UNIVERSIDADE FEDERAL FLUMINENSE
ESCOLA DE ENGENHARIA
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

MARCOS VINICIUS SALGADO MONTEIRO

UM ESTUDO DO PROTOCOLO HCC/UDT SOBRE REDES COM ALTO PRODUTO LARGURA DE BANDA-ATRASO

NITERÓI
2012

MARCOS VINICIUS SALGADO MONTEIRO

UM ESTUDO DO PROTOCOLO HCC/UDT SOBRE REDES COM ALTO PRODUTO LARGURA DE BANDA-ATRASO

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

Orientador: Prof^{o} Luiz Cláudio Schara Magalhães, PhD

NITERÓI

2012

MARCOS VINICIUS SALGADO MONTEIRO

UM ESTUDO DO PROTOCOLO HCC/UDT SOBRE REDES COM ALTO PRODUTO LARGURA DE BANDA-ATRASO

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

Aprovada em 10 de dezembro de 2012

BANCA EXAMINADORA

Prof. Dr. Luiz Cláudio Schara Magalhães
Universidade Federal Fluminense
(Orientador)

Prof. Dr. Kleber Vieira Cardoso
Universidade Federal de Goiás

Prof. Dr. Alexandre Sztajnberg
Universidade do Estado do Rio de Janeiro

Prof. Dr. Michael Anthony Stanton
Universidade Federal Fluminense

Niterói

2012

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

M775 Monteiro, Marcos Vinicius Salgado
Um estudo do protocolo HCC/UDT sobre redes com alto produto largura de banda-atraso / Marcos Vinicius Salgado Monteiro. – Niterói, RJ : [s.n.], 2012.
100 f.

Dissertação (Mestrado em Engenharia de Telecomunicações) - Universidade Federal Fluminense, 2012.
Orientador: Luiz Cláudio Schara Magalhães.

1. TCP/IP (Protocolo de redes de computação). 2. Algoritmo de controle de congestionamento homeostático. 3. Rede de computação. 4. HCC/UDT (Protocolo de redes de computação). 5. Comunicação de dados multimídia. I. Título.

CDD 004.6

Para Carolina e Raquel.

Potência não é nada sem controle. (Comercial de uma marca de pneus)

AGRADECIMENTOS

Gostaria de agradecer à minha esposa Raquel pelo incentivo e compreensão ao longo de todo este percurso.

À minha família e em especial aos meus pais (pai *in memorian*) pela minha formação. Também, aos meus irmãos por me proporcionarem condições de solidificar ainda no ensino médio a base de conhecimentos que me levou até aqui.

Ao meu orientador, Prof. Schara, pelo apoio sempre que necessário tanto nas orientações e esclarecimentos das dúvidas levantadas, como também por prover a infraestrutura necessária para os experimentos, além da sugestão do tema.

Aos demais professores, funcionários e alunos do curso de Mestrado e do laboratório Midiacom que de alguma forma, cada um ao seu modo, também contribuíram para este trabalho. Em especial à Livia Gerk e ao Diego Passos pelas constantes ajudas e pacientes explicações.

Aos ex-colegas de trabalho da Oi que fizeram com que a minha ausência não fosse impactante e em especial ao meu ex-gerente Islan Neves que além de permitir as minhas ausências foi um grande incentivador.

Ao meu amigo Miguelângelo que com muito samba e preciosos conhecimentos de programação tanto me auxiliou com suas consultorias.

E, por fim, aos colegas de trabalho da Petrobras que também me ajudaram. Em especial à Débora Santos que ajudou na revisão do texto de trabalho.

SUMÁRIO

LISTA DE ILUSTRAÇÕES 10
LISTA DE TABELAS 11
LISTA DE ABREVIATURAS E SIGLAS 12
RESUMO 13
ABSTRACT 14
1 INTRODUÇÃO 15
1.1 PROTOCOLOS DE TRANSPORTE 16
1.2 CONTROLE DE CONGESTIONAMENTO 18
1.3 OBJETIVOS DO TRABALHO 19
1.4 ORGANIZAÇÃO DO TRABALHO 19
2 TRABALHOS RELACIONADOS 20
2.1 INTRODUÇÃO 20
2.2 O CONTROLE DE CONGESTIONAMENTO PADRÃO DO TCP 22
2.3 O PROBLEMA 24
2.4 MODIFICAÇÕES NO TCP 24
2.4.1 HIGH SPEED TCP (HS-TCP) 24
2.4.2 SCALABLE TCP (S-TCP) 27
2.4.3 HAMILTON TCP (H-TCP) 28
2.4.4 BIC (BINARY INCREASE CONGESTION CONTROL) 29
2.4.5 CUBIC 31
2.4.6 FAST 32
2.4.7 C-TCP (*COMPOUND* TCP) 34
2.5 APLICAÇÕES QUE UTILIZAM O UDP 35
2.6 MUDANÇAS NA ESTRUTURA DA REDE 35
2.7 RESUMO 36
3 CONTROLE DE CONGESTIONAMENTO HOMEOSTÁTICO, HCC 38
3.1 INTRODUÇÃO 38
3.2 MECANISMOS BASEADOS EM TAXA 39
3.3 VARIAÇÃO DO ATRASO – JITTER 41
3.4 AJUSTES NO MECANISMO DE CONTROLE DO CONGESTIONAMENTO 43
3.5 O ALGORITMO DE CONTROLE DE CONGESTIONAMENTO HOMEOSTÁTICO 44
3.6 SUMÁRIO DO CAPÍTULO 3 45
4 A IMPLEMENTAÇÃO 46
4.1 INTRODUÇÃO 46
4.2 O UDT 46
4.2.1 ARQUITETURA 47
4.2.2 QUESTÕES DE IMPLEMENTAÇÃO 49
4.2.3 ESTRUTURA DOS PACOTES 52
4.2.4 TEMPORIZADORES 54
4.2.5 ENVIO E RECEBIMENTO DE DADOS 55
4.2.6 ESQUEMA DE COMPRESSÃO DE INFORMAÇÃO DE PERDA 55
4.2.7 CONTROLE DE CONGESTIONAMENTO CONFIGURÁVEL 56
4.3 IMPLEMENTAÇÃO DO HCC NO UDT 57
4.3.1 O ALGORITMO 58
4.4 SUMÁRIO DO CAPÍTULO 4 58
5 A AVALIAÇÃO DO HCC 60
5.1 INTRODUÇÃO 60
5.2 ESTRATÉGIA DE AVALIAÇÃO 60
5.3 TESTES REALIZADOS 61
5.3.1 A EFICIÊNCIA EM RELAÇÃO À VAZÃO 61
5.3.2 EQUIDADE ENTRE DIFERENTES FLUXOS UTILIZANDO O HCC 62
5.3.3 EQUIDADE COM O TCP 62
5.3.4 CONVERGÊNCIA PARA A BANDA DISPONÍVEL 63

5.3.5 ESTABILIDADE....63
5.4 CENÁRIO DE TESTES....64
5.4.1 EMULANDO A WAN....67
5.5 REALIZAÇÃO DOS TESTES....67
5.5.1 AS APLICAÇÃOES....67
5.5.2 GERAÇÃO DOS ARQUIVOS COM AS ANÁLISES....69
5.6 RESULTADOS DOS TESTES....69
5.6.1 EFICIÊNCIA EM RELAÇÃO À VAZÃO.....69
5.6.2 EQUIDADE ENTRE DIFERENTES FLUXOS UTILIZANDO O HCC....72
5.6.3 EQUIDADE COM O TCP....73
5.6.4 CONVERGÊNCIA PARA A BANDA DISPONÍVEL....81
5.6.5 ESTABILIDADE....82
5.7 SUMÁRIO DO CAPÍTULO 5....85
5.7.1 SUMÁRIO DOS RESULTADOS....85
6 CONCLUSÕES E TRABALHOS FUTUROS....87
6.1 TRABALHOS FUTUROS....88
REFERÊNCIAS....89
APÊNDICE A – SCRIPTS DE AUTOMATIZAÇÃO....93
APÊNDICE B – MODIFICAÇÕES NOS SISTEMAS OPERACIONAIS....99
ANEXO A - NETEM....100

LISTA DE ILUSTRAÇÕES

Figura 1-1 - Arquitetura OSI e TCP/IP....17
Figura 2-1 - Variantes do TCP para rede alta velocidade 21
Figura 2-2 - Dinâmica da Janela de Congestionamento do TCP Padrão 23
Figura 2-3 - Objetivo do HS-TCP 25
Figura 2-4 - A dinâmica da janela de congestionamento do HS-TCP 27
Figura 2-5 - Janela de Congestionamento do S-TCP....28
Figura 2-6 - Busca Binária do BIC....30
Figura 2-7 - Dinâmica do BIC....31
Figura 2-8 - Dinâmica da Janela de Congestionamento do CUBIC....32
Figura 3-1 - Método do par de pacotes.....40
Figura 3-2 - Compressão e Expansão do tempo entre os pacotes consecutivos 41
Figura 3-3 - Medida de *Jitter* Positivos e Negativos 42
Figura 4-1 - Arquitetura UDT/CC....47
Figura 4-2 - Implementação UDT/CC.....49
Figura 4-3 - Inserção do buffer de aplicação no buffer do protocolo.....51
Figura 4-4 - Estrutura do cabeçalho de um pacote de dados UDT 52
Figura 4-5 - Estrutura de um pacote de controle UDT 53
Figura 5-1 - Topologia de rede....65
Figura 5-2 - Fila do Kernel....66
Figura 5-3 - Índice de Eficiência do HCC/UDT e TCP/UDT....71
Figura 5-4 - Índice de equidade intraprotocolo....72
Figura 5-5 - Teste de equidade para o cenário de fluxo B 75
Figura 5-6 - Fluxos FB com 150ms de atraso e taxa de perda de 1E-4....75
Figura 5-7 - Fluxos FF com 150ms de atraso e taxa de perda de 1E-4....76
Figura 5-8 - Fluxos FD com 150ms de atraso e taxa de perda de 1E-4 77
Figura 5-9 - Teste de Equidade para o cenário FC 77
Figura 5-10 - Teste equidade para o cenário FD....78
Figura 5-11 - Teste equidade para o cenário FE....80
Figura 5-12 - Convergência do HCC....82
Figura 5-13 - Índice de estabilidade DE para o cenário de Fluxos FA 83
Figura 5-14 - Estabilidade entre fluxos HCC com valores mais baxos de perda e atraso. 83
Figura 5-15 - Estabilidade entre fluxos HCC com valores mais altos de perda e atraso. 84
Figura 5-16 - Índice de estabilidade DE para os cenários de Fluxo FB, FC, FD e FE 85

LISTA DE TABELAS

Tabela 2-1 – Variantes do TCP 36
Tabela 5-1 – Exemplo de resultado impresso pela aplicação cliente. 68
Tabela 5-2 - Resultado dos testes de Eficiência do HCC/UDT 70
Tabela 5-3 - Resultado dos testes de Eficiência do TCP/UDT 71
Tabela 5-4 - Resultado dos testes de equidade intraprotocolo com os valores das médias calculadas de cada fluxo e o desvio-padrão. 73
Tabela 5-5 - Resultado dos testes de equidade com o TCP no cenário FB, com quatro Fluxos HCC e um Fluxo TCP 74
Tabela 5-6 - Resultado dos testes de equidade com o TCP no cenário FC, com três Fluxos HCC e dois fluxos TCP 79
Tabela 5-7 - Resultado dos testes de equidade com o TCP no cenário FD, com dois Fluxos HCC e três fluxos TCP 79
Tabela 5-8 - Resultado dos testes de equidade com o TCP no cenário FE, com um Fluxo HCC e quatro fluxos TCP. 81

LISTA DE ABREVIATURAS E SIGLAS

ACK	:	*Acknowledge*
AIMD	:	*Additive Increase Multiplicative Decrease*
API	:	*Application Program Interface*
ARP	:	*Address Resolution Protocol*
CBR	:	*Constant Bit Rate*
CC	:	*Congestion Control*
CCC		Controle de Congestionamento Configurável
CPU	:	*Central Processing Unit*
DAIMD	:	*AIMD with decreasing increases*
E/S	:	Entrada e Saída
FIFO	:	*First In First Out*
HCC	:	*Homeostatic Congestion Control*
IETF	:	*Internet Engineering Task Force*
IP	:	*Internet Protocol*
KLT	:	*Kernel Level Thread*
MIMD	:	*Multiplicative Increase Multiplicative Decrease*
NAK	:	*Negative Acknowledgde*
PLBA	:	Produto Largura de Banda - Atraso
OSI	:	*Open Systems Interconnection*
RAM	:	*Random Access Memory*
RFC	:	*Request For Comment*
RTO	:	*Retransmission Timeout*
RTT	:	*Round Trip Time*
SCTP	:	*Stream Control Transmission Protocol*
SO	:	Sistema Operacional
TCP	:	*Transport Control Protocol*
UDP	:	*User Datagram Protocol*
UDT	:	*UDP Data Transfer Protocol*
ULT	:	*User Level Thread*
VLAN	:	*Virtual Local Area Network*
WAN	:	*Wide Area Network*

RESUMO

A ineficiência do protocolo de transporte TCP padrão para redes de alta velocidade ficou evidente no momento em que estas redes começaram a ser implantadas. Com isto, surgiram diversas propostas para resolver esta ineficiência.

Dentre essas propostas, há a substituição do mecanismo de controle de congestionamento baseado em janela, utilizado no TCP, pelo mecanismo baseado em taxa no qual a largura de banda é medida e depois os pacotes são enviados em períodos pré-definidos de forma a utilizar melhor os recursos disponíveis da rede. Na implementação desse mecanismo, utiliza-se o UDP como protocolo de transporte e faz-se o controle de congestionamento na camada de aplicação.

Este trabalho apresenta um estudo do desempenho do algoritmo de controle de congestionamento homeostático, HCC, implementado sobre o arcabouço do protocolo UDT para redes de alta velocidade com um alto produto largura de banda-atraso.

Os resultados demonstraram que o HCC/UDT utiliza eficazmente a largura de banda disponível, apresenta boa equidade tanto entre fluxos somente HCC quanto entre fluxos HCC e TCP compartilhados, converge rapidamente tanto para alcançar a vazão máxima quanto para diminuir a taxa de envio de pacotes na presença de tráfego interveniente. E, apresenta boa estabilidade após alcançar o equilíbrio dinâmico.

Palavras chave: Protocolo de Transporte, Rede com alto produto largura de banda-atraso; Ineficiência do TCP; Controle de Congestionamento; Controle baseado em taxa.

ABSTRACT

The standard TCP inefficiency for high-speed networks was evident at the time these networks started being deployed. Along with this, various proposals have emerged to address this inefficiency.

Among these proposals, there is one which is a replacement of the congestion control mechanism based on window, the same mechanism used by TCP, to the rate based mechanism. In this mechanism the bandwidth is measured and then the packets are sent by pre-defined periods to better utilize the available resources of the network.

In the deployment of this mechanism, one uses the UDP as transport protocol and the congestion control is done at the application layer.

This paper presents a study of the performance of the UDT transport protocol implemented together with the homoestatic congestion control algorithm, HCC, for high speed networks with large bandwidth-delay product (BDP networks).

The results showed that HCC/UDT uses effectively the available bandwidth, presents good intra-protocol fairness and good TCP Friendliness index. It convergences fast as to the maximum throughput as to decrease the rate in case of intervenient traffic. It also presents good stability once reached the dynamic balance.

Keywords: Transport Protocol; BDP networks; TCP Inefficiency; Congestion Control; Rate Based Control.

1 INTRODUÇÃO

O aumento do volume de tráfego na Internet aliado aos avanços na implementação das redes, principalmente óticas, tem gerado um círculo virtuoso entre o aumento da largura de banda e a intensidade dos dados trafegados nestas redes, uma vez que esta disponibilidade de banda incentiva o desenvolvimento de novas aplicações distribuídas na rede que, por sua vez, requerem cada vez mais largura de banda.

Pesquisadores de diversas áreas têm utilizado as redes de alta velocidade para a colaboração internacional com a transferência massiva de dados e o compartilhamento de recursos computacionais e de redes. As redes GIGA(1), INTERNET2 (2), CANARIE (3), são alguns exemplos de redes de alta velocidade. Como algumas redes são distribuídas internacionalmente, o atraso de propagação se torna representativo quando as localidades estão separadas por grandes distâncias, por exemplo, por enlaces transcontinentais.

O aumento da velocidade aliado à distribuição geográfica realçou a subutilização das redes de alta velocidade que possuem um alto produto largura de banda-atraso (PLBA) devido à ineficiência do protocolo de transporte, o TCP (*Transport Control Protocol*) (4), o protocolo de transporte de *facto* da Internet (5). Essa ineficiência é derivada do controle de congestionamento utilizado pelo TCP, com aumento aditivo e diminuição multiplicativa, AIMD (*Additive Increase Multiplicative Decrease*) que é muito conservador para atingir rapidamente a capacidade máxima da rede e muito drástico quando se depara com um evento de perda (6).

A janela de congestionamento do TCP padrão, em segmentos, é aproximadamente de $1{,}2 / \sqrt{p}$, por RTT (*Round Trip Time* ou atraso de ida e volta) onde p é taxa de perda de pacotes no caminho(7). Isto é consequência da diminuição pela metade da janela de congestionamento em caso de evento de perda (de pacotes) e o aumento aditivo caso contrário. Esta função da resposta do TCP limita o valor máximo alcançável da janela de congestionamento.

Por exemplo, em uma conexão utilizando o TCP padrão com pacotes de 1500bytes e o atraso de ida e volta de 100ms, a janela de congestionamento média para alcançar a utilização máxima de um enlace de 10 Gbps, deveria ser de 83.333 segmentos e a perda de pacotes poderia ocorrer, no máximo, a cada 5.000.000.000 pacotes ou o equivalente a um evento de congestionamento a cada 1hora e 40 minutos. A taxa média de perdas de pacotes para atingir

a utilização máxima seria de 10^{-10} o que daria uma taxa de erro de bit em torno de 10^{-14} que é inviável para as tecnologias atualmente implantadas (8).

Diversos trabalhos foram publicados propondo alternativas ao TCP padrão para redes de alta velocidade, desde modificações no próprio TCP a mudanças drásticas na infraestrutura das redes existentes, passando pela utilização do UDP (*User Datagram Protocol*) (9) ao invés do TCP.

Esta dissertação faz um estudo sobre a utilização do controle de congestionamento homeostático, HCC (*Homeostatic Congestion Control*) (10), em um protocolo sobre o UDP como uma opção ao algoritmo de congestionamento do TCP para as redes com alto PLBA.

1.1 PROTOCOLOS DE TRANSPORTE

A camada de transporte, tanto no modelo OSI quanto no modelo TCP/IP, representados na Figura 1-1, é responsável pela transferência dos dados entre as máquinas de origem e destino, independentemente do tipo, topologia ou configuração das redes físicas existentes entre elas. Uma entidade dessa camada, na origem, comunica-se diretamente com outra entidade dessa mesma camada no destino. Por isso, ela é chamada fim a fim.

Os protocolos mais amplamente conhecidos dessa camada são, além do TCP, o UDP e o SCTP (*Stream Control Transmission Protocol*) (11) e o DCCP (12). O UDP é um protocolo sem conexão que não provê controle de fluxo, de erros ou retransmissão após a recepção de um datagrama incorreto. O SCTP é orientado a mensagem (O TCP é orientado a byte), preserva os limites das mensagens individuais e provê transmissões confiáveis das mensagens, como o TCP. O DCCP implementa datagramas não confiáveis, bidirecionais e com conexões *unicast* de congestionamento controlado.

Os protocolos da camada de transporte em geral têm quatro objetivos fundamentais que geralmente são transparentes para as camadas superiores: a eficiência, a equidade, a convergência e a distribuitividade, definidos a seguir.

O protocolo de transporte confiável deve ser eficiente na utilização da largura de banda disponível fim a fim. Para que isso ocorra, ele tem que investigar a largura de banda disponível e se recuperar para a taxa máxima de envio de dados rapidamente após uma queda, devido a perdas de pacotes em uma das máquinas ou de destino ou de origem ou, devido ao congestionamento da rede. E, depois de atingida a máxima taxa, o protocolo deve permanecer enviando nessa taxa até que ocorra alguma mudança na rede, ou seja, as oscilações devem ser tão pequenas quanto o possível.

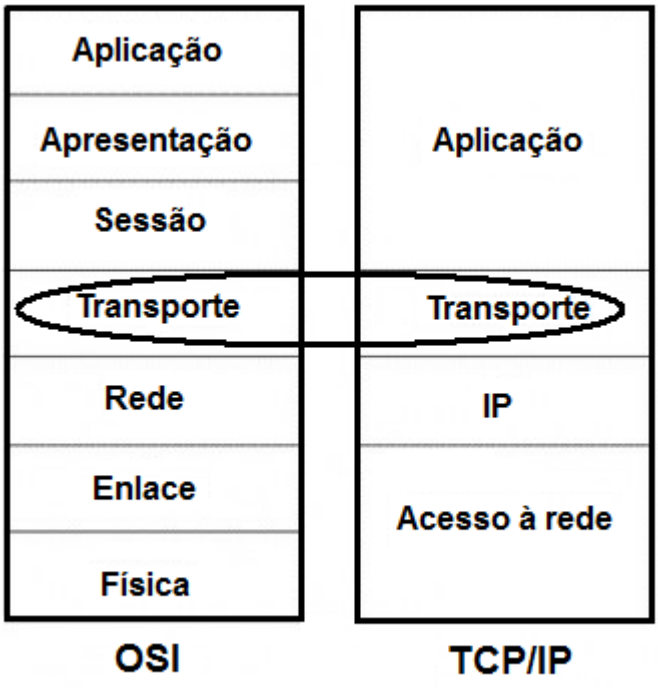

Figura 1-1 - Arquitetura OSI e TCP/IP

A equidade é a propriedade do protocolo de distribuir a largura de banda equitativamente entre todos os fluxos simultâneos. A imparcialidade entre fluxos do mesmo protocolo é chamada equidade intra-protocolo. Um caso especial da equidade é quando os fluxos têm RTT diferentes. Esse é um caso não contemplado pelo TCP, onde fluxos com RTT pequeno tendem a ocupar a maior parte da largura de banda disponível. Outro caso é a imparcialidade entre novos protocolos e o TCP, já que esse é o protocolo de transporte mais utilizado na Internet. Nesse caso, a imparcialidade é chamada de TCP *Friendliness*, ou quanto o novo protocolo é amigável ao TCP no sentido de quando existirem fluxos concorrentes do TCP e do outro protocolo, o TCP deve atingir o mesmo desempenho que teria se o outro fluxo também fosse TCP. Isto é, o fluxo do novo protocolo não deve ocupar largura de banda que antes era destinada ao TCP.

Em condições estáticas, um protocolo de transporte deve convergir para um ponto de equilíbrio (a máxima taxa mencionada antes) a partir de qualquer ponto de partida. Alguma oscilação pode ocorrer em torno desse ponto de equilíbrio uma vez que confirmações binárias são utilizadas para notificar as mudanças no estado da rede. Essa é uma propriedade dos protocolos de transporte confiáveis na Internet.

Por fim, os protocolos de transporte devem controlar suas taxas de envio de dados entre as máquinas, de forma fim a fim, ou seja, distribuída. O princípio fim a fim diz que, sempre que

possível, as operações de protocolos de transporte devem ocorrer somente entre os nós finais. O princípio de fim a fim aumenta a distribuidade do sistema.

1.2 CONTROLE DE CONGESTIONAMENTO

O controle de congestionamento é um componente crítico no protocolo de transporte confiável para a realização dos objetivos descritos acima. O protocolo de transporte utiliza um algoritmo de controle de congestionamento para ajustar a taxa de envio dos dados através de um sistema realimentado. Essa realimentação pode ser explicitamente gerada pelos nós intermediários, como os roteadores, ou pode ser estimada através de outros elementos como as perdas de pacotes, as tendências de aumento no atraso dos pacotes ou de eventos de estouro de temporizadores. A realimentação explícita dos roteadores traz uma informação mais precisa, porém, requer um processamento maior nos roteadores, além dos custos de implantação.

A taxa de envio dos dados pode ser ajustada através do intervalo de tempo entre os pacotes ou pelo número de pacotes pendentes a serem transmitidos. O primeiro método é conhecido como controle de congestionamento baseado em taxa e o segundo como controle de congestionamento baseado em janela.

O TCP utiliza o método de ajuste de envio de dados baseado em janela. Na primeira versão desse protocolo não havia controle de congestionamento (13) e os dados eram enviados em rajadas numa janela de controle de fluxo. Apesar de funcionar bem quando a rede estava levemente carregada, esse método provocava um círculo vicioso fazendo com que a vazão útil decrescesse à medida que a utilização da rede aumentava, pois quando havia perdas de pacotes, o protocolo retransmitia os pacotes perdidos, o que congestionava ainda mais os enlaces até que toda a capacidade da rede estivesse ocupada pelas retransmissões. O controle de congestionamento foi adicionado ao TCP por Jacobson (14) e desde então surgiram várias propostas de modificações e melhorias.

É importante salientar que o controle de fluxo é diferente de controle de congestionamento. Enquanto o primeiro tem como objetivo evitar a sobrecarga do receptor, o segundo tem como objetivo evitar a sobrecarga na rede que interconecta o transmissor ao receptor (15).

O controle de congestionamento homeostático, HCC, utiliza o método de controle de envio dos dados baseado em taxa. Nesse método a largura de banda disponível na rede é estimada, através do envio de pares de pacotes, e, de posse desse valor, os pacotes são

enviados em intervalos regulares para se aproveitar toda largura de banda disponível. Além disso, a informação de variação do atraso (*jitter*) é utilizada para determinar se a rede está ficando sobrecarregada. Por esse motivo, o HCC pode ser uma alternativa ao controle de congestionamento do TCP para aproveitar de forma mais eficaz a largura de banda de disponível em enlaces com alto PLBA.

1.3 OBJETIVOS DO TRABALHO

Os objetivos da presente dissertação são implementar um protocolo utilizando o algoritmo de controle de congestionamento homeostático e verificar se ele é uma alternativa ao TCP para redes com um alto PLBA, já que a ineficiência desse é devida, principalmente, ao seu algoritmo de controle de congestionamento AIMD.

Essa verificação foi realizada através de experimentos de uma implementação do protocolo UDT utilizando o algoritmo de controle de congestionamento HCC, numa rede real controlada, onde os atrasos e perdas de pacotes foram emulados. Foi observado o seu desempenho e comparado com o desempenho do algoritmo de congestionamento do TCP padrão, também implementado no arcabouço do protocolo UDT, sob as mesmas condições.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em mais cinco capítulos.

O Capítulo 2 apresenta alguns trabalhos relacionados com a proposição de alternativas à ineficiência do TCP padrão para os tipos redes com alto PLBA, apresentando desde modificações no próprio protocolo TCP a mudanças radicais na implementação da estrutura das redes.

O Capítulo 3 apresenta o conceito teórico do HCC, o algoritmo de controle de congestionamento homeostático, uma análise explicativa do seu funcionamento e sua utilização para redes de alta velocidade com alto PLBA.

O Capítulo 4 descreve a implementação realizada do protocolo utilizando algoritmo de congestionamento homeostático (HCC). Essa implementação foi realizada sobre o arcabouço UDT (*UDP Data Transfer)* (16) para a realização dos testes comparativos.

O Capítulo 5 descreve uma série de experimentos realizados para a avaliação do protocolo utilizando o HCC para redes de alta velocidade com alto PLBA.

Por fim, o Capítulo 6 apresenta as conclusões e sugestões para trabalhos futuros.

2 TRABALHOS RELACIONADOS

2.1 INTRODUÇÃO

A ineficiência do protocolo TCP em redes de alta velocidade motivou a pesquisa de alternativas capazes de usar toda a largura de banda disponível. Diversas propostas surgiram desde modificações no próprio protocolo TCP até a mudança na estrutura dos protocolos existentes com abordagens baseadas em informações dos roteadores sobre o estado da rede. Surgiram, também, propostas do controle de congestionamento na camada de aplicação utilizando o protocolo UDP.

Por ser o protocolo de transporte mais utilizado na Internet, o TCP foi utilizado como base para algumas sugestões de melhoria em seu desempenho nas redes com alto PLBA. Essas sugestões propuseram modificações no mecanismo de controle de congestionamento que é o principal responsável pela ineficiência do TCP padrão.

Essas modificações utilizam a indicação de congestionamento para acionar o mecanismo de controle. Esses mecanismos podem ser classificados em dois tipos: baseadas em perdas e baseadas em atraso de pacotes.

Os mecanismos que utilizam a perda de pacotes como indicação do congestionamento são considerados métodos reativos, uma vez que é necessário ocorrer o congestionamento (perda de pacotes) para o mesmo ser acionado. Já os mecanismos que se baseiam no atraso dos pacotes, tentam prever e evitar o congestionamento e, por isto, são considerados proativos.

Na Figura 2-1, podem-se observar essas variantes do TCP. Na primeira linha, pode-se citar o *High Speed TCP* (7), *Scalable TCP* (17), *H-TCP* (18), *BIC* (19) e o *CUBIC* (20). Cada um desses protocolos citados será apresentado com detalhes na seção seguinte.

Dentre as diversas propostas que surgiram, este trabalho voltou a atenção para algumas propostas que foram disponibilizadas, mesmo que como opcionais, no núcleo dos sistemas operacionais Linux e Windows.

Entre as variantes que utilizam o atraso como indicação principal de congestionamento pode-se citar o *FAST TCP* (21) que é um protocolo para redes de alta velocidade derivado do *TCP Vegas* (22). Já o *Compound TCP* (23) é uma sinergia entre os dois mecanismos. Da mesma forma, na seção seguinte haverá uma apresentação mais detalhada desses protocolos.

