MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

MARCELO REIS DA SILVA

ADAPTAÇÃO DA CAMADA DE TRANSPORTE ATRAVÉS DA CORRETA SELEÇÃO DE ALGORITMO DE CONTROLE DE CONGESTIONAMENTO

Rio de Janeiro
26 de Janeiro de 2009

INSTITUTO MILITAR DE ENGENHARIA

MARCELO REIS DA SILVA

ADAPTAÇÃO DA CAMADA DE TRANSPORTE ATRAVÉS DA CORRETA SELEÇÃO DE ALGORITMO DE CONTROLE DE CONGESTIONAMENTO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Ronaldo Moreira Salles - Ph.D.

Rio de Janeiro
26 de Janeiro de 2009

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

S586a Silva, M. R.
Adaptação da Camada de Transporte Através da Correta Seleção de Algoritmo de Controle de Congestionamento/ Marcelo Reis da Silva. – Rio de Janeiro: Instituto Militar de Engenharia, 26 de Janeiro de 2009.
94 p.: il.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 26 de Janeiro de 2009.

1. Redes de comunicao de dados. 2. Transporte adaptativo. I. Título. II. Instituto Militar de Engenharia.

CDD 004.06

INSTITUTO MILITAR DE ENGENHARIA

MARCELO REIS DA SILVA

ADAPTAÇÃO DA CAMADA DE TRANSPORTE ATRAVÉS DA CORRETA SELEÇÃO DE ALGORITMO DE CONTROLE DE CONGESTIONAMENTO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Ronaldo Moreira Salles - Ph.D.

Aprovada em 26 de Janeiro de 2009 pela seguinte Banca Examinadora:

Prof. Ronaldo Moreira Salles - Ph.D. do IME - Presidente

Prof. Magnos Martinello - Ph.D. da UFES

Prof. Sidney Cunha de Lucena - DSc. da UNIRIO

Prof. Raquel Coelho Gomes Pinto - D.Sc. do IME

Rio de Janeiro
26 de Janeiro de 2009

Dedico esta dissertação ao meu DEUS, o Pai das luzes, de quem procede toda boa dádiva e todo dom perfeito, em quem não há mudança nem sombra de variação.

AGRADECIMENTOS

Agradeço, acima de tudo, a JESUS, que sustenta todas as coisas pela Palavra do seu Poder.

Às irmãs de oração da Assembléia de Deus em Parque Colúmbia, Pastoreada pelo servo do SENHOR Artur Luís de Meireles, pela forma simples e fervorosa com que sempre apresentavam minhas causas diante do SENHOR.

Ao meu Orientador, Major Salles, por sua paciência, dedicação e pelos excelentes conselhos.

A todos os meus familiares, em especial à minha Esposa Beatriz, cujo apoio não se pode descrever com palavras e ao meu irmão Luciano pelos equipamentos cedidos.

Agradeço a todas as pessoas que contribuíram com o desenvolvimento desta dissertação de mestrado, tenha sido por meio de críticas, idéias, apoio, incentivo ou qualquer outra forma de auxílio.

Por fim, a todos os professores e funcionários do Departamento de Engenharia de Sistemas (SE/8) do Instituto Militar de Engenharia.

Marcelo Reis da Silva

Se o degrau alcançado não revelar um pouco mais da minha pequenez, eu, na verdade, deslizei.

O AUTOR

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

C2	-	*Comando e Controle*
ACK	-	*Acknowledge*
AIMD	-	*Additive Increase Multiplicative Decrease*
BDP	-	*Bandwidth delay product*
BIC	-	*Binary Increase Congestion*
COTER	-	*Comando de Operações Terrestres*
cwnd	-	*congestion window*
DNS	-	*Domain Name System*
DSL	-	*Digital Subscriber Line*
ERE	-	*Eligible Rate Estimation*
FDT	-	*Fio Duplo Telefônico*
FTP	-	*File Transfer Protocol*
GEO	-	*Geo-estacionário*
HF	-	*High frequency*
HSTCP	-	*HighSpeed TCP*
HTTP	-	*Hypertext Transfer Protocol*
IME	-	*Instituto Militar de Engenharia*
med	-	*Médio*
min	-	*Mínimo*
MSS	-	*Maximum segment size*
MTU	-	*Maximum transmit unit*
Netem	-	*Network Emulation*
OSI	-	*Open Systems Interconnection*
PER	-	*Packet Error Rate*
RTT	-	*Round-trip time*
SISCOMIS	-	*Sistema de Comunicações Militares por Satélite*
SNT	-	*Sistema Nacional de Telecomunicações*
ssthresh	-	*Slow Start threshold*
STCP	-	*Scalable TCP*

