulo 2 será a apresentação dos controles de congestionamento do HSTCP, MulTCP, BIC TCP e CUBIC TCP.

No capítulo 3 será apresentado o RMTP, mostrando a sua arquitetura, o cabeçalho do pacote 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 foram realizados testes com o TCP compartilhando banda com os protocolos de alta velocidade um a um. Ou seja, foram feitos cinco experimentos: TCP x HSTCP, TCP x MulTCP, TCP x BIC TCP, TCP x CUBIC TCP e TCP x RMTP.

Na segunda parte dos testes, todos os protocolos compartilham banda em um ambiente de alto tráfego, enquanto que no terceiro experimento cada protocolo é testado de forma isolada para mostrar a sua capacidade de vazão em uma rede *gigabit*.

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 que podem se transformar em novas teses nesta área.

Capítulo 2 Protocolos de Transporte para Redes de Alta Velocidade

2.1 TCP em Links de Alta Velocidade

O protocolo IP [PO181] [KU003] [PE003] é um protocolo da camada de rede e possui como característica marcante uma grande conectividade, pois consegue atender a vários tipos de conexões físicas tais como: linhas discadas, conexões dedicadas de baixa velocidade e de alta velocidade, satélites etc. Ademais, o protocolo IP consegue uma boa interação com o nível de enlace das redes de dados, fato que contribuiu para a explosão da Internet.

Já o TCP é um protocolo 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 conectar aplicações localizadas em diferentes máquinas.

Entretanto, quando o TCP foi projetado, 10 Mbps era uma velocidade altíssima. Assim, este protocolo possui basicamente quatro características de projeto que quando utilizado em redes com taxas disponíveis em *gigabits* por segundo podem comprometer o desempenho de uma conexão:

1. Números de seqüência dos segmentos TCP
2. Tamanho dos pacotes
3. Tamanho da janela de transmissão
4. Controle de congestionamento

2.1.1 Números de seqüência dos segmentos TCP

Números de seqüência são usados para identificar os segmentos, pois no caso de atraso ou perda destes, o receptor consegue reordenar os segmentos ou solicitar uma

retransmissão. Em [PA098] coloca-se de forma bem clara que o TCP em uma rede de 10Mbps, necessita de 1700 segundos para que o contador do número de seqüência reutilize 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 com o mesmo número de seqüência estarem na rede ao mesmo tempo. Porém, em velocidades de *gigabit* por segundo, os 1700 segundos citados acima se transformam em 17 segundos em redes de 1Gbps e 1,7 segundos em redes de 10Gbps. Portanto, em redes de alta velocidade, existe a possibilidade de vários segmentos com o mesmo número de seqüência estarem na linha em um determinado instante, e no caso de reordenação ou retransmissão, receptor e/ou transmissor não sabem qual pacote está se tratando. Este problema pode ser solucionado usando a técnica proposta em [JA092].

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)

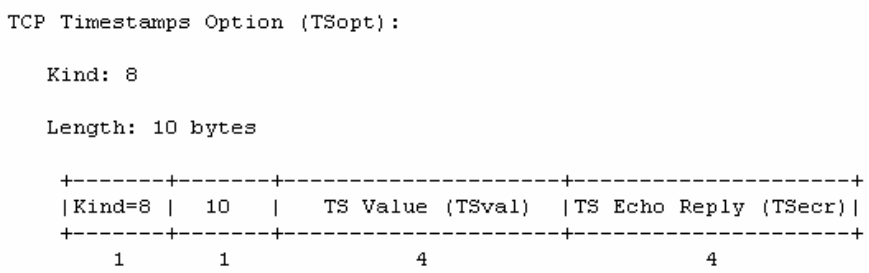


Figura 1:TCP Timestamp Option

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 echo do valor de TSval recebido pelo host. Só é válido se o bit *ack* estiver setado. Quando TSecr não é válido o seu valor será zero

Utilizando o TSopt, criou-se um mecanismo chamado PAWS (*Protect Against Wrapped Sequence Number*) o qual possui um algoritmo que rejeita segmentos antigos que possuem 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 grande problema do PAWS é que se o relógio do *timestamp* for de uma granularidade de 1ms conexões com mais de 50 dias terão o problema do contador do *timestamp* voltar ao valor inicial. Se a granularidade do relógio for de μs (granularidade compatível com redes de alta velocidade) os 50 dias se transformam em cerca de 1 hora inviabilizando esta técnica para redes de alta velocidade.

2.1.2 Tamanho dos pacotes (Jumbopackets)

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 em uma rede WAN, deve-se também se preocupar com este tópico, pois a melhora de uma conexão no nível de transporte pode ser alcançada

apenas aumentando a quantidade de dados em um pacote. Como o desempenho do TCP em WAN's, principalmente Internet, foi e continua sendo objeto de muito estudo. [MA097] mostra que a vazão de uma conexão TCP está limitada pela seguinte expressão:

Throuput = ~ 0,7 * MSS / (RTT * Sqrt (Packet_ Loss))

Porém, [DY099] verifica que quanto maior for o tamanho do pacote, maior será a probabilidade de erro (*Packet Loss*), mantendo a mesma taxa de erro de bit (*bit error rate)*. Assim não se pode afirmar plenamente que apenas aumentando o tamanho do pacote tem-se uma maior vazão atingida.

Outra observação importante é a seguinte: utilizando *jumbopackets* sempre que ocorrer uma perda, a quantidade de informação perdida será maior do que a quantidade de informação perdida no caso de utilização de pacotes menores. Isto pode gerar sérios problemas para a camada de aplicação.

2.1.3 Tamanho da janela de transmissão

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

1- Se em um *link* de 1Gbps e com distância intercontinental, o receptor solicita ao transmissor a diminuição do tamanho da janela de transmissão, isso só acontecerá cerca de 30 ms depois da solicitação. Com isso teremos a transmissão de cerca 3 milhões de bits fora da vazão ideal. Apesar de parecer um problema, isso não preocupa, pois alguns milhões de bits de dados é menos que 1 Mbyte, quantidade insignificante nos dias de hoje.

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 zero e passa a ter um limiar que é a metade do tamanho da janela que teve a perda. Esse valor serve como limite para que, nas próximas tentativas, o TCP passe a crescer dobrando o tamanho de sua janela de transmissão (partida lenta) até esse limiar. 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 sub utilizar uma rede *gigabit*, por exemplo, além de desinteressante é dispendioso. As novas propostas de protocolos de transporte para redes *gigabit*, alteram a variação do tamanho da janela de transmissão atuando assim no controle de congestionamento.

2.1.4 Controle de Congestionamento

Sabendo que a quantidade máxima de informação que está na linha (*link*) em um determinado instante é dada pelo produto atraso x banda (*delay x bandwidth*) precisa-

se analisar este produto com muita atenção quando se trata de um projeto de protocolos para redes de alta velocidade.

Apesar de hoje o TCP ser o protocolo mais usado na Internet, ele demonstra uma limitação. Em redes de alta velocidade, o TCP não consegue atingir a taxa de um *link* da ordem de *gigabit* por segundo, pois antes de chegar à vazão oferecida pela rede, ocorrem eventos que fazem com que o algoritmo do TCP diminua a taxa de envio de pacotes. Devido a isso, a comunidade científica vem 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 que se acredita ser o 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*, conseguiremos naturalmente obter este controle de congestionamento. Uma vez que estes protocolos espaçam seus segmentos de forma regular na unidade de tempo, temos uma estabilidade bem maior na rede, o que facilita de sobremaneira o controle de congestionamento.

Portanto, será mostrado a seguir como funciona o controle de congestionamento do TCP e logo após as novas propostas dos protocolos, HSTCP, MulTCP, BIC TCP e CUBIC TCP, para redes de alta velocidade e finalmente no próximo capítulo o protocolo RMTP que possui um controle de congestionamento baseado em taxa conhecido como HCC (Homeostatic Congestion Control) [MA005].

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 1 e vai aumentando em 1 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 2, no final do 2º RTT será 4 (pois recebeu-se 2 *acks*), no final do 3º RTT será 8 (pois recebeu-se 4 *acks*) e assim sucessivamente.

Após a partida lenta, 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, $b = 0,5$

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. Sabendo que para uma conexão

TCP que transmite w segmentos de tamanho MSS *bytes* a cada RTT segundos a vazão é dada por:

1. (w x MSS) / RTT

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

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

Fazendo uma relação entre 1 e 2 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.

2.2 HSTCP, MulTCP, BicTCP e seus Controles de Congestionamento

2.2.1 HSTCP

O HSTCP é um aperfeiçoamento do TCP para redes de alta velocidade. A idéia de Sally Floyd em [FL003] foi de alterar o controle de congestionamento do já conhecido e utilizado TCP, fazendo com que este ganhe desempenho em *links* de alta velocidade.

Para isso acontecer Sally Floyd criou a seguinte função resposta para o HSTCP:

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

onde:

$S = (\log High\_Window - \log Low\_Window) / (\log High\_P - \log Low\_P)$

e

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

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

Low_P = taxa de perda de segmentos para Low_window

High_P = taxa de perda de segmentos para High_window

Substituindo o valor de S em 3, função resposta do HSTCP, temos:

$w = (p / Low\_P)^S Low\_window$

Para valores padrões de Low-Window = 38, High_Window = 83000, Low_P = 10^{-3} e High_P = 10^{-7} temos como função resposta para o HSTCP o seguinte valor final:

$w = 0,12/p^{0,83}$

Sendo w, a quantidade de segmentos que podemos transmitir em um RTT, temos que nos momentos de AIMD de uma conexão HSTCP as funções que determinam o tamanho da janela são:

4. $W = w + a(w) / w$
5. $W = w – b(w) * w$

Analisando 4 e 5 verifica-se que os valores futuros de w estão em função de w's atuais. Ademais, em [FL003] os valores de a(w) e b(w) são respectivamente:

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

$$B(w) = \frac{(\text{High Decrease} - 0{,}5)\ (\log w - \log \text{Low Window})}{\text{Log High Window} - \log \text{Low Window}} + 0{,}5$$

Onde:

High_Decrease = determina o valor de B(w) para w = High_window

Pode-se observar através das duas expressões - A(w) e B(w) - que para valores muito pequenos de w 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 protocolos já existentes (no caso o TCP).

Outra característica do HSTCP é o seu desempenho para grandes valores da janela de transmissão, pois para valores de w muito grandes os futuros valores de w crescem mais rápido e diminuem mais devagar do que o TCP.

2.2.2 MulTCP

Na fase de prevenção de congestionamento o TCP ajusta a sua janela de transmissão seguindo a idéia AIMD que apresenta a seguinte característica:

Recebendo um *ack*: (incremento):

1. $w = w + a / w$ onde $a = 1$

Percebendo uma perda: (decremento):

2. $w = w - b * w$ onde $b = 0{,}5$

A partir das afirmações acima encontra-se um valor médio do tamanho da janela em condições estacionárias, que é dado por:

3. $W_{med} = 1{,}2 / p^{1/2}$

Onde p é a taxa média de perda de pacotes.

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.

2.2.2.1 MulTCP1

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:

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

Se N = 1 em (4), obtém-se o mesmo valor de (3), porém para N > 1, o valor de (4) é maior que o valor de N x (3) criando assim uma incoerência.