Ainda utilizando o TCP, houve outras abordagens que propuseram a utilização de fluxos paralelos concorrentes de TCP para aproveitar melhor a banda disponível, como *GridFTP* (24) e o MulTCP (25).

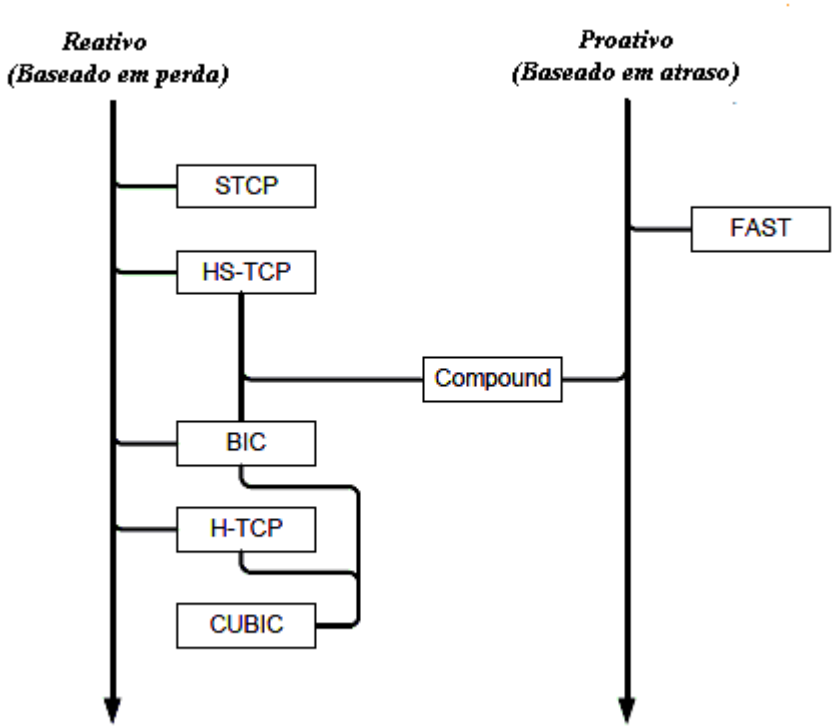

Figura 2-1 - Variantes do TCP para rede alta velocidade (5)

Em outras propostas, o controle de congestionamento é realizado na camada de aplicação e utilizam o UDP como protocolo de transporte ao invés do TCP. O UDT(16) é um exemplo dessa proposta.

Há também uma linha que propõe uma mudança radical na estrutura dos protocolos de transportes com a introdução de um novo protocolo de transporte envolvendo modificações nos roteadores de forma que os mesmos possam informar aos nós sobre as condições da rede. São exemplos desse tipo de proposta o *Explict Congestion Notification* (26) e e*Xplict Congestion Protocol* (27).

E, o próprio algoritmo de congestionamento HCC, através do protocolo RMTP (*Reliable Multiplexing Transport Protocol*) foi observado em redes de alta velocidade. Porém, o foco deste trabalho (28) foi as redes de alta velocidade sem considerar, no entanto, os altos valores de atraso que acarretam o alto PLBA. Outra diferença é que o desempenho do protocolo foi verificado apenas através de simulações no NS-2 (*The Network Simulator*). (29)

2.2 O CONTROLE DE CONGESTIONAMENTO PADRÃO DO TCP

Apesar do surgimento de diversas propostas para o controle de congestionamento do TCP, o padrão globalmente aceito e implementado, às vezes com algumas pequenas variações (30), é o definido pela RFC (*Request for Comments*) 5681 do IETF (*Internet Engineering Task Force)* (31) que descreve os quatros algoritmos: Partida Lenta (*Slow Start*), Prevenção de Congestionamento (*Congestion Avoidance*), Retransmissão Rápida (*Fast Restransmit*) e Recuperação Rápida (*Fast Recovery*).

Os algoritmos Partida Lenta e Prevenção de Congestionamento são utilizados pelo transmissor para controlar o quanto de dados é injetado na rede. Para implementá-los foram adicionados ao TCP as variáveis de estado da conexão: a janela de congestionamento do lado transmissor, w, e a janela de congestionamento do lado receptor.

No início da conexão, o algoritmo partida lenta é utilizado. Apesar do nome, esse mecanismo aumenta exponencialmente o tamanho da janela de congestionamento que é atualizada da seguinte maneira para cada segmento reconhecido (ack):

$$w \leftarrow w + 1 \tag{2.1}$$

Como a cada RTT são enviados w segmentos, se não houver perdas, w segmentos são reconhecidos. Logo, o algoritmo dobra o valor da janela a cada RTT.

Esse algoritmo é utilizado até que o valor da janela w atinja o valor de outra variável de estado da conexão, $ssthresh$. Essa variável é utilizada para determinar o limite de atuação dos algoritmos Partida Lenta e Prevenção de Congestionamento.

Quando o algoritmo Prevenção de Congestionamento entra em funcionamento, a janela de congestionamento aumenta da seguinte forma para cada segmento reconhecido:

$$w \leftarrow w + TS/w \tag{2.2}$$

Onde TS é o tamanho do segmento, ou seja, durante a fase de prevenção de congestionamento o incremento da janela é de somente um segmento por RTT, por isto o aumento aditivo no nome do algoritmo de congestionamento, AIMD (*Additive Increase Multiplicative Decrease*).

Caso haja perda, a janela diminui conforme segue:

$$w \leftarrow \big(1 - \beta(w)\big) \times w \tag{2.3}$$

Onde $\beta(w)$ é igual ½, ou seja, a janela diminui à metade.

A retransmissão rápida é utilizada da seguinte forma: quando o receptor detecta um pacote com número de sequência diferente de um valor esperado ele envia uma confirmação duplicada, pacote ACK (repete o mesmo pacote que enviou antes de receber esse último fora da sequência). Isso permite ao transmissor enviar o pacote faltante. Porém para evitar a retransmissão de um pacote que possa chegar atrasado, o transmissor espera receber três confirmações duplicadas e, assim retransmite o pacote que fora perdido na rede, antes que o temporizador expire.

A recuperação rápida permite ao transmissor evitar a fase de crescimento exponencial, não diminuindo a janela para um segmento e, recomeçando o crescimento linear a partir de um tamanho de janela igual à metade do tamanho da janela quando o terceiro ACK duplicado foi recebido. A ideia é que cada pacote de confirmação recebido sugere que o pacote que o originou deixou a rede uma vez que o receptor somente gerará pacotes ACK quando os segmentos chegarem até ele. Dessa forma, esses segmentos não estarão consumindo recursos na rede, permitindo ao transmissor continuar enviando segmentos, porém com uma janela menor, já que houve indicação de congestionamento.

A dinâmica da janela de congestionamento do TCP padrão é mostrada na Figura 2-2.

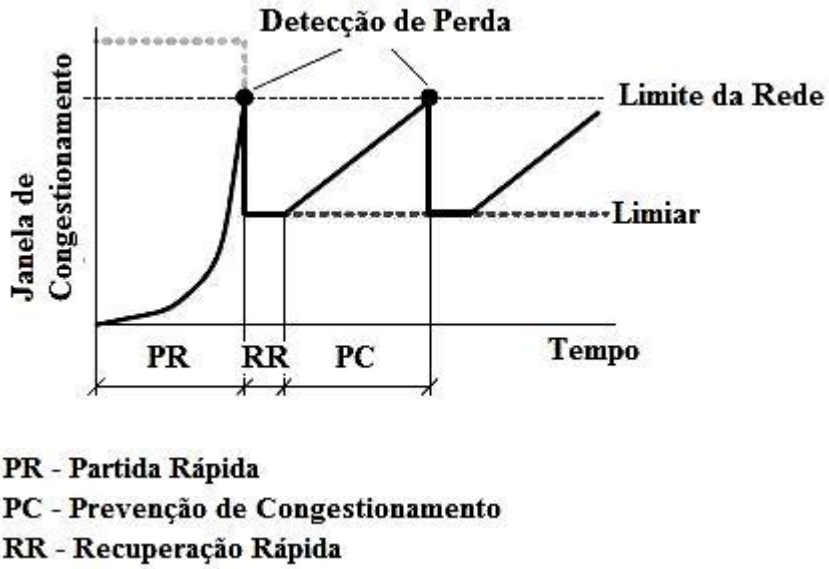

Figura 2-2 - Dinâmica da Janela de Congestionamento do TCP Padrão (5)

2.3 O PROBLEMA

No regime estacionário, a taxa de envio de pacotes do TCP padrão (T), medida em pacotes por RTT, *ppr*, é:

$$T = \sqrt{1,2} \Big/ \sqrt{p} \qquad (2.4)$$

Onde p é a taxa de perda de pacotes.

Para alcançar uma vazão média de B bps, uma conexão TCP padrão num cenário com R segundos de atraso e pacotes de D bytes, necessitaria que o período entre eventos de perdas ocorressem não menos do que a cada B*R/12*D RTTs.

Assumindo que os pacotes possuam jumbo frames de 9000 bytes e o RTT seja 100ms, a taxa de envio máxima por RTT será de (9000 x 8 bits x T) / 100ms. Para uma conexão TCP manter a taxa de envio de 10Gbps, seria necessário, utilizando o TCP padrão, que um evento de perda não ocorresse num intervalo menor que 15 minutos.

Para atingir taxas maiores, os usuários podem estabelecer múltiplas conexões simultâneas como faz, por exemplo, o *GridFTP* (24). Com isso ele conseguirá agregar N conexões virtuais TCP. Isso pode ser suficiente para um usuário ocasional numa rede bem preparada. Contudo o parâmetro N fica a cargo do usuário o que pode acarretar num desempenho mais agressivo quando houver um período de congestionamento de moderado a alto. Dessa forma essa não é uma boa solução para a rede como um todo (7).

2.4 MODIFICAÇÕES NO TCP

Para solucionar o problema descrito acima, algumas modificações no algoritmo de controle de congestionamento do TCP foram sugeridas. A seguir serão apresentados os protocolos *High Speed TCP* (7), *Scalable TCP* (17), *H-TCP* (18), *BIC* (19), *CUBIC* (20), *FAST TCP* (21) e o *Compound TCP* (23).

2.4.1 HIGH SPEED TCP (HS-TCP)

O *High Speed TCP* (HS-TCP) proposto por Floyd (7) tem por objetivo, dentre outros, prover eficiência na utilização da largura de banda disponível nas redes com alto PLBA sem depender de baixas taxas irreais de perda de pacotes. Outro objetivo é prover equidade com fluxos do TCP padrão em ambientes com alta taxa de perdas.

Para isso, substitui o coeficiente de crescimento e o fator de decrescimento do TCP padrão por funções do tamanho da janela de congestionamento. Com isso, ele consegue manter uma conexão com uma vazão de 10Gbps num cenário de atraso de 100ms e tamanho do pacote de 1500 bytes, conforme mostrado na Figura 2-3.

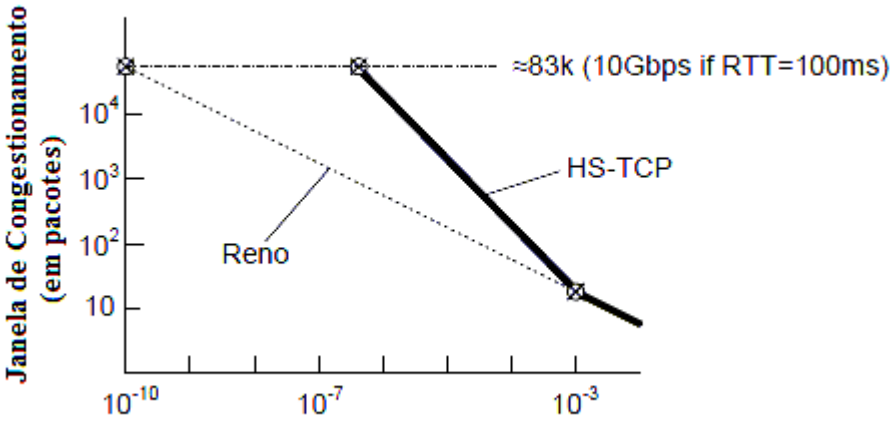

Figura 2-3 - Objetivo do HS-TCP (5)

No HS-TCP três novos parâmetros são especificados Janela_Pequena, Janela_Grande, e Perda_Alta. O parâmetro Janela_Pequena é utilizado para determinar o comportamento do protocolo. Se a janela de congestionamento for menor ou igual ao valor da Janela_Pequena, que foi definido em 38 segmentos, o comportamento do protocolo é o mesmo do TCP padrão, caso contrário o protocolo adota a função HS-TCP.

Os valores de Janela-Grande, tamanho máximo da janela de congestionamento, definido como 83.000 segmentos e Perda_Alta, taxa de perda de pacotes, definido com 10^{-7} limitam a função HSTCP.

O valor da janela de congestionamento no recebimento de uma confirmação será:

$$w \leftarrow w + \alpha(w)/w \tag{2.5}$$

E, quando houver um evento de congestionamento a janela será decrescida:

$$w \leftarrow \big(1 - \beta(w)\big) \times w \tag{2.6}$$

Quando o tamanho da janela de congestionamento for menor ou igual ao valor da variável Janela_Pequena, $\alpha(w)$ será igual a um e $\beta(w)$ igual a meio. Isso permite a compatibilidade com o TCP Padrão.

Quando janela de congestionamento tiver o valor igual ao da Janela_Grande (JG), a variável

Perda_Alta (*PA*) é especificada e a relação entre $\alpha(w), \beta(w)$ e as variáveis é a seguinte:

$$\alpha(w) = \frac{JG^2 * PA * 2 * \beta(w)}{2 - \beta(w)} \quad (2.7)$$

O parâmetro Diminuição_Grande, *DG*, é utilizado para especificar o $\beta(w)$

$$\beta(w) = DG \quad (2.8)$$

Quando o tamanho da janela de congestionamento é maior do que JanelaPequena e menor que JanelaGrande, tem-se:

$$\beta(w) = \frac{(\mathrm{DG} - 0{,}5) * (\log(w) - \log(JP))}{\log(JG) - \log(JP)} \quad (2.9)$$

E,

$$\alpha(w) = \frac{w^2 * p(w) * 2 * \beta(w)}{2 - \beta(w)} \quad (2.10)$$

Onde $p(w)$ é a taxa de perda de pacotes para a janela de congestionamento w dada por:

$$p(w) = \frac{0.078}{\sqrt{w}} \quad (2.11)$$

Essas funções permitem ao HS-TCP utilizar todo um enlace de 10Gbps com a taxa de perda de pacotes em torno de 10^{-7}. O TCP padrão necessita de uma taxa de perda de pacotes de 10^{-10} para atingir a ocupação máxima.

Durante os períodos de congestionamentos, quando a probabilidade de perdas de pacotes é alta, o HS-TCP se comporta como o TCP Reno.

Para as redes com alto PLBA, o HS-TCP não considera a equidade com os fluxos TCP Reno como um fator importante visto que esses não conseguem utilizar eficazmente os recursos da rede. Já a equidade intraprotocolo é importante e o HS-TCP, durante o período de Prevenção de Congestionamento, não altera o algoritmo AIMD e, a janela de congestionamento é acrescida de um tamanho constante de pacotes por RTT.

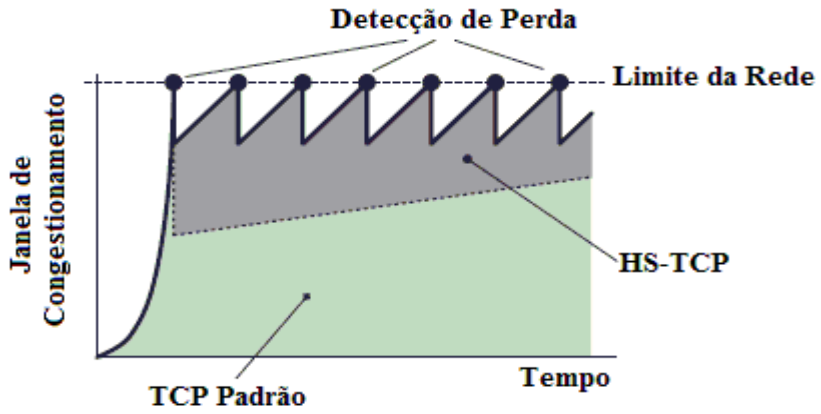

Figura 2-4 - A dinâmica da janela de congestionamento do HS-TCP (5)

A Figura 2-4 mostra a dinâmica da janela de congestionamento do HS-TCP no domínio do tempo e também a comparação com o TCP Padrão.

O HS-TCP foi apresentado em 2003 e está disponível a partir da versão do 2.6.13 do *kernel* do Linux.

2.4.2 SCALABLE TCP (S-TCP)

O *Scalable* TCP foi proposto como uma alternativa ao HS-TCP para o problema de eficiência na transferência de dados em redes com alto PLBA. Utiliza o algoritmo MIMD (*Multiplicative Increase* e *Multiplicative Decrease*) ao invés do AIMD. Durante o período de Prevenção de Congestionamento, o S-TCP aumenta a sua janela de congestionamento por uma fração do tamanho atual da janela por RTT.

$$w = w + \alpha(w) * w \tag{2.12}$$

Onde $\alpha(w) = 0{,}01$, ou seja, a janela cresce 1% a cada RTT.

E, quando ocorrer congestionamento, diminui a janela de um oitavo:

$$w = w\,(1 - \beta(w)) \tag{2.13}$$

Onde $\beta(w) = 0{,}125$

A função da janela de congestionamento do S-TCP é mostrada na Figura 2-5. É similar à HS-TCP, porém com uma frequência de aumentos e diminuição maior.

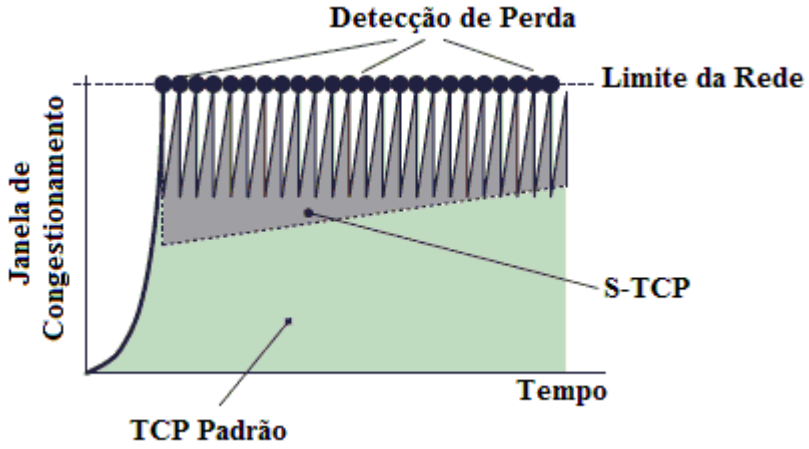

Figura 2-5 - Janela de Congestionamento do S-TCP (5)

Dessa forma, o S-TCP utiliza melhor a banda que o TCP Padrão e é escalável em diferentes ambientes por seguir funções exponenciais. No entanto, esse algoritmo de aumento multiplicativo e decréscimo multiplicativo não utiliza equitativamente a largura de banda disponível. Os fluxos com maior janela tendem a permanecer com maior janela uma vez que o MIMD preserva a proporção entre as janelas dos diversos fluxos.

O S-TCP foi apresentado em 2003 e está disponível a partir da versão do 2.6.13 do kernel do Linux.

2.4.3 HAMILTON TCP (H-TCP)

O Hamilton TCP (H-TCP) é mais uma alternativa para o controle de congestionamento do TCP, com o intuito de prover uma boa equidade entre fluxos interprotocolos (entre fluxos H-TCP e outros fluxos como o TCP), intraprotocolo (diversos fluxos H-TCP) e fluxos com diferentes valores de RTT.

Para isso, o H-TCP generaliza o algoritmo AIMD ajustando os fatores de aumento da taxa de envio de pacotes em função do tempo decorrido desde o último evento de congestionamento.

Seja Δ o tempo transcorrido desde o último evento de perda. O fator de aumento da taxa de, $\alpha(\Delta)$, é ajustado como segue:

$$\alpha(\Delta) = \begin{cases} 1 & para\ \Delta \leq \Delta L \\ 1 + 10 * (\Delta - \Delta L) + 0.25 * (\Delta - \Delta l)^2 & para\ \Delta > \Delta L \end{cases} \quad (2.14)$$

Onde ΔL é um parâmetro limiar de compatibilidade para troca entre o modo de operação TCP Padrão e o H-TCP.

Dessa forma, a janela de congestionamento é dada por,

$$w \leftarrow w * w/\alpha \tag{2.15}$$

A cada evento de perda:

$$\beta \leftarrow \frac{RTTmin}{RTTmax}, \ \beta \in [0.5, 0.8] \tag{2.16}$$

E, a janela de congestionamento,

$$w \leftarrow w * \beta \tag{2.17}$$

A simetria da rede e equidade entre os fluxos ocorre devido ao fato do ajuste ser baseado no tempo decorrido desde o último evento de perda, e, assim, não faz diferença se o fluxo está no modo de alta velocidade ou no TCP padrão em relação ao tamanho de janela.

No entanto, pode ser obervado que a função $\alpha(\Delta)$ conduz a certo grau de injustiça quanto se tem RTT diferentes. Se dois fluxos H-TCP com diferentes RTT competem entre si pela largura de banda disponível, o fluxo com menor RTT sempre ganhará a disputa.

O H-TCP foi apresentado em 2004 e está disponível a partir da versão do 2.6.13 do kernel do Linux.

2.4.4 BIC (BINARY INCREASE CONGESTION CONTROL)

O *Binary Increase Congestion Control (BIC)* estende o TCP Padrão com uma nova fase operacional chamada Convergência Rápida. Nessa fase, uma busca binária é utilizada para rapidamente se descobrir o valor ótimo para a janela de congestionamento.

Essa busca é ilustrada na Figura 2-6. Funciona da seguinte maneira: a janela de congestionamento é atualizada para o valor médio entre os valores das variáveis Janela Máxima (Wmax) e Janela Mínima (Wmin). Wmax é o valor da janela logo antes do algoritmo entrar na fase recuperação rápida e, inicialmente, é arbitrariamente um valor alto e Wmin é o valor logo após o algoritmo sair da fase recuperação rápida e, inicialmente, é configurado com 1 segmento. Se não houver perda, a variável Wmin é atualizada com valor da Janela de Congestionamento e, se houver, Wmax recebe esse valor.

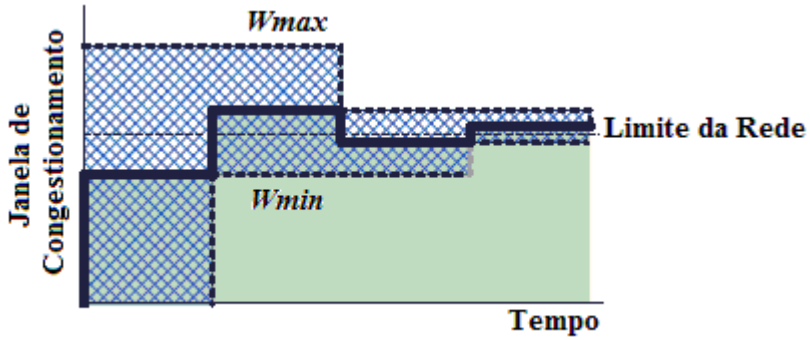

Figura 2-6 - Busca Binária do BIC (5)

Não havendo perda, o algoritmo repetidamente atualiza a janela com valor médio entre Wmax e Wmin até que a diferença entre Wmax e Wmin seja menor que o valor atribuído à variável de limiar Smin.

Essa busca binária permite que o algoritmo descubra rapidamente a largura de banda disponível, usando uma função logarítmica, crescendo mais agressivamente no início e diminuindo o crescimento à medida que a janela se aproxima do valor máximo disponível.

Para que o crescimento da janela no início da busca não seja demasiado grande para provocar impactos na rede, existe outra variável de limiar chamada de Smax que limita esse crescimento. Se a diferença entre o Wmax e Wmin é maior que Smax, a janela é acrescida de Smax, que é o limite máximo de crescimento da janela por RTT.

Quando uma perda é detectada, o valor da janela de congestionamento é atribuído a Wmax. Esse valor agora pode ser considerado acima do limite da rede por ter havido congestionamento e o algoritmo entra no algoritmo de recuperação rápida.

A função da janela de congestionamento do BIC é mostrada na Figura 2-7. Xu et al (19) mostraram, teoricamente, que o BIC não é menos injusto que o TCP padrão em situações onde há perdas sincronizadas. A razão entre o tamanho da janela de congestionamento de dois fluxos com RTT diferentes muda de $e^{\left(\frac{1}{RTT_1}-\frac{1}{RTT2}\right)*t*ln2}$ para tamanhos de janela pequenos para $RTT1/_{RTT2}$ para valores de janela grandes. Porém alguns experimentos posteriores (20) e (32) mostraram que em algumas situações o BIC pode ter uma equidade entre diferentes RTTs e entre fluxos BIC e de outros algoritmos de controle de congestionamento pior que o TCP padrão.

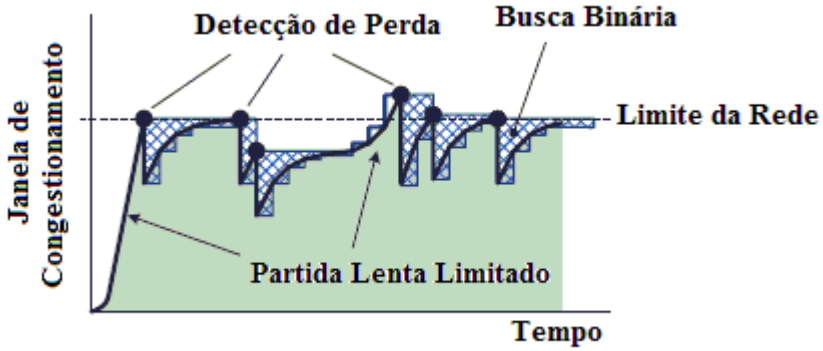

Figura 2-7 – Dinâmica do BIC (5)

O BIC foi disponibilizado em 2004 e está disponível a partir da versão do 2.6.12 do kernel do Linux. Foi o protocolo padrão do Linux até a versão 2.6.16.

2.4.5 CUBIC

Para solucionar o problema da equidade entre diferente RTTs, Rhee at al (20) propuseram o CUBIC, uma melhoria em relação ao BIC, no qual o tamanho da janela de congestionamento, W, é uma função do tempo decorrido desse o último evento de perda, t.

$$W(t) = C * (t - K)^3 + Wmax \quad (2.18)$$

Onde C é uma constante predefinida, $Wmax$ é o tamanho da janela imediatamente antes do último evento de perda detectado e K é o tempo que a função acima necessita para crescer de W para $Wmax$. Quando não ocorrer evento de perda. K é calculado da seguinte forma:

$$K = \sqrt[3]{\beta * Wmax/C} \quad (2.19)$$

A função permite um crescimento rápido do tamanho da janela quando W possui um valor bem menor que o tamanho máximo estimado, $Wmax$, e, se torna mais conservativo à medida que W se aproxima de $Wmax$.

Ao receber uma confirmação durante o período de Prevenção de Congestionamento, o CUBIC calcula a taxa de crescimento da janela durante o próximo período de RTT usando a Equação.(2.19). Ele define W (t + RTT) como valor alvo da janela de congestionamento. Dependendo do tamanho da janela atual, CUBIC funciona em três modos diferentes. Primeiro, se a janela atual é menor que o tamanho da janela que o TCP padrão alcançaria no tempo t depois do último evento de perda, o CUBIC está no modo TCP. Caso contrário, se a janela é menor que Wmax, CUBIC está na região côncava da função (região 1 e segunda parte

da região 3 da Figura 2-8). E, se a janela é maior que Wmax, CUBIC está na região convexa (região 2 e primeira parte da região 3).

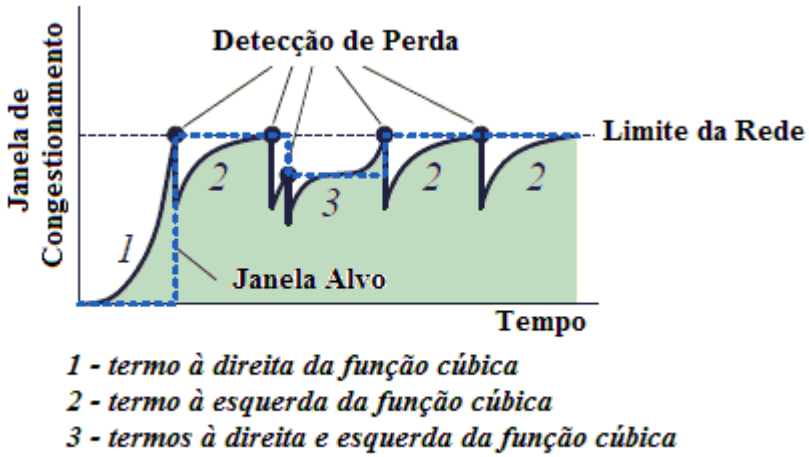

Figura 2-8 - Dinâmica da Janela de Congestionamento do CUBIC (5)

O CUBIC foi incorporado ao Linux na versão do 2.6.15 do kernel e, ele próprio estava, de acordo com os autores, na versão 2.0. A partir da versão 2.6.18 do kernel ele passou a ser o algoritmo de congestionamento padrão. Depois algumas modificações foram realizadas tanto pelos desenvolvedores do Linux (para aperfeiçoar o método de cálculo) quanto pelos autores. Na versão 2.1 do CUBIC o valor máximo de incremento da janela por RTT de 32 pacotes foi removido na fase côncava da função. Esse valor foi herdado do Smax do BIC, mas a sua remoção melhorou a escalabilidade do protocolo e não ocasionou problema adverso (Linux 2.6.22). Houve melhoria na fase partida lenta e de segurança. E, na versão 2.2 do CUBIC a amarração do valor máximo de incremento também foi removida na fase convexa. Assim, houve melhora na velocidade de convergência sem prejuízo para equidade intraprotocolo e com o TCP padrão.