TCP	-	*Transmission Control Protocol*
UDP	-	*User Datagram Protocol*
VHF	-	*Very High Frequency*
WAN	-	*Wide Area Network*

RESUMO

Hoje em dia, as transmissões sem fio, junto com a rápida expansão de redes de alta velocidade, impõem novos desafios aos protocolos de transporte de dados. Dentre estes desafios, destacam-se as conexões de longo *round trip time* (RTTs), as conexões com taxas de erros de pacotes (PER) não desprezíveis (HF, por exemplo), e grandes largura de banda. Para superar estes desafios, um grande número de variantes TCP é apresentado na literatura com finalidades e propósitos diferentes. Entretanto, como a maioria das propostas são desenvolvidas para solucionar problemas diferentes, elas representam otimizações para redes específicas. Conseqüentemente, dado o nível crescente de heterogeneidade das redes atuais e futuras, a escolha da melhor abordagem de transporte parece um problema em aberto. Este trabalho propõe uma metodologia para a seleção de diferentes versões de protocolos de transporte caracterizando uma camada de transporte adaptativa. A proposta é simples, de baixo custo e baseada em experimentos reais. Alguns critérios, a serem adotados para a seleção do transporte de dados, são discutidos. São fornecidos resultados relativos a topologias de rede reais, escolhidas para ilustrar as vantagens de uma camada de transporte adaptativa. Tais resultados são extremamente encorajadores e justificam as observações que se seguem sobre a viabilidade e implementação de uma camada deste tipo.

ABSTRACT

Nowadays, the wireless transmissions, together with the rapid growth of high speed networks, pose new challenges to the data transport protocols. Among them, the most prominent are long round trip times (RTTs), not negligible (HF, for instance) packet error rates (PER), and very large bandwidths. To overcome them, a wide variety of TCP enhancements has been presented in the literature with different purposes and capabilities. However, as most proposals aim to address specific impairments, they are optimizations for specific network environments. Therefore, given the increasing level of heterogeneity of present and future networks, the choice of the best transport enhancement seems a quite open problem. This work proposes a selection methodology among different transport protocol versions, characterizing an adaptive transport layer. Such proposal is simple, low cost and built from real experiments. Some criteria, to be adopted for the data transport selection, are discussed. Results, referring to real network topologies, chosen to ilustrate the adaptive transport layer advantages, are provided. They are quite encouraging and justify the following remarks about viability and such a layer implementation.

1 INTRODUÇÃO

1.1 ABRANGÊNCIA E COMPLEXIDADE DAS REDES DE COMUNICAÇÃO

Ao considerar-se as redes de comunicação, é fácil observar que elas têm permeado praticamente todas as áreas da atividade humana. Seja na calmaria do campo, seja no agito das grandes cidades, quer no lazer ou nas atividades profissionais, na guerra ou na paz, as conexões de dados tornam as distâncias entre os diversos ambientes cada vez menor. A expansão das redes de dados é, portanto, algo inevitável em uma sociedade que, por diversos fatores, principalmente econômicos, busca cada vez mais a globalização. Tão diversas quantas as situações permeadas pelas redes de comunicação são as características dos componentes da sua malha. Essa heterogeneidade exige a adaptação das camadas superiores para que haja comunicação entre quaisquer dois pontos desta rede. A configuração das redes está cada vez mais complexa, como resultado da combinação de diferentes:

- **Plataformas**

  No mundo das plataformas, entidades agregadoras dos componentes de *hardware* utilizados nas redes de comunicação, há variações em praticamente todos os aspectos: tamanho, capacidade de processamento, capacidade de armazenamento, arquitetura, etc.