2.2.2.2 MulTCP2

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:

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

Para N = 1, Sendo "a" o mesmo incremento de (1) e "b" o mesmo decremento de (2), a relação entre w e p pode ser escrita da seguinte forma:

6. $W_{mul2} = (a * (2 - b))^{1/2} / (2bp)^{1/2}$

De (5) e (6) chega-se em uma relação entre a e b que se faz necessária para que a função resposta do MulTCP seja a esperada para qualquer valor de N:

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

2.2.3 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 (RTT *Fairness*)
- Amigável ao TCP (TCP *Friendliness*)
- Escalabilidade (*Scalability*)

Em [LI004] é proposto um novo controle de congestionamento para redes de alta velocidade, que será descrito a seguir. Além disso, o grupo de estudo do BIC TCP já apresentou em 2005 uma nova versão deste, chamada CUBIC TCP [IN005] que também será descrita na seção 2.2.4 e tem como sua principal característica o fato de ser mais amigável (TCP *Friendly* [HA003]) que o BIC TCP quando testado com o TCP.

1ª parte: Busca de Incremento Binário (*Binary Search Increase)*

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:

Target Window = $(W_{Max} – W_{Min}) / 2$

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 Binary Search Increase* a janela corrente passa a ser W_{Max} e a nova janela após o decremento passa a ser o novo W_{Min}.

2ª parte: Incremento Aditivo (*Additive Increase)*

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 Min e Max ser menor que S_{max}

Ocorrendo perdas nos momentos de *Additive Increase*, 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 *Binary Search Increase* 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 *Additive Increase*, o BIC TCP roda um algoritmo chamado de partida lenta onde o incremento será Cwnd + 1, Cwnd + 2, ..., Cwnd + $S_{max.}$

   Onde Cwnd = Janela Corrente

Após a partida lenta ser rodada o BIC TCP passa para o modo *Binary Search Increase.*

2) Convergência Rápida (*Fast Convergence*): No *Binary Search Increase*, após uma redução da janela de transmissão, novos W_{Max} e W_{Min} são definidos. Se o novo W_{Max} é menor que o anterior, esta janela teve uma tendência descendente, com isso para garantir uma maior *fairness*[1] (justiça), quando existe mais de um fluxo deve-se reajustar o novo W_{Max} como sendo o primeiro valor de *Target Window*, ou seja:

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

2.2.4 CUBIC TCP

O CUBIC TCP é uma versão melhorada do BIC TCP e proposta pelo mesmo grupo que criou o BIC TCP. Em [IN005], Injong Rhee e Lisong Xu criam uma nova variante para o BIC TCP chamada de CUBIC TCP, que como o nome já diz, 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

[1] Fairness – Fluxos com menores RTT tendem a capturar uma maior banda

$K = ((ßW_{MAX}/C))^{1/3}$

ß – Constante que decrementa o tamanho da janela no momento de uma perda, ou seja, a janela reduz de $ßW_{MAX}$ no momento de uma perda. A figura abaixo mostra o comportamento das janelas de transmissão do BIC e do CUBIC.

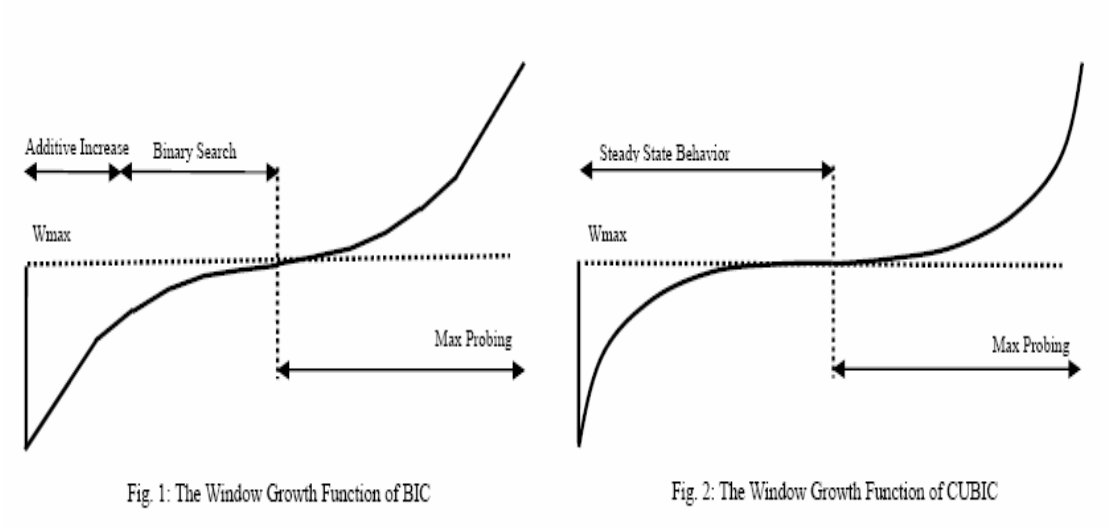


Figura 2:Curvas de crescimento das janelas do BIC e do CUBIC

Capítulo 3 O RMTP

3.1 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 *rate-based* com *acks* seletivos (*sacks*) para garantir confiabilidade
- Estimativa de banda através de medição de banda disponível

3.2 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.

Em outras palavras, na transmissão a aplicação envia dados para a camada de transporte que verifica os vários canais de rede disponíveis e determina qual canal irá utilizar para a transmissão dos dados naquele momento. Na recepção, a camada de transporte recebe os dados através dos vários canais da camada de rede e oferece estes dentro dos requisitos solicitados para a camada de aplicação.

Através da técnica de par de pacotes (*packet-pair),* utilizada pelo RMTP, tem-se como estimar o menor intervalo de tempo entre os pacotes, para que não ocorram problemas na comunicação entre transmissor e receptor dentre os quais destaca-se: 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 dos seguintes mecanismos deste protocolo tais como: confiabilidade, controle de congestionamento e controle de fluxo.

Para manter a confiabilidade e o seqüenciamento, o RMTP utiliza um mecanismo *window-based* com *acks* seletivos. O tamanho da janela de confiabilidade, chamado *window size*, define o espaço disponível no receptor para mensagens que devem ser aceitas e armazenadas. O valor de *window size* depende da estimativa de banda e atraso de todos os canais em uso naquele instante.

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

Como o tamanho da janela de confiabilidade deve acomodar uma quantidade de pacotes para que o protocolo não pare se acontecer uma perda, o RMTP define o tamanho da janela como sendo um múltiplo de 32 imediatamente maior que duas vezes a soma do produto *delay x bandwidth* de cada canal.

A variável responsável pelo controle do espaço disponível no *buffer* do receptor é chamada de *free buffer space*. A quantidade de espaço livre em *free buffer space* é que permite o avanço da janela de transmissão, se *free buffer space* é igual a zero o

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

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

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 *probes* nos canais ativos, para atingir a banda acessível. Quando o valor *de maximum rate* 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 *intial rate*, sendo seu valor obtido através da técnica de par de pacotes, porém, como o par de pacotes pode super estimar a banda na maioria de suas estimativas, o valor de *initial rate* será a meta metade do valor obtido no primeiro teste de par de pacotes.

3.3 Confiabilidade

O RMTP usa confiabilidade baseada em retransmissão e detecção de descontinuidades nos números de seqüência (*gap)* para identificar perdas. Os *frames* recebidos pelo receptor são notificados para o transmissor através do recebimento de

acks. Existem dois tipos de *acks*: acumulativo e seletivo. O *ack* acumulativo carrega o número do próximo *frame* esperado (último recebimento + 1). O *ack* seletivo é um mapa de bits (*bit map)* do estado da fila do receptor, sendo o bit 0 a posição do próximo *frame* esperado. O transmissor mantém um *bit map* individual por canal para identificar quais *frames* já foram enviados.

Para detectar um *gap*, o RMTP verifica espaços no *bit map* de *acks* seletivos em cada canal. Quando o transmissor envia um determinado *frame* o *bit map* do canal deste *frame* é marcado. Desta forma, ao ser recebido o *ack* seletivo é comparado com o *bit map* de cada canal, se for encontrado *gap* ou *gaps*, os *frames* perdidos são retransmitidos. É necessário usar a comparação dos *bit maps* porque quadros podem ser transmitidos por canais de diferentes atrasos, o que gerará fatalmente recepção fora de ordem. Um pacote só pode ser considerado perdido se o pacote enviado depois dele naquele canal já tiver sido recebido.

Uma vez retransmitido o *frame*, o protocolo não pode mais continuar a seqüência de detecção dos *gaps*, pois o *frame* retransmitido está fora de ordem. Devido a este fato torna-se necessário utilizar dois *bits maps* para controlar as retransmissões. Um *bit map* é chamado de *retransmitted frames*, que controla todos os *frames* retransmitidos e evita múltiplas retransmissões do mesmo pacote antes de verificar se este teve a chance de ser recebido com sucesso. Para verificar se a retransmissão foi perdida, o RMTP possui um segundo *bit map* chamado de *marker frames*, usado para indicar o *frame* que foi enviado imediatamente após a retransmissão. Se um *ack* do *marker frames* chega antes do *ack* do *retransmitted frames* o *frame* retransmitido foi perdido e com isso deve-se enviá-lo de novo.

O *bit map marker frames* possui uma função interessante à saber: mesmo depois de um *gap* ser detectado e um *frame* perdido ser retransmitido, todos os *acks* gerados antes do *frame* perdido que chegam ao receptor irão conter o mesmo *gap*. Assim, várias retransmissões desnecessárias vão ocorrer. Para evitar estas retransmissões, um *frame* no *bit map* de *retransmitted frames* só pode ser retransmitido novamente após a sua marcação no *bit map* de *marker frames* ser apagada. A posição no *bit map* de *marker frames* é apagada quando um *ack* ou um *gap* daquela posição é detectado. Essa técnica permite ao protocolo fazer todas as suas retransmissões sem usar *timers* explícitos.

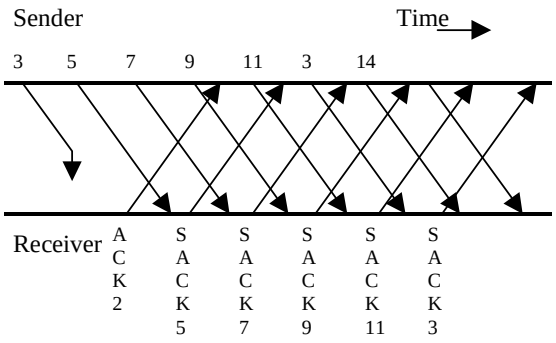


Figura 3:Detecção e retransmissão de um frame perdido

Para ilustrar o problema observe o exemplo da figura 3, usado por Magalhães em [MA005]. Um canal que transmite os *frames* 3 e 5, e recebe um *ack* do 5, teve o seu *frame* 3 perdido. Assim, o *frame* 3 será retransmitido pelo primeiro canal disponível, sendo assim retirado (apagado) do *bit map* do canal original e ligado no *bit map* do canal responsável pela retransmissão. Será também ligado no *bit map* de *retransmitted frames* e o *frame* seguinte deste canal será ligado no *bit map* de *marker frames*. Uma vez que o *frame* é retransmitido, ele só será retransmitido de novo, se o seu *marker frames* receber o *ack*. Se o *frame* 14 segue a retransmissão do *frame* 3 neste exemplo,

o *frame* 3 só pode ser retransmitido após o *ack* do *frame* 14 ser recebido. Como o *frame* 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 *frames* fora de ordem nos canais. *Frames* 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 *gap* pode falhar se todos os pacotes da *reliability window* forem perdidos. Para resolver este problema, em vez de usar um *time out* explicito, faz-se o seguinte: se *reliability window* estiver cheia, e nenhum frame acessível para retransmissão na fila do canal, então, as taxas caem à metade em todos os canais e os frames são retransmitidos, começando do último *frame* sem *ack*. Isto funciona como um mecanismo de *time out* natural. As taxas são reduzidas à metade para evitar um repentino descompasso entre o tamanho da janela e o produto *delay x bandwidth*, o que pode causar retransmissões desnecessárias.