O CUBIC foi apresentado em 2008 e está disponível a partir da versão do 2.6.16 do kernel do Linux tornando-se o protocolo padrão até as versões atuais.

2.4.6 FAST

Diferentemente dos protocolos anteriores, o FAST TCP utiliza a medida de atraso como indicação primária do congestionamento baseando-se no princípio do Vegas (22). O mecanismo de controle de congestionamento é divido em quatro componentes funcionalmente independentes:

- Controle de dados – determina quais pacotes serão transmitidos.
- Controle da janela – determina quantos pacotes serão transmitidos.
- Controle de rajada – determina quando os pacotes serão transmitidos.
- Estimativa – calcula o atraso de enfileiramento e as perdas de pacotes.

Quando a confirmação é positiva, o algoritmo calcula o atraso que será utilizado para computar os valores de atraso mínimo e médio utilizados.

A janela de congestionamento é atualizada da seguinte forma:

$$w \leftarrow min\left\{2w, (1-\gamma) * w + \gamma * (\frac{RTTmin}{RTT} * w + \alpha)\right\} \qquad (2.20)$$

Onde $\gamma \in (0,1)$, $RTTmin$ é o RTT mínimo observado e α é um parâmetro do protocolo que determina o número total de pacotes enfileirados nos roteadores ao longo do caminho no estado estacionário. O período de atualização da janela é fixo em 20ms no protótipo desenvolvido pelos autores.

De acordo com a equação (2.20), se estiver ocorrendo congestionamento na rede o RTT será maior que $RTTmin$ e o algoritmo irá diminuir a janela de congestionamento na proporção de $RTTmin/RTT$. Caso contrário, a janela irá crescer com base unicamente no parâmetro α.

A escolha do parâmetro α determina o compromisso entre estabilidade e escalabilidade. Se α for muito grande o protocolo será escalável em qualquer rede com alto PLBA, porém terá problemas com a convergência.

Embora experimentos tanto de simulações quanto implementações em redes reais tenham mostrado valores notáveis de equidade intraprotocolo e com diferentes valores de RTT, estabilidade e escalabilidade, novas pesquisas (34) reconheceram alguns problemas na concepção do FAST. Primeiramente, a falta de exatidão nas medidas de RTT herdada do Vegas, uma vez que o protocolo depende dos valores reais de $RTTmin$ e esse é um valor difícil de calcular, principalmente em redes onde rotas tendem a ser dinâmicas. Depois, o RTT pode não ser um bom substituto para o atraso de enfileiramento especialmente quando há congestionamento no caminho reverso ou há mudanças de rota. E, por fim, a equidade com o TCP (*TCP Friendliness*) não é boa mesmo para redes com baixo valor de largura de banda-atraso.

2.4.7 C-TCP (*COMPOUND* TCP)

Tan et al (23) depois de observar que os métodos de controle de congestionamento baseados em atraso tinham a característica de ajustarem a sua agressividade de acordo com a utilização do enlace, porém não eram competitivos com os métodos baseados em perdas, propuseram o *Compound* TCP que é a combinação de ambos os métodos.

Essa sinergia é implementada adicionando um novo componente escalável baseado em atraso no algoritmo de Prevenção de Congestionamento do TCP padrão. Esse novo componente, dw, faz com que a janela de congestionamento cresça rapidamente quando a rede está subutilizada e diminui esse crescimento gradualmente à medida que rede começa a ficar sobrecarregada. A janela de congestionamento será a soma da janela atualizada:

$$w \leftarrow (cw + dw) \tag{2.21}$$

Onde cw é janela de congestionamento convencional do TCP padrão e dw é janela baseada em atraso.A atualização de cw se dá através do mesmo mecanismo de Prevenção de Congestionamento do TCP.

O C-TCP mantém no início da conexão o mesmo comportamento do mecanismo de Partida Lenta do TCP padrão. Isso porque, o incremento exponencial da janela é rápido o suficiente mesmo para redes com alto PLBA visadas pelos autores. Nessa fase, dw é colocado em zero e essa componente da janela baseada em atraso somente será efetiva na fase de Prevenção do Congestionamento.

O algoritmo do componente baseado em atraso é derivado do TCP Vegas. A variável $RTTmin$ foi mantida e atualizada com o menor valor medido do RTT. A outra variável, $RTTatual$, é constantemente atualizada. Assim, o número de pacotes acumulados na conexão pode ser estimado. A vazão estimada será $cw/RTTmin$, a vazão real instantânea será $cw/RTTatual$. A diferença entre as vazões estimada e real multiplicada pelo valor do $RTTmin$, *diff*, infere a quantidade de pacotes que foram injetados e enfileirados na rede. Um limiar, γ, é definido para indicar congestionamento. Se o $\gamma > diff$, a rede está subutilizada. Caso contrário, a rede é considerada carregada e o componente baseado em atraso suavemente diminui o tamanho da janela.

É atualmente o algoritmo de controle de congestionamento mais implantado nas redes (5), sendo o algoritmo padrão para os sistemas Windows 2008 Server e disponível para sistemas Vista e Windows 7. Além disso, há remendos (*patches*) para implementá-lo nos sistemas Windows 2003 Server e XP.

2.5 APLICAÇÕES QUE UTILIZAM O UDP

Conforme mencionado, para resolver a ineficiência do TCP para redes de alta velocidade houve abordagens que propuseram a utilização do UDP ao invés de TCP, como o *FOBS* (34), *SABUL* (35), *Reliable Blast UDP* (36), *Hurricane* (37) *e FRTP* (38). Todos esses protocolos foram projetados para redes privadas ou com qualidade de serviço (QoS) habilitada. Não possuem algoritmo de controle de congestionamento ou têm algoritmo apenas com a finalidade de prover rapidamente alta utilização da largura de banda disponível.

Uma melhoria em relação a estas abordagens foi o UDT, que é um protocolo com algoritmo de congestionamento e que foi a sequencia do trabalho do SABUL(16).

O UDT foi desenvolvido para utilizar eficazmente as redes óticas de alto desempenho desenvolvidas e as que ainda estão em desenvolvimento. Construído sobre UDP com controle de congestionamento e confiabilidade, permite utilizar de forma eficaz grande parte da largura de banda disponível nos enlaces. Além disto, um conjunto de interfaces de programação de aplicativos (*API*) foi desenvolvido para suportar facilmente a implementação de aplicações, incluindo fluxo de dados e mensagens confiáveis. A biblioteca original também foi estendida para suportar a fácil implementação de outros algoritmos de controle de congestionamento. Esta facilidade foi utilizada para os testes desde trabalho, por isto, o UDT será estudado mais profundamente no capítulo três.

2.6 MUDANÇAS NA ESTRUTURA DA REDE

Finalmente, uma proposta mais radical surgiu: adicionar nos roteadores a realimentação explícita das informações da rede. Esta foi a proposta do *XCP (eXplicit Control Protocol*) que é a generalização da proposta do *ECN (Explicit Congestion Notification*), onde os pacotes são atualizados com as informações de incremento ou decremento calculados pelo roteadores, e, essas informações, após chegarem ao receptor dos pacotes, são enviados ao transmissor via pacotes de reconhecimento.

As informações da janela de congestionamento e RTT são enviadas pelo transmissor no cabeçalho de cada pacote. Os roteadores monitoram o tráfego de entrada para cada uma das filas de saída. Baseado na diferença entre a largura de banda do enlace e taxa do tráfego de entrada, o roteador informa aos fluxos que estão dividindo o enlace para aumentar ou diminuir suas janelas de congestionamento. Ele faz isto atualizando o cabeçalho de congestionamento

de cada pacote de dados. Um roteador mais congestionado à frente no caminho pode reduzir ainda mais o valor da janela de congestionamento. Assim, o pacote irá conter a informação do enlace de gargalo no caminho. Estas informações ao chegarem ao receptor são enviadas ao transmissor por um pacote de reconhecimento. Desta forma o transmissor pode atualizar sua janela de congestionamento.

No XCP os controladores de eficiência e equidade são desassociados. O controlador de eficiência utiliza o MIMD para ajustar a taxa de dados de acordo com a largura de banda disponível no enlace de menor capacidade do caminho. Enquanto que o controlador de equidade utiliza o algoritmo AIMD para distribuir equitativamente a largura de banda entre todos os fluxos concorrentes.

Apesar de demonstrar bom desempenho, o XCP sofre com problemas de implantação, pois é necessária a modificação nos roteadores e nos sistemas operacionais das máquinas finais. Além disto, Braden et al (39) demonstrou que a implantação gradual do XCP provoca um queda significativa no seu desempenho.

2.7 RESUMO

Esse capítulo descreveu algumas propostas de solução para a ineficiência do protocolo TCP padrão para redes com alto PLBA.

Na Além das propostas apresentadas na tabela, foi apresentada também a proposta de utilização do UDP ao invés do TCP como protocolo de transporte e o controle de congestionamento sendo realizado na camada de aplicação.

Outra linha de proposta apresentada, foi a modificação da estrutura de rede com a implementação do Explicit Congestion Notification e sua generalização com o eXplit Congestion Protocolo (XCP) que utiliza os nós intermediários (roteadores) para melhorar a qualidade da informação do estado da rede.

Tabela 2-1 pode ser visto um sumário destas soluções baseadas no TCP.

Além das propostas apresentadas na tabela, foi apresentada também a proposta de utilização do UDP ao invés do TCP como protocolo de transporte e o controle de congestionamento sendo realizado na camada de aplicação.

Outra linha de proposta apresentada, foi a modificação da estrutura de rede com a implementação do Explicit Congestion Notification e sua generalização com o eXplit Congestion Protocolo (XCP) que utiliza os nós intermediários (roteadores) para melhorar a qualidade da informação do estado da rede.

Tabela 2-1 – Variantes do TCP (5)

Variantes do TCP	Ano	Base	Características	Estado	Implementação
HS-TCP	2003	New Reno	Fases de aumento aditivo e fatores de decrecimento multiplicativo como funções do tamanho da janela de congestionamento; Partida Lenta limitada.	Experimental	Linux >= 2.6.13
STCP	2003	New Reno	Política de prevenção de congestionamento com aumentos multiplicativos e decrescimentos multiplicativos	Experimental	Linux >= 2.6.13
H-TCP	2004	New Reno	Fases de aumento aditivo como uma função do tempo decorrido desde a última detecção de perda de pacote; Fase de aumento escalável para um RTT de referência; Adaptação do coeficiente de decresimento multiplicativo.	Experimental	Linux >= 2.6.13
BIC-TCP	2004	HS-TCP	Busca binária da janela de congestionamento; Partida Lenta limitada.	Experimental; Padrão do Linux até 2.6.15	Linux >= 2.6.12
CUBIC	2008	BIC-TCP	O controle da janela de congestionamento como uma função do tempo decorrido desde o último evento de congestionamento	Experimental; Padrão do Linux.	Linux >= 2.6.16
Compound TCP	2005	HS-TCP e Vegas	Dois componentes no cálculo da janela de congestionamento (lento e escalável)	Experimental; Padrão do Windows Server 08, Disponível Vista, Seven, XP* e Server 2003*.	Windows Server 08, Disponível Vista, Seven, XP* e Server 2003*.
FAST	2003	Vegas	Atualização da janela de congestionamento a taxa constante baseada em equação	Experimental	

3 CONTROLE DE CONGESTIONAMENTO HOMEOSTÁTICO, HCC

3.1 INTRODUÇÃO

Este capítulo descreve o controle de congestionamento homeostático, HCC (*Homeostatic Congestion Control*), desenvolvido por Magalhães (40) para redes sem fios que por ser baseado em taxa pode ser uma alternativa ao TCP para redes de alta velocidade com alto PLBA. O nome homeostático vem do propósito de alcançar a homeostase (ou homeostasia), que é a propriedade de um sistema de regular o seu ambiente interno para manter uma condição estável, mediante múltiplos ajustes de equilíbrio dinâmico, controlados por mecanismos de regulação interrelacionados. No HCC, esses mecanismos são o par de pacotes (*packet-pair*) e o monitoramento da variação do atraso (*jitter*). Enquanto o par de pacotes tende a superestimar a largura de banda disponível, aumentando a taxa de envio de pacotes, o monitoramento da variação do atraso (*jitter*) é usado para corrigir essa estimativa, diminuindo a taxa.

O principal desafio de um controle de congestionamento é detectá-lo. E, para isso, a perda de pacotes pode ser utilizada como uma indicação. Dessa forma, quando um pacote é perdido, o receptor pode inferir que está ocorrendo congestionamento e informar essa situação ao transmissor para que ele diminua a taxa de envio de pacotes a fim de parar com a sobrecarga. Porém, esse processo demanda um intervalo de tempo entre a ocorrência do congestionamento e a sua prevenção, pois o receptor tem que perceber a falha, o que envolve em receber um ou mais pacotes depois de ocorrida a perda ou expiração de um temporizador, e, depois disso, informar ao transmissor.

Outra forma do transmissor controlar o congestionamento é ter ciência da largura de banda disponível até o receptor. Porém, sem reserva prévia é quase impossível saber de antemão quanta largura de banda está disponível porque esse valor varia de forma imprevisível.

Uma opção são os mecanismos baseados em taxa. Nesse caso, os pacotes são transmitidos em intervalos regulares o que impede as rajadas inerentes ao modelo baseado em janela (iniciadas pelo recebimento da confirmação), limitando a quantidade da variação do tempo entre envios (jitter na origem) na transmissão desses pacotes.

3.2 MECANISMOS BASEADOS EM TAXA

Nos mecanismos de transmissão baseados em taxa, os pacotes são enviados a uma taxa determinada, ou de outra maneira, são enviados com um intervalo de tempo pré-estabelecido entre eles. Desse modo, geram uma correspondência melhor entre o intervalo de chegada dos pacotes no receptor e as condições da rede, permitindo uma melhor estimativa da largura de banda disponível. Isso diferencia esses dos mecanismos de transmissão baseados em janela, onde o tempo de transmissão dos pacotes é ditado pelo recebimento no transmissor de uma confirmação (pacote ACK) oriunda do receptor, o que torna esses mecanismos inerentemente de rajadas, à medida que vários pacotes podem ser enviados quando do recebimento de um ACK pelo transmissor.

Não havendo tráfego concorrente, o intervalo de tempo entre dois pacotes consecutivos permanecerá constante ao longo da rede se a taxa de transmissão do fluxo estiver abaixo da capacidade da rede, excluindo-se nesse caso os efeitos da fronteira, quando há mudança na taxa – nesse caso o intervalo de tempo entre dois pacotes consecutivos irá mudar. Isso ocorre porque se a taxa está abaixo da capacidade da rede, não há enfileiramento em nenhum dos roteadores ao longo do caminho e os pacotes subsequentes ao primeiro encontrarão as mesmas condições em cada um dos roteadores e, sendo assim, o tempo de trânsito total do transmissor ao receptor será o mesmo para todos os pacotes. Por outro lado, se a taxa estiver acima da capacidade da rede, filas se formarão no roteador que precede o enlace de menor capacidade e, portanto, a taxa observada no receptor será a taxa de serviço desse enlace, também denominado enlace de gargalo.

Logo, no caso de não haver tráfego concorrente, pode-se controlar a taxa de envio dos pacotes informando ao transmissor a taxa observada no receptor e ajustando-a, no transmissor, no caso dela ter sido superestimada em relação à capacidade do enlace de gargalo. Se a taxa for subestimada nenhuma alteração será percebida no receptor. Por isso, uma forma para encontrar a capacidade real do enlace de gargalo é enviar os pacotes na maior taxa que o transmissor conseguir transmitir e medir o intervalo de tempo de chegada entre os pacotes. Esse tempo é a dispersão causada pelo enlace de gargalo. Se os pacotes forem enviados a essa taxa nenhum atraso adicional será causado no enlace de gargalo porque o espaçamento dos pacotes corresponde à taxa de serviço desse enlace.

O método do par de pacotes (41) é baseado na ideia exposta acima. Quando um pacote é transmitido por um enlace, ele enfrenta um atraso de transmissão relacionado ao relógio do

hardware do transmissor. Em um enlace de capacidade C, o atraso de transmissão de um pacote de tamanho L é $\Delta t = L/C$. A medida do par de pacotes consiste de dois pacotes com o mesmo tamanho L enviados em sequência do transmissor para o receptor. Sem tráfego interveniente o par de pacotes alcançará o receptor com uma dispersão δt (por exemplo, o espaço de tempo do último bit do primeiro pacote para o último bit do segundo pacote) igual a L/C. Esse é o atraso de transmissão no enlace gargalo no caminho. O receptor pode estimar a capacidade desse enlace a partir da dispersão da seguinte maneira, $C=L/\delta t$, conforme mostrado na Figura 3-1.

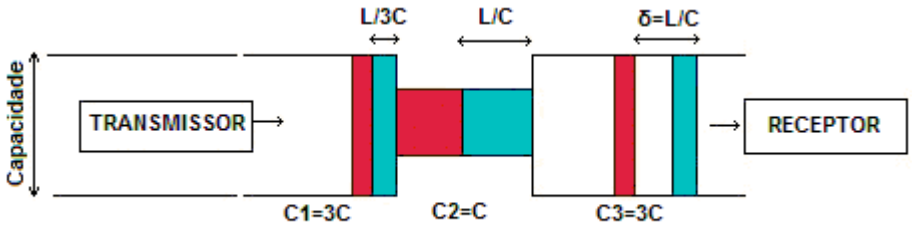

Figura 3-1 – Método do par de pacotes – Dois pacotes em sequência saem do transmissor e chegam ao receptor com uma dispersão δ determinada pelo enlace de gargalo (42).

Sem tráfego concorrente, a política de enfileiramento nos roteadores não é importante uma vez que não terá efeito sobre a dinâmica do par de pacotes. No entanto, se houver tráfego concorrente, a política de enfileiramento torna-se muito importante, por exemplo, se o enfileiramento justo (*fair queueing*) for utilizado, cada fluxo conseguirá a mesma parcela de banda. Nesse caso, a política de enfileiramento isolará cada fluxo do efeito de rajada de outros tráfegos. O par de pacotes, então, medirá a parcela de banda de cada fluxo. Por outro lado, se a política utilizada for FIFO (*First In First Out* – o primeiro a entrar é o primeiro a sair), a política mais comum na maioria dos roteadores da Internet, acontecerá um comportamento mais complexo. Dois efeitos ocorrerão, a compressão de tempo, quando há fila no roteador e o primeiro pacote atrasa; e, a expansão de tempo, quando um ou mais pacotes se situam entre o primeiro e o segundo pacote do par de pacotes, como pode ser visualizada na Figura 3-2. Os efeitos podem afetar o mesmo par de pacotes em diferentes rotadores ao longo do caminho, o que gerará uma dispersão grande nos valores medidos do período de tempo entre as chegadas dos pacotes no receptor.

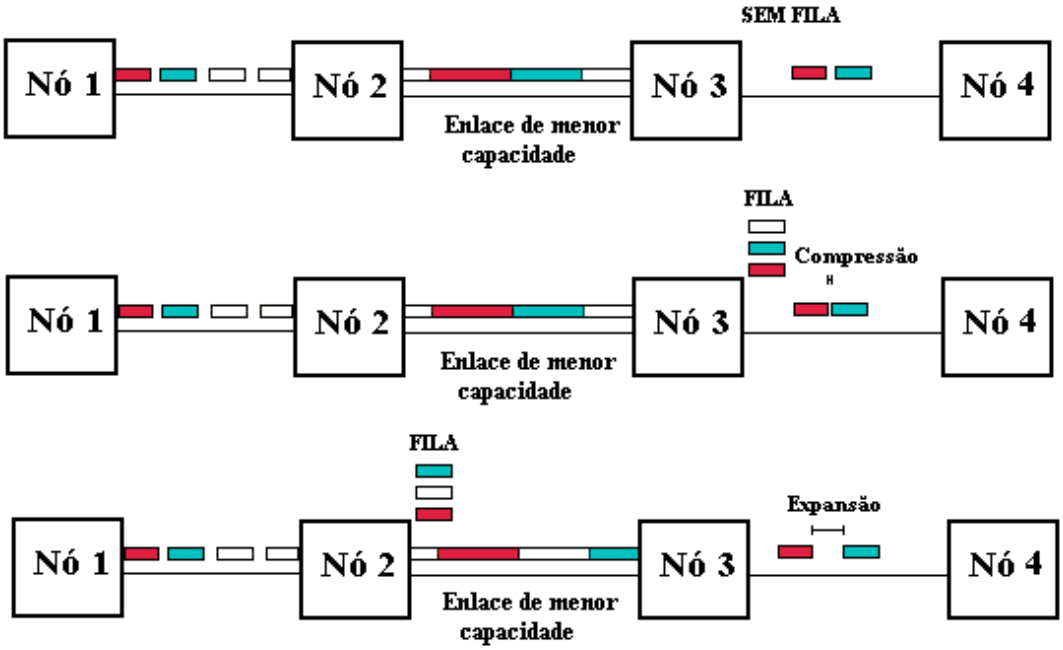

Figura 3-2 – Compressão e Expansão do tempo entre os pacotes consecutivos (28) A segunda linha representa a compressão do tempo e a terceira a expansão..

3.3 VARIAÇÃO DO ATRASO – JITTER

Para medir a largura de banda disponível de maneira mais exata, duas variantes do algoritmo de dispersão de pacotes são utilizadas: o primeiro é o método do par de pacotes citado acima, onde dois pacotes são enviados fim a fim. E, o segundo é a medição regular da variação entre o tempo de chegada dos pacotes (*jitter*). Em ambos os casos, a dispersão dos pacotes é medida. A primeira técnica tem sido utilizada extensivamente para medir a capacidade mínima do enlace do caminho fim a fim (e não parte da largura de banda), mas requer experimentos com muitas repetições e análises estatísticas (43) e (44). A segunda técnica tem sido também utilizada para capacidade, com requerimentos similares (45).

Os tempos de compressão e de expansão dos pacotes devem ser filtrados, ao se usar par de pacotes para medir o enlace de menor capacidade. Como mostrado por (42), a distribuição dos tempos de chegada entre pacotes é multimodal e o modo principal não necessariamente corresponde à banda do enlace de menor capacidade.

Se o tempo de trânsito de todos os pacotes for o mesmo, eles podem manter entre eles o mesmo intervalo de tempo em que foram transmitidos. Porém, devido à presença de tráfego interveniente mesmo os pacotes sendo enviados a uma taxa menor que a taxa de serviço do

caminho, terão tempos de trânsito diferentes (pacotes enviados acima da taxa máxima de serviço estarão, certamente, atrasados e, se essa taxa for mantida por um período, é provável que alguns deles serão perdidos). Se um pacote está atrasado em relação ao pacote anterior, o receptor computará um *jitter* positivo. Se o *jitter* é calculado observando a diferença entre o tempo de chegada entre dois pacotes consecutivos e, também, a diferença de tempo quando eles foram transmitidos, *jitters* negativos também ocorrerão. Se o primeiro pacote está atrasado e o segundo não sofre qualquer atraso, eles chegarão ao receptor com uma diferença de tempo menor do que a diferença com que eles foram transmitidos. Na Figura 3-3 é possível visualizar como os *jitters* positivos e negativos são gerados.

É esperada a alternância de *jitters* positivos e negativos. Quando uma série de *jitters* positivos é percebida, é indício de que a taxa de serviço da rede foi violada. O *jitter* médio desta série é uma boa aproximação da diferença entre o período de transmissão ideal e o valor atual do período. Quando se usa o controle homeostático, dois *jitters* positivos consecutivos são tomados como uma indicação da violação da taxa e o período de transmissão é corrigido para um valor que resulta numa taxa menor que a taxa de serviço da rede, dada as condições observadas pelos dois pacotes.

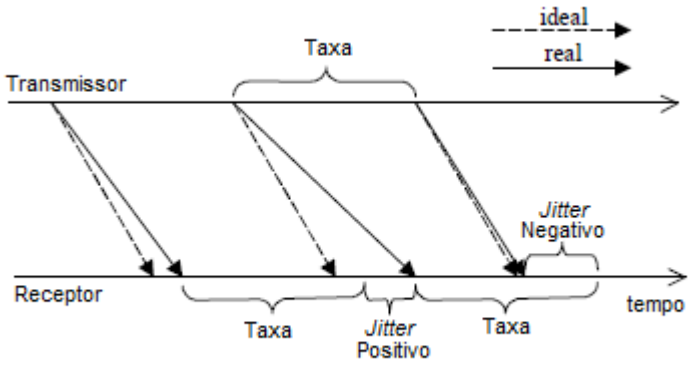

Figura 3-3 – Medida de *Jitter* Positivos e Negativos (10)

A monitoração do *jitter* previne que o período de transmissão se desvie do ponto operacional da rede. Infelizmente, isso não é suficiente para garantir que a rede irá operar sem perdas de pacotes. O mecanismo de correção deve decrementar a taxa para um ponto abaixo do ponto operacional, permitindo que qualquer fila que tenha se formado, reduza-se. Nesse ponto é importante evitar a sincronização, adicionando alguma aleatoriedade. Se diferentes fluxos baseados em taxa se tornam sincronizados, eles aumentarão e diminuirão a taxa conjuntamente. Se todos os fluxos aumentarem a utilização da rede e violarem a taxa de

serviço ao mesmo tempo, eles poderão experimentar perdas simultâneas. Se essas perdas são acopladas com um mecanismo de *backoff*, então todos os fluxos irão decrementar suas taxas ao mesmo tempo, sendo observado um grande desperdício de banda (46). Variando a quantidade de decréscimo da taxa usando uma função randômica, os diferentes fluxos não ficarão sincronizados, ao custo de nunca se atingir a banda total do enlace.

3.4 AJUSTES NO MECANISMO DE CONTROLE DO CONGESTIONAMENTO

Três parâmetros podem ser alterados para mudar o comportamento dos protocolos que utilizam o mecanismo proposto acima. O primeiro é o parâmetro usado na média movente utilizado no cálculo da taxa após o recebimento de um par de pacotes. Esse parâmetro regula quanto da taxa anterior é mantida no cálculo da nova taxa, e pode ser ajustado de maneira que o protocolo se adapte mais lenta ou rapidamente às condições atuais da rede. Apesar de parecer óbvio que a melhor escolha será a adaptação rápida, é possível que ela provoque a ocorrência de oscilações, uma vez que cada ajuste de taxa tem influência em todos os tráfegos no caminho e se o valor ultrapassar o ponto operacional desejado, o próximo ajuste deve tentar corrigir isso.

O segundo parâmetro é o mecanismo de monitoração de *jitter*. Se a taxa ultrapassar o ponto operacional, *jitters* positivos serão computados e uma adaptação na taxa será realizada, pois, idealmente, a taxa deve ser diminuída para o ponto operacional. Os valores de *jitter* são uma indicação de quanto erro contém a taxa atual, e um valor proporcional ao *jitter* médio é usado para corrigir a taxa. Os valores que podem ser alterados nesse caso são: o limiar de *jitter* positivos consecutivos e quanto do valor do *jitter* é utilizado para corrigir a taxa. Então, o protocolo recua mais lenta ou rapidamente nos casos de erros no calculo da taxa de equilíbrio.

O terceiro parâmetro é o quanto de decréscimo terá a taxa quando se detectar congestionamento. O TCP usa decrescimento multiplicativo, diminuindo pela metade a sua janela em caso de perdas. Uma abordagem similar é usada e a taxa é diminuída pela metade em caso de perdas. Embora diminuir a janela e a taxa não é exatamente a mesma coisa, o efeito é o mesmo, ou seja, o recuo suficiente para que as filas formadas na rede possam desaparecer. Se cada protocolo usasse a mesma técnica, seria possível recuar menos, mas nas condições atuais é mais seguro recuar agressivamente (10).

3.5 O ALGORITMO DE CONTROLE DE CONGESTIONAMENTO HOMEOSTÁTICO

No algoritmo de controle de congestionamento homeostático existem três fases distintas:

1 – O Aumento Exponencial;

2 – A Prevenção do Congestionamento;

3 – O Controle de Congestionamento.

Na fase de aumento exponencial a ideia é convergir rapidamente para largura de banda disponível, então os pares de pacotes são enviados a cada cinco pacotes (10). O período (P) no qual os pacotes são enviados é ajustado de acordo com a equação 3.1. O erro ou a diferença entre o período ótimo e período atual é dado pela equação 3.2. Se o período ótimo é constante, isto é, se a largura de banda disponível não mudar, o erro decrescerá exponencialmente para um cenário estático.

$$P_{n+1=(1-\alpha)*} P_n + \alpha * P_{medido} \quad 3.1$$

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

$$Erro = (1 - \alpha)^n * (P_{medido} - P_n) \quad 3.2$$

A fase de prevenção do congestionamento é sinalizada por uma sequência de *jitters* positivos, mostrando que a rede está ficando carregada e as filas estão crescendo. Nesta fase, o período é corrigido pelo erro entre o período atual e o período ótimo. A magnitude do erro é dada pela média da somatória dos valores de *jitter*. Então o novo período é calculado de acordo com a equação 3.3.

$$P_{n+1} = P_n + (jitter1 + \cdots + jittern)/n) \quad 3.3$$

Onde n é o número de medidas .

Na terceira fase, o controle de congestionamento, o protocolo percebeu que a rede está congestionada, observando-se perdas de pacotes. O protocolo então inicia um decréscimo multiplicativo, dobrando o período a cada RTT, conforme equação 3.4, seguindo a seguinte expressão:

Se (horário_atual > horário_da_última_perda + RTT + 2* P_n)

$$P_{n+1} = P_n * 2 \quad (3.4)$$

Sendo P_C o período atual, P_{medido} o período medido e P_{otimo} o período ótimo, teremos três casos:

Caso 1 – Muito pequeno: $P_{medido} < \frac{(P_{otimo} - (1-\alpha)*P_c)}{\alpha}$

Caso 2 – Região ótima: $P_{otimo} \geq P_{medido} \geq \frac{(P_{otimo} - (1-\alpha)*P_c)}{\alpha}$

Caso 3 – Muito grande: $P_{medido} > P_{otimo}$

Se o período é muito pequeno (caso 1), a taxa será muito grande e poderá causar congestionamento. Se o período é muito grande (caso 3), o protocolo não alcançará a máxima vazão. Se o período cair na região ótima (caso 2), o novo período ficará próximo daquele que resultará no uso de quase toda a largura de banda disponível do enlace. A natureza homeostática do controle de congestionamento atua para corrigir os erros nas medidas. Por haver um viés para superestimar a largura de banda disponível, normalmente as medidas caem nos casos um ou dois. O caso dois é o ótimo. O caso um gerará *jitters* positivos e a correção do *jitter* atuará abaixando a taxa. O caso três é mais raro. O valor de *jitter* somente pode ser tão grande quanto à soma de tempo que um pacote pode despender nas filas dos roteadores ao longo do caminho, dando um limite sobre o valor máximo absoluto (10).

3.6 SUMÁRIO DO CAPÍTULO 3

Nesse capítulo foi apresentado o algoritmo de controle de congestionamento homeostático, HCC, que utiliza dois mecanismos que tendem a resultar em comportamentos opostos para manter o equilíbrio dinâmico: o par de pacotes que tende a superestimar a largura de banda disponível e a medição sistemática da variação do atraso que tende a corrigir o valor superestimado pelo par de pacotes.

Foi apresentado também as fases que contemplam o algoritmo: o aumento exponencial para rapidamente atingir a largura de banda disponível, a prevenção do congestionamento com a diminuição da taxa de envio de pacotes baseada nas medições da variação do atraso e, por último, o controle de congestionamento, dobrando o período de envio dos pacotes para diminuir a taxa de envio pela metade.

4 A IMPLEMENTAÇÃO

4.1 INTRODUÇÃO

Este capítulo discorre sobre a forma como o algoritmo de controle de congestionamento homeostático, HCC, foi implementado no arcabouço do protocolo UDT (*UDP-based Data Transfer Protocol)* para a realização dos experimentos utilizados na avaliação desse novo protocolo.

A utilização do UDT deveu-se a uma importante característica desse arcabouço que é disponibilizar uma classe genérica de controle de congestionamento que pode ser utilizada para a rápida implementação de um novo algoritmo de congestionamento. Com isso, o trabalho voltou a atenção para o estudo do comportamento de um protocolo que utiliza o algoritmo de congestionamento HCC sem ter de implementá-lo completamente.

Na sequência serão apresentadas as principais características desse arcabouço e o desenvolvimento do código específico do algoritmo HCC.

4.2 O UDT

De acordo com Gu et al (47) (48) a motivação original do UDT foi superar o problema da ineficiência do TCP sobre redes de alta velocidade, discutido nos capítulos anteriores.

O outro objetivo foi permitir que pesquisadores, estudantes e desenvolvedores pudessem implementar e testar facilmente novos algoritmos e protocolos de transferência de dados.

Uma vez que a Internet continua evoluindo, novos desafios e requerimentos para protocolos de transporte sempre emergirão, e o UDT poderá ser utilizado por pesquisadores para desenvolver e testar, de forma ágil, esses novos algoritmos e protocolos, em particular o algoritmo de congestionamento, e assim, conduzir experimentos sobre redes reais.

O arcabouço do protocolo UDT, desenvolvido na linguagem de programação C++, disponibiliza uma classe genérica de controle de congestionamento que pode ser utilizada para a rápida implementação desses novos algoritmos de congestionamento. Em (48) (16) foram mostradas as implementações de alguns algoritmos derivadas desta classe.

O UDT foi concebido para funcionar sobre o protocolo UDP (9). Mas, ao contrário desse, ele é orientado à conexão, de unidifusão (*unicast*) e duplex, suporta tanto fluxo de dados confiáveis quanto de mensagens. A semântica do fluxo de dados é similar ao do TCP, enquanto que a semântica das mensagens pode ser considerada como um subconjunto do

SCTP (11). Ambos os pacotes de dados e controle são transferidos usando UDP. O módulo de controle de congestionamento é um *framework* aberto que pode ser utilizado para implementar diferentes algoritmos de controle, conforme mencionado anteriormente. Além disso, o UDT tem também um algoritmo de controle nativo baseado no controle de taxa AIMD, chamado DAIMD (*additive increase multiplicative decrease with decreasing increases* - AIMD com aumentos decrescentes).

4.2.1 ARQUITETURA

O arcabouço UDT tem uma arquitetura de camadas, conforme mostrado na Figura 4-1. O UDT usa o UDP através da interface de soquete disponibilizada pelos sistemas operacionais e, da mesma forma, provê a interface de soquete para as aplicações. Dessa forma o arcabouço *k* pode ser considerado um *middleware* entre as aplicações e a camada de transporte.

As aplicações podem chamar as interfaces de programação de aplicativos (API - *Application Programming Interface*) dos soquetes UDT da mesma maneira que elas chamam as API dos soquetes do sistema. Elas também podem prover uma instância de classe de controle de congestionamento (CC na Figura 4-1) para o UDT processar os eventos de controle definidos pelo usuário ou usar o algoritmo de controle de congestionamento padrão disponibilizado pelo UDT (47).

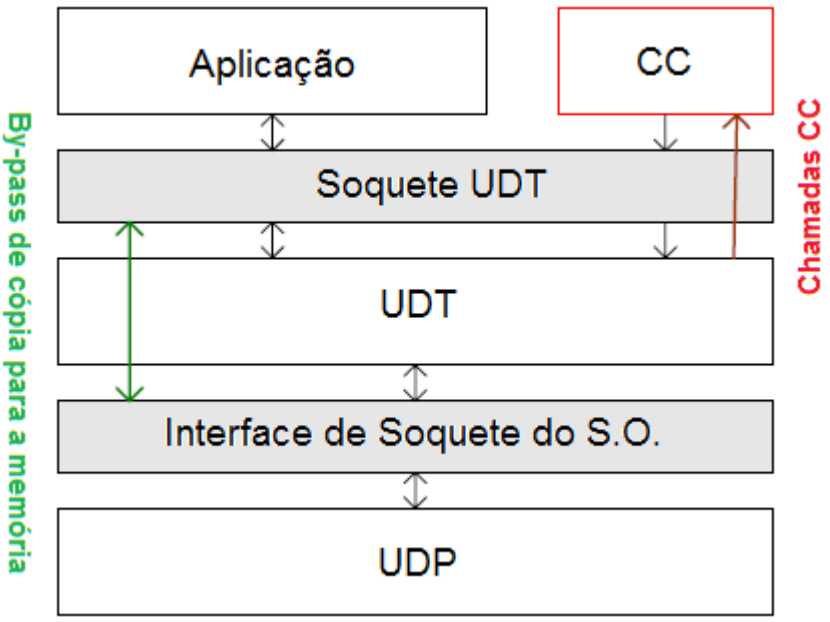

Figura 4-1– Arquitetura UDT/CC (16)

A Figura 4-2 descreve a arquitetura do software UDT/CC. A camada UDT tem cinco componentes de função: o módulo API, o transmissor, o receptor, o *listener* e o canal UDP, assim como quatro componentes de dados: o *buffer* do transmissor, o *buffer* do receptor, a lista de perdas do transmissor e a lista de perdas do receptor. Uma vez que o UDT é bidirecional, todas as entidades UDT têm a mesma estrutura.

O módulo API é responsável por interagir com as aplicações. Os dados a serem transmitidos são passados para o buffer do transmissor e enviados pelo transmissor no canal UDP. No outro lado da conexão (não mostrado na figura, mas com a mesma estrutura), o receptor lê os dados do canal UDP no buffer de recepção, reordena-os, se necessário, e verifica as perdas de pacotes. As aplicações podem ler os dados recebidos do buffer do receptor.

O receptor também processa as informações de controle. Ele atualizará a lista de perdas do transmissor (quando um pacote NAK é recebido) e a lista de perdas do receptor (quando uma perda é detectada). Certos eventos de controle desencadearão a atualização, pelo receptor, do módulo de controle de congestionamento, o qual, por sua vez, atualizará a taxa de envio de pacotes no transmissor.

O UDT utiliza o modelo cliente-servidor. Uma das entidades UDT é iniciada primeiro como servidor (*listener*), aceita e processa requisições recebidas e cria novos soquetes para cada nova conexão. Uma vez que as estações de trabalho normalmente têm múltiplos processadores, o UDT usa uma implementação *multi-threading*. O transmissor, o receptor e o *listener* são *threads* simultâneas, mas elas são iniciados somente sob demanda.

Uma *thread* (linha de execução) é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente. O suporte à *thread* é fornecido pelo próprio sistema operacional (SO), no caso da linha de execução no nível do núcleo (em inglês: *Kernel-Level Thread* (KLT)), ou implementada através de uma biblioteca de uma determinada linguagem, no caso de uma *User-Level Thread* (ULT).

Um *thread* permite que o usuário de programa, por exemplo, utilize uma funcionalidade do ambiente enquanto outras *threads* realizam outros cálculos e operações.

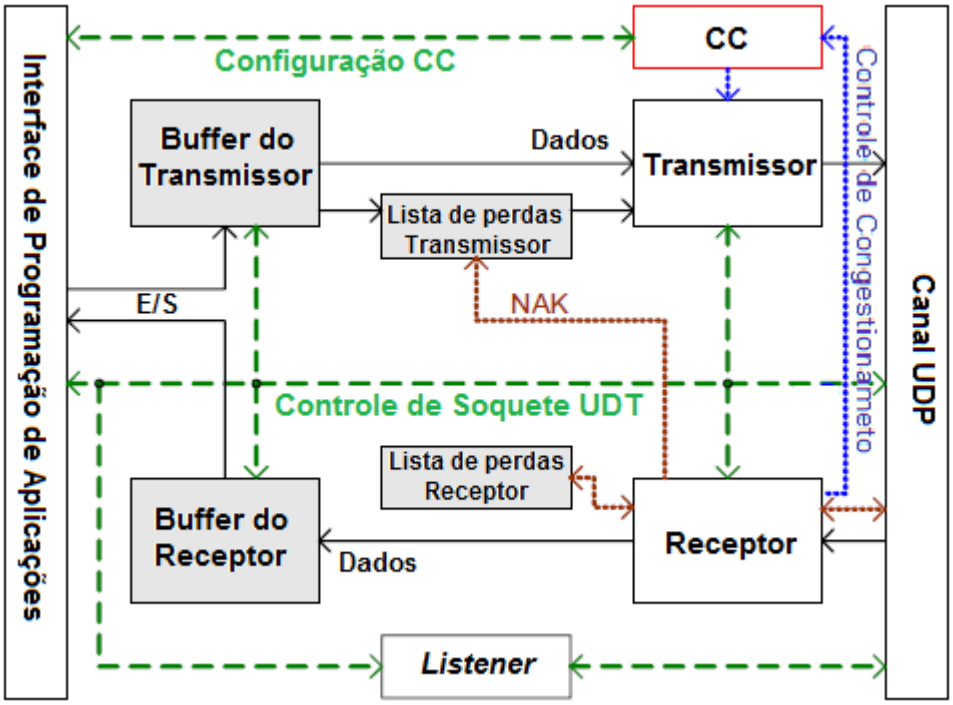

Figura 4-2 – Implementação UDT/CC: as linhas sólidas representam o fluxo de dados, as linhas tracejadas representam o fluxo de controle. Os blocos cinza (buffers e lista de perdas) são os componentes de dados e os blocos em branco são os componentes de função (16).

Em hardwares equipados com uma única CPU, cada *thread* é processada de forma aparentemente simultânea, pois a mudança entre uma linha e outra é feita de forma tão rápida que para o usuário isso está acontecendo paralelamente. Em hardwares com múltiplas CPUs ou com múltiplos núcleos, as *threads* podem ser executados realmente de forma simultânea.

Os sistemas que suportam apenas uma única *thread* são chamados *monothread* e aqueles sistemas que suportam múltiplas *threads* são chamados de *multithread*

4.2.2 QUESTÕES DE IMPLEMENTAÇÃO

Jain et al (49) percebeu, há mais de duas décadas, a dificuldade em processar transferências de dados na casa dos Gigabits por segundo. Apesar de certas restrições da época em relação ao hardware não existirem mais, a forma como um protocolo de transporte é implementado na camada de aplicação determina o seu desempenho.

A leitura dos dados que chegam nos pacotes mais a geração de informações de controle, no receptor, podem levar um tempo muito longo se comparado à alta taxa de chegada dos pacotes em redes de alta velocidade. Dessa forma, uma implementação ruim pode ocasionar estouro de temporizadores e perdas frequentes de pacotes ou até mesmo uma

avalanche de perdas de pacotes à medida que a perda de processamento pode causar mais perdas de pacotes.

O UDT usa o modo explícito de informação de perdas e, para isso, o receptor mantém uma lista de perdas para armazenar essas informações. Acessar essa lista, que pode conter milhares de pacotes, pode despender tanto tempo que a quantidade de pacotes que chega exceda a capacidade do *buffer* do protocolo.

Nas próximas duas seções, são mostradas as formas como o UDT trata esse problema e também a cópia para memória.

4.2.2.1 GERENCIAMENTO DE INFORMAÇÕES DE PERDAS

Quando ocorrem as perdas durante o período de congestionamento elas frequentemente são contínuas. Por isso, ao invés de utilizar todos os números de sequência dos pacotes perdidos, podem ser usados dois valores para representar um evento de perda: o primeiro e o último da sequência. Por exemplo, se os pacotes com números de sequência de 200 a 500 são perdidos, o par [200, 500] pode ser utilizado para gravar a perda ao invés de se usar 301 números.

A complexidade de busca é muito menor que aquela de varrer um vetor regular ou uma lista, porque há menos eventos de perdas que pacotes perdidos. Além do que, cada acesso relativo a um evento consome um período de tempo similar.

Com o armazenamento das informações de perdas, não se faz necessários vetores ou mapas de bits para confiabilidade dos dados.

4.2.2.2 PREVENÇÃO DE CÓPIA NA MEMÓRIA

A prevenção de cópia na memória em protocolos de transporte para redes de alta velocidade é muito mais crítica do que em protocolos para aplicações de internet convencional devido ao grande volume de dados transferidos. A ideia nesse caso é evitar a perda de pacotes quando a CPU ou o barramento do sistema estiverem ocupados copiando grandes blocos de dados grandes.

Em Fast Sockets (50) e Zero Copy TCP (51) foi apresentada a prevenção de cópia entre o modo de acesso ao usuário e de acesso ao kernel. Porém, na camada de aplicação há uma outra cópia para memória que deve ser evitada: entre o buffer do protocolo e a aplicação.

O UDT implementa operações assíncronas de E/S (Entrada e Saída) para reduzir a cópia na memória entre o buffer do protocolo e a aplicação. Adicionalmente insere um buffer da aplicação como uma extensão lógica no buffer do protocolo, conforme mostrado na Figura 4-3.

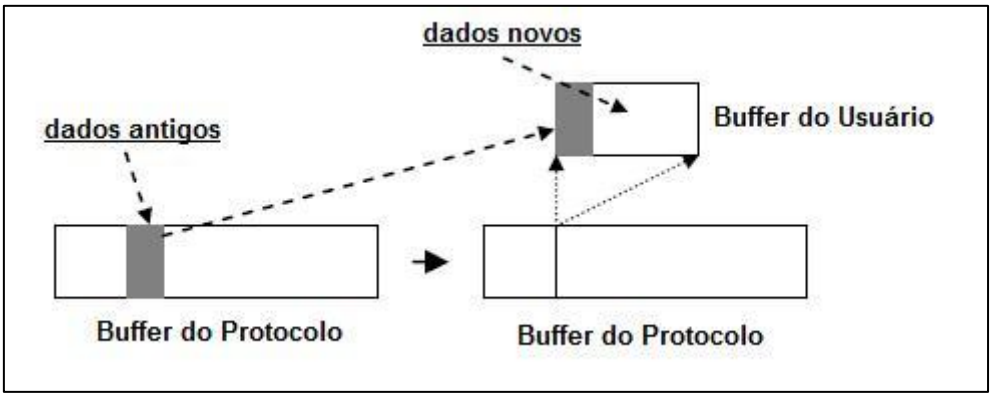

Figura 4-3 - Inserção do buffer de aplicação no buffer do protocolo. O lado esquerdo representa o buffer do protocolo antes da inserção e, o lado direito representa o resultado final. A área cinza do buffer representa os dados já existentes no buffer. Os dados novos serão escritos diretamente no buffer do usuário (47).

4.2.2.3 TEMPORIZADORES DE ALTA PRECISÃO

Para suportar o controle de taxas de transferência de dados na velocidade de gigabits por segundo, a precisão de relógio dever ser pelo menos no nível de microssegundos.

A granularidade é limitada pela resolução do relógio do núcleo do sistema. E, com os temporizadores de alta resolução, *High Resolution Timers* (52), a medida do tempo é realizada diretamente na fonte de relógio do hardware, permitindo uma resolução da ordem de microssegundos (53). Como o núcleo dos sistemas Linux modernos usam o *tickless design*, já é possível para programas de usuário atingir precisões menores que 1ms, Desta forma, acredita-se ser possível escalonar com razoável precisão o envio de pacotes.

4.2.2.4 IMPLEMENTAÇÃO E PROJETO DA API

A API, ou interface de programação de aplicativo, precisa ser considera quando se implementa um protocolo de transporte. Geralmente, é uma boa prática cumprir com as semânticas do soquete. No entanto, devido aos requerimentos especiais e uso em cenários com aplicações de alto desempenho, modificações adicionais em relação à API original são necessárias.

No passado os programadores de rede adotaram o método *sendfile* (54). Em adição ao *sendfile*, um novo metodo, *recvfile*, também foi acrescentado para receber dados diretamente no disco.

O UDT também implementa a operação assíncrona de E/S em ambos os lados do transmissor e receptor. Os métodos e parâmetros relacionados à operação assíncrona foram adicionados na API. A operação assíncrona de E/S é um método efetivo para a redução de cópias na memória (47).

Se a camada de transporte souber se uma perda de pacotes é fruto do congestionamento ou de erro no enlace, esta informação será muito útil para o controle de congestionamento em enlaces com taxas altas de erro de bit. Por isto, algumas API de nível inferior devem estar expostas para as aplicações. O UDT expõe muitas interfaces do UDP para dar às aplicações mais flexibilidade para configurar suas características de transporte.

4.2.3 ESTRUTURA DOS PACOTES

O UDT possui dois tipos de pacote: um de dados e o outro de controle. Eles se distinguem pelo primeiro bit de cabeçalho do pacote, conhecido como bit de *flag*.

O cabeçalho do pacote de dados inicia-se com o primeiro bit igual zero. O número de sequência dos pacotes utiliza os trinta e um bits seguintes depois desse primeiro bit de *flag*. O sequenciamento de pacotes do UDT é realizado com o número de sequência sendo incrementado de um para cada pacote de dados enviado, na ordem do envio. O número de sequência é sobreposto depois de ser incrementado no número máximo de 2^{31} - 1. A Figura 4-4 abaixo mostra a estrutura do cabeçalho de um pacote de dados com a representação dos trinta e dois (a largura da figura) bits de cada campo (linha da figura).

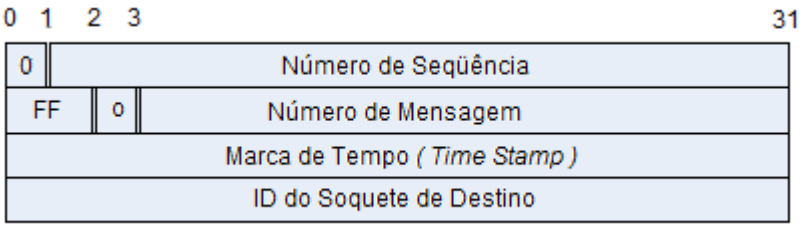

Figura 4-4 – Estrutura do cabeçalho de um pacote de dados UDT (16)

O próximo campo de trinta e dois bits no cabeçalho do pacote de dados é usado para a identificação da mensagem. Os primeiros dois bits, representado na figura por FF, indicam a posição do pacote na mensagem. A sequência “10” representa o primeiro pacote, “01” o

último, “11” identifica que há somente um pacote e “00” qualquer pacote no meio. O terceiro bit “o” significa se a mensagem deve ser entregue em ordem (1) ou não (0). A mensagem a ser entregue em ordem requer que todas as mensagens anteriores sejam ou entregues ou descartadas. Os vinte e nove bits restantes são usados para identificar o número da mensagem, similar ao número de sequência de pacotes, mas independente. Uma mensagem UDT pode conter múltiplos pacotes UDT.

Na sequência, estão os trinta e dois bits da marca de tempo (*time stamp*). A marca de tempo é um valor relativo se iniciando no momento da criação da conexão. A informação de marca de tempo não é requerida pelo UDT e nem pelo seu algoritmo de controle nativo, mas é incluído para o caso do algoritmo de controle definido pelo usuário requerer essa informação.

O identificador (ID) do destino é usado pelo multiplexador de UDP. Múltiplos soquetes do UDT podem ser associados à mesma porta UDP e esse identificador do soquete é usado para diferenciar as diversas conexões do UDT.

Se o bit de *flag* do pacote UDT é igual a um, então se trata de um pacote de controle que possui a estrutura conforme Figura 4-5.

Existem oito tipos de pacotes de controle no UDT e a informação de qual tipo se trata é colocada nos campos dos bits de um a quinze do cabeçalho conforme está representado na Figura 4-5. O teor dos campos seguintes depende do tipo de pacote. Os 128 primeiros bits devem sempre existir no cabeçalho do pacote, enquanto que, dependendo do tipo de pacote, pode haver campos de informações de controle vazios.

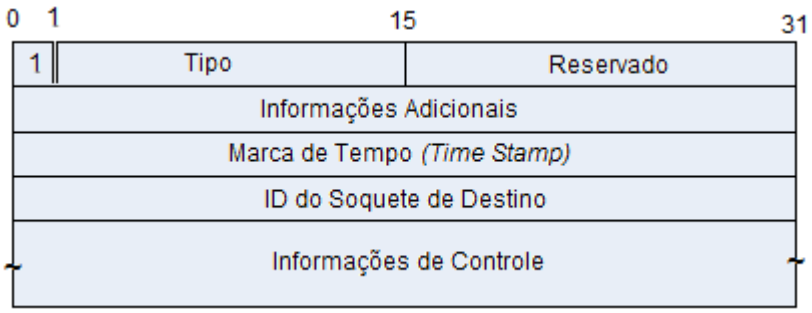

Figura 4-5 - Estrutura de um pacote de controle UDT (16)

Os oito pacotes de controle definidos são:

- Tipo 0x0: *Handshake* de Conexão do Protocolo – Contem informações para o estabelecimento da conexão.

- Tipo 0x1: Keep-alive – utilizado para verificar o estado da conexão.

- Tipo 0x2: Confirmação (ACK) – confirmação de recebimento de determinados pacotes. Contém diversas informações da conexão.

- Tipo 0x3: Confirmação Negativa (NAK) – informação de perdas.
- Tipo 0x5: Desconexão – utilizado para informar a intenção de desconexão.
- Tipo 0x6: Confirmação da Confirmação (ACK2) – confirmação dos pacotes ACK.
- Tipo 0x7: Requisição de Descarte de Mensagem.
- Tipo 0x7FFF: Explicados pelos bits 16 a 31, reservado para definições do usuário.

O UDT utiliza subsequenciamento para os pacotes de ACK. A cada pacote ACK é associado um número de sequência único de 16 bits que é independente do número de sequência dos pacotes de dados. O número de sequência de ACK utiliza os bits de 32-63 ("Informações adicionais") no cabeçalho de pacote de controle. O número de sequência do ACK varia de zero a 2^{31} - 1.

Finalmente, os campos de marca de tempo (*Time Stamp*) e ID do soquete de destino sempre existem em todos os tipos de pacotes de controle do UDT, completando os 128 primeiros bits do cabeçalho do pacote.

4.2.4 TEMPORIZADORES

O UDT utiliza quatro temporizadores para disparar diferentes eventos periódicos. Cada evento tem seu próprio período e eles são todos independentes. Eles utilizam o horário do sistema como fonte.

Os quatros temporizadores são ACK, NAK, EXP e SND. SND é usado somente no transmissor para o envio de pacotes baseado em taxa, enquanto que os outros três são usados somente no receptor.

O temporizador ACK é usado para desencadear uma confirmação (ACK). Seu período é determinado pelo módulo de controle de congestionamento. Contudo, o UDT enviará um ACK com frequência de pelo menos um a cada 0,01 segundos, mesmo que o controle de congestionamento não precise de temporização baseada em ACK. Aqui, 0,01 segundos (10 milissegundos) é definido como um período SYN, ou período de sincronização, e afeta muitos dos outros temporizadores usados no UDT.

O temporizador NAK é usado para disparar uma confirmação negativa (NAK). Seu período é atualizado dinamicamente para 4 * RTT + RTTVar + SYN,_onde RTTVar é a variância de amostras RTT.

O temporizador EXP é usado para desencadear retransmissões de pacotes de dados e manter estado de conexão. Seu período é atualizado dinamicamente para N * (4 * RTT_ + RTTVar + SYN), onde N é o número de *timeouts* contínuos. Para evitar *timeouts*

desnecessários, um limiar mínimo (por exemplo, 0,5 segundos) dever ser usado na implementação.

A granularidade recomendada para os períodos dos temporizadores é da ordem de microssegundos. No entanto, a manutenção da precisão de tempo não é necessária, exceto para o temporizador SND.

4.2.5 ENVIO E RECEBIMENTO DE DADOS

Cada entidade UDT tem duas partes lógicas: o transmissor e o receptor. O transmissor envia (e retransmite) os dados de aplicação de acordo com os controles de fluxo e congestionamento. O receptor recebe ambos os pacotes de dados e controle e envia os pacotes de controle de acordo com os pacotes recebidos e os temporizadores.

O receptor é responsável por iniciar e processar todos os eventos de controle, incluindo o controle de congestionamento e controle de confiabilidade e seus mecanismos relacionados.

O UDT sempre tenta empacotar os dados da aplicação em pacotes com tamanho fixo (o tamanho máximo do pacote negociado durante o estabelecimento da conexão), a menos que não existam dados suficientes para serem enviados.

O tamanho da janela de controle de fluxo é inicialmente 16 pacotes. No recebimento do pacote de ACK, o valor da janela de controle de fluxo é atualizado para o tamanho disp\onível do buffer no receptor, no caso do algoritmo de congestionamento original.

4.2.6 ESQUEMA DE COMPRESSÃO DE INFORMAÇÃO DE PERDA

A informação de perda disponível no pacote NAK é um *array* de 32 bits (inteiros). Ele pode conter informações individuais de pacotes ou um intervalo de pacotes. Se o primeiro bit de um inteiro no vetor for zero significa que é um número de sequência normal, ou seja, que o pacote com esse número de sequência foi perdido; se o primeiro bit for um, significa que todos os pacotes começando por esse número de sequência (31 bits restantes do inteiro) até o próximo número no vetor (cujo primeiro bit dever ser 0) foram perdidos.

Por exemplo, se a seguinte informação é transportada no pacote NAK:

0X00000002, 0X**8**0000006, 0X**0**000000B, 0X0000000E

significa que os pacotes com as sequencias 2, de 6 a 11 (0xB) e 14 (0xE) foram perdidos.

4.2.7 CONTROLE DE CONGESTIONAMENTO CONFIGURÁVEL

O controle de congestionamento no UDT é um *framework* aberto e por isso um algoritmo de controle de congestionamento definido pelo usuário pode ser facilmente implementado e modificado. Particularmente, o algoritmo de controle nativo é também implementado nesse arcabouço .

O algoritmo definido pelo usuário pode redefinir diversas rotinas de controle que listam e ajustam vários parâmetros do UDT. As rotinas serão chamadas quando certos eventos ocorrerem. Por exemplo, quando um ACK é recebido, o algoritmo de controle pode incrementar o tamanho da janela de congestionamento.

4.2.7.1 INTERFACE CCC

O UDT permite ao usuário acessar dois parâmetros do controle de congestionamento configurável (CCC): o tamanho da janela de congestionamento e o intervalo de envio dos pacotes. Os usuários podem ajustar esses dois parâmetros para realizar o controle baseado em janela, baseado em taxa ou uma abordagem híbrida.

Em adição, os seguintes parâmetros podem ser informados.

1) RTT;
2) Tamanho máximo do segmento;
3) Banda estimada;
4) O último número de sequência que foi enviado até então;
5) A taxa de chegada de pacotes no lado do receptor.

Uma implementação do UDT pode também expor parâmetros adicionais. Essa informação pode ser utilizada nos algoritmos de controle de congestionamento definidos pelo usuário para ajustar a taxa de envio de pacotes.

Os seguintes eventos de controle podem ser redefinidos via CCC (por exemplo, por uma chamada de função).

1) *init*: quando o soquete UDT é conectado.

2) *close*: quando o soquete UDT é encerrado.

3) onACK: quando um ACK é recebido.

4) onLoss: quando um NAK é recebido.

5) onTimeout: quando ocorre timeouts.

6) onPktSent: quando um pacote de dados é enviado.