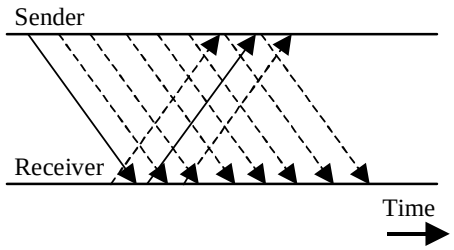


Figura 4:Produto Delay-Bandwidth em um ambiente Rate-Based

Se apenas um canal está sendo usado, o enchimento da *reliability window* pode ser causado por um tamanho da *reliability window* sub estimado. Para isso não acontecer,

o tamanho mínimo da *reliability window* deve ser maior que o dobro do *delay* da propagação da banda do canal mais um período extra para contabilizar a sincronização.

Na figura 4 existe um exemplo, também citado por Magalhães em [MA005], de um canal capaz de enviar 7 *frames* antes de receber um *ack*, neste caso, o tamanho mínimo da *reliability window* é 8. Se um tamanho menor for escolhido, todo o *frame* deve ser retransmitido duas vezes no caso de alguma perda. A *reliability window* deve poder armazenar nos *buffers frames* suficientes para permitir que o *ack* de um *frame* recebido a tempo evite a sua retransmissão. Se mais de um canal está sendo usado, então o *minimum window size* (MWS) é dado pela razão das bandas e do maior atraso de propagação. O MWS é definido como sendo o tamanho que os *buffers* precisam ter para acomodar todos os pacotes que são transmitidos entre o tempo que o pacote é enviado no canal de maior atraso e o tempo que o seu *ack* é recebido. O tempo total de propagação é o atraso de propagação de ida do canal mais lento mais o tempo de volta do canal mais rápido (*ack*). O número de pacotes transmitidos neste intervalo é o tempo total de propagação da banda no canal. O total de pacotes é a soma do número de pacotes transmitidos em cada canal. Portanto o MWS é dado por:

$$MWS = \sum_{i} (delay\_slowest\_channel + delay\_fastest\_channel) * rate\_channel(i)$$

Figura 5:Minimun Widow Size (MWS)

Canais podem ser adicionados depois de uma negociação inicial do *window size*. Um novo canal tem que obedecer ao requisito acima ou descompassos irão causar retransmissões desnecessárias. Se um canal violar os requisitos de tempo do MWS e for inserido no grupo de canais acessíveis, todo *frame* enviado será retransmitido por

um dos outros canais. Os *acks* dos *frames* enviados no canal lento não retornarão ao transmissor a tempo de evitar que a fila fique cheia, causando assim retransmissões.

Fixar o tamanho da *reliability window* com um valor igual ao MWS não é uma boa idéia. Se pacotes forem perdidos no canal com o maior atraso de propagação, isto pode fazer com que o protocolo trabalhe de forma intermitente, similar à síndrome conhecida como TCP *silly-window*.

3.4 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 um 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 negociada no início da transmissão, *maximum rate*. A *maximum rate* define um limite no número de pacotes 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 *maximum rate*. Quando o protocolo alcançar a *maximum rate* ele irá parar de testar a conexão, até acontecer perdas. 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. O controle de fluxo do RMTP trabalha como o do TCP, mas com uma fila de tamanho fixo: o cabeçalho do pacote carrega quantos *slots* estão vagos na fila do receptor. O transmissor irá parar de transmitir dados se a fila ficar cheia, diminuir a taxa à metade, enviar apenas *acks* do receptor mostrando que existe espaço novamente na fila do receptor, normalizando assim todo o processo.

3.5 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 banda disponível
- ✓ Estabilidade quando utilizado sozinho e na presença de outro tráfego
- ✓ Tenta ser um protocolo justo *(Fairness)*

O TCP superestima a banda do caminho a ser percorrido pelo pacote, causando perdas mesmo em condições estáveis. Esta característica nos encoraja a aceitar o desafio de tentar medir a banda disponível da rede sem causar problemas à mesma.

O HCC usa a técnica de par de pacotes e a medição do intervalo de chegada entre os pacotes para estimar a banda de transmissão ideal. O AIMD sem dúvida nenhuma é uma boa alternativa ao HCC, embora a existência de múltiplos caminhos (fluxos) no RMTP tenha levado a idéia da medição de banda.

3.5.1 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) [MO003] [MO103] [RA001] nos roteadores diminuir este tempo, sempre existirá um Δt 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 pacotes 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 pacotes.

3.5.2 A Técnica do Par de Pacote

O espaçamento de pacotes adjacentes causado pelo *link* de gargalo entre o transmissor e o receptor [figura 6], 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 pacote (*packet-pair*) 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 utilizam filas do tipo FIFO (*First In First Out*) estas medições podem ser sub-estimadas ou super-estimadas.

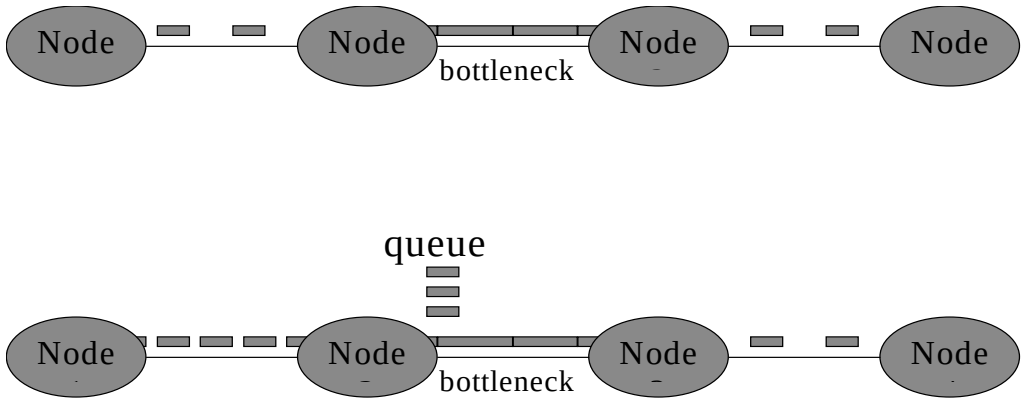


Figura 6: Se a taxa de envio está abaixo da taxa de serviço no link de gargalo, o tempo dos pacotes é mantido, se não uma fila é formada e os pacotes atrasados

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 7], 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].

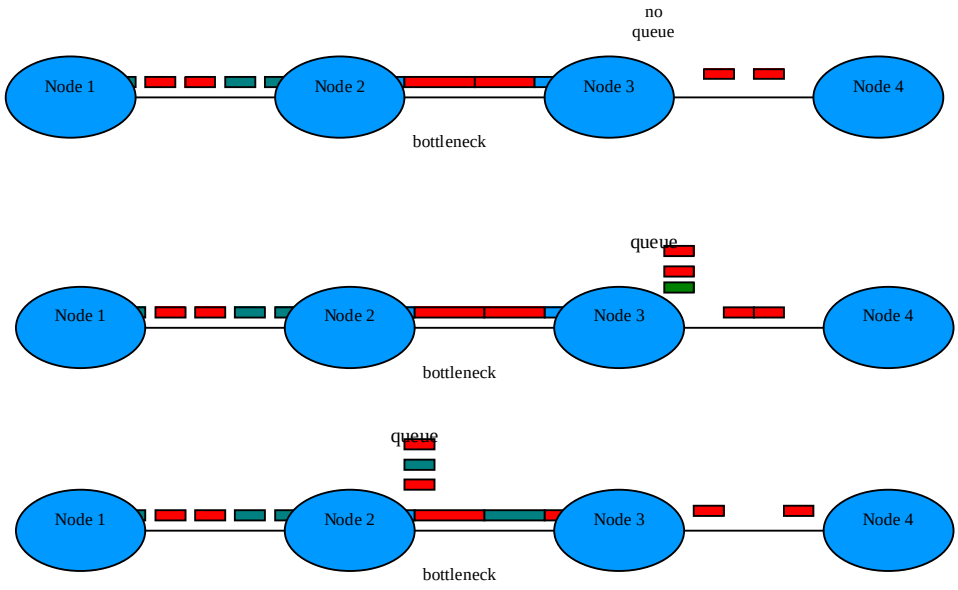


Figura 7: O tempo de chegada de dois pacotes consecutivos pode mudar de pendendo das condições da rede

3.5.3 Controle Homeostático (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 dos *buffers*, os roteadores são equipamentos que conseguem durante um certo intervalo de tempo receber uma quantidade de pacotes maior do que a capacidade que estes têm de encaminhá-los. Isto pode levar à violação da taxa máxima de serviço durante um tempo limitado.

Como o TCP tem sua transmissão baseada em *acks*, ele, naturalmente, gera rajadas que são enfileiradas nos *buffers* 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 os mesmos pacotes são recebidos [Figura 8], existe a possibilidade de *jitters* positivos e *jitters* negativos (isso vai depender de como estará a rede no momento).

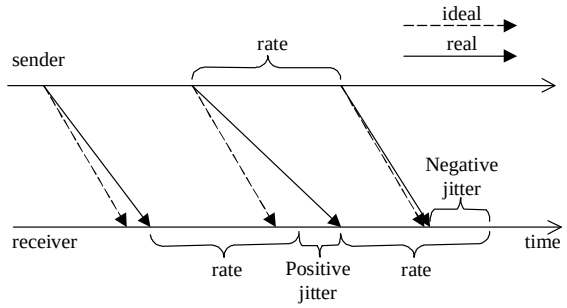


Figura 8: Exemplo de jitter positivo e jitter negativo

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 congestionado. 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 *probe packets* e são responsáveis em medir a banda acessível. A medição do tempo de chegada entre os dois *probe packets* indica o tempo mínimo de separação entre pacotes que a rede pode atingir.

Como o par de pacotes mede a capacidade do *link* e não a banda acessível, para impedir uma mudança abrupta, um novo valor é misturado com o valor corrente. Quanto mais passado, mais suave a curva, 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.

3.5.4 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

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

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 *link*. 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.

3.6 Cabeçalho do Pacote

O tamanho do cabeçalho do RMTP é variável, depende do tamanho do *bit map* dos *acks* seletivos. Os campos são os seguintes:

- *Port_number: integer*
- *type*: o tipo de pacote, *byte*
- *seq*: número de sequência, *integer*
- *last_sent*: número de sequencia do último pacote enviado neste canal, *integer*
- *rate*: a taxa do canal, *integer double*
- *expected*: o *acknowledgement* acumulado, o número do último pacote recebido em todos os canais mais um, *integer*
- *window*: tamanho do *buffer* livre do receptor, *int*
- *ack*: o *acknowledgement* seletivo, *variable*
- *checksum*: *integer*

O significado dos bits do campo *type* podem ser visto abaixo (Tabela 1):

Bit	7	6	5	4
If ON	Add Interface	Remove Interface	Interface Removed	Initial Packet
Bit	3	2	1	0
If ON	End Packet	Probe Packet	Ack Packet	Data Packet

Tabela 1: Tabela de [MA005] com o significado dos bits do campo type

Existem campos adicionais para mensagens com os bits 5, 6 e 7:

- IP: ip da interface
- PORT: porta da interface

Capítulo 4 Experimentos Realizados

4.1 Introdução

Neste capítulo serão apresentados os experimentos que verificam três questões importantes na análise de protocolos de transporte. Primeiramente, será feita uma comparação entre os protocolos para redes de alta velocidade, citados no capítulo 2, para verificar qual deles é mais amigável ao TCP na condição de rede proposta.

O segundo experimento deste capítulo tem a finalidade de mostrar 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 *links* de acesso constantes e variando os parâmetros do *link* de backbone.

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

4.2 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 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.