7) onPktRecv: quando um pacote de dados é recebido.

Os usuários podem também ajustar os seguintes parâmetros nos algoritmos de controle definidos pelo usuário:

(1) Intervalo de ACK: Um ACK pode ser enviado a cada número fixo de pacotes. O usuário pode definir esse intervalo. Se o valor é -1, então significa que nenhum ACK será enviado baseado no intervalo dos pacotes.

(2) Temporizador de ACK: um ACK também será enviado baseado num intervalo fixo de tempo. Isto é mandatário no UDT. O intervalo de tempo máximo e padrão do ACK é SYN.

(3) Temporizador de retransmissão: RTO: UDT utiliza 4 * RTT + RTTVar para computar o RTO. Usuários podem redefinir isso.

A descrição e a discussão detalhada sobre UDT/CCC podem ser encontradas em (48).

4.3 IMPLEMENTAÇÃO DO HCC NO UDT

O HCC, por ser baseado em taxa, deixa a janela de controle de fluxo no tamanho máximo para que não haja restrição quanto a esse valor, e fica atualizando o período de envio de pacotes baseado nas informações recebidas no transmissor através dos pacotes de ACK e NAK oriundos do receptor.

No algoritmo original do HCC, o par de pacotes era enviado a cada cinco pacotes, no entanto, como o UDT também utiliza esse mecanismo, ele foi aproveitado. Contudo, no UDT o envio do par de pacotes acontece a cada dezesseis pacotes. Esta modificação não constitui uma mudança significativa no comportamento do protocolo.

4.3.1 O ALGORITMO

O funcionamento do algoritmo é o seguinte: quando do recebimento de um pacote de ACK, o evento *onACK* será disparado. Assim, o algoritmo lê as informações de RTT e da banda disponível recebidas no pacote de ACK. A banda é utilizada para saber o valor do período de envio dos pacotes, medido em microssegundos, que será o inverso do valor da largura de banda em Mbps multiplicado 10^6.

O outro valor utilizado é o RTT, pois o algoritmo faz modificações a cada RTT, pelo menos, para evitar oscilações. O horário da última modificação é armazenado e quando um ACK é recebido, o algoritmo verifica se a diferença entre o horário atual e a última modificação é maior ou igual ao valor de RTT. Se for, ele atuará e caso contrário nenhuma modificação ocorrerá.

No caso do recebimento de um ACK, o período será modificado de acordo com a seguinte lógica: o novo período de envio dos pacotes será trinta por cento do valor medido nesse pacote recebido mais setenta por cento do valor anterior. Um gerador de número aleatório gera um número randômico entre 0,9 e um para evitar sincronização. Além disso, um valor de *jitter* é calculado, o período medido menos o valor do período anterior. Se o valor de duas dessas medidas consecutivas de *jitter* for positivo, significa que a rede está ficando carregada e o algoritmo precisa diminuir a taxa de envio de pacotes. Ele faz isso, adicionando a média dos valores de *jitters* calculados ao valor do novo período de envio de pacotes.

O outro evento observado pelo o algoritmo é o *onLoss* que ocorre no recebimento de um pacote de NAK, indicando que houve perda de pacotes. Nesse caso o algoritmo duplica o valor do período de envio dos pacotes, a fim de diminuir a taxa à metade e desafogar a rede. Da mesma forma que no evento de ACK, o RTT previamente armazenado é utilizado para determinar se já se passou pelo menos um RTT para o protocolo efetuar a modificação a fim de evitar oscilações.

4.4 SUMÁRIO DO CAPÍTULO 4

Nesse capítulo foi apresentado como o algoritmo de controle de congestionamento homeostático foi implementado sobre o arcabouço do protocolo UDT.

Algumas características e a arquitetura desse arcabouço também foram apresentadas para esclarecer detalhes de implementação que podem interferir nos resultados dos testes.

O gerenciamento das informações de perda, a prevenção de cópia na memória, a utilização de temporizadores de alta resolução e como se projetará e implementará a API são detalhes importantes para o desempenho de um protocolo em redes com alto PLBA.

5 A AVALIAÇÃO DO HCC

5.1 INTRODUÇÃO

Este capítulo descreve como foi realizada a avaliação do protocolo que utiliza o algoritmo de controle de congestionamento homeostático HCC. Contempla desde a estratégia de avaliação até os resultados encontrados, passando pelo cenário utilizado e testes realizados.

5.2 ESTRATÉGIA DE AVALIAÇÃO

Neste trabalho, o algoritmo de controle de congestionamento homeostático HCC foi implementado sobre o arcabouço do protocolo UDT.Essa implementação foi utilizada nos testes para avaliação do desempenho do algoritmo, uma vez que o objetivo do trabalho foi a avaliação do algoritmo de controle de congestionamento homeostático sobre o UDT e não de um novo protocolo propriamente dito.

Conforme visto no capítulo quatro, o arcabouço do protocolo UDT, desenvolvido em C++, disponibiliza uma classe genérica de controle de congestionamento que pode ser utilizada para a rápida implementação de um novo algoritmo de congestionamento. Em (48) e (16), foram mostradas as implementações de alguns algoritmos derivadas dessa classe. Dentre elas está a implementação do TCP *New Reno*, chamada por Gu et al de CTCP (16)

Como existem diversas versões de algoritmos de controle de congestionamento do protocolo TCP que foram desenvolvidas desde a primeira proposição de Jacobson, a ideia foi usar a versão mais amplamente utilizada nas redes hoje em dia. Porém, determinar a versão do protocolo mais amplamente utilizada na Internet hoje em dia não é uma tarefa trivial (30).

Nos sistemas operacionais Linux, a versão padrão do TCP é o CUBIC (20) desde a versão do *kernel* 2.6.19, disponibilizado em novembro de 2006 (55). Antes desta versão, foi o BIC (19) o algoritmo padrão do *kernel* do Linux desde a versão 2.6.8, liberada em agosto de 2004 (56). O *Compound* TCP (23) foi desenvolvido para sistemas operacionais Windows. Os sistemas Operacionais Windows Vista, Windows 7 e Windows 2008 Server têm esse algoritmo disponível, sendo que no último é configurado como padrão para o TCP (57).

O IETF (*Internet Engineering Task Force*), organismo padronizador da Internet descreve essas novas versões como experimentais (58), inclusive o TCP *New Reno* (59). O padrão é definido pelos quatros algoritmos que compõem o controle de congestionamento: partida lenta, controle de congestionamento, retransmissão rápida e recuperação rápida. Esses algoritmos são atualmente descritos na RFC 5681 (31), atualização da RFC 2581 (60).

Por isso, a implementação do algoritmo de controle de congestionamento do TCP, CTCP, descrita em Gu (48) e (16) foi utilizada como referência para os testes de desempenho. Apesar dela ter sido baseada na RFC 2581, pois a atual RFC foi liberada depois dos experimentos de Gu em 2007, não houve grandes mudanças na implementação: o tamanho da janela que era de dois segmentos na RFC 2581 passou para até 4380 bytes e algumas das modificações que foram tornadas padrão na nova RFC já haviam sido sugeridas antes e foram implementadas no CTCP.

É importante salientar que essa implementação do TCP sobre o arcabouço do UDT não provê exatamente os mesmos mecanismos que o TCP implementado diretamente no núcleo do sistema operacional, somente o controle de congestionamento desse. Porém, provê as mesmas funcionalidades. Por exemplo, apesar de o protocolo TCP utilizar o sequenciamento baseado em bytes e o CTCP usar sequenciamento baseado em pacotes, os resultados dos testes de vazão, imparcialidade e estabilidade são bastante similares, conforme apresentado por Gu (16).

5.3 TESTES REALIZADOS

Com o surgimento de diversas propostas de modificações nos algoritmos de controle de congestionamento existentes e mesmo a proposição de novos algoritmos, tornou-se necessário estipular parâmetros para a avaliação do desempenho desses e certificar que esses novos algoritmos não impactassem no funcionamento das redes já implantadas. Em (61), os princípios básicos do controle de congestionamento são descritos e em (58) são fornecidas orientações para a avaliação de algoritmos propostos quando esses diferirem significativamente dos princípios descritos em (61).

Dentre esses princípios e orientações, este trabalho observou o impacto que o protocolo UDT utilizando o algoritmo de controle de congestionamento HCC provocaria no padrão TCP existente, através do teste de equidade com o TCP (*TCP Friendliness*). Observou-se também a eficiência em relação à vazão, a equidade entre diferentes fluxos utilizando o HCC, a convergência e a estabilidade.

5.3.1 A EFICIÊNCIA EM RELAÇÃO À VAZÃO

É a medida do quão eficiente é o algoritmo para utilizar toda a largura de banda de disponível no enlace. É definida como a vazão total de todos os fluxos simultâneos.

Supondo que existam n fluxos HCC na rede e o fluxo i-th tem uma vazão média xi, então o índice de eficiência, IE, é definido como (16):

$$IE = \sum_{i=1}^{n} \frac{\bar{x}\iota}{Vmax} \quad 5.1$$

Onde $Vmax$ é a vazão máxima possível no enlace.

5.3.2 EQUIDADE ENTRE DIFERENTES FLUXOS UTILIZANDO O HCC

Mede quanto equitativamente os n fluxos concorrentes dividem a largura de banda disponível. É medido pelo índice de imparcialidade de Jain (62), descrito aqui como IJ:

$$IJ = \frac{(\sum_{i=1}^{n} \overline{x\iota})^2}{n * \sum_{i=1}^{n} \bar{x}\iota^2} \quad 5.2$$

Calcula-se a vazão média de cada fluxo, $\bar{x}\iota\cdot$, e o índice de *Fairness* de Jain é calculado a partir desses valores. Quanto mais próximo de um for índice, melhor a distribuição da banda entre os diversos fluxos.

5.3.3 EQUIDADE COM O TCP.

A convivência harmoniosa com o TCP é uma medida mais difícil de analisar, uma vez que é quase impossível para um protocolo com algoritmo de controle de congestionamento diferente ter o mesmo comportamento do TCP (16), e assim obter a mesma fração da banda disponível. Por outro lado, não é razoável limitar, em redes com alto PLBA, a vazão do novo protocolo ao do TCP, dado que esse último é reconhecidamente ineficiente (58).

Suponha-se que existam m fluxos HCC coexistindo com n fluxos TCP na rede. Com a mesma configuração de rede é iniciado outro teste com m + n fluxos TCP, somente. A vazão média para o i-th fluxo TCP será xi para o primeiro caso e yi e para o segundo. O índice de equidade com o TCP, IT, será (16):

$$IT = \frac{\frac{1}{n} * (\sum_{i=1}^{n} \bar{x}\iota)}{\frac{1}{m+n} * (\sum_{i=1}^{m+n} \bar{y}\iota)} \quad 5.3$$

Dessa forma, é comparada a vazão média quando há somente fluxos TCP com o cenário onde eles coexistem com HCC. O ideal é que o HCC não interfira na utilização da largura de banda do TCP.

Como o comportamento do TCP é diferente do HCC, a vazão média foi calculada levando-se em consideração somente o período de tempo onde todos os fluxos eram simultâneos, uma vez que o teste foi realizado com a transferência de um número fixo de dados entre o transmissor e o receptor para cada fluxo, e, o protocolo com melhor desempenho termina sua transferência mais rapidamente. Pelo fato do HCC terminar a sua transferência de dados antes do TCP, a simples comparação dos valores médios de vazão não dará uma medida real, pois quando há somente fluxos TCP ele transmite por mais tempo e quando há compartilhamento de fluxos HCC e TCP o tempo observado para o TCP pode não ter sido suficiente para atingir a sua capacidade máxima de transmissão. Assim, o valor da vazão precisa ser normalizado.

A Equidade ideal é com IT igual a um. Se IT é maior que um significa que o HCC é bastante amigável com o TCP e se IT é menor que um demonstra que o HCC se torna agressivo utilizando a maior parte da largura de banda para ele.

5.3.4 CONVERGÊNCIA PARA A BANDA DISPONÍVEL

Teste realizado para mensurar a habilidade do HCC de responder às mudanças da banda. Habilita-se um fluxo de HCC e aguarda até que ele atinja o valor máximo de banda ocupada. Depois um novo fluxo CBR é inserido e verifica-se se o HCC diminui a taxa de envio durante a concorrência e depois retoma o valor inicial depois que o novo fluxo CBR é retirado.

5.3.5 ESTABILIDADE

Teste para medir as características de oscilação de um fluxo de dados. Uma oscilação pequena é considerada o comportamento desejado para a maioria das situações e frequentemente provoca uma vazão melhor.

Para medir as oscilações, foi considerada a vazão média de cada intervalo de tempo (amostra). A oscilação foi expressa pelo desvio padrão dos valores das amostras da vazão de cada fluxo normalizado pela média dos valores da vazão (21).

$$DE = \frac{1}{n} * \sum_{i=1}^{n} \left(\frac{1}{\bar{xi}} * \sqrt{\frac{1}{m-1} * \sum_{k=1}^{m} (xi(k) - \bar{xi})^2} \right) \quad 5.4$$

Onde n é o número de fluxos concorrentes, m o número de amostras de vazão, $xi(k)$ é o valor da vazão da amostra *k*-th do fluxo *i* e $\bar{xi}$ é a vazão média de cada fluxo *i*.

Quando menor o valor do índice mais estável é o fluxo.

5.4 CENÁRIO DE TESTES

Como o objetivo do trabalho foi a observação da implementação do algoritmo HCC sobre redes com alto PLBA, foi utilizado um enlace de 1Gbps com o atraso configurado no subsistema de controle de tráfego do Linux, o módulo Netem (*Network Emulation*) (63) simulando um enlace transoceânico, conforme proposto por (64).

Foi montado um cenário com três máquinas ligadas através de um switch Gigabit Ethernet, modelo Planet GSD800S, com versão de software v1.2b090511, representado na Figura 5-1.

Na máquina 1 foi instalado o aplicativo do transmissor, na máquina 2 o aplicativo do receptor e, a máquina 3, funcionou como o roteador entre as duas primeiras.

As três máquinas foram equipadas da seguinte maneira. Máquinas 1 e 2:

- processador Quad-core, Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz
- 4GBytes memória RAM;
- controlador de interface de rede Ethernet Realtek RTL-8169 Gigabit Ethernet (rev 10) 66MHz.
- Distribuição Sistema Operacional Linux Ubuntu 9.04 com versão de kernel 2.6.31-22, x86_64.

Já a máquina 3 foi configurada da seguinte maneira:

- processador Intel(R) Core (TM) i7 CPU 860 @ 2.80GHz
- 3 GBytes memória RAM,
- Controlador de interface de rede Intel Corporation 82578DC Gigabit Network Connection 66MHz
- Distribuição Sistema Operacional OpenSuse Linux 11.1 com versão de Kernel 2.6.27.56 .

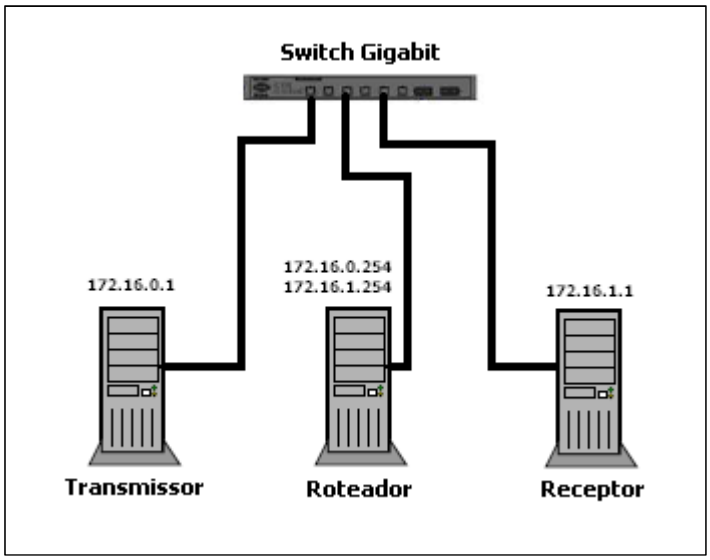

Figura 5-1 - Topologia de rede

Na máquina 3,com função de roteamento, foi utilizado o subsistema de controle de tráfego do Linux, o módulo Netem (*Network Emulation*). Esse módulo, incorporado ao núcleo do Linux a partir da versão 2.6.8, provê a facilidade de modificação das características de uma conexão fim a fim tais como atraso, variação do atraso (*jitter*), perda, reordenação e duplicação de pacotes para fins de testes, manipulando os pacotes na fila de saída do núcleo do Netem. É composto de duas partes: um pequeno módulo do núcleo para a disciplina da fila e um comando de linha para a configuração desse módulo.

Na Figura 5-2, pode-se observar a disciplina de enfileiramento do núcleo na saída do sistema entre a camada de rede IP e o dispositivo de rede. É nesse processo que as modificações são inseridas.

A manipulação do tamanho da fila no Netem foi desabilitada a partir da versão 2.6.28 do núcleo do Linux, quando o Netem foi modificado de *classful* para *classless*. Por isto, foi necessário utilizar na máquina com função de roteamento a versão 2.6.27 do *kernel*, uma vez que nas emulações com altos valores de PLBAé necessário um tamanho da fila suficiente para armazenar o número de pacotes enviados por RTT. Esse tamanho da fila é calculado pelo valor do PLBA dividido pelo tamanho dos pacotes. No caso dos testes o valor do tamanho do pacote foi de 1472 bytes já que o HCC foi implementado sobre o protocolo UDP. Por exemplo, um cenário com 200ms de atraso e 652 Mbps de largura de banda, tem-se em torno de 88.587 pacotes. Valor bem superior aos 1000 pacotes configurados por padrão nas interfaces de rede do sistema operacional Linux.

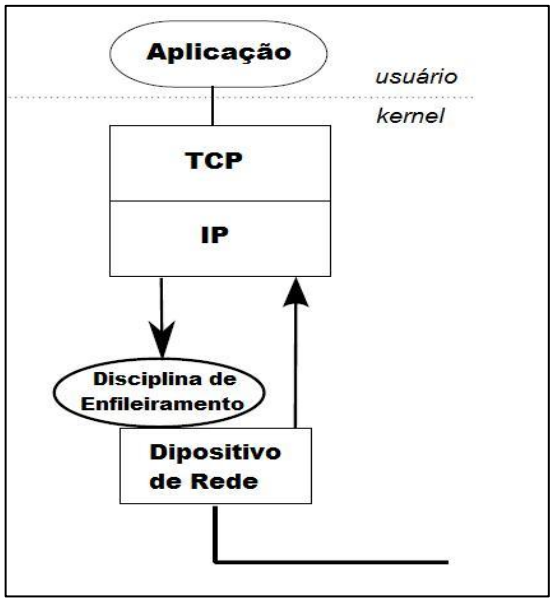

Figura 5-2 - Fila do Kernel (63)

Além das configurações do Netem, foi necessário realizar também alguns ajustes nas configurações padrões dos sistemas operacionais. Como nas redes com alto PLBA se transmite muitos dados dentro do mesmo RTT é necessário que os buffers dos sistemas operacionais estejam adaptados para esta quantidade de informação. Para isto os seguintes parâmetros foram modificados:

- tamanhos padrão e máximo do buffer de leitura para o soquete do sistema operacional;
- tamanhos padrão e máximo do buffer de escrita para o soquete do sistema operacional;
- tamanho do comprimento da fila para a entrega ao dispositivo de rede.

A outra configuração importante no cenário proposto é desabilitar o redirecionamento dos pacotes ARP (65). Como todas as máquinas estão conectadas ao mesmo switch sem configuração de VLAN, se não se desabilitar o redirecionamento dos pacotes ARP, as máquinas 1 e 2 se comunicarão diretamente sem passar pelo roteador onde a WAN é emulada.

Houve também a tentativa de utilização da rede PlanetLab (66) nos testes. Um *slice* foi criado e o protocolo HCC/UDT foi implementado. No entanto não foi possível a reserva de mais do que 100Mbps de banda nas máquinas disponíveis. Assim, como os testes de desempenho requeriam altas taxas durante grandes períodos de tempo, não houve prosseguimento com esses testes. A conclusão foi que a rede do PlanetLab é mais indicada para os testes de escalabilidade.

5.4.1 EMULANDO A WAN

O atraso simulado pelo Netem é por padrão baseado na distribuição uniforme. Outras distribuições como pareto e paretonormal também podem ser utilizadas. Nos experimentos foi utilizada a distribuição uniforme.

A granularidade do atraso é limitada pela resolução do relógio do *kernel* do sistema. Até a versão do *kernel* 2.6.21, os temporizadores mediam o tempo em *jiffies*, que é a unidade de tempo do temporizador de interrupções do sistema. A dimensão do *jiffy* é determinada pelo valor da variável HZ dos sistemas operacionais Linux, cujos valores poderiam ser 100, 250 e 1000 o que daria uma precisão de 10ms, 4ms e 1ms respectivamente.

O advento dos temporizadores de alta resolução, *High Resolution Timers* (52), dissociou a manipulação dos temporizadores do relógio do sistema. Em vez disso, os temporizadores de alta resolução medem o tempo diretamente da fonte de relógio do hardware que monoliticamente aumenta a contagem dos nanossegundos, permitindo uma resolução da ordem de microssegundos (53). Além de permitir, também, a funcionalidade do *kernel tickless*, ou seja, quando uma CPU fica em estado ocioso, o arcabouço do temporizador avalia o momento do próximo evento agendado e, caso o próximo evento esteja num período de tempo mais distante que a próxima interrupção periódica, ele reprograma o relógio por CPU para esse acontecimento futuro. Isso permite que o processador ocioso possa entrar em períodos mais longos de ociosidade sem as desnecessárias interrupções periódicas (67).

5.5 REALIZAÇÃO DOS TESTES

5.5.1 AS APLICAÇÃOES

Foram utilizadas duas aplicações para os testes. Uma do tipo cliente que foi instalada na máquina 1 (transmissora) e outra do tipo servidor que foi instalada na máquina 2 (receptora).

As aplicações do tipo servidor tanto do protocolo HCC/UDT quanto TCP/UDT foram iniciadas na máquina dois em *backgroud*:

```
./appserver_HCC &
./appserver_TCP &
```

Dessa forma, as aplicações ficavam preparadas para receber conexões nas portas 22222 e 22223 respectivamente.

Na máquina transmissora eram iniciadas as aplicações clientes da seguinte maneira:

```
./appclient_HCC 172.16.1.1 22222
```

./appclient_TCP 172.16.1.1 22223

Essas aplicações ativam a transmissão de uma quantidade fixa de dados para o endereço IP e porta especificados utilizando os algoritmos de controle de congestionamento do HCC e TCP respectivamente, implementados no UDT.

Como resultado, a aplicação cliente imprime na tela os dados de cada segundo do experimento, conforme Tabela 5-1 Esses dados foram gravados em arquivo para a posterior análise.

Tabela 5-1 – Exemplo de resultado impresso pela aplicação cliente.

SendRate (Mb/s)	RTT(ms)	PktSndPeriod (us)	Janela	TimeStamp (s)	NAK	Packet Flight	TX Retrans
86,2269	112,644	6,31642	83333	1	0	1254	16
128,53	112,872	5,38858	83333	2	0	1176	16
129,366	112,756	7,01917	83333	3	0	1314	16
128,113	112,305	10,2352	83333	4	0	1274	16
129,108	112,318	12,103	83333	5	0	1319	16
129,014	117,429	12,4453	83333	6	0	1356	16
128,024	117,264	7,00939	83333	7	0	1377	16
129,142	117,029	7,74446	83333	8	0	1287	16
127,733	117,366	9,4953	83333	9	0	1283	16
129,935	117,963	7,96655	83333	10	0	1330	16

Cada cenário foi repetido trinta vezes e os resultados gravados em arquivos. Então, para automatizar a iniciação dos testes foram desenvolvidos scripts que estão no Apêndice A desta dissertação.

Para os testes de equidade foram executados seis cenários diferentes com a combinação de fluxos assim denominados:

FA – Cenário de Fluxos A – cinco fluxos HCC

FB – Cenário de Fluxos B – quatro fluxos HCC e um fluxo TCP

FC – Cenário de Fluxos C – três fluxos HCC e dois TCP

FD – Cenário de Fluxos D – dois fluxos HCC e três fluxos TCP

FE – Cenário de Fluxos E – um fluxo HCC e quatro fluxos TCP

FF – Cenário de Fluxos F – cinco fluxos TCP

Para a análise dos testes de eficiência e equidade intraprotocolo (*Fairness*) foram utilizados os resultados do cenário FA. Para a análise dos testes de equidade com o TCP (*TCP Friendliness)* foram utilizados os cenários de FB a FE. Os resultados do cenário FF serviram de base para a comparação com os testes de equidade com o TCP.

5.5.2 GERAÇÃO DOS ARQUIVOS COM AS ANÁLISES

O primeiro passo na análise dos resultados foi calcular a média do valor da vazão de cada amostra (segundos) de cada cenário do experimento e, gravar em um novo arquivo somente as informações de tempo do experimento (coluna *Timestamp* da Tabela 5-1) e a média da vazão calculada anteriormente para o respectivo tempo (coluna *SendRate* da Tabela 5-1). A partir daí os resultados obtidos nesses arquivos foram utilizados para o cálculo da média, desvio padrão e intervalo de confiança de 95% das amostras.

No caso dos experimentos de equidade com o TCP como foram executados cinco fluxos simultâneos e o protocolo UDT/HCC, por ser mais eficiente, termina primeiro a sua transferência dos dados, é preciso descobrir quando foi o momento do término para somente considerar o período de fluxos simultâneos.

Para isto, foi desenvolvido um script para ler todos os arquivos de média gerados no passo anterior e descobrir dentre os cinco fluxos simultâneos qual deles possuía o menor valor de *TimeStamp* na última linha. Então, nos cálculos realizados somente foram consideradas as amostras até aquele momento.

No apêndice A, estão também os scripts utilizados para automatizar a análise dos resultados.

5.6 RESULTADOS DOS TESTES

Nesta seção serão apresentados os resultados obtidos nos testes realizados para avaliação do protocolo UDT utilizando o algoritmo de controle homeostático, HCC.

5.6.1 EFICIÊNCIA EM RELAÇÃO À VAZÃO.

Esse teste foi realizado com intuito de observar o quão eficiente é a implementação do protocolo UDT utilizando o algoritmo de congestionamento HCC.

Primeiramente, observou-se a capacidade máxima de transmissão entre a máquina 1 e a máquina 2 sem ativar o módulo Netem na máquina com função de roteamento. Utilizando-se de um aplicativo de transferência CBR, o Iperf (68), observou-se uma vazão máxima de 652Mbps. Esse valor serviu de referência do valor máximo a ser alcançado pela aplicação. Esse valor é fruto das limitações de hardware e firmware do cenário de testes, uma única

interface de rede na máquina com função de roteamento onde há além do tráfego entre transmissor e receptor, o tráfego reverso, .

Depois, foram iniciadas diversas baterias de testes, com cinco fluxos simultâneos, modificando-se os valores de atraso e perda no Netem. Os experimentos foram repetidos trinta vezes e foram calculados a média, o desvio padrão e o intervalo de confiança de 95%. A quantidade de fluxos simultâneos foi baseada nos experimentos do UDT nos quais foram utilizados três e quatro fluxos em cenários diferentes (16). Neste trabalho foi utilizado o valor de cinco fluxos para que se pudesse obsevar ainda mais a divisão de largura de banda entre os diferentes fluxos e também para que o valor não fosse grande demais a ponto de interferir nos resultados, dadas a limitações do *hardware* utilizado nos experimentos.

Na Tabela 5-2, pode-se observar que o protocolo utilizando o controle de congestionamento HCC mantém uma eficiência de utilização da largura banda disponível de aproximadamente 98% do valor máximo independentemente dos valores de atraso ou da porcentagem de perda de pacotes. Isso demonstra que o algoritmo, por ser baseado em taxa, é realmente mais eficiente nos cenários onde o PLBA é alto, mesmo havendo perdas.

Tabela 5-2 - Resultado dos testes de Eficiência do HCC/UDT

Atraso (ms)	Perda(%)	Total (Mbps)	IE
50	1,00E-04	640,19	98,19%
50	1,00E-03	639,78	98,13%
50	1,00E-02	639,96	98,15%
50	1,00E-01	641,96	98,46%
100	1,00E-04	639,25	98,04%
100	1,00E-03	639,91	98,15%
100	1,00E-02	639,18	98,03%
100	1,00E-01	642,15	98,49%
150	1,00E-04	638,85	97,98%
150	1,00E-03	639,07	98,02%
150	1,00E-02	638,9	97,99%
150	1,00E-01	638,78	97,97%
200	1,00E-04	639,01	98,01%
200	1,00E-03	638,87	97,99%
200	1,00E-02	638,96	98,00%
200	1,00E-01	639,38	98,06%

Se comparado ao TCP/UDT, a eficiência do HCC/UDT é bastante superior. Na Tabela 5-3 têm-se os resultados do teste de eficiência para o TCP/UDT para os mesmos cenários de

atraso e taxa de perda de pacotes utilizados no teste de eficiência do HCC/UDT. Na Figura 5-3, são mostrados os índices de eficiência tanto para o HCC/UDT quanto para o TCP/UDT.

Pode-se observar que o TCP/UDT ainda tem um desempenho bom para os cenários onde o atraso não é muito alto e as taxas de perdas são mais baixas, apesar de menor que o HCC/UDT. Contudo, à medida que esses valores aumentam o desempenho do UDT/TCP diminui consideravelmente.