4.2.1 Instalação do NS (Network Simulator)

O NS (*Network Simulator*) é um simulador de rede que oferece um excelente suporte para o estudo de redes de computadores, sendo usado como o padrão para diversas pesquisas. Atualmente, o NS possui uma grande variedade de simulações 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.

O NS foi criado como uma variante do REAL *Network Simulator* em 1989, e vem sendo desenvolvido durante todos estes anos. Em 1995, o NS passou a ser desenvolvido pela DARPA através do projeto VINT na LBL, XEROX Parc, UCB e na USC/ISI.

Atualmente o NS é desenvolvido principalmente através da DARPA com a SAMAN e através da NSF com a CONSER, todos em colaboração com a ACIRI, apesar da UCB Daeddelus, CMU Monarch e Sun Microsystems estarem também realizando grandes contribuições na área de redes sem fio.

O código do NS 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 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 além de outros tópicos também interessantes para o uso do simulador.

Após a instalação do NS 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. 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 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. Para isso em ns-allinone-2.26/ns-2.26 digitou-se "make clean" <enter> e em seguida "make" <enter>.

4.2.2 Instalação dos Protocolos HSTCP, BICTCP e CUBICTCP

Para a instalação dos controles de congestionamento dos protocolos HSTCP, BICTCP e CUBICTCP no NS, 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, fazendo com que, todo e qualquer agente TCP criado no NS 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.

4.2.3 Instalação do Protocolo MulTCP

Foi solicitado ao Sr. Nabeshima, autor do protocolo MulTCP, o algoritmo do controle de congestionamento do MulTCP para o NS. 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 4.2 criando assim uma terceira versão destes arquivos com os controles de congestionamento dos quatro protocolos de alta velocidade estudados durante este trabalho.

4.2.4 Instalação do RMTP

Em [MA005], Magalhães descreve um controle de congestionamento para o NS 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 Multplexing Transport Protocol) [MA005].

Após a obtenção dos arquivos do RMTP (rmtp.cc e rmtp.h) foi preciso realizar várias configurações no NS 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 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
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 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.

4.3 Testes "TCP Friendly"

Nesta primeira seção de experimentos procuramos verificar se realmente os protocolos de alta velocidade são amigáveis ao TCP *(TCP Friendly)* [HA003] [DO000] [PA099], ou seja, se em certas condições de rede estes protocolos conseguem compartilhar a banda oferecida com fluxos TCP sem deteriorá-los.

À 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 nossa idéia aqui é fazer uma comparação entre os protocolos sob uma determinada condição de rede, ou seja, dos protocolos escolhidos (BIC TCP, CUBIC TCP,

HSTCP, MulTCP e RMTP), descobrir qual é o mais amigável ao TCP na situação proposta.

A topologia utilizada para esta seção de testes [Figura 9], corresponde ao tipo mais simples, ou seja, duas conexões (uma TCP e uma de alta velocidade) passando por um *backbone*.

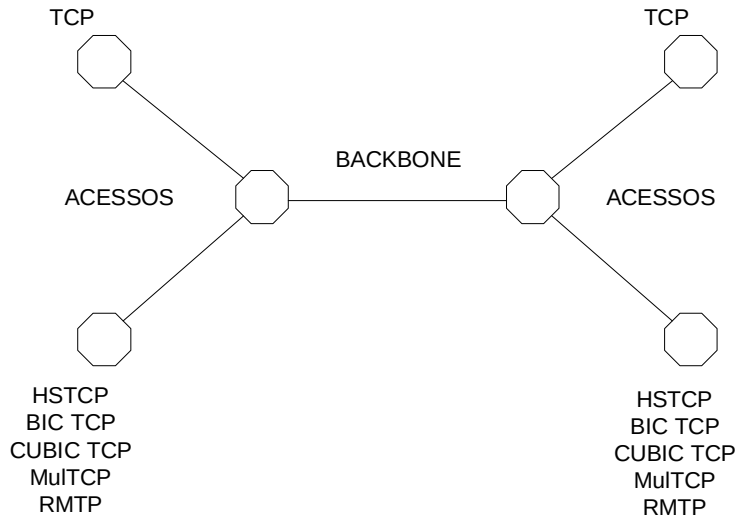


Figura 9: topologia de testes da seção 4.3

Portanto, esses protocolos foram testados um a um com o TCP em duas condições de rede conforme a tabela abaixo. Este ambiente de rede foi simulado através do *script* OTcl que encontra-se no anexo I deste trabalho.

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

Tabela 2: Condições de rede utilizadas na primeira bateria de testes

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 1. Na condição 1 temos um congestionamento razoável pois os dois fluxos de 750Mbps geram um excesso de banda de 50% no *backbone.* Já na condição 2 temos um alto grau de congestionamento, com a capacidade do *backbone* sendo ultrapassada em 100%.

Para o atraso de pacote na rede, os *links* 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 de *buffer* escolhido para os seguintes experimentos, foi o produto atraso x banda dos *links* multiplicado por 5. Já o tipo de fila escolhido para estes *buffers* 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*.

Todas as simulações tiveram 30 segundos de duração, este valor é considerado razoável para o alcance do estado estacionário entre dois fluxos em redes de alta velocidade.

4.3.1 TCP X BICTCP

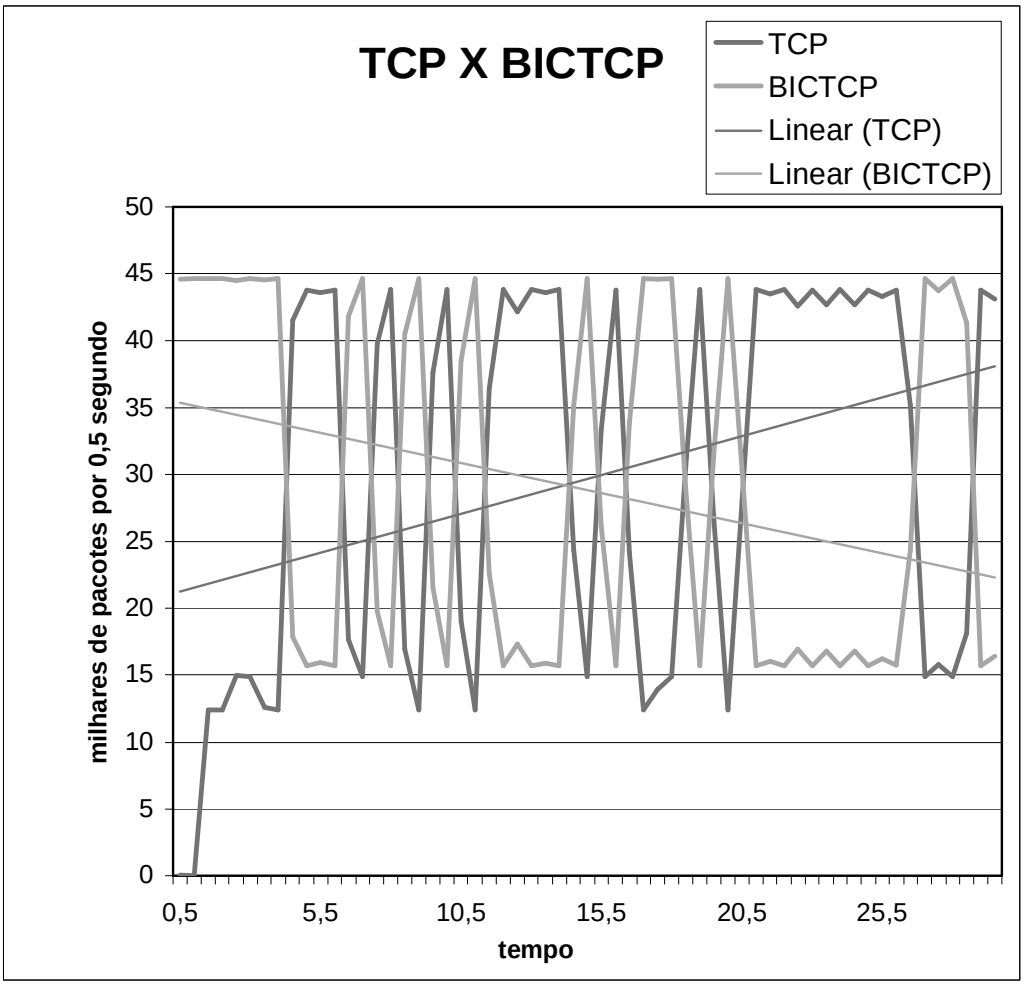


Figura 10: TCP x BICTCP sob Condição 1

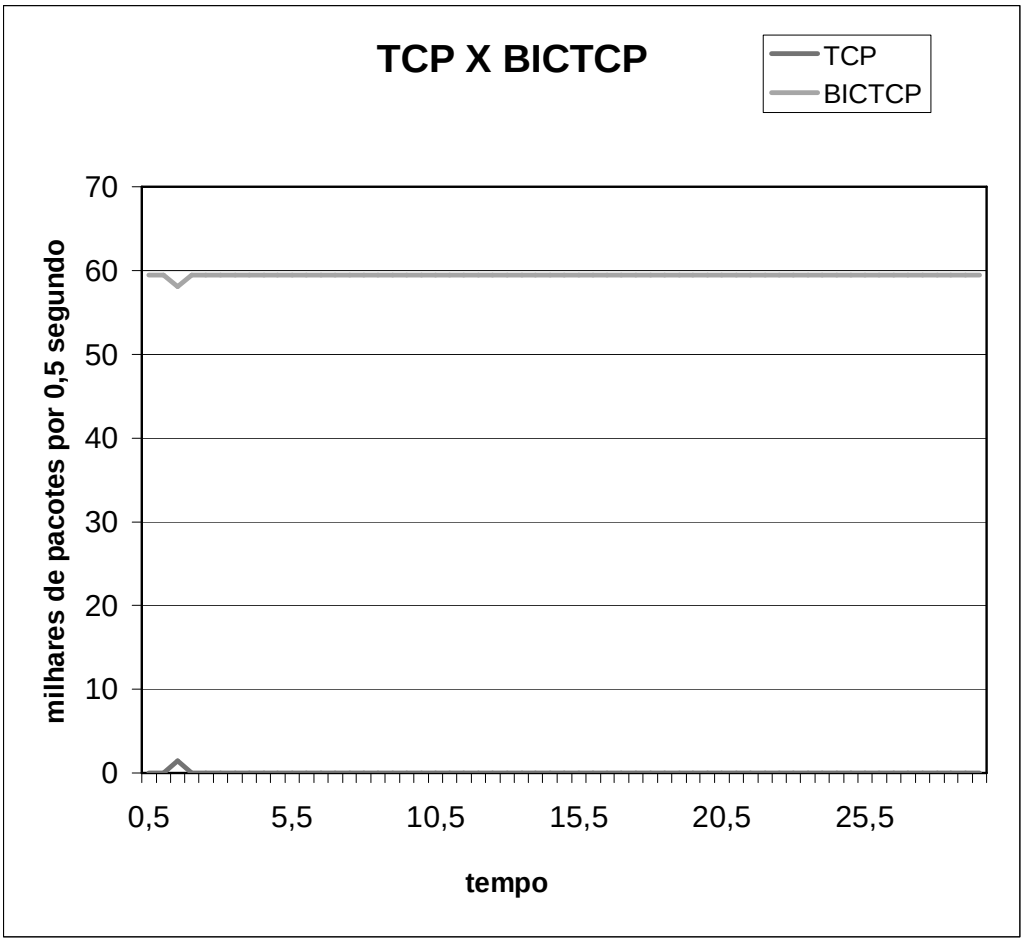


Figura 11: TCP x BICTCP sob Condição 2

Nesta seção podemos verificar que apesar do BICTCP ser amigável ao TCP na condição 1, na condição 2, que possui um grau de alto 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.