Tabela 5-3 - Resultado dos testes de Eficiência do TCP/UDT

Atraso(ms)	Perda(%)	Vazão (Mbps)	IE
50	1,00E-04	588,69	90,29%
50	1,00E-03	602,42	92,40%
50	1,00E-02	606,61	93,04%
50	1,00E-01	204,01	31,29%
100	1,00E-04	591,48	90,72%
100	1,00E-03	576,75	88,46%
100	1,00E-02	419,28	64,31%
100	1,00E-01	154,46	23,69%
150	1,00E-04	277,54	42,57%
150	1,00E-03	258,89	39,71%
150	1,00E-02	175,55	26,92%
150	1,00E-01	60,8	9,33%
200	1,00E-04	210,26	32,25%
200	1,00E-03	203,09	31,15%
200	1,00E-02	166,96	25,61%
200	1,00E-01	75,76	11,62%

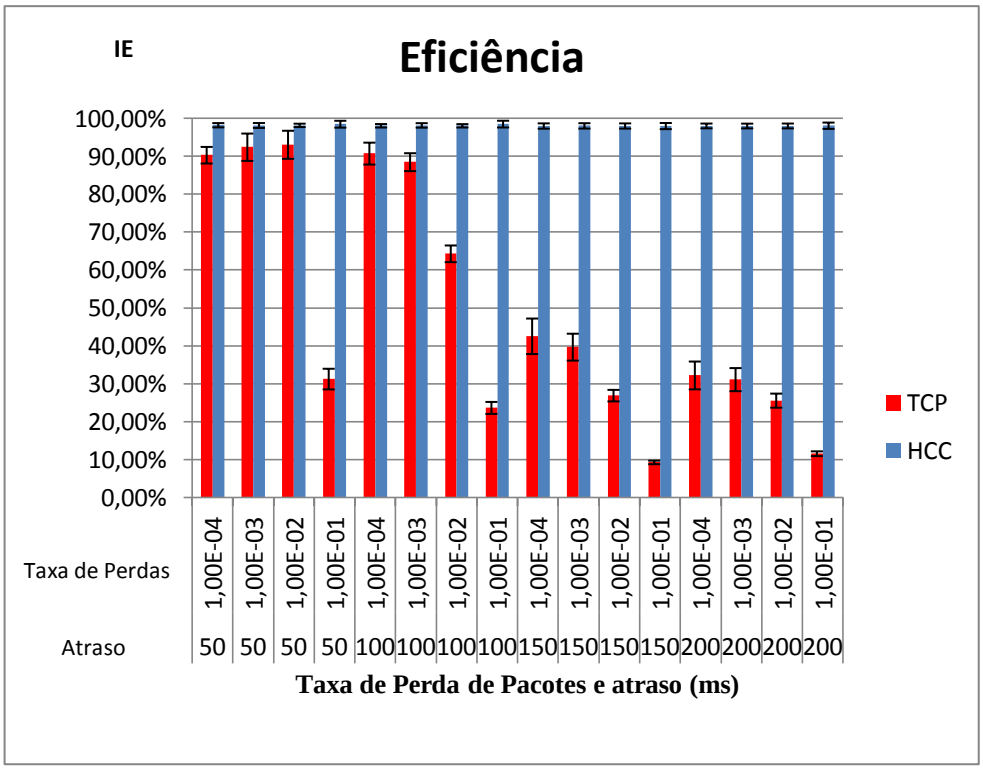

Figura 5-3 – Índice de Eficiência do HCC/UDT e TCP/UDT

5.6.2 EQUIDADE ENTRE DIFERENTES FLUXOS UTILIZANDO O HCC

Nesse cenário, da mesma forma que no teste anterior, foram realizados experimentos com cinco fluxos simultâneos HCC/UDT para diversos valores de atraso e perda. A partir do índice de Jain, foi observado o quão a banda foi distribuída equitativamente entre todos os fluxos. Quanto mais próximo de um for o índice de Jain melhor a equidade.

A Figura 5-4 e a Tabela 5-4 mostram que independentemente do valor do atraso e do valor da perda, o índice de Jain, IJ, é muito próximo de um, o que demonstra que o protocolo HCC/UDT compartilha a largura de banda disponível igualitariamente entre todos os fluxos.

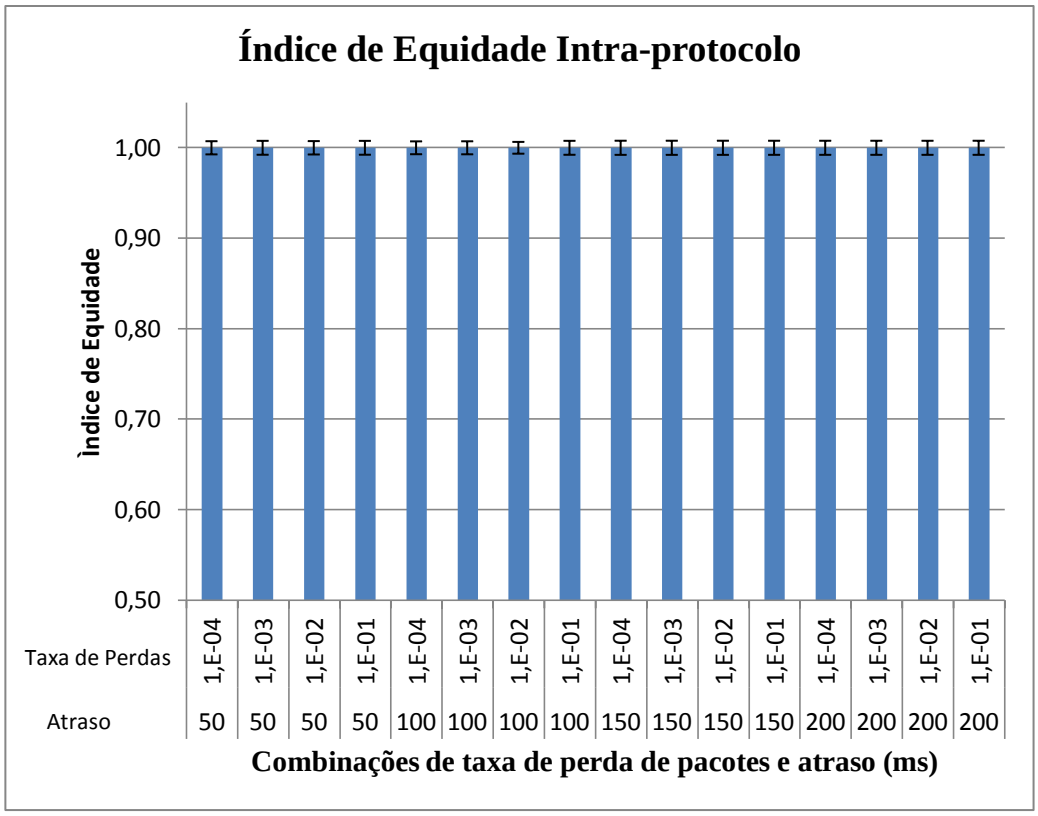

Figura 5-4 – Índice de equidade intraprotocolo

Observa-se também que o desvio-padrão é pequeno em relação aos valores médios de vazão.

Tabela 5-4 - Resultado dos testes de equidade intraprotocolo com os valores das médias calculadas de cada fluxo e o desvio-padrão.

Atraso (ms)	Perda (%)	HCC1		HCC2		HCC3		HCC4		HCC5		IJ
		Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	
50	1,00E-04	128,02	2,35	128,37	4,04	127,97	3,09	127,87	5,29	127,96	3,57	1,00
50	1,00E-03	127,84	4,30	128,37	4,29	127,86	3,96	127,83	4,91	127,88	3,31	1,00
50	1,00E-02	127,89	3,73	128,28	2,59	127,97	1,99	127,88	3,66	127,94	2,52	1,00
50	1,00E-01	128,27	6,34	128,56	4,87	128,44	6,38	128,24	6,69	128,45	5,18	1,00
100	1,00E-04	127,78	4,14	127,81	2,58	127,89	3,15	127,76	3,53	128,01	1,38	1,00
100	1,00E-03	127,96	5,95	128,04	2,91	128,07	3,35	127,88	3,96	127,96	2,92	1,00
100	1,00E-02	127,87	2,75	127,70	5,37	127,87	1,77	127,87	2,68	127,87	1,92	1,00
100	1,00E-01	128,44	5,27	128,36	6,45	128,41	6,57	128,53	5,34	128,41	5,38	1,00
150	1,00E-04	127,82	4,70	127,76	4,79	127,75	4,36	127,73	4,35	127,79	4,66	1,00
150	1,00E-03	127,85	4,84	127,78	4,60	127,8	4,52	127,86	4,62	127,78	4,42	1,00
150	1,00E-02	127,84	4,86	127,80	4,27	127,76	4,40	127,8	4,47	127,70	4,27	1,00
150	1,00E-01	127,77	5,51	127,69	5,62	127,78	4,68	127,92	5,04	127,62	5,52	1,00
200	1,00E-04	127,81	4,03	127,76	4,50	127,82	3,81	127,82	3,67	127,80	4,32	1,00
200	1,00E-03	127,80	3,77	127,76	4,58	127,73	3,95	127,8	3,93	127,78	4,16	1,00
200	1,00E-02	127,80	4,23	127,80	4,36	127,77	4,22	127,82	3,46	127,77	4,53	1,00
200	1,00E-01	127,77	5,06	127,70	6,11	128,03	5,34	127,97	4,94	127,91	5,57	1,00

5.6.3 EQUIDADE COM O TCP

Neste teste foi observado quão amigável o protocolo HCC/UDT é do TCP/UDT. Uma consideração importante a ser feita é o fato de o TCP padrão ser reconhecidamente ineficiente em enlaces com alto PLBA. Portanto, também não é razoável limitar a vazão do novo protocolo à vazão do TCP. Assim, foi feita a comparação do desempenho do TCP/UDT quando há compartilhamento da largura de banda com o HCC/UDT com o desempenho quando há somente fluxos TCP/UDT.

Nas tabelas e figuras a seguir são mostrados os resultados para diferentes cenários em relação à quantidade de fluxos com HCC e TCP e também aos valores de atraso e percentagem de perda. O valor de referência foi o valor médio da vazão quando há somente fluxos com TCP, representado nas tabelas pela coluna “Media TCP”,

Na Tabela 5-5 são mostrados os resultados do cenário FB, composto por quatro fluxos HCC e um fluxo TCP. É apresentada a vazão do fluxo TCP/UDT quando há compartilhamento da largura de banda com os outros quatro fluxos HCC (coluna TCP1) e, também, os valores de vazão quando há somente fluxos TCP.

Tabela 5-5 - Resultado dos testes de equidade com o TCP no cenário FB, com quatro Fluxos HCC e um Fluxo TCP

Atraso (ms)	Perda (%)	HCC1		HCC2		HCC3		HCC4		TCP1		Media TCP	IT
		Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro		
50	1,00E-04	122,63	3,09	123,07	2,50	122,72	1,85	122,75	2,65	136,05	5,01	117,74	1,54
50	1,00E-03	122,97	3,69	123,28	0,52	123,19	1,02	123,24	1,18	134,24	4,09	120,48	1,51
50	1,00E-02	123,01	1,45	123,05	0,64	122,89	2,18	122,79	3,83	135,77	4,66	121,32	1,52
50	1,00E-01	135,62	6,18	136,00	5,24	135,84	6,24	135,65	6,84	93,02	20,88	40,80	8,31
100	1,00E-04	133,92	7,96	133,03	8,49	133,18	8,68	133,04	10,15	103,81	11,33	118,30	1,42
100	1,00E-03	129,28	9,27	129,35	8,44	129,31	7,90	129,33	7,65	116,27	13,23	115,35	1,83
100	1,00E-02	130,99	4,90	130,97	4,13	130,93	4,66	130,84	4,88	106,83	8,41	83,86	3,14
100	1,00E-01	146,36	9,34	146,87	14,12	146,85	12,58	146,58	11,75	50,25	17,08	30,89	8,53
150	1,00E-04	154,63	6,24	154,66	6,44	154,74	6,16	154,63	6,41	17,99	10,18	55,51	0,93
150	1,00E-03	154,70	6,52	154,72	6,22	154,76	6,28	154,67	6.34	17,69	9,94	51,78	1,24
150	1,00E-02	155,80	6,20	155,76	6,26	155,75	5,57	155,77	6,14	13,67	6,32	35,11	2,07
150	1,00E-01	158,29	8,57	158,21	8,45	158,76	8,78	158,35	7,71	6,33	1,88	12,16	1,51
200	1,00E-04	156,83	5,58	156,87	5,84	156,86	5,78	156,83	5,15	10,38	5,75	42,05	0,92
200	1,00E-03	156,94	5,74	156,87	5,88	156,92	5,99	156,84	5,36	10,32	5,69	40,62	1,13
200	1,00E-02	157,48	6,07	157,37	6,37	157,36	6,46	157,33	5,66	8,66	4,19	33,39	1,53
200	1,00E-01	154,01	7,42	154,54	7,54	154,37	7,80	154,74	7,23	3,56	1,00	18,39	1,44

Na análise dos resultados duas observações precisam ser feitas: a primeira é o que o TCP é reconhecidamente ineficiente para esse tipo de rede (mesmo no melhor caso há uma taxa de perda 1E-4) e, somente foi considerado na análise o período em que todos os fluxos eram simultâneos. Significa dizer que o HCC/UDT terminou a sua transferência sem que o TCP/UDT tenha conseguido atingir sua capacidade máxima.

Na Figura 5-5 é mostrado o resultado do índice de equidade com o TCP para o cenário FB, calculado a partir dos valores normalizados das vazões, já que, como explicado anteriormente, não se pode comparar os valores diretamente uma vez que o HCC termina primeiro a sua transmissão.

A normalização foi feita dividindo-se a média da vazão pelo tempo de observação, tanto no caso de haver compartilhamento quanto no caso de todos os fluxos serem TCP.

Pode ser visualizado que o HCC é bastante amigável ao TCP em quase todos os cenários de atraso e taxa de perda. Lembrando que o ideal é o valor do índice igual a um e que acima de um, o protocolo é bastante amigável e menor que um o protocolo é agressivo. Somente nos casos de taxa perda 10^{-4} e atrasos 150 e 200 ms, o índice foi menor que um, porém muito próximo, 0,93 e 0,92, respectivamente.

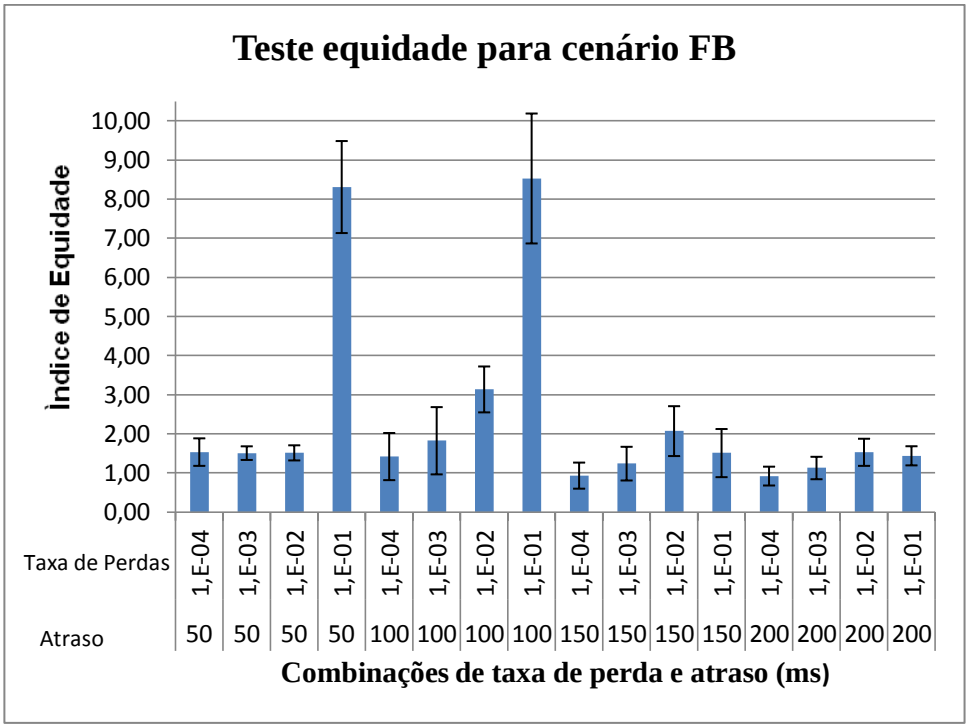

Figura 5-5 – Teste de equidade para o cenário de fluxo B

Em relação ao fato do HCC/UDT terminar a sua transferência de dados mais rapidamente que o TCP/UDT, fica claro, por exemplo, nas Figura 5-6 e Figura 5-7 onde há atraso de 150ms e a taxa de perda de 10^{-4}. Pode ser visto na Figura 5-7 que o TCP necessita mais de 400 segundos para atingir a vazão máxima. E, o HCC termina sua transmissão em menos de 160 segundos, conforme pode ser visto na Figura 5-6. Por isto, quando se calcula a média do TCP durante os 160 segundos ela é muito mais baixa do que quando é levado em consideração todos os 400 segundos.

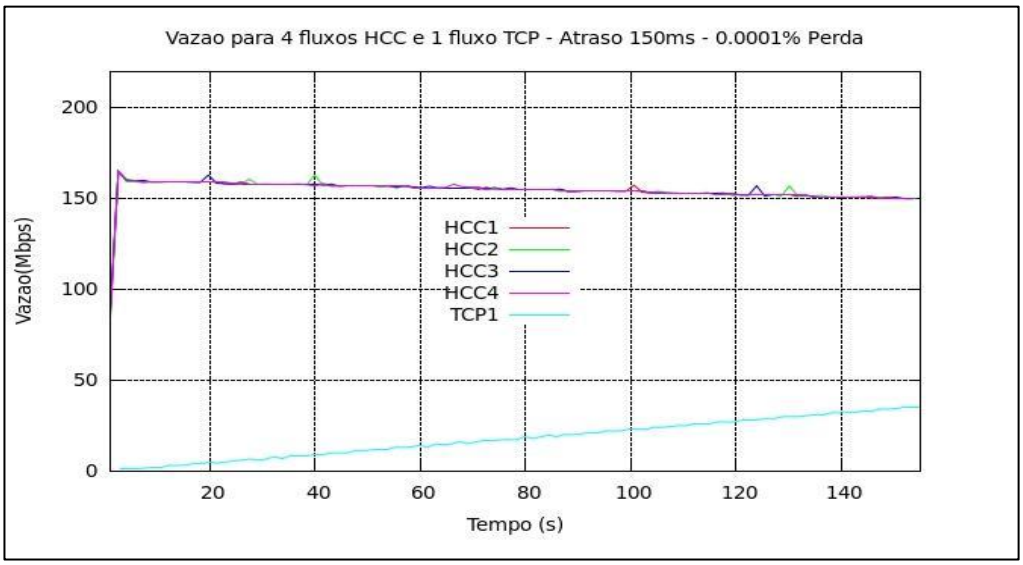

Figura 5-6 – Fluxos FB com 150ms de atraso e taxa de perda de 1E-4

Observa-se também na Figura 5-6 que o HCC vai diminuindo sua taxa de transmissão à medida que o TCP vai aumentando a sua. O que é mais um indício de boa equidade com o TCP.

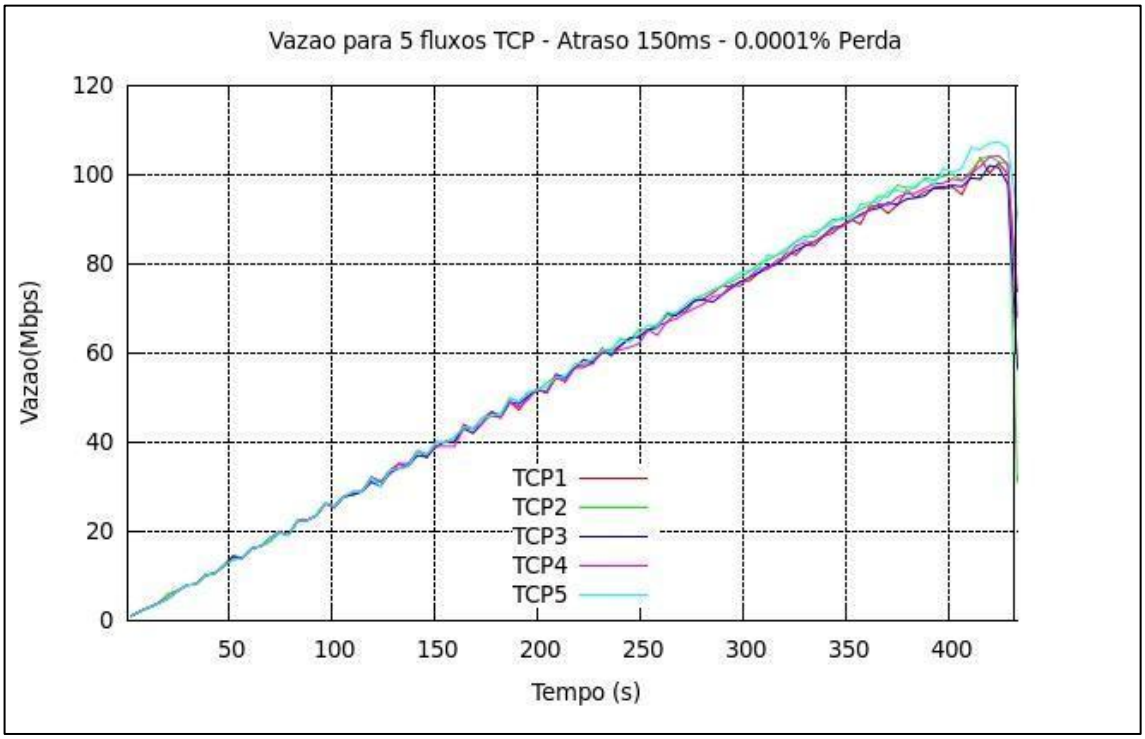

Figura 5-7 – Fluxos FF com 150ms de atraso e taxa de perda de 1E-4

Na Figura 5-8, tem-se o cenário com o fluxo FD e os mesmos valores de atraso e perda que na Figura 5-6. Aqui se observa um comportamento similar. Porém, como a maior parte de largura de banda é dividida somente por dois fluxos HCC, eles atingem uma vazão que é praticamente o dobro do valor do fluxo HCC do cenário FB, em torno de 300 Mbps. Da mesma forma, a vazão dos fluxos HCC vai diminuindo à medida que a vazão dos fluxos TCP vai aumentando.

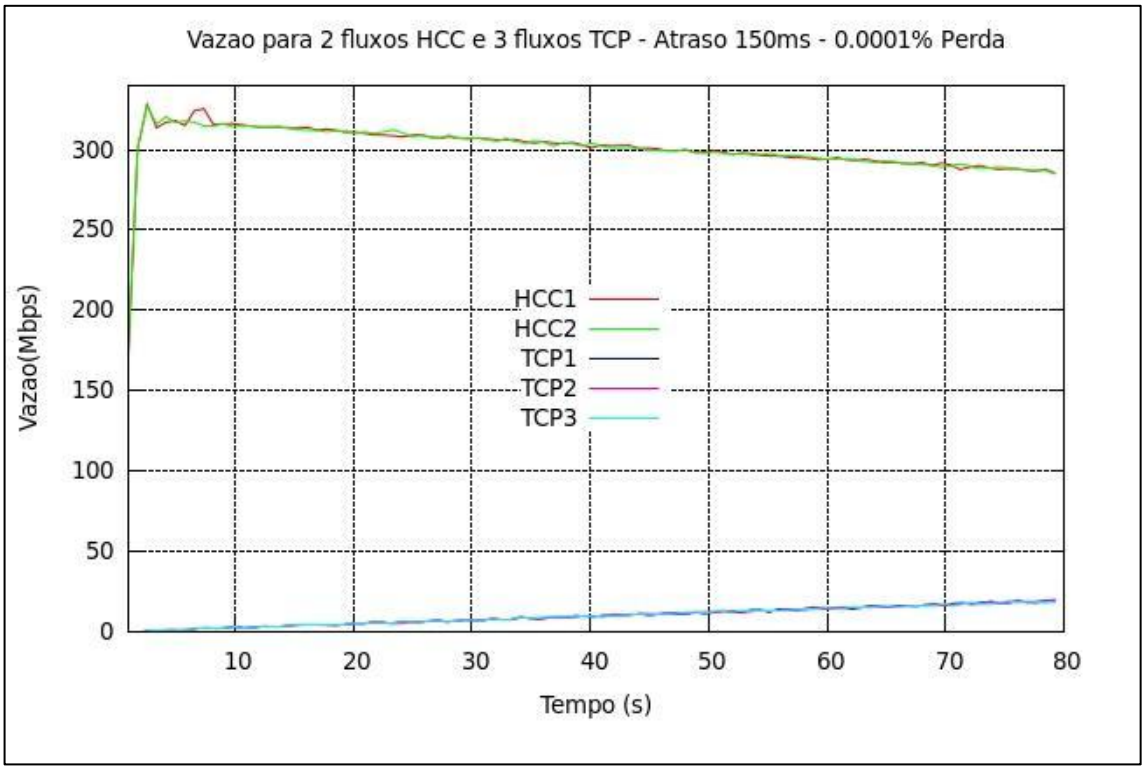

Figura 5-8 - Fluxos FD com 150ms de atraso e taxa de perda de 1E-4

Na Figura 5-9 e Figura 5-10 são mostrados os resultados do índice IT para os cenários FC e FD. Também nesses casos somente para os cenários de taxa perda de 10^{-4} e atrasos 150 e 200 ms, o índice foi menor que um, porém muito próximo.