4.3.2 TCP X CUBICTCP

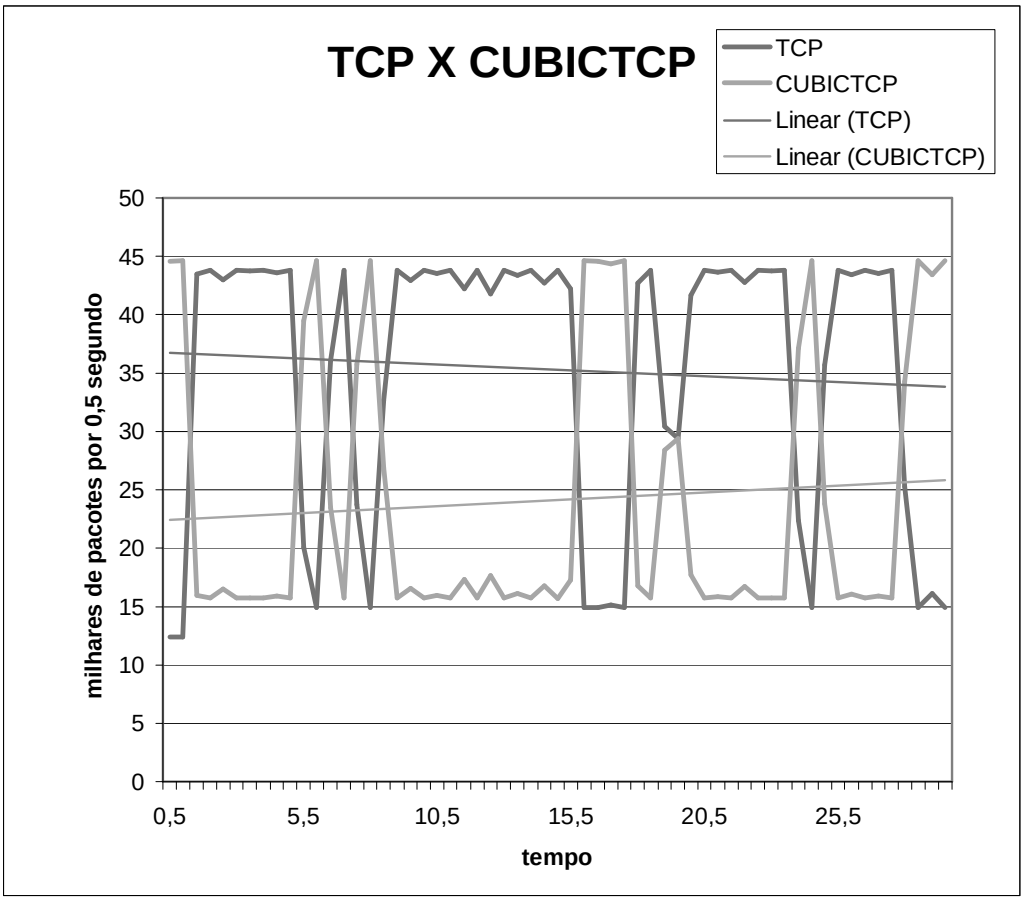


Figura 12: TCP x CUBICTCP sob Condição 1

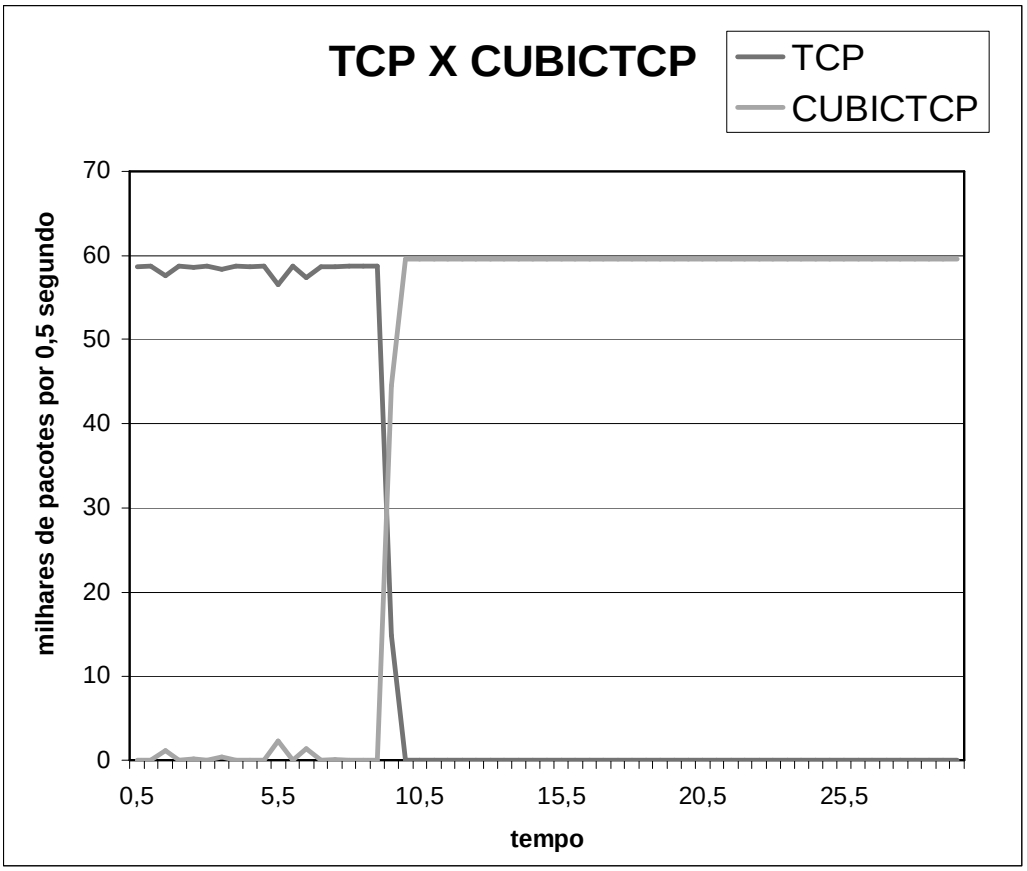


Figura 13: TCP x CUBICTCP sob Condição 2

No caso do CUBIC TCP realmente este se mostra mais suave em condições de alto grau de congestionamento, pois na condição 2 o fluxo TCP para somente com cerca de 13 segundos de duração do teste, o que não aconteceu com o BIC TCP. 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 manter a rede capaz de escoar os dois fluxos ao mesmo tempo.

4.3.3 TCP X HSTCP

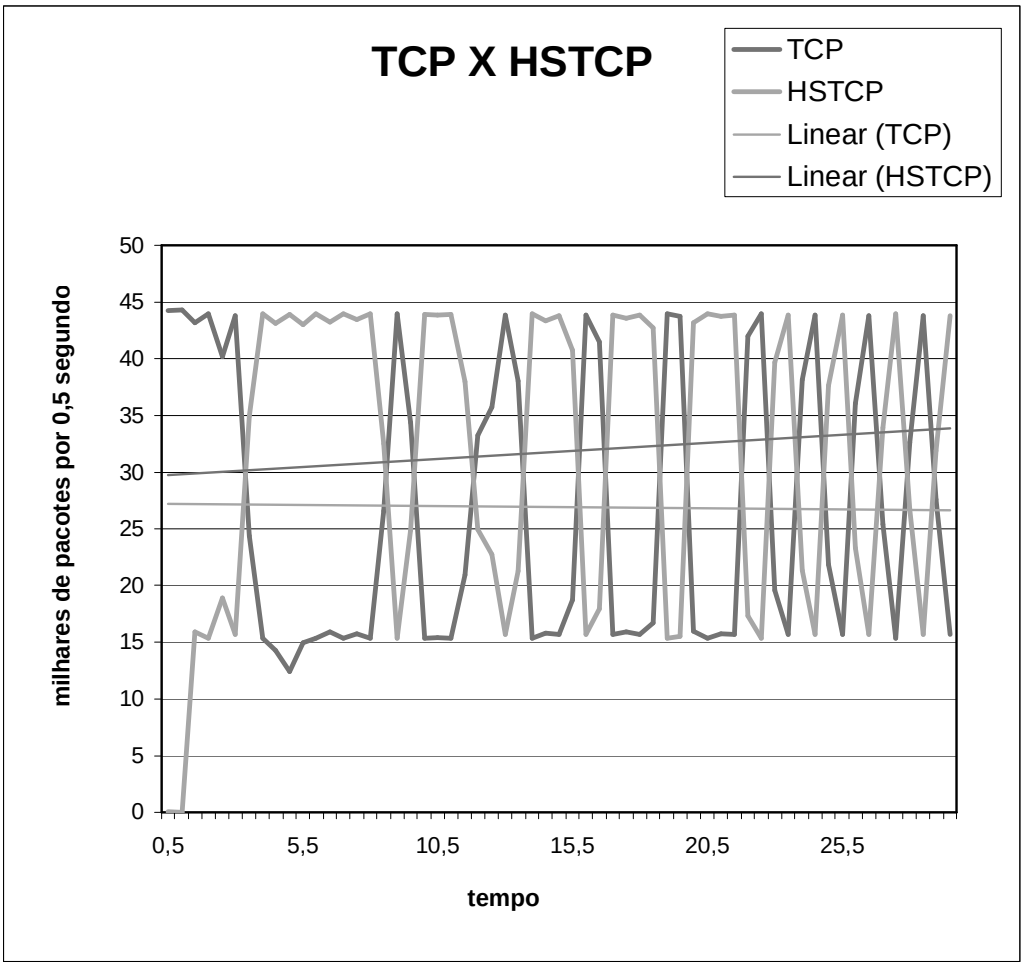


Figura 14: TCP x HSTCP sob Condição 1

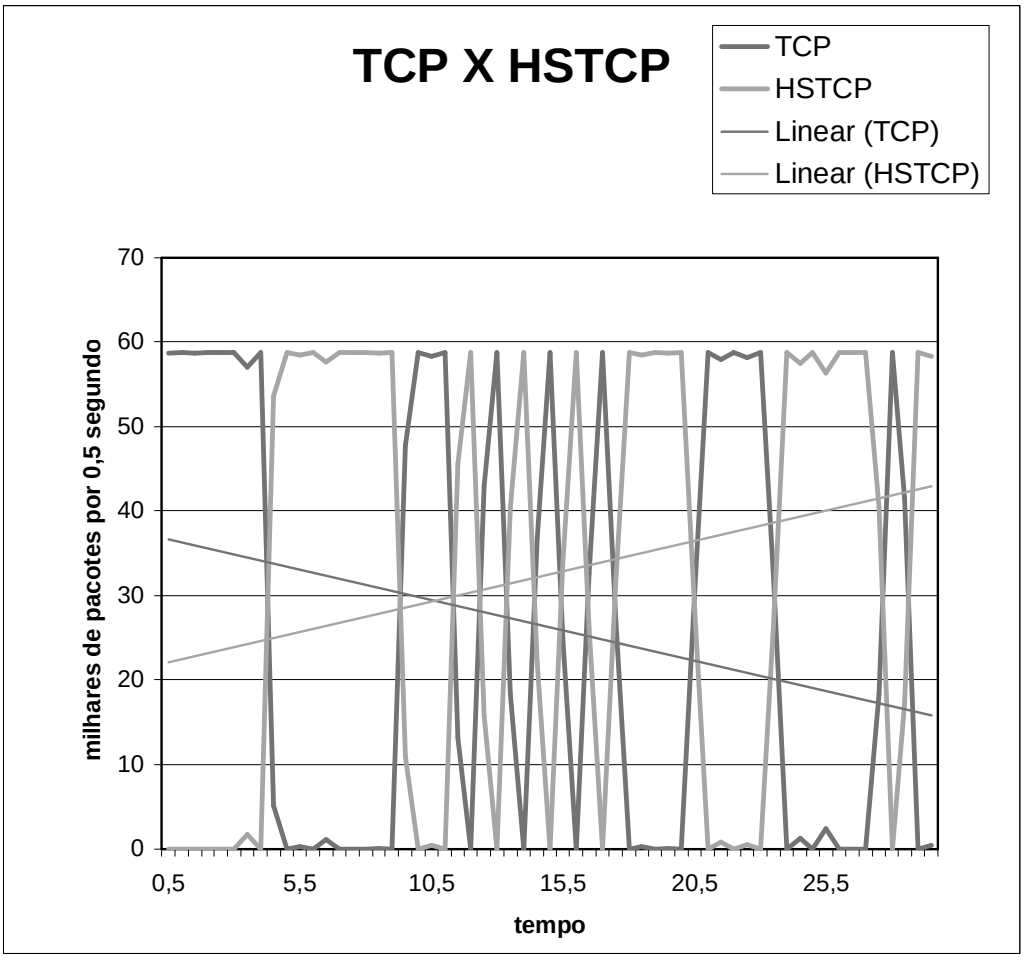


Figura 15: TCP x HSTCP sob Condição 2

Os gráficos acima provam que o HSTCP é o protocolo mais amigável ao TCP de todos os escolhidos para este 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, a única diferença é quando o tamanho da janela de transmissão chega ao valor 38 bytes, a partir daí como é verificado um fluxo de alta

velocidade esta janela de transmissão passa a crescer mais rápido e diminuir mais lentamente.

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.

4.3.4 TCP X MulTCP

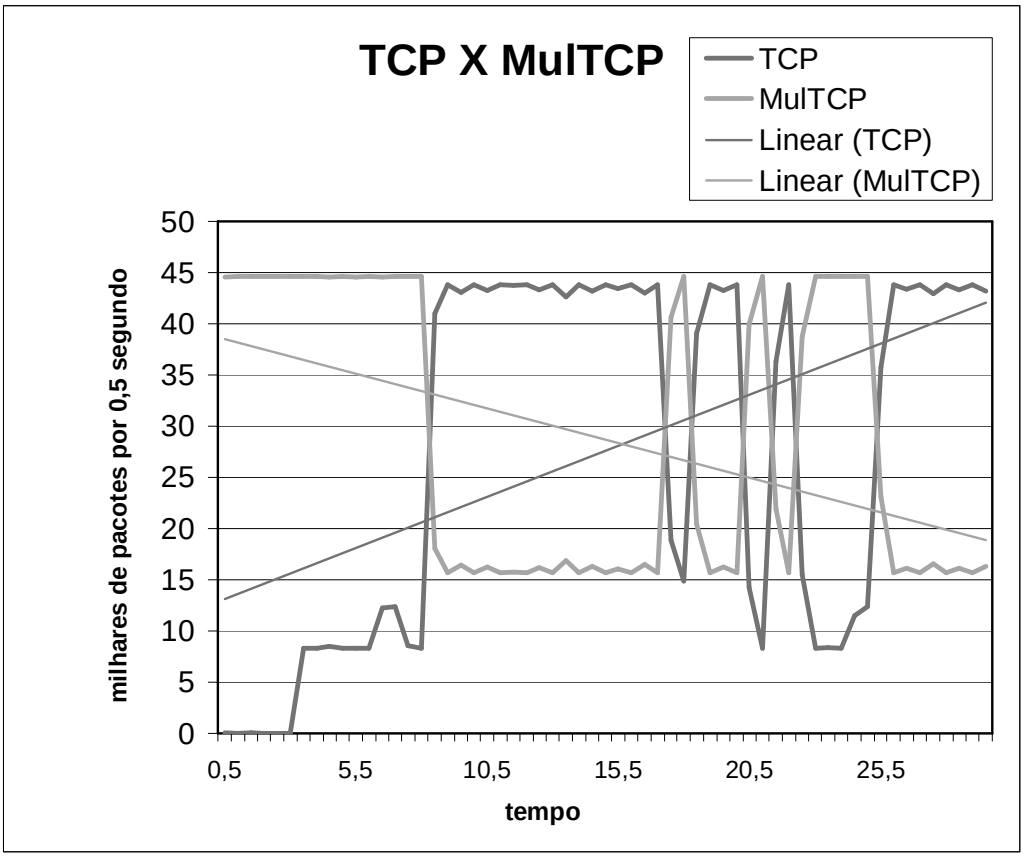


Figura 16: TCP x MulTCP sob Condição 1

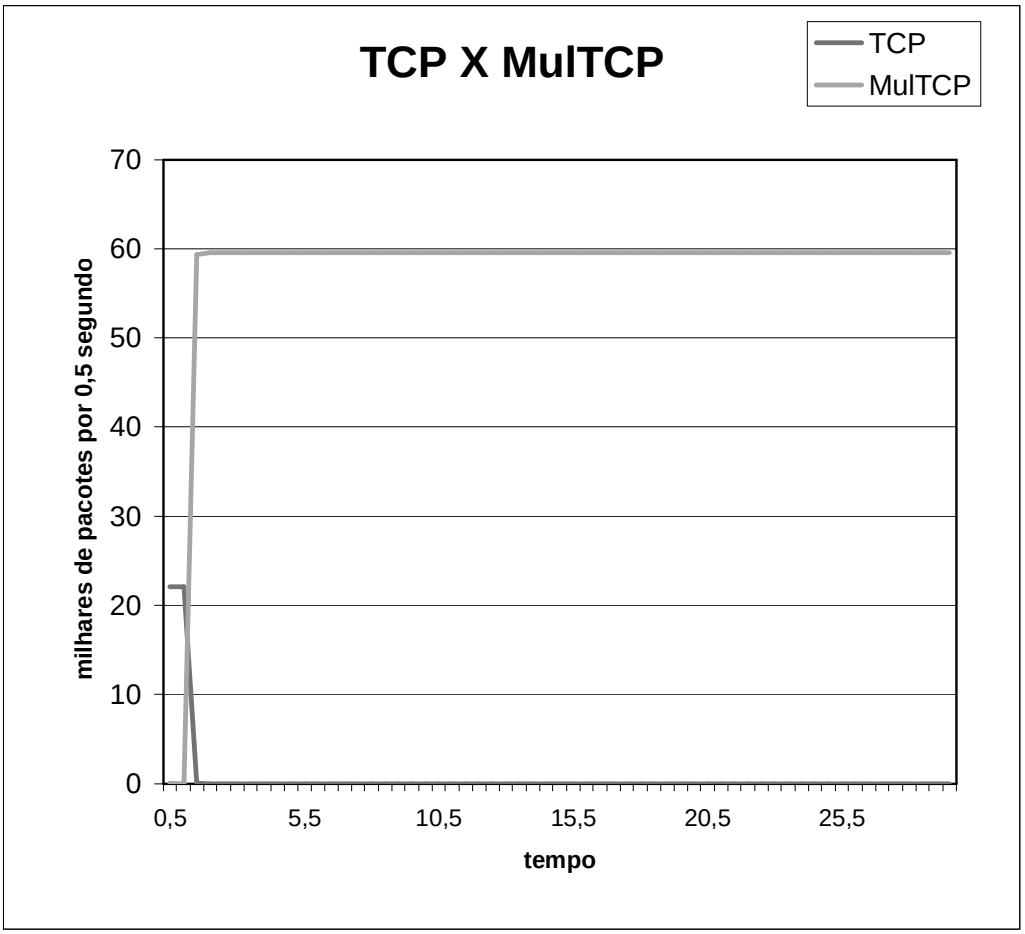


Figura 17: TCP x MulTCP sob Condição 2

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 com sensibilidade na condição de alto congestionamento (condição 2).