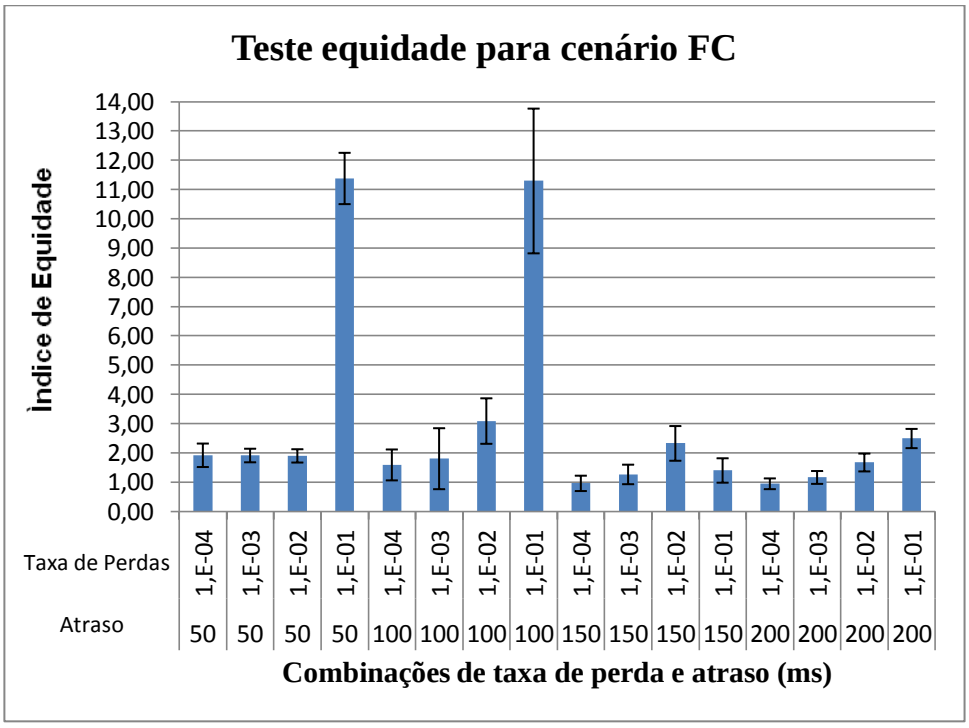

Figura 5-9 – Teste de Equidade para o cenário FC

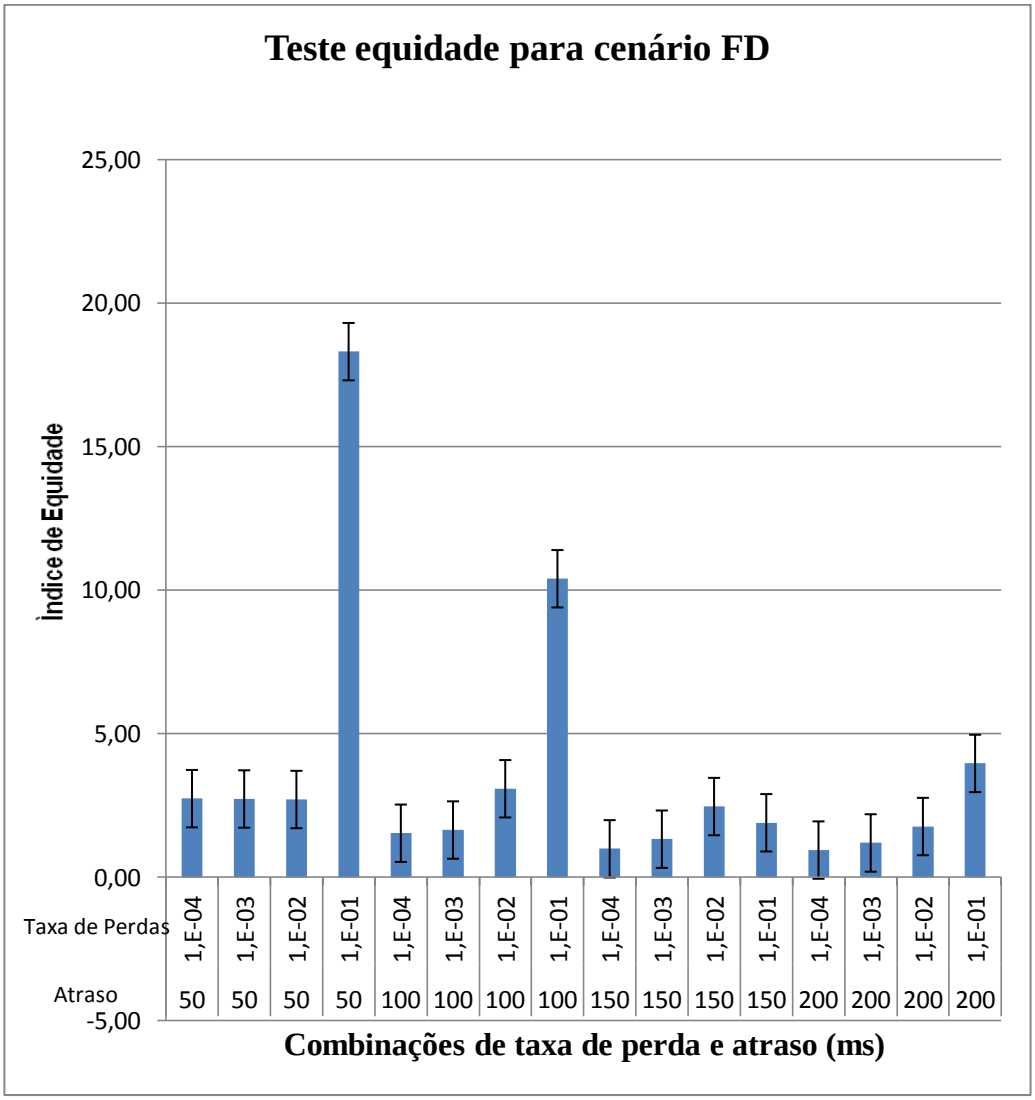

Figura 5-10 - Teste equidade para o cenário FD

As

Tabela 5-6 e 5-7 mostram os resultados de vazão dos cenários FC e FD

Tabela 5-6 - Resultado dos testes de equidade com o TCP no cenário FC, com três Fluxos HCC e dois fluxos TCP

Atraso (ms)	Perda (%)	HCC1		HCC2		HCC3		TCP1		TCP2		IT
		Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	
50	1,00E-04	117,93	3,04	118,41	1,30	118,40	1,28	130,61	5,78	130,59	5,66	1,92
50	1,00E-03	117,94	2,99	118,29	1,04	118,25	1,87	130,73	5,41	130,37	5,48	1,91
50	1,00E-02	118,11	1,27	118,46	2,48	117,92	3,6	130,43	5,48	130,76	5,49	1,90
50	1,00E-01	144,81	12,22	145,15	9,91	145,14	10,91	96,87	17,22	98,73	13,89	11,38
100	1,00E-04	130,78	22,90	130,81	23,00	130,70	22,18	114,97	9,71	122,19	10,17	1,59
100	1,00E-03	128,90	22,10	129,21	21,96	129,28	22,03	124,71	18,31	110,94	13,76	1,80
100	1,00E-02	134,32	14,80	134,42	14,49	134,41	14,61	114,14	12,11	101,78	10,18	3,09
100	1,00E-01	167,54	15,47	166,87	15,09	166,88	13,46	71,05	27,71	52,18	23,17	11,29
150	1,00E-04	201,82	10,12	201,74	9,94	201,83	9,6	14,18	8,00	14,20	8,03	0,96
150	1,00E-03	202,06	10,04	202,03	9,74	202,01	9,64	13,74	7,61	13,97	7,80	1,27
150	1,00E-02	203,71	9,35	203,55	9,70	203,69	9,22	11,94	5,83	11,62	5,93	2,33
150	1,00E-01	201,21	10,24	201,18	10,47	201,08	10,68	4,28	1,17	4,79	1,38	1,40
200	1,00E-04	206,47	8,56	206,4	8,16	206,53	8,72	8,05	4,40	8,09	4,38	0,95
200	1,00E-03	206,51	8,24	206,46	8,58	206,67	9,08	8,05	4,35	8,06	4,37	1,16
200	1,00E-02	204,28	9,87	204,72	9,86	204,60	10,34	7,50	3,82	7,23	3,55	1,67
200	1,00E-01	167,77	8,78	166,83	8,47	167,34	8,69	3,96	1,24	4,01	1,45	2,49

Tabela 5-7 - Resultado dos testes de equidade com o TCP no cenário FD, com dois Fluxos HCC e três fluxos TCP

Atraso (ms)	Perda (%)	HCC1		HCC2		TCP1		TCP2		TCP3		IT
		Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	
50	1,00E-04	112,28	1,11	112,19	2,97	125,44	6,60	125,51	6,71	125,49	6,62	2,74
50	1,00E-03	112,05	3,57	112,88	2,45	125,71	6,74	125,55	6,83	125,04	6,76	2,73
50	1,00E-02	112,81	1,68	113,03	2,45	125,58	6,84	125,14	6,99	125,61	6,93	2,71
50	1,00E-01	152,21	17,41	152,80	17,02	103,65	16,84	116,05	10,09	97,83	13,37	18,32
100	1,00E-04	134,13	42,00	134,45	45,15	107,45	9,29	124,77	10,43	117,34	8,99	1,54
100	1,00E-03	140,36	42,62	140,66	43,80	108,48	9,55	108,34	10,32	111,84	10,95	1,65
100	1,00E-02	137,48	33,55	137,54	33,79	114,83	16,82	113,61	13,01	98,71	11,56	3,09
100	1,00E-01	193,44	27,13	191,87	27,18	48,07	16,60	64,02	31,15	66,01	32,82	10,41
150	1,00E-04	300,66	17,92	300,60	17,26	9,91	5,48	9,88	5,45	9,90	5,42	1,00
150	1,00E-03	300,88	17,55	300,81	17,01	9,95	5,45	9,73	5,31	9,72	5,20	1,33
150	1,00E-02	284,87	15,54	283,79	16,58	9,42	4,68	8,83	4,26	8,86	4,19	2,47
150	1,00E-01	219,70	14,10	220,06	14,19	3,83	1,22	4,19	1,26	4,41	1,52	1,90
200	1,00E-04	307,37	16,47	307,38	15,94	5,47	2,94	5,49	2,95	5,43	2,87	0,95
200	1,00E-03	299,72	16,11	299,67	15,70	5,83	3,13	5,86	3,19	5,74	3,02	1,20
200	1,00E-02	241,00	16,07	238,99	12,88	6,71	3,42	6,79	3,54	6,56	3,46	1,77
200	1,00E-01	173,34	8,62	172,91	8,28	4,34	1,78	4,74	1,87	4,69	1,86	3,97

Na Figura 5-11 e na Tabela 5-8, respectivamente, são apresentados os resultados para o cenário FE, com somente um fluxo HCC e quatro fluxos TCP.

Observa-se analisando todas as tabelas que à medida que há mais fluxos TCP piora a utilização da largura de banda disponível..

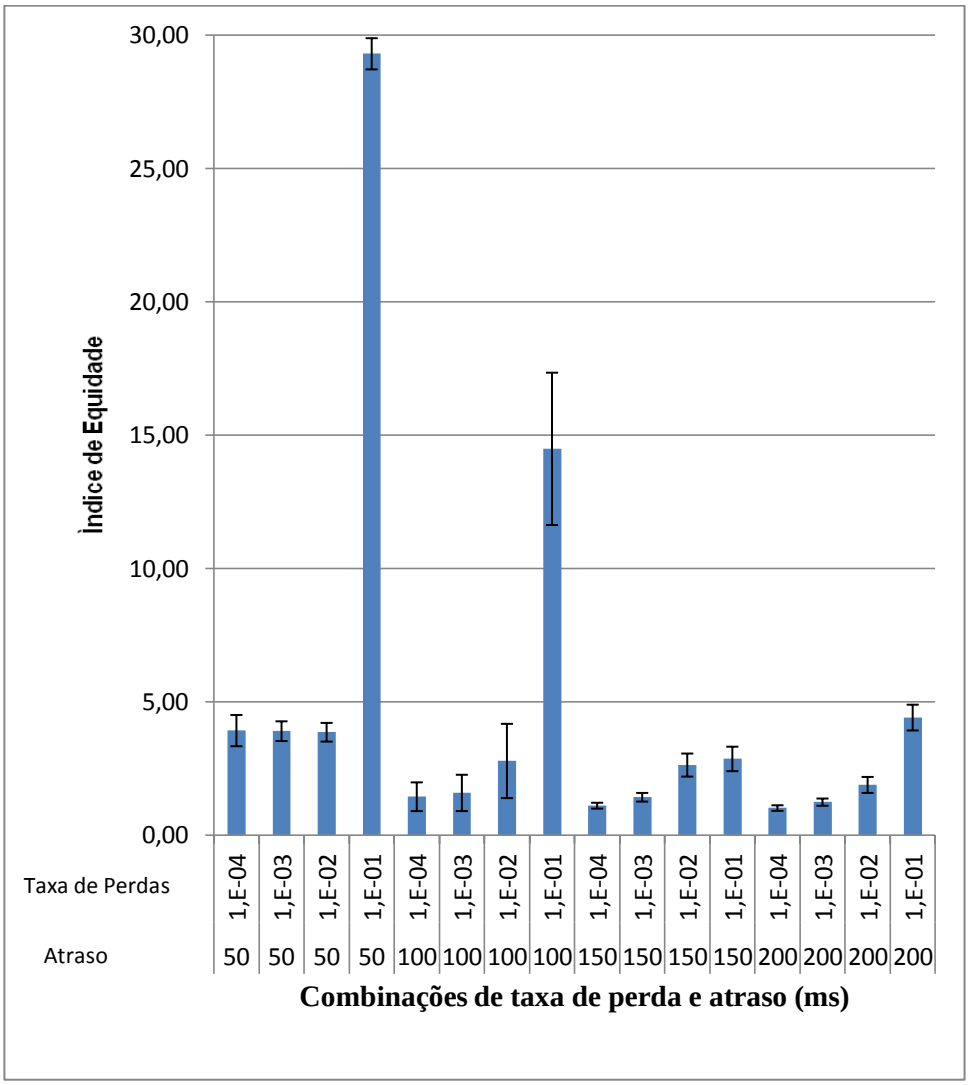

Figura 5-11 - Teste equidade para o cenário FE

Tabela 5-8 - Resultado dos testes de equidade com o TCP no cenário FE, com um Fluxo HCC e quatro fluxos TCP.

Atraso (ms)	Perda (%)	HCC1		TCP1		TCP2		TCP3		TCP4		IT
		Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	Vazão	Erro	
50	1,00E-04	111,45	1,91	120,43	8,17	121,15	8,67	117,98	8,97	115,74	7,60	3,92
50	1,00E-03	112,85	3,63	117,49	8,73	119,6	8,76	119,42	8,97	117,98	8,27	3,90
50	1,00E-02	111,72	1,96	114,81	7,83	120,47	8,52	118,96	8,84	118,62	8,64	3,86
50	1,00E-01	162,55	20,25	102,6	7,77	125,4	10,57	100,91	9,39	119,72	13,82	29,30
100	1,00E-04	138,32	58,89	121,5	9,08	97,21	10,76	123,41	11,65	114,48	9,04	1,44
100	1,00E-03	146,14	61,6	107,71	8,73	114,71	10,87	116,88	13,45	96,29	8,86	1,59
100	1,00E-02	157,08	59,27	103,94	21,69	88,25	16,15	108,32	22,06	100,53	19,98	2,78
100	1,00E-01	239,38	39,91	62,45	23,78	71,87	22,68	67,23	39,15	64,02	31,92	14,49
150	1,00E-04	501,54	31,25	6,33	3,34	6,33	3,34	6,33	3,43	6,30	3,33	1,11
150	1,00E-03	465,94	29,87	6,98	3,76	6,88	3,73	6,94	3,71	6,94	3,77	1,42
150	1,00E-02	317,27	17,94	8,84	4,34	8,96	4,54	8,63	4,21	8,49	4,06	2,63
150	1,00E-01	228,79	16,84	3,74	1,32	3,96	1,63	3,78	1,47	3,46	1,19	2,86
200	1,00E-04	379,02	19,84	4,8	2,50	4,78	2,50	477	2,47	4,80	2,50	1,02
200	1,00E-03	357,26	18,46	5,12	2,71	5,12	2,71	5,11	2,67	5,09	2,66	1,24
200	1,00E-02	250,37	18,48	6,85	3,60	6,89	3,64	6,87	3,54	7,01	3,71	1,89
200	1,00E-01	176,32	7,98	5,17	2,30	4,77	1,88	4,61	1,79	4,80	1,95	4,41

5.6.4 CONVERGÊNCIA PARA A BANDA DISPONÍVEL

Neste teste foi observada a capacidade do protocolo UDT/HCC de convergir para a largura de banda disponível.

Para isso foi realizado o experimento com um fluxo HCC num cenário com atraso de 100ms. O teste foi iniciado somente com fluxo HCC e depois de dez segundos do início, quando o fluxo já havia atingindo a sua capacidade máxima de transmissão (em torno de 640 Mbps) , iniciou-se a transmissão de um fluxo CBR que perdurou por vinte segundos. Dessa forma, observou-se o comportamento do UDT/HCC com dez e trinta segundos de experimento, conforme pode ser visualizado na Figura 5-12.

O fluxo CBR foi executado utilizando-se a ferramenta Iperf configurada para uma vazão constante de 200 Mbps durante os vinte segundos.

Nesse caso, percebe-se que o protocolo UDT/HCC converge rapidamente para banda disponível tanto no momento da aplicação do fluxo CBR quando precisa diminuir sua taxa de transmissão do valor da taxa do fluxo CBR, quanto no momento que cessa a aplicação do fluxo CBR, quando o HCC deve convergir novamente para a máxima taxa permitida.

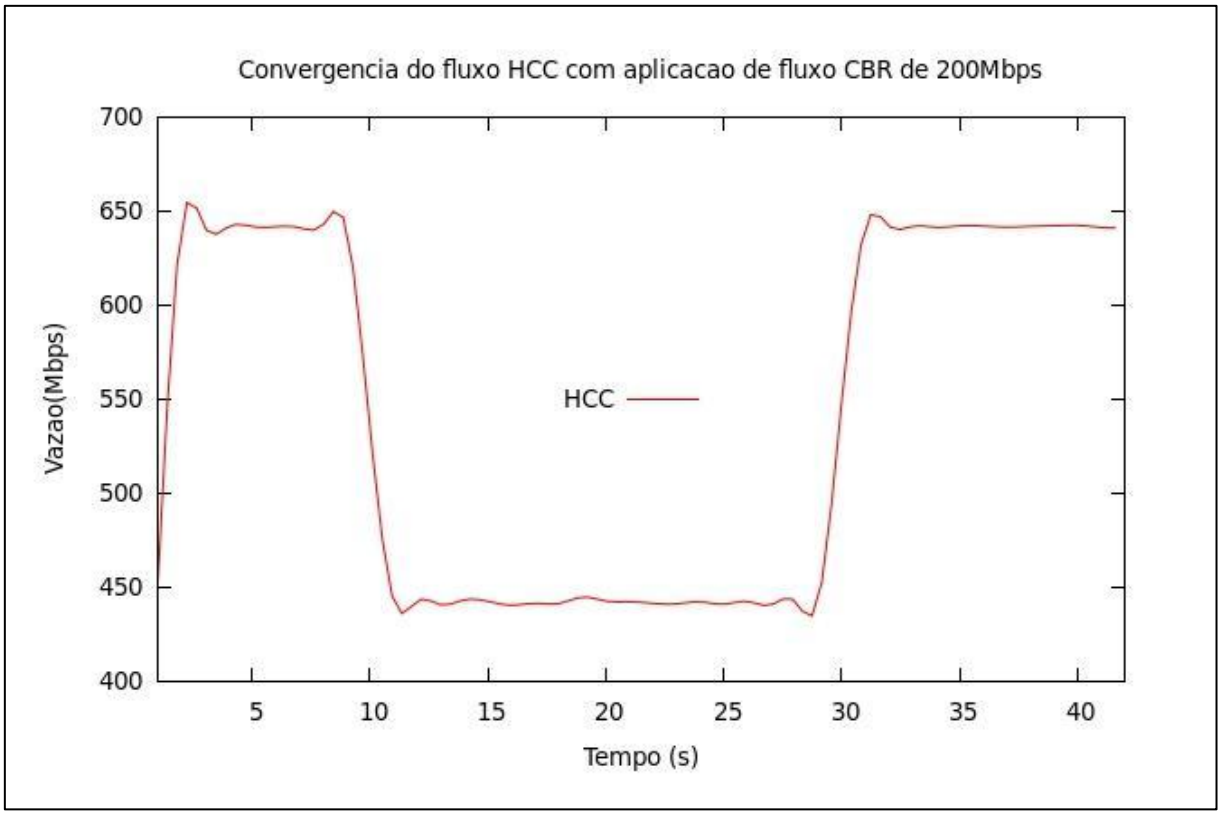

Figura 5-12 – Convergência do HCC

5.6.5 ESTABILIDADE

Uma vez que o HCC convirja é requisitado que não varie demasiadamente em torno do ponto de convergência. Essa medida é feita com o desvio padrão dos valores de vazão coletadas normalizado pelo valor da vazão média dos fluxos.

Pode-se observar nas Figuras Figura 5-13, Figura 5-14 e Figura 5-15 que a variação dos fluxos HCC é pequena, tanto para o cenário com os mais baixos valores de atraso e percentagem de perdas quanto para o cenário com os maiores valores.

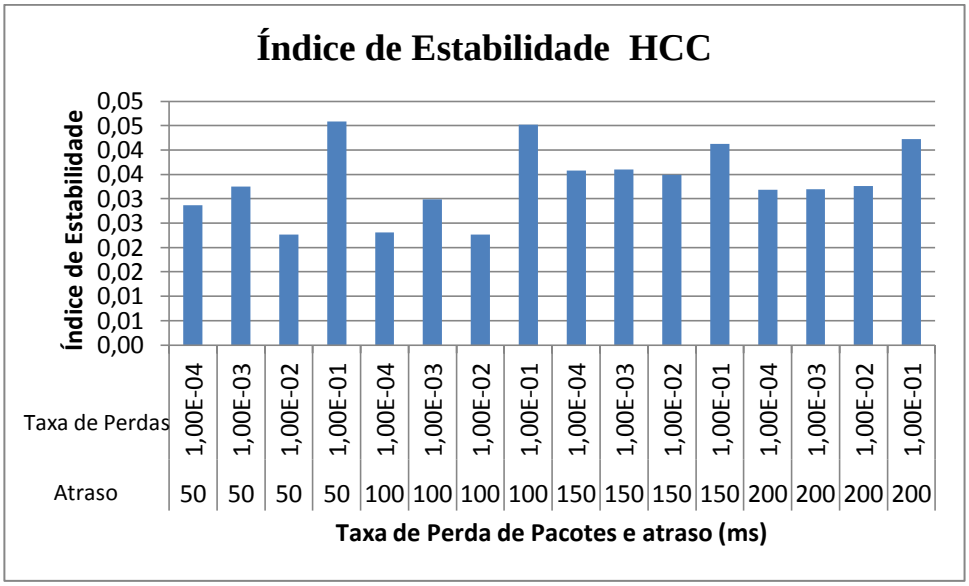

Figura 5-13 – Índice de estabilidade DE para o cenário de Fluxos FA

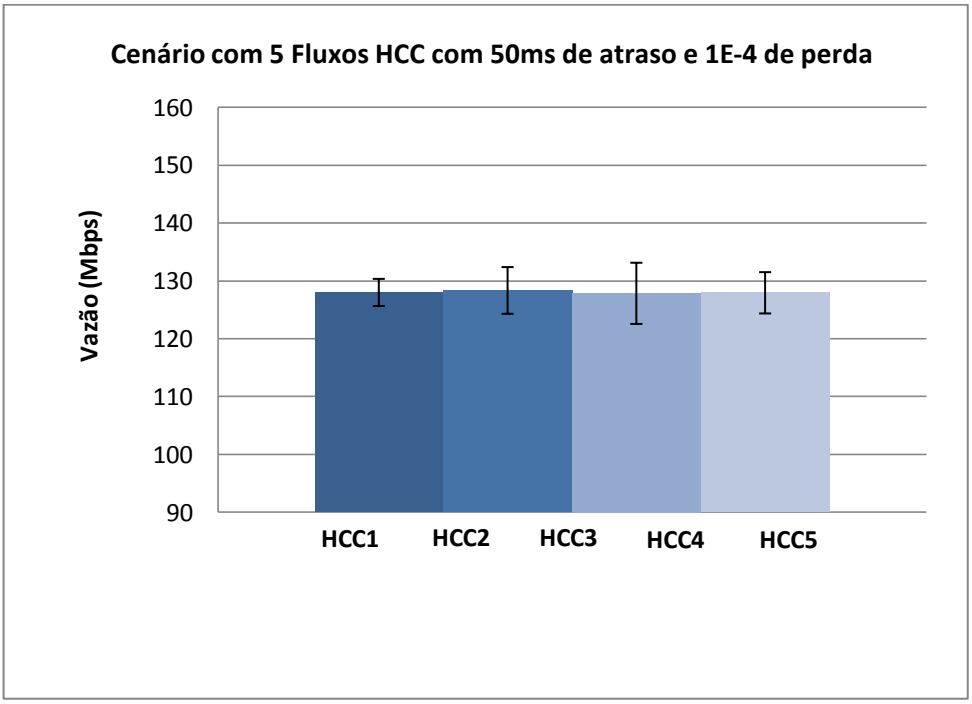

Figura 5-14 - Estabilidade entre fluxos HCC com valores mais baixos de taxa de perda e atraso.

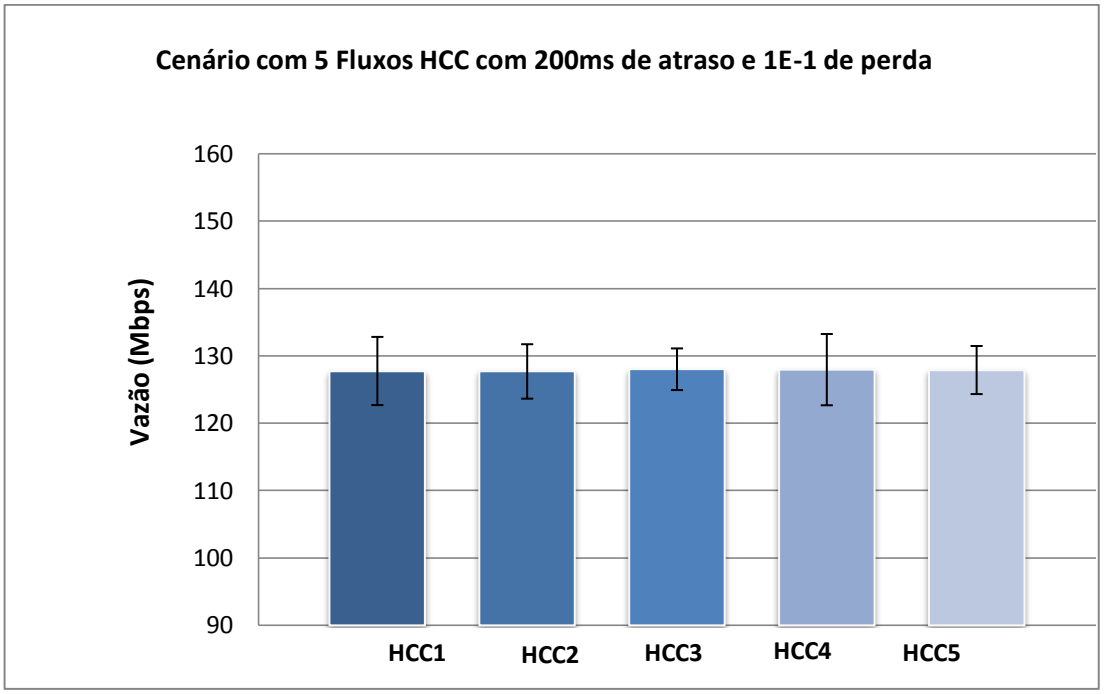

Figura 5-15– Estabilidade entre fluxos HCC com valores mais altos de taxa de perda e atraso.

Já para os cenários em que há compartilhamento dos fluxos do HCC com o TCP, o desvio aumenta, conforme por ser visto na Figura 5-16. Isso ocorre porque o HCC vai acompanhando a variação do TCP, diminuindo a sua taxa de transmissão quando o TCP aumenta e aumentando quando o TCP diminui. Pode ser visualizado que o cenário FE, onde há somente um fluxo com HCC, possui o maior índice para todas as combinações justamente pelo fato de outros quatro fluxos com HCC aproveitarem pouco a banda disponível.

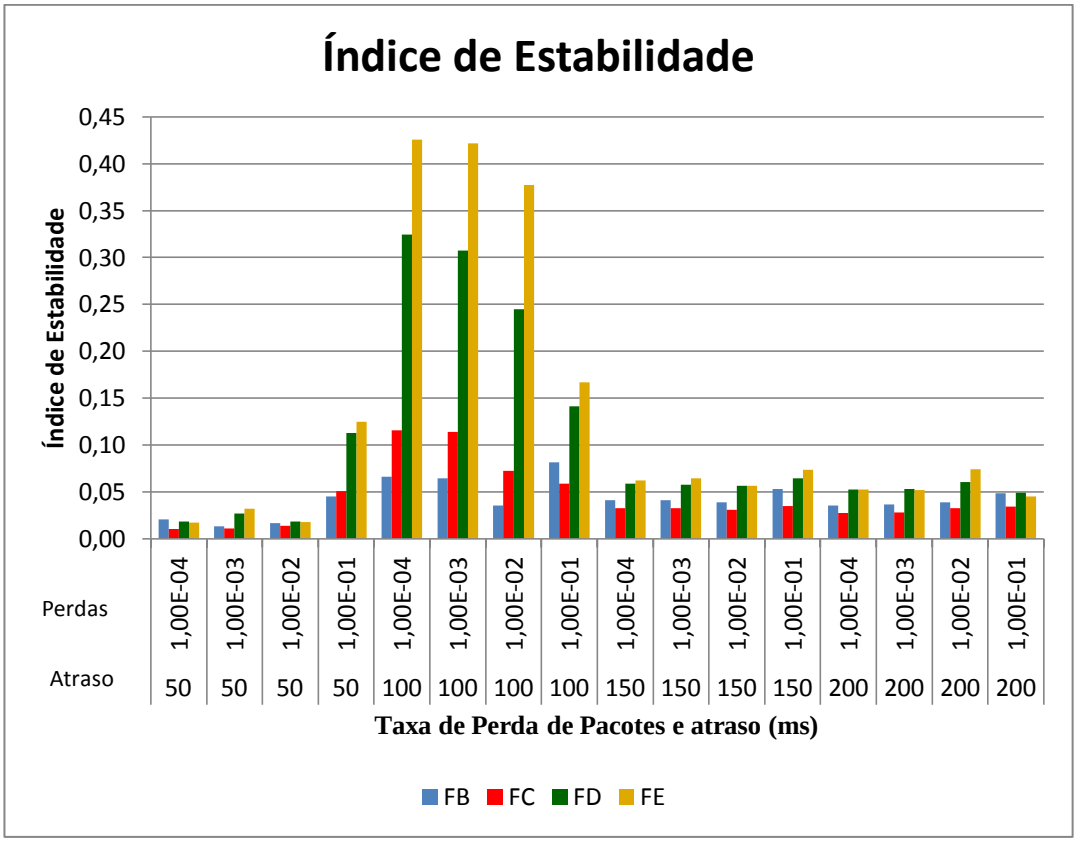

Figura 5-16 - Índice de estabilidade DE para os cenários de Fluxo FB, FC, FD e FE

5.7 SUMÁRIO DO CAPÍTULO 5

Neste capítulo foi apresentado como os testes foram realizados. Os cenários utilizados e quais testes foram realizados para a avaliação do protocolo HCC/UDT em redes com alto PLBA

5.7.1 SUMÁRIO DOS RESULTADOS

Pelos testes realizados, é possível observar que o HCC é um excelente mecanismo de controle de congestionamento para redes de alta velocidade, principalmente para aquelas que apresentam atrasos elevados (como redes que usam enlaces via satélite, ou, em menor importância, redes transcontinentais) e redes que tenham enlaces sujeitos a erros (recentemente, com enlaces de redes sem fio atingindo 400Mbps[1], esta característica torna-se cada vez mais importante).

[1] Redes IEEE 802.11n podem em teoria atingir até 600Mbps usando quatro fluxos espaciais de 150Mbps cada um, no entanto produtos comerciais normalmente trabalham apenas com 300Mbps e poucos são capazes de atingir taxas maiores.

Em termos de eficiência em relação à vazão, o protocolo UDT/HCC apresentou uma taxa em torno de 98% da vazão máxima disponível, valor bem superior à taxa obtida pelo UDT/TCP. Já em relação à equidade intraprotocolo, o resultado também foi excelente. Muito próximo de um, o valor ideal do índice de Jain.

Nos resultados dos testes de equidade com o TCP, dentre todos os sessenta e quatro cenários diferentes em relação à quantidade fluxos HCC e TCP, atraso e taxa de perda, somente em cinco o índice de equidade foi menor que um. E, ainda assim em todos os cinco os valores foram muito próximos de um.

No teste de convergência o UDT/HCC mostrou bom resultado, diminuindo rapidamente a transmissão quando detectou a presença de outro tráfego e convergindo para a vazão máxima após o término da aplicação do tráfego interveniente.

E, por fim, os testes de estabilidade mostraram que o UDT/HCC varia pouco ao redor da média quando há somente fluxos UDT. Quando há fluxos TCP compartilhados o desvio aumenta porque o HCC vai acompanhando a variação do TCP, aumentando a sua taxa de transmissão quando o TCP diminui e diminuindo quando o TCP aumenta.