4.3.5 TCP X RMTP

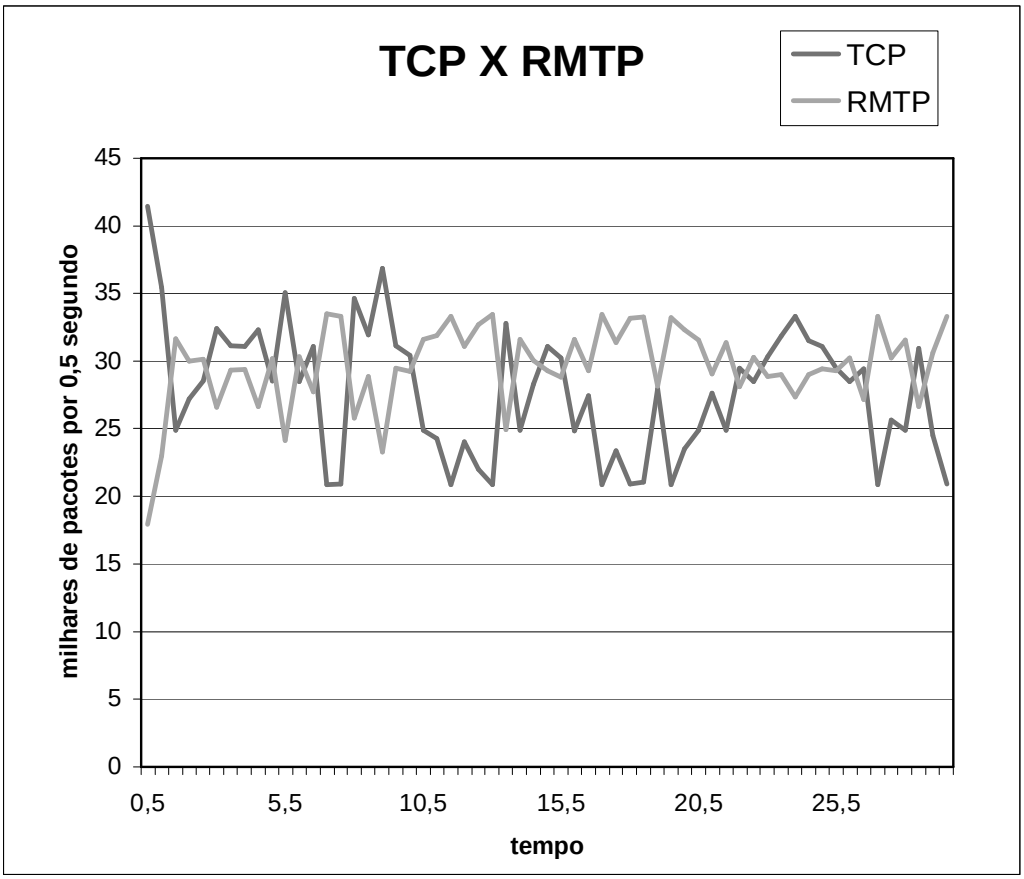


Figura 18: TCP x RMTP sob Condição 1

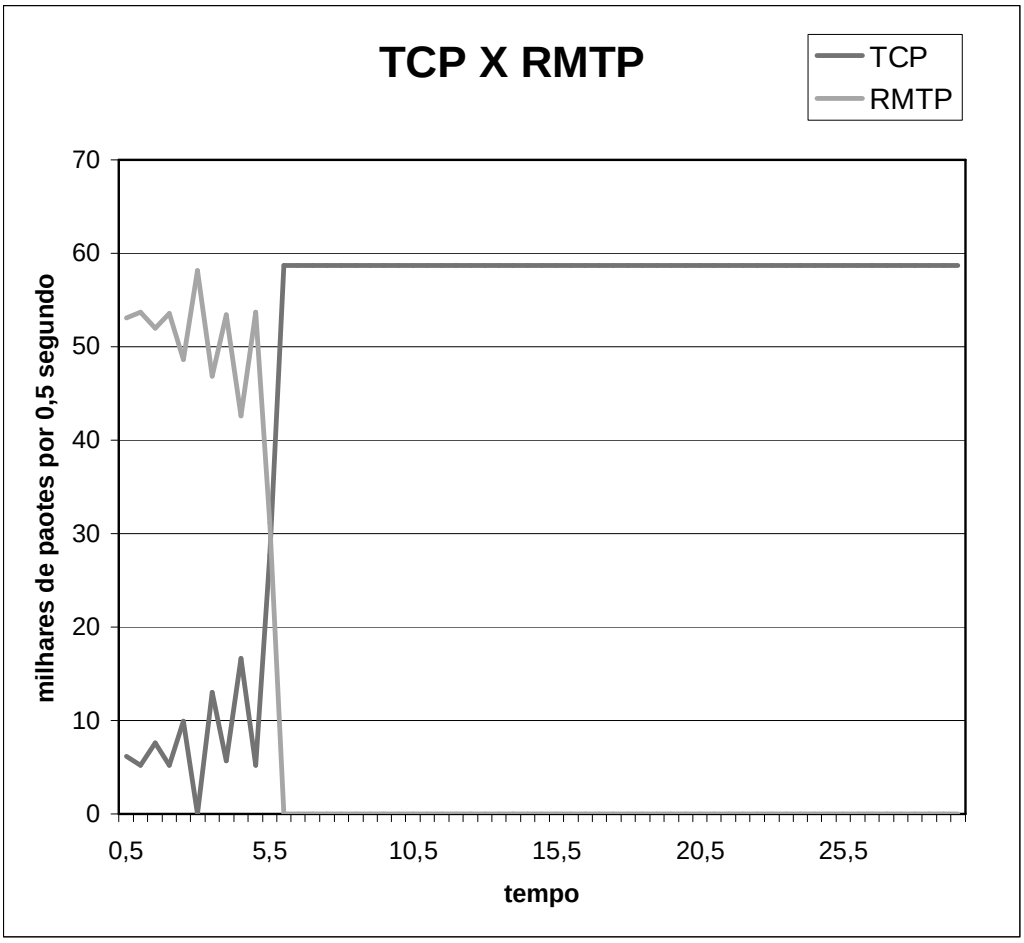


Figura 19: TCP x RMTP sob Condição 2

O RMTP possui junto com o HSTCP o melhor desempenho desta bateria de testes. Apesar de na condição 2 o RMTP ser amigável até demais com o TCP (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 estabilidade maior que nas seções anteriores, fazendo com que a vazão dos dois fluxos fique em torno de 30000 pacotes por 0,5 segundo. Podemos 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*.

4.4 Teste de Vazão com TCP, HSTCP, BIC TCP, CUBIC TCP, MulTCP e RMTP

Nesta seção, foram feitos experimentos 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 capacidade de escoamento de pacotes.

Foi verificado no capítulo 2, que como o TCP não possui capacidade de encher *links* da ordem de *gigabit* por segundo surgiram novos protocolos de transporte para atender a esta necessidade. Devido a esta afirmação o teste desta seção foi considerado o mais importante, pois mostrou qual dos protocolos foi capaz de através de um único fluxo utilizar toda abanda disponível de um enlace de dados.

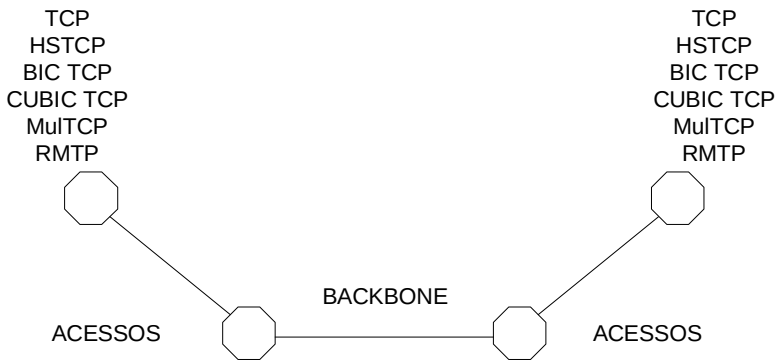


Figura 20: Topologia para os testes de vazão

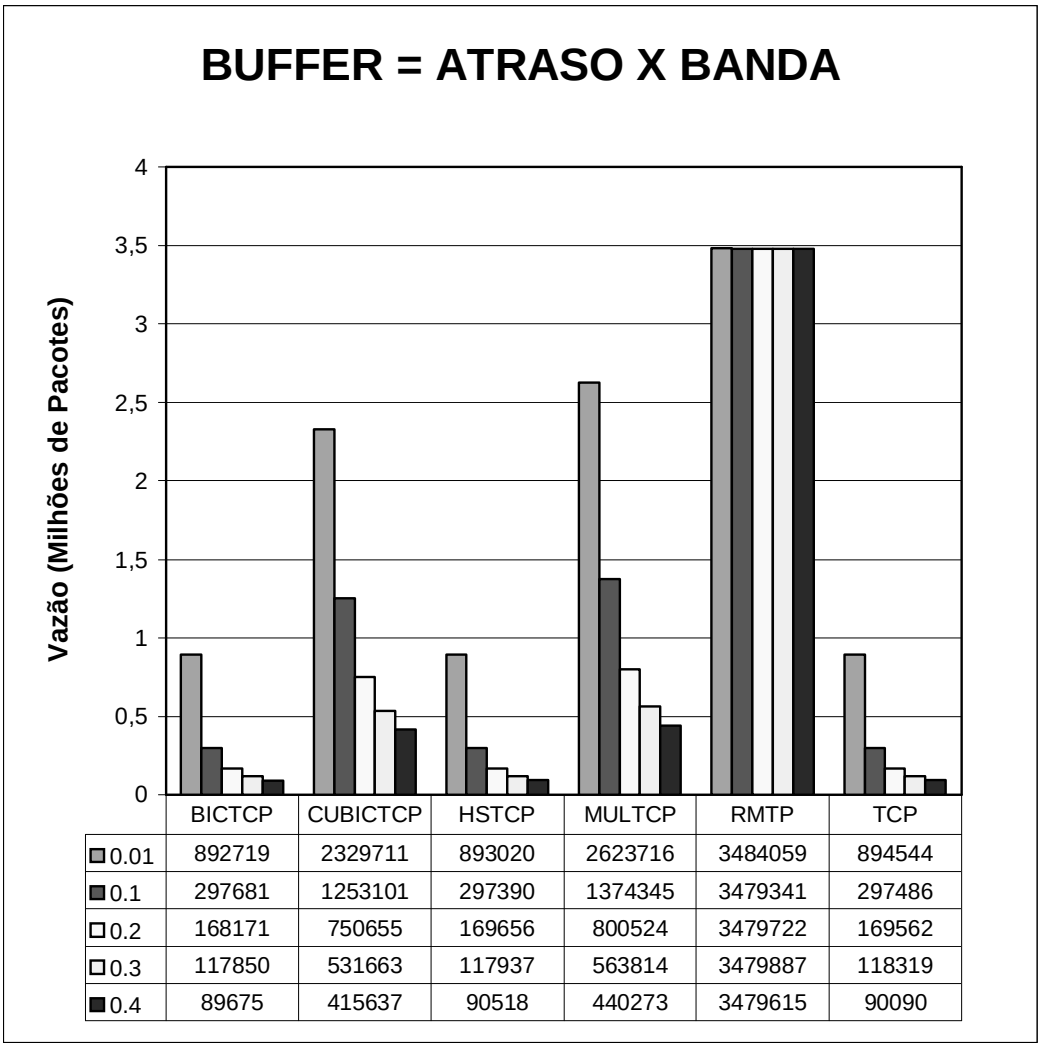
Para este teste também foi utilizada uma topologia simples [figura 20], com apenas um acesso que será utilizado pelos seis protocolos um de cada vez. Porém, para obter uma boa idéia da realidade, foram feitos testes em várias condições de rede. Apesar de

manter constante a velocidade, o atraso e o *buffer* dos acessos, no *backbone* foram realizadas variações no atraso e no tamanho do *buffer* com a intenção de simular um ambiente real de uma rede de dados. A tabela a seguir mostra como foi montado o teste desta seção.

ACESSO			BACKBONE		
BANDA	ATRASO	BUFFER	BANDA	ATRASO	BUFFER
1Gbps	0.01 ms	produto atraso x banda	1Gbps	0.01 ms 0.1 ms 0.2 ms 0.3 ms 0.4 ms	produto atraso x banda
1Gbps	0.01 ms	produto 5 x atraso x banda	1Gbps	0.01 ms 0.1 ms 0.2 ms 0.3 ms 0.4 ms	produto 5 x atraso x banda

Tabela 3: Situacões de rede para teste de vazão

O valor 0.01 ms de atraso nos *links* de acesso e no primeiro caso do *link* de *backbone* foi retirado de [KO003]. Neste artigo foi feito um estudo em um *backbone* real e verificou-se que em 155 Mbps o valor do atraso em um único *hop* era de 0.1 ms e em 622 Mbps o valor era 0.025 ms, logo para 1Gbps chegamos ao valor de 0.01 ms. Como temos na internet hoje 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 chegamos aos seguintes gráficos [Figura 21] e [Figura 22]. Estes experimentos também tiveram a duração de 30 segundo para cada situação descrita na tabela.



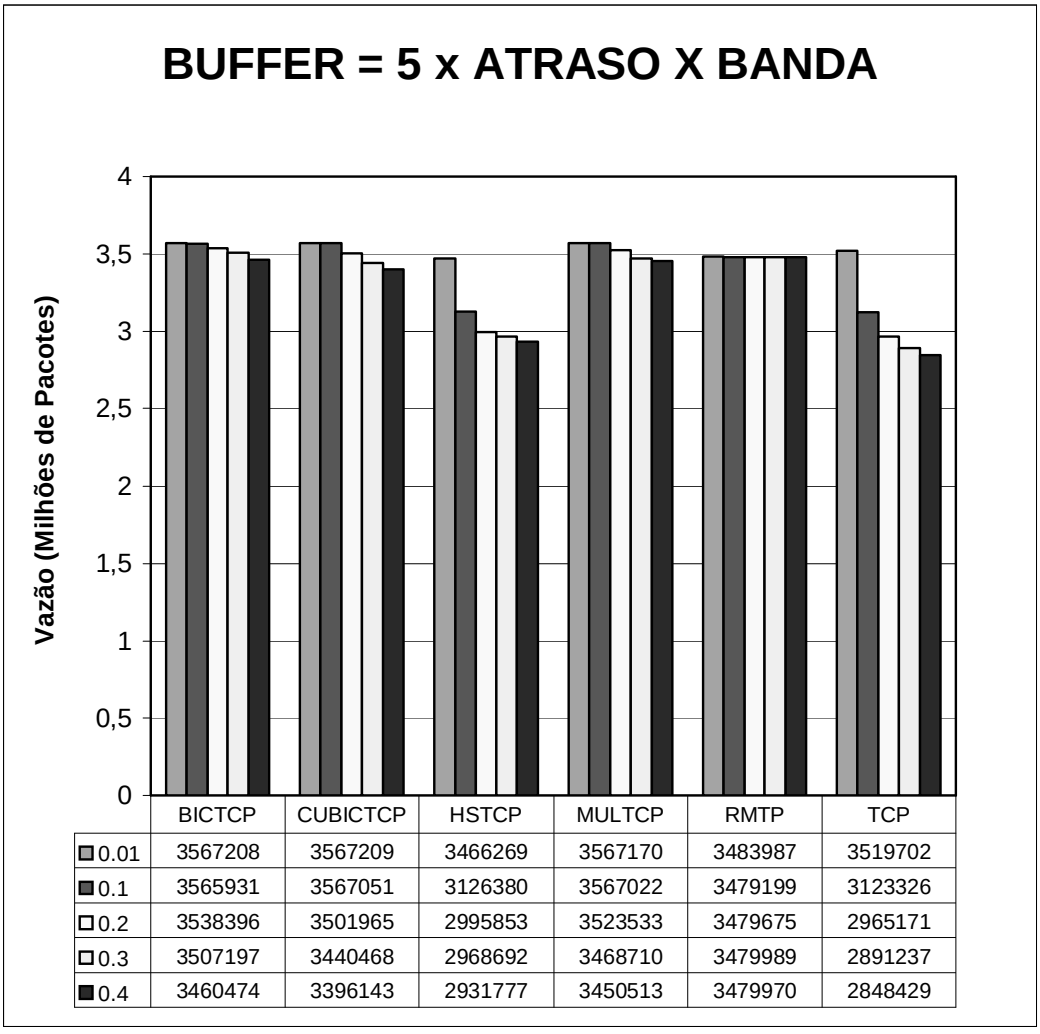
	BICTCP	CUBICTCP	HSTCP	MULTCP	RMTP	TCP
0.01	892719	2329711	893020	2623716	3484059	894544
0.1	297681	1253101	297390	1374345	3479341	297486
0.2	168171	750655	169656	800524	3479722	169562
0.3	117850	531663	117937	563814	3479887	118319
0.4	89675	415637	90518	440273	3479615	90090

Figura 21: Gráfico com vazão dos protocolos utilizando buffer = atraso x banda

Os dois gráficos desta seção mostram uma grande estabilidade do RMTP, apesar de o teste da seção 4.5 nos mostrar esta estabilidade em função do tempo, aqui, verificamos uma grande estabilidade da vazão do RMTP na presença de uma variação de 40 vezes o atraso.

Outra observação interessante é verificada no segundo gráfico, com o aumento do tamanho dos *buffers* nos *links* de acesso e de *backbone*, os protocolos de alta velocidade conseguem atingir uma vazão semelhante a do RMTP, o problema que este valor de 5 vezes o produto atraso banda é entendido como um valor grande além

de claro ser difícil para um roteador hoje em dia reservar esta enorme quantidade de memória para um *link* de 1Gbps



	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

Figura 22: Gráfico com vazão dos protocolos utilizando buffer = 5 x atraso x banda

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 os *buffers* sendo 5 vezes o tamanho do produto atraso banda. Esta é sem dúvida a grande motivação para estudos na área de protocolos de transporte para redes de alta velocidade.

4.5 Testes de Desempenho com TCP, HSTCP, BIC TCP, CUBIC TCP, MulTCP e RMTP

Esta nova fase de testes consistiu na busca do protocolo de maior estabilidade e portanto melhor desempenho em condições de alto tráfego de rede. Estes testes 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).

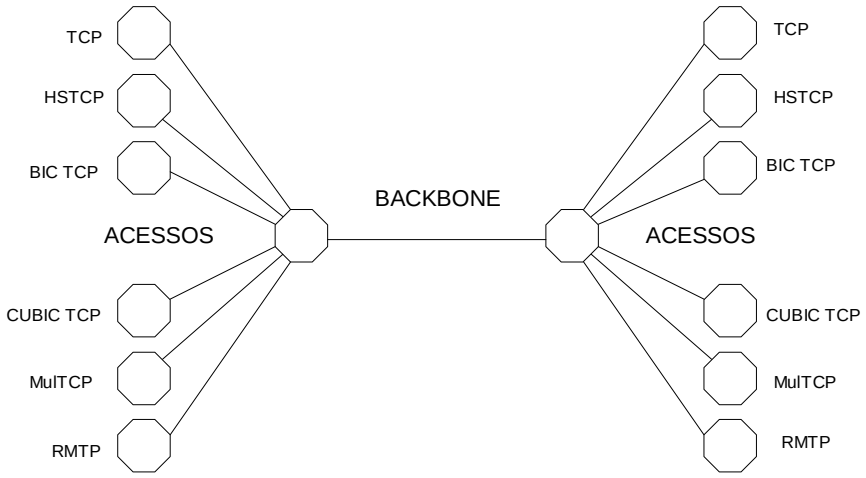


Figura 23: Topologia dos testes da seção 4.26

Utilizando a topologia da figura 23, criou-se uma condição de rede para a avaliação de desempenho dos controles de congestionamento, quando estes disputam recursos da rede entre si. Para este teste foi utilizado um segundo *script* feito em linguagem OTcl [Anexo II] que gera seis fluxos de acesso sendo um fluxo para cada protocolo estudado e com velocidade de 250 Mbps cada, 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 dos *buffers* tanto nos *links* de acesso como nos *links* de *backbone* utilizamos o produto atraso x banda. O tipo de fila também nos dois tipos de *links* foi o *drop* tail. O tempo de duração deste teste assim como os testes do item anterior foram de 30 segundos.

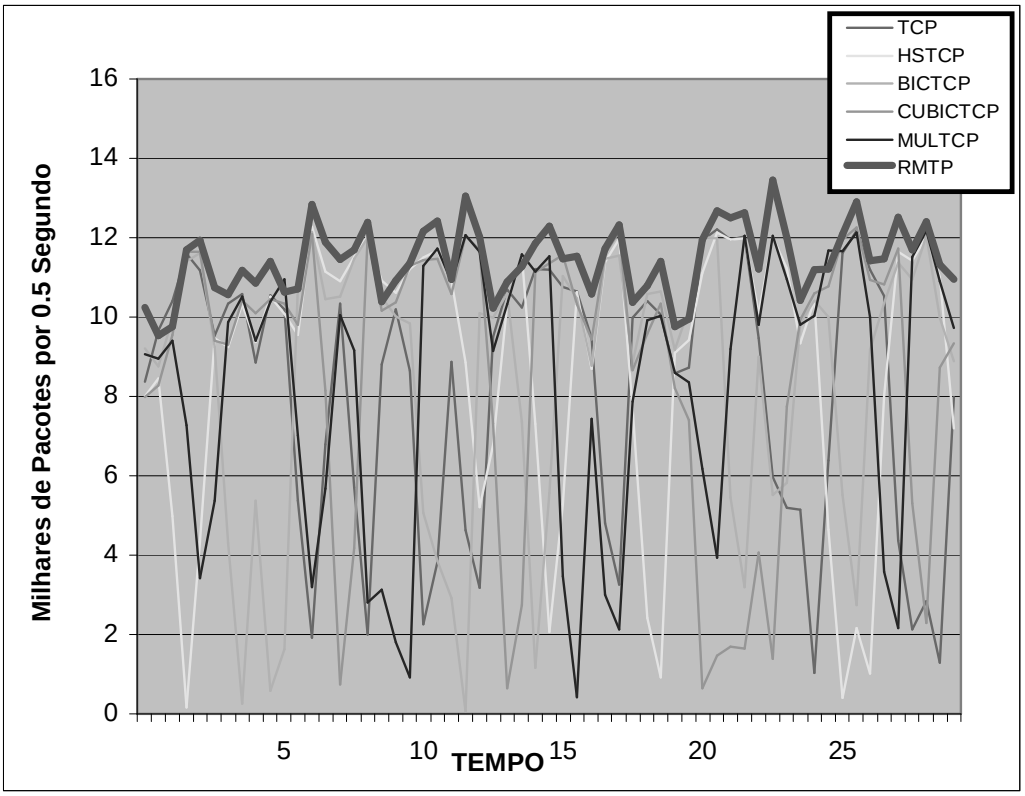


Figura 24: Gráfico de vazão com todos os protocolos disputando banda

Analisando o gráfico da figura 24, observamos que o protocolo RMTP mantém uma banda maior que os outros (linha grossa e verde escura), isto se deve as 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 abaixo 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.

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

Tabela 4: vazão dos protocolos no teste da seção 4.5

Capítulo 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, BIC TCP, CUBIC TCP, MulTCP e RMTP) enquanto que a segunda realiza um estudo comparativo entre estes protocolos.

Após a bateria de experimentos do capítulo 4, acreditamos ter atingido o nosso 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 principalmente às características do seu controle de congestionamento e do seu modo de enviar pacotes para a rede, atender as necessidades de um protocolo de transporte para rede de alta velocidade.

Dentro do nosso ponto de vista 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 nos *buffers* dos roteadores, protocolos baseados em taxa são menos agressivos a 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 4.3 o RMTP chega a estabilizar o fluxo TCP (que é naturalmente um fluxo de rajadas).

Sobre um ponto negativo do RMTP nestes testes é justamente a granularidade das variáveis no NS-2. Para hoje produzirmos estas simulações do capítulo 4 em uma rede

real, precisamos de 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 consequentemente de difícil utilização em larga escala, limitando assim o uso do RMTP.

Uma solução para este problema seria utilizar 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 de uma empresa conhecida como Big Foot Networks (www.bigfootnetworks.com) e utilizando como infra-estrutura uma rede de pesquisa da RNP conhecida como rede GIGA.

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 esta ligada diretamente à uma característica intrinsica deles que é transmitir de forma sempre crescente os dados e reduzindo de forma agressiva a sua taxa quando da ocorrência de congestionamento. Uma tentativa de acertar esta "falha" foi o TCP Pacing [AG000] que infelizmente se mostrou pior que o TCP.

Portanto, os protocolos baseados em taxa podem ser uma real 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* que é a seguinte: 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.

Outra questão interessante tange a caracterísrica diferente dos fluxos baseados em *acks* e os fluxos baseados em taxa, pensando também nos novos serviços que estão sendo viabilizados na internet tais como a TV Digital, podemos afirmar que fluxos com as características dos fluxos baseados em taxa podem ser mais interessantes para este tipo 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 desta tese ter sido realizada no NS-2 esperamos 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 sistemas operacionais para que através da utilização de distribuições Linux seja viável a utilização do RMTP em redes de alta velocidade.

Anexo I

```
# 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 II

```
# 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 III

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

Referências Bibliográficas:

[AG000] A. Aggarwal, S. Savage, e T. Anderson. Understanding the performance of TCP pacing. In Proceedings of the IEEE INFOCOM, páginas 1157--1165, Tel-Aviv, Israel, Março 2000.

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

[DO001] Dovrolis, C., P. Ramanathan, e D. Moore: 2001, "What do packet dispersion techniques measure?", In: IEEE Infocom, 2001

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

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

[HA003] M. Handley, S. Floyd, J. Padhye, J. 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

[JA092] Jacobson, V., e Braden, R., e Borman, D., “TCP Extensions for High Performance”, RFC 1323, Maio 1992

[KE092] S. Keshav, "A Control-Theoretic Approach to Flow Control," presented at Proceedings of the SIGCOMM '92 Symposium, 1992.

[KO003] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, C. Diot “Measurement and Analysis of Single-Hop Delay on an IP Backbone Network”, IEEE Journal on Selected Areas in Communications 2003

[KU003] Kurose, James, e Ross, Keith, “Redes de Computadores e a Internet Uma

Nova Abordagem – Pearson Education do Brasil – 1ª Edição”, 2003

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

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

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

[MO003] Marek Malowidzki, "Simulation-based Study of ECN Performance in RED Networks", SPECTS, 2003.

[MO103] Marek Malowidzki, "ECN is Fine - But Will It Be Used?", PDCN, 2003

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

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

[PA099] J. Padhye, J. Kurose, D. Towsley, and R. 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.

[PE003] Peterson, Larry, e Davie, Bruce, “Computer Networks A System Approach – Morgan Kaufmann – 3ª Edição”, 2003

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

[PO181] Postel, Jon, “Internet Protocol”, RFC791, Setembro 1981

[RA001] K. Ramakrishnan, S. Floyd, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC3168, Setembro 2001