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi apresentada a avaliação da implementação do protocolo UDT utilizando o algoritmo de controle de congestionamento homeostático, HCC em redes com alto PLBA. O intuito foi verificar se o protocolo com o algoritmo de congestionamento era realmente uma alternativa à utilização do TCP padrão para esses tipos de redes, onde esse é reconhecidamente ineficiente.

No capítulo dois foram vistas as diversas soluções propostas para a ineficiência do TCP padrão. Foram comentadas propostas de modificações no TCP, principalmente no algoritmo de controle de congestionamento, propostas de modificação na estrutura da rede com as informações de congestionamento sendo fornecidas pelos roteadores, e, a implementação do controle de congestionamento na camada de aplicação e utilização do UDP como protocolo de transporte.

No capítulo três, foi mostrado o mecanismo de controle de congestionamento baseado em taxa, que tem desempenho melhor que o TCP padrão para redes de alta velocidade e, também, a utilização do *jitter* como indício de congestionamento. Esses dois mecanismos compõem a homeostasia do algoritmo HCC, com o par de pacotes tendendo a superestimar a largura de banda disponível e a utilização do *jitter* tendendo a corrigir esse valor.

No capítulo quatro foi mostrado como o algoritmo de congestionamento homeostático foi implementado no arcabouço do UDT. A arquitetura do arcabouço bem como algumas características também forma apresentadas para esclarecer detalhes de implementação que podem interferir nos resultados dos testes realizados como o gerenciamento das informações de perda, prevenção de cópia na memória, temporizadores de alta resolução e projeto e implementação da API.

E, no capítulo cinco, foram apresentados quais os testes que foram realizados e como eles foram realizados.

Foi observado nos testes realizados que a implementação do HCC/UDT tem bom desempenho nas redes com alto PLBA mesmo quando os valores de perda e atraso são elevados. Isso mostra que o algoritmo é realmente uma alternativa ao TCP.

Outra conclusão foi sobre a facilidade de se usar o UDT para o rápido desenvolvimento do algoritmo não sendo necessário escrever todo o protocolo.

6.1 TRABALHOS FUTUROS

Como trabalho futuro pode-se citar o desenvolvimento de um protocolo utilizando o controle de congestionamento homeostático, HCC, diretamente no núcleo do sistema operacional para a realização dos testes de desempenho. Esta implementação deixará o HCC disponível no sistema operacional para a utilização de qualquer usuário e permitirá a utilização de aplicações que requer altas taxas de transmissão e facilitará a realização de novos experimentos.

Nos testes realizados as retransmissões não foram analisadas. Assim, uma investigação sobre as retransmissões é um trabalho futuro importante. O HCC é apenas um mecanismo de controle de congestionamento e um protocolo completo precisa implementar outros mecanismos como controle de fluxo e confiabilidade via retransmissões, por exemplo. Estudar quais mecanismos de controle de fluxo e confiabilidade ainda mantém as características de regularidade do fluxo necessárias ao bom funcionamento do HCC é um desafio a ser vencido.

Uma investigação que pode ser realizada no futuro é a comparação do HCC/UDT ou de um novo protocolo utilizando o HCC implementado no núcleo com os protocolos CUBIC e *Compound* TCP, já que esses são os mais utilizados nos sistemas operacionais atualmente. E, também, com o próprio mecanismo de controle de congestionamento do UDT.

Outro teste que se pode realizar é a verificação do comportamento do protocolo UDT/HCC sobre enlaces de 10 Gbps ou superiores.

REFERÊNCIAS

1 PROJETO, G. I. G. A. Web Site oficial–Disponível em< http://www. giga. org. br>. **Acesso em**, 15 set. 2012.

2 INTERNET2. Disponivel em: <http://www.internet2.edu/>. **Acesso em,** 15 set. 2012.

3 CANARIE. Disponivel em: <http://www.canarie.ca/en/network/overview>. **Acesso em,** 12 abr. 2012.

4 POSTEL, Jon. RFC 793: **Transmission control protocol**, September 1981.

5 AFANASYEV, Alexander et al. Host-to-host congestion control for TCP.**Communications Surveys & Tutorials, IEEE**, v. 12, n. 3, p. 304-342, 2010..

6 XU, Wenjun et al. Hybrid congestion control for high-speed networks.**Journal of Network and Computer Applications**, v. 34, n. 4, p. 1416-1428, 2011.

7 FLOYD, Sally. RFC 3649. **HighSpeed TCP for large congestion windows**, 2003..

8 YAMAZAKI, Etsushi et al. Fast optical channel recovery in field demonstration of 100-Gbit/s Ethernet over OTN using real-time DSP. **Optics Express**, v. 19, n. 14, p. 13179-13184, 2011..

9 POSTEL, Jon; RFC 768. **User datagram protocol**, p. 1-3, 1980.

10 MAGALHÃES, Luiz Claudio Schara; MONTEIRO, Marcos V.S.; CARRANO, Ricardo C. Homeostatic Congestion Control. **SBRC 2010**, 2010. 393-406.

11 STEWARD, R. RFC 4960. **Stream Control Transmission Protocol**, 2007.

12 KOHLER, Eddie; HANDLEY, Mark; FLOYD, F. RFC 4340: **Datagram congestion control protocol (DCCP)**. 2006.

13 NAGLE, John. RFC 896: **Congestion control in IP/TCP internetworks.**1984..

14 JACOBSON, Van. Congestion avoidance and control. In: **ACM SIGCOMM Computer Communication Review**. ACM, 1988. p. 314-329.

15 PETERSON, Larry L.; DAVIE, Bruce S. **Computer networks: a systems approach**. Morgan Kaufmann, . 3rd Edition. p.379. 2003.

16 GU, Yunhong; GROSSMAN, Robert L. UDT: UDP-based data transfer for high-speed wide area networks. **Computer Networks**, v. 51, n. 7, p. 1777-1799, 2007.

17 KELLY, Tom. Scalable TCP: Improving performance in highspeed wide area networks. **ACM SIGCOMM Computer Communication Review**, v. 33, n. 2, p. 83-91, 2003.

18 LEITH, Douglas; SHORTEN, Robert. H-TCP: TCP for high-speed and long-distance networks. In: **Proceedings of PFLDnet**. 2004..

19 XU, Lisong; HARFOUSH, Khaled; RHEE, Injong. Binary increase congestion control (BIC) for fast long-distance networks. In: **INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies**. IEEE, p. 2514-2524, 2004.

20 HA, Sangtae; RHEE, Injong; XU, Lisong. CUBIC: a new TCP-friendly high-speed TCP variant. **ACM SIGOPS Operating Systems Review**, v. 42, n. 5, p. 64-74, 2008..

21 JIN, Cheng; WEI, David X.; LOW, Steven H. FAST TCP: motivation, architecture, algorithms, performance. In: **INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies**. IEEE, p. 2490-2501, 2004.

22 BRAKMO, L. . P. L. TCP Vegas: end to end congestion avoidance on a global Internet. **IEEE J. Sel. Areas Commun**, 13, n. 8, October 1995. 1465–1480.

23 TAN, K. E. A. A Compound TCP Approach for High-Speed and Long Distance Networks. **INFOCOM 2006**, April 2006.

24 ALLCOCK, William et al. GridFTP: Protocol extensions to FTP for the Grid.**Global Grid ForumGFD-RP**, v. 20, 2003.

25 NABESHIMA, Masayoshi. Performance evaluation of multcp in high-speed wide area networks. **IEICE transactions on communications**, v. 88, n. 1, p. 392-396, 2005.

26 RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. RFC 3168. **The addition of Explicit Congestion Notification (ECN) to IP. The Internet Society**, 2001.

27 KATABI, Dina; HANDLEY, Mark; ROHRS, Charlie. Congestion control for high bandwidth-delay product networks. In: **ACM SIGCOMM Computer Communication Review**. p. 89-102, ACM, 2002.

28 GUERRA, Fabio. Protocolos de Transporte para Redes de Alta Velocidade**:** um estudo comparativo. **[Dissertação de Mestrado]**. Niteroi, UFF , Outubro de 2006.

29 SIMULADOR, NS Network. Disponivel em: <http://www.isi.edu/nsnam/ns>. Acesso em: 28 jul. 2012.

30 CASAGRANDE, G.; GRANELLI, F.; MIORANDI, D. TCPMoon: Monitoring the Diffusion of TCP Congestion Control Variants in the Internet. **2011 IEEE International Conference on Communications (ICC)**, p. 1-6. Kyoto, 5-9 June 2011

31 ALLMAN, M.; PAXSON, V. E. Blanton,**" TCP Congestion Control**. RFC 5681, September, 2009..

32 HA, Sangtae et al. A step toward realistic performance evaluation of high-speed TCP variants. In: **Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)**. 2006.

33 BELHAJ, Salem; TAGINA, Moncef. Vfast tcp: An improvement of fast tcp. In:**Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on**. IEEE, p. 88-93, 2008.

34 DICKENS, Phillip M. FOBS: A lightweight communication protocol for grid computing. In: **Euro-Par 2003 Parallel Processing**. Springer Berlin Heidelberg, p. 938-946, 2003.

35 GU, Yunhong; GROSSMAN, Robert. SABUL: A transport protocol for grid computing. **Journal of Grid Computing**, v. 1, n. 4, p. 377-386, 2003.

36 HE, Eric et al. Reliable blast UDP: Predictable high performance bulk data transfer. In: **Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on**. IEEE, p. 317-324, 2002.

37 WU, Qishi; RAO, Nageswara SV. Protocol for high-speed data transport over dedicated channels. In: **Third International Workshop on Protocols for Long-Distance Networks (PFLDnet 2005), Lyon, France**. 2005.

38 ZHENG, Xuan; MUDAMBI, Anant Padmanath; VEERARAGHAVAN, Malathi. Frtp: Fixed rate transport protocol–a modified version of sabul for end-to-end circuits. **Pathnets2004 on Broadnet**, 2004.

39 BRADEN, Robert et al. Studies of XCP deployment issues. In: **Proc. PFLDNet 2005 Workshop**. 2005.

40 MAGALHÃES, Luiz Claudio Schara; A transport layer approach to host mobility **[Tese de Doutorado]**. University of Illinois at Urbana-Champaign,2005.

41 KESHAV, S. A control-theoretic approach to flow control. **ACM SIGCOMM Computer Communication Review**, v. 25, n. 1, p. 188-201, 1995.

42 DOVROLIS, Constantinos; RAMANATHAN, Parameswaran; MOORE, David. Packet-dispersion techniques and a capacity-estimation methodology.**Networking, IEEE/ACM Transactions on**, v. 12, n. 6, p. 963-977, 2004..

43 LIU, Xiliang et al. Single-hop probing asymptotics in available bandwidth estimation: sample-path analysis. In: **Proceedings of the 4th ACM SIGCOMM conference on Internet measurement**. ACM, p. 300-313, 2004.

44 PARK, Kyung-Joon et al. Feedback-assisted robust estimation of available bandwidth. **Computer Networks**, v. 53, n. 7, p. 896-912, 2009.

45 STRAUSS, Jacob; KATABI, Dina; KAASHOEK, Frans. A measurement study of available bandwidth estimation tools. In: **Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement**. ACM, p. 39-44, 2003.

46 KLEIN, Thierry E.; LEUNG, Kin; ZHENG, Haitao. Enhanced scheduling algorithms for improved TCP performance in wireless IP networks. In: **IEEE Globecom**. p. 2744-2759, 2004.

47 GU, Yunhong; HONG, Xinwei; GROSSMAN, Robert L. Experiences in design and implementation of a high performance transport protocol. In:**Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference**. IEEE, p. 22-22, 2004.

48 GU, Yunhong; GROSSMAN, Robert L. Supporting configurable congestion control in data transport services. In: **Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference**. IEEE, . p. 31-31, 2005.

49 JAIN, Niraj; SCHAWRTZ, M.; BASHKOW, Theodore. Transport protocol processing at GBPS rates. In: **ACM SIGCOMM Computer Communication Review**. ACM, p. 188-199, 1990.

50 RODRIGUES, Steven H.; ANDERSON, Thomas E.; CULLER, David E. High-performance local area communication with fast sockets. In: **Proceedings of the USENIX Technical Conference,** p. 257-274.1997.

51 CHU, Hsiao-keng Jerry. Zero-copy TCP in Solaris. In: **Proceedings of the 1996 annual conference on USENIX Annual Technical Conference**. Usenix Association, p. 21-21,1996.

52 GLEIXNER, Thomas; NIEHAUS, Douglas. Hrtimers and beyond: Transforming the linux time subsystems. In: **Linux Symposium**. p. 333-346, 2006.

53. Internet MontaVista, Real-Time-Guide-R21. Disponivel em: <https://export.writer.zoho.com/public/rreginelli/Chapter-5---High-Resolution-Timers-Final1/fullpage>.**Acesso** em 15 set.2012

54 JOUBERT, Philippe et al. High-performance memory-based web servers: Kernel and user-space performance. In: **Proceedings of the USENIX Annual Technical Conference**. 2001.

55 KERNEL_LINUX_2_6_19 , 2006. Disponivel em: <http://kernelnewbies.org/Linux_2_6_19>. **Acesso** em: 15 set. 2012.

56 KERNEL_LINUX_2_6_8 , 2004. Disponivel em: <http://kernelnewbies.org/Linux_2_6_8>. **Acesso** em: 2012 set. 15.

57 PHILIP, 2008. Disponivel em: <http://www.speedguide.net/articles/windows-7-vista-2008-tweaks-2574>. **Acesso** em: 15 set. 2012.

58 FLOYD, Sally; ALLMAN, Mark. RFC 5333. **Specifying New Congestion Control Algorithms**, 2007.

59 FLOYD, Sally; HENDERSON, Tom; GURTOV, Andrei. RFC 2582. **The NewReno modification to TCP's fast recovery algorithm**, April, 1999..

60 ALLMAN, Mark; PAXSON, Vern; STEVENS, William. RFC 2581: **TCP congestion control**. 1999.

61 FLOYD, Sally. RFC 2914. **Congestion Control Principles**, 2000.

62 JAIN, Raj. The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling. **Wiley- Interscience**, New York, NY, April 1991.

63 HEMMINGER, S. Network Emulation with NetEm. **Proceedings of the 6th Australia's National Linux Conference (LCA2005)**, Canberra, Australia, April 2005.

64 ANDREW, Lachlan et al. Towards a common TCP evaluation suite. In: **Proc. PFLDnet**. 2008.. Towards a Common TCP Evaluation Suite. **in: PFLDnet 2008**, 2008.

65 PLUMMER, David C. RFC 826: **Ethernet Address Resolution Protocol**. November 1982.

66 PLANETLAB. Disponivel em: <http://www.planet-lab.org/>. **Acesso em**: 01 jul. 2012.

67 SIDDHA, Suresh; PALLIPADI, Venkatesh; VEN, A. V. D. Getting maximum mileage out of tickless. In: **Linux Symposium**. p. 201-207, 2007.

68 IPERF. Disponivel em: <http://iperf.sourceforge.net/>. **Acesso em**: 20 set. 2012.

APÊNDICE A – SCRIPTS DE AUTOMATIZAÇÃO

Para facilitar e assegurar a correta iniciação dos testes e também a análise dos resultados foi desenvolvido scripts para a automatização tanto das iniciações dos testes quanto da análise dos resultados.

Este anexo descreve de forma mais detalhada esses scripts desenvolvidos

A.1 INICIAR OS EXPERIMENTOS

Foram desenvolvidas duas aplicações para os testes uma cliente que foi instalada na máquina transmissora e outra do tipo servidor que foi instalada na máquina receptora.

As aplicações do tipo servidor tanto do protocolo utilizando HCC quanto TCP foram iniciadas na máquina receptora:

./appserver_HCC &

./appserver_TCP &

Desta forma, as aplicações ficavam prontas para receber conexões nas portas 22222 e 22223 respectivamente.

Na máquina transmissora eram iniciadas as aplicações clientes.

Ex:

./appclient_HCC 172.16.1.1 22222

./appclient_TCP 172.16.1.1 22223

Estas aplicações ativam a transmissão de dados para o endereço IP e porta especificados.

Como são iniciados cinco fluxos simultâneos em cada rodada de experimento e estas rodadas são repetidas trinta vezes, foram desenvolvidos scripts para cada um dos testes realizados de forma a automatizar o processo.

A seguir é apresentado o *shell* script utilizado para o teste de eficiência

```
#!/bin/bash
#
# Script to run a test 30 times in sequence
# Proccess Name
proccess=appclient
# Destination IP Address
IPADDR=172.16.1.1
# File Name Prefix
Name=mtu1500_delay_100ms_
# Date
Date=`date +%F_%R_`
# Directory
Dir=/home/mvsmonteiro/udt4.7a
# Auxiliar Variable
i=0
# Eficiency HCC 1 flow
  while [ $i -le 30 ] ; do
          num_procs=`ps ax | grep $proccess | grep -v grep| wc -l`
          if [ $num_procs -eq 0 ] ; then
            $Dir/app/appclient_HCC $IPADDR 22222 > $Dir/Eficiencia_HCC_'$Name$Date$i &
            i=`expr $i + 1`
        else
                sleep 60
       fi
  done.
```

O script executa trinta vezes a aplicação cliente e grava os resultados no diretório especificado com nome do arquivo também especificado.

Para os testes de equidade foram executados seis cenários diferentes com a combinação de fluxos assim denominados:

FA – Cenário de Fluxos A – cinco fluxos HCC

FB – Cenário de Fluxos B – quatro fluxos HCC e um fluxo TCP

FC – Cenário de Fluxos C – três fluxos HCC e dois TCP

FD – Cenário de Fluxos D – dois fluxos HCC e três fluxos TCP

FE – Cenário de Fluxos E – um fluxo HCC e quatro fluxos TCP

FF – Cenário de Fluxos F – cinco fluxos TCP

Para o teste de equidade intraprotocolo (*Fairness*) foram utilizados os resultados no fluxo A. Já para os testes de equidade com o TCP (TCP *Friendliness*) foram utilizados os fluxos de FB a FF. Sendo que os resultados do fluxo FF foram usados para a comparação de quando há somente fluxos FF e de quando há fluxos HCC e TCP compartilhados.

Da mesma forma foi desenvolvido um script para automatizar o teste. A seguir segue parte do script para os fluxos FA e FB.

```
#!/bin/bash
#
# Script to run a test 30 times in sequence
# By Marcos Monteiro
# Variables
# Proccess Name
Delay=100
Loss=0.001
proccess=appclient
# Destination IP Address
IPADDR=172.16.1.1
# File Name Prefix
Name=mtu1500_delay_$Delay'_'
# Date
Date=`date +%F_%R_`
# Directory
Dir=/home/mvsmonteiro/udt4.7a
/bin/mkdir $Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss
# Auxiliar Variable
i=0
# for ((j=1; j <= 6; j++)) do
# Fairness A - 5 HCC
  while [ $i -le 30 ] ; do
    num_procs=`ps ax | grep $proccess | grep -v grep| wc -l`
    if [ $num_procs -eq 0 ] ; then
    $Dir/app/appclient_HCC $IPADDR 22222  >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FA_HCC1_'$Name$Date$i &
    $Dir/app/appclient_HCC $IPADDR 22222 >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FA_HCC2_'$Name$Date$i &
    $Dir/app/appclient_HCC $IPADDR 22222 >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FA_HCC3_'$Name$Date$i &
    $Dir/app/appclient_HCC $IPADDR 22222 >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FA_HCC4_'$Name$Date$i &
    $Dir/app/appclient_HCC $IPADDR 22222 >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FA_HCC5_'$Name$Date$i &
        i=`expr $i + 1`
     else
        sleep 60
     fi
  done
# Fairness B - 4 HCC & 1 TCP
  while [ $i -le 60 ] ; do
    num_procs=`ps ax | grep $proccess | grep -v grep| wc -l`
    if [ $num_procs -eq 0 ] ; then
    $Dir/app/appclient_HCC          $IPADDR          22222          >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FB_HCC1_'$Name$Date$i &
    $Dir/app/appclient_HCC          $IPADDR          22222          >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FB_HCC2_'$Name$Date$i &
    $Dir/app/appclient_HCC          $IPADDR          22222          >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FB_HCC3_'$Name$Date$i &
    $Dir/app/appclient_HCC          $IPADDR          22222          >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FB_HCC4_'$Name$Date$i &
    $Dir/app/appclient_TCP          $IPADDR          22223          >
$Dir/teste/resultados/Fairness/$Delay'ms_Perda_'$Loss/'FB_TCP1_'$Name$Date$i &
        i=`expr $i + 1`
     else
        sleep 60
     fi
done
```

A partir do armazenamento em arquivo dos resultados, foi realizada a análise dos dados.

Para automatizar a análise dos resultados, foram desenvolvidos scripts para ajustar os arquivos com resultados e calcular a média, desvio padrão e intervalo de confiança.

Como resultado, a aplicação cliente imprime os dados na tela (que foram gravadas em arquivo) conforme Tabela A.1.

SendRate(Mb/s)	RTT(ms)	PktSndPeriod(us)	Janela	TimeStamp(s)	NAK	PacketFlight	TXRetrans
86.2269	112644	6.31642	83333	1	0	1254	16
128.53	112872	5.38858	83333	2	0	1176	16
129366	112756	7.01917	83333	3	0	1314	16
128113	112305	10.2352	83333	4	0	1274	16
129,108	112,318	12,103	83333	5	0	1319	16
129,014	117,429	12,4453	83333	6	0	1356	16
128,024	117,264	7,00939	83333	7	0	1377	16
129,142	117,029	7,74446	83333	8	0	1287	16
127,733	117,366	9,4953	83333	9	0	1283	16
129,935	117,963	7,96655	83333	10	0	1330	16

Tabela A.1: Resultado da transmissão de dados do cliente

Primeiramente é calculada a média de cada amostra dentre as trinta rodadas de experimentos e o resultado salvo em arquivo.

```
#!/bin/bash
# Script para preparar os arquivos de resultados para os cálculos e gráficos
#
HORARIO=2012-09-28_06:58
DELAY=120_
PERDA=0.001
TESTE=mtu1500_delay_$DELAY$HORARIO
DIR1=/home/mvsmonteiro/udt4.7a/teste/resultados/UFF/Fairness
DIR2=/home/mvsmonteiro/udt4.7a/teste/resultados/temp/
CENARIO=$DELAYms_Perda_$PERDA
##############################################################################################
########## *******          FA - 5 FLUXOS HCC       ********* ##############
##############################################################################################
for (( j = 1; j < 6; j++ )); do
TIPO=FA
FLUXO=_HCC$j'_'
ARQUIVO=$TIPO$FLUXO$TESTE
Name=$TIPO$FLUXO$TESTE'_'$i
for (( i = 0; i < 31; i++ )); do
Name=$TIPO$FLUXO$TESTE'_'$i
#ARQUIVO=FA_HCC1_mtu1500_delay_100_2011-02-20_23_33
# Separa somente as colunas tempo e vazao
awk  '{print ($5 , $1);}'  $DIR1/$CENARIO/$Name >> $DIR2$ARQUIVO'1'
done
# Limpa o arquivo deixando somente os valores de tempo e vazao e salva no arquivo_2.
awk '{if ($1 !~ /socket/ && $1 !~ /TimeStamp/ && $1 !~ /Invalid/) {print ($1 , $2);}}' $DIR2$ARQUIVO'1'
> $DIR2$ARQUIVO'2'
# Ordena o arquivo_2 e salva no arquivo_3
sort -n $DIR2$ARQUIVO'2' > $DIR2$ARQUIVO'3'
```

```
# Roda o arquivo awk para calcular a media de cada iteracao com o mesmo valor de tempo
awk -f media.awk $DIR2$ARQUIVO'3' > $DIR2$ARQUIVO'_mediatemp'
awk '{if ($2 > 0) {print ($1 , $2);}}' $DIR2$ARQUIVO'_mediatemp' > $DIR2$ARQUIVO'_media'
rm $DIR2$ARQUIVO'1'
rm $DIR2$ARQUIVO'2'
rm $DIR2$ARQUIVO'3'
rm $DIR2$ARQUIVO'_mediatemp'
done
```

O comando “awk -f media.awk” executa o arquivo abaixo:

```
 BEGIN{
tempo = 0;
contador = 1;
bw=0;
}
{
if ($1 != tempo) {
  printf("%d %d\n", tempo, bw/(contador));
  tempo = $1;
  bw=$2;
  contador = 1;
} else
  {
  bw = bw + $2;
  contador = contador +1;
 }
}
END {
 printf("%d %d\n", tempo, bw/(contador));
}
```

O resultado desse processo gera um único arquivo com duas colunas. A primeira contem as amostras (na verdade o tempo do experimento em segundos – coluna “time stamp” da tabela A.1). E, a segunda coluna contém a média (dentre as trinta rodadas) para cada segundo do experimento, calculada com base na coluna “SendRate” da Tabela A.1. E, a partir destas informações a média, desvio padrão e intervalo de confiança são calculados.

No caso dos experimentos de equidade com o TCP como foram rodados cinco fluxos simultâneos e o protocolo com HCC termina primeiro a transferência dos dados, por ser mais eficiente, é preciso descobrir quando foi o momento desse término para somente considerar o período fluxos simultâneos.

Para isto, foi desenvolvido um script para ler todos os arquivos de média gerados antes e descobrir qual o menor valor de TimeStamp (Tab. A1).

```
for (( j = 1; j < 6; j++ )); do
TIPO=FA
FLUXO=_HCC$j
ARQUIVO=$TIPO$FLUXO$TESTE
tail -n 1 $DIR2/$ARQUIVO'_media' >> $DIR2/Menorvalor_$TIPO'_1'
done
awk  '{print ($1);}'  $DIR2/Menorvalor_$TIPO'_1' >> $DIR2/Menorvalor_$TIPO'_2'
MenorValor=`awk -f teste.awk $DIR2/Menorvalor_$TIPO'_2'`
for (( i = 1; i < 6; i++ )); do
```

```
TIPO=FA
ARQUIVO=$TIPO$FLUXO$TESTE
FLUXO=_HCC$i
awk -v Valor=$MenorValor '{ if ($1 <= Valor )  {print( $1 , $2)}}' $DIR2/$ARQUIVO'_media' >>
$DIR1/$CENARIO/resultados/$ARQUIVO
done
```

Para calcular a média geral de cada fluxo do experimento, o desvio padrão e intervalo de confiança foi utilizado o script abaixo.

```
#!/bin/awk -f
#
#     Programa para calcular média e desvio padrão e int. confiança com awk
{
  soma+=$2 # Soma a coluna do arquivo com os valores da vazão
  somaquad+=$2*$2 # Soma quadrática para o desvio
}
END {
     media=soma/NR # Media - Soma total pelo número de linhas
     desvpad=sqrt((somaquad - NR*media^2)/(NR - 1)) # Desvio Padrão
     intConf1 = media + 1.96*desvpad/sqrt(NR) # Intervalo Superior
     intConf2 = media - 1.96*desvpad/sqrt(NR) # Intervalo Inferior
     printf "Tamanho da amostra t(s) %d\n"  \
           "          Média: %.2f    \n"  \
           "   Desvio Padrão: %.2f    \n" \
           "Intervalo de Confiança: de %.2f  a %.2f    \n", \
           NR, media , desvpad, intConf2, intConf1
   }
```

APÊNDICE B – MODIFICAÇÕES NOS SISTEMAS OPERACIONAIS

Abaixo está a sintaxe dos comandos e os valores utilizados para as modificações nos sistemas operacionais Linux utilizados nos testes:

- Modificação dos tamanhos padrão e máximo, em bytes, do buffer de leitura para o soquete do sistema operacional;

sysctl -w net.core.rmem_default = 8388608

sysctl -w net.core.rmem_max = 16777216

- Modificação dos tamanhos padrão e máximo, em bytes, do buffer de escrita para o soquete do sistema operacional;

sysctl -w net.core.wmem_default = 8388608

sysctl -w net.core.wmem_max = 16777216

- Modificação dos tamanho do comprimento da fila, em pacotes, para a entrega ao dispositivo de rede.

sysctl -w net.core.netdev_max_backlog = 20000

- Desabilitar ARP redirection:

sysctl -w net.ipv4.conf.all.accept_redirects=0

sysctl -w net.ipv4.conf.all.send_redirects=0

ANEXO A - NETEM

A seguir seguem alguns exemplos de configuração do Netem. No exemplo abaixo foi aplicado um atraso de 100ms na fila de saída da interface eth0.

tc qdisc add dev eth0 root netem delay 100ms

A variação do atraso (*jitter*) também pode ser inserida de forma randômica. Abaixo temos uma variação de 100ms ± 10ms.

tc qdisc add dev eth0 root netem delay 100ms 10ms

Contudo, o *jitter* na rede não é puramente aleatório. Assim, é possível inserir uma correlação. Abaixo foi inserida uma dependência de 25% do *jitter* em relação ao valor anterior. Esse valor não representa uma correlação estatística, mas sim uma aproximação.

tc qdisc add dev eth0 root netem delay 100ms 10ms 25%

A perda é implementada de forma randômica, de acordo com o valor definido no comando. Por exemplo, o comando abaixo provoca uma perda randômica de 1 pacote a cada 1000.

tc qdisc add dev eth0 root netem loss 0.1%

Esta aleatoriedade pode ser modificada fazendo com que a perda seja dependente da anterior. Assim, no comando abaixo, a perda será de 0.3% e, dependente de um quarto (25%) da perda anterior, de acordo com a expressão:

Perda(n) = 0.25 * Perda (n −1) + 0.75 random.

tc qdisc add dev eth0 root netem loss 0.3% 25%

Há também a possibilidade de se modificar o tamanho do buffer da fila de saída. O parâmetro “limit”, medido em pacotes, é utilizado para esse fim.

tc qdisc change dev eth1 root netem delay 50ms **limit** 250.