- **Aplicações**

  Diversas são as aplicações clientes das redes de dados, cada qual com seus requisitos e exigências sobre o desempenho das mesmas para atingirem seus propósitos. Aplicações em tempo real , por exemplo, requerem que novos dados sejam entregues a todo instante e no tempo correto; aplicações elásticas devem primar pela correção e recuperação dos dados trocados; já as aplicações multimídias exigem pequeno *jitter*, e assim por diante.

- **Tecnologias de transmissão**

  Interligando as plataformas existem diversos tipos de enlaces diferentes com características próprias quanto à taxa de transmissão, largura de banda, sensibilidade à

interferência, latência, grau de mobilidade, etc.

A diversificação das tecnologias envolvidas na comunicação digital tem influenciado de forma decisiva os requisitos dos projetos, inclusive aqueles de cunho bélico. No Exército Brasileiro esta influência pode ser comprovada pelas características do projeto C2 (Comando e Controle) em Combate.

1.2 MOTIVAÇÃO

1.2.1 PROJETO C2 EM COMBATE

Para auxiliar as atividades de comando e controle do Exército Brasileiro, foi criado, através da portaria número 102 - Comando da Força, de 18 de março de 2003, o projeto C2 em combate (COTER, 2002), composto por dois grandes vetores: O Programa C2 em Combate e o Módulo de Telemática. O Módulo de Telemática provê diversas tecnologias de enlace através das quais o Programa C2 em Combate trafegará suas informações (FIG. 1.1). Sendo assim, um dos principais requisitos do programa C2 em combate é a capacidade de trocar seus dados independente da tecnologia de enlace disponibilizada pelo módulo de telemática em um dado momento. Dentre os enlaces possíveis de serem

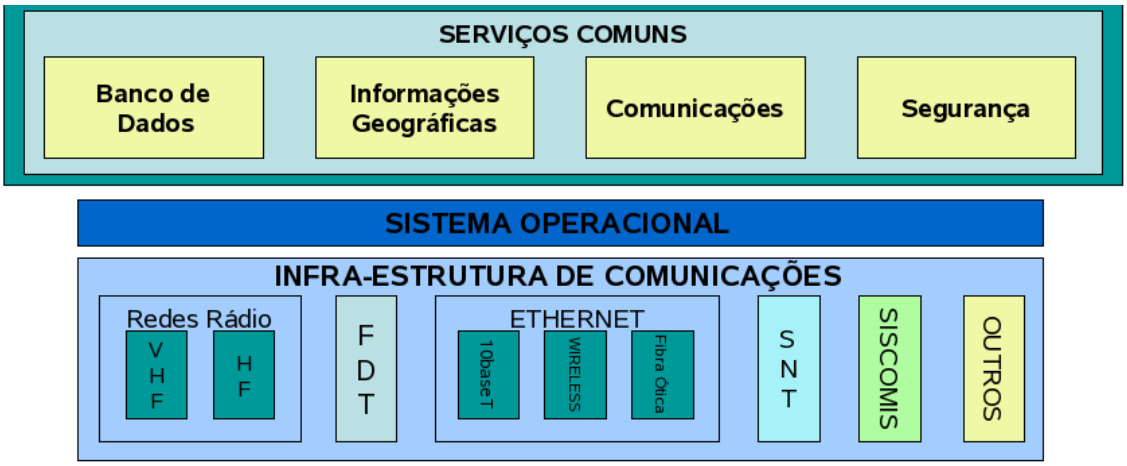

FIG. 1.1: Arquitetura Programa C2.

escolhidos estão: rádio (HF, VHF), satélite (SISCOMIS), ethernet, DSL (FDT), etc . Certamente a manutenção de um mesmo algoritmo de transporte quando do chaveamento entre uma ligação DSL (velocidades da ordem de centenas de kilobits ou até Megabits por segundo) (KESSLER, 1992) e uma em HF (no máximo 1200 bps, com taxa de erro média de 50% (JOHNSON, 1995)), poderia comprometer a troca de informações.

1.2.2 DISCREPÂNCIA NAS LIGAÇÕES FIM-A-FIM

Um fato observado nas grandes redes de computadores (WAN´s, por exemplo) é que as ligações entre os clientes de um servidor são as mais diversas, sofrendo variações principalmente na capacidade fim-a-fim dos enlaces. Para exemplificar, consideremos a topologia da rede rio (FIG 1.2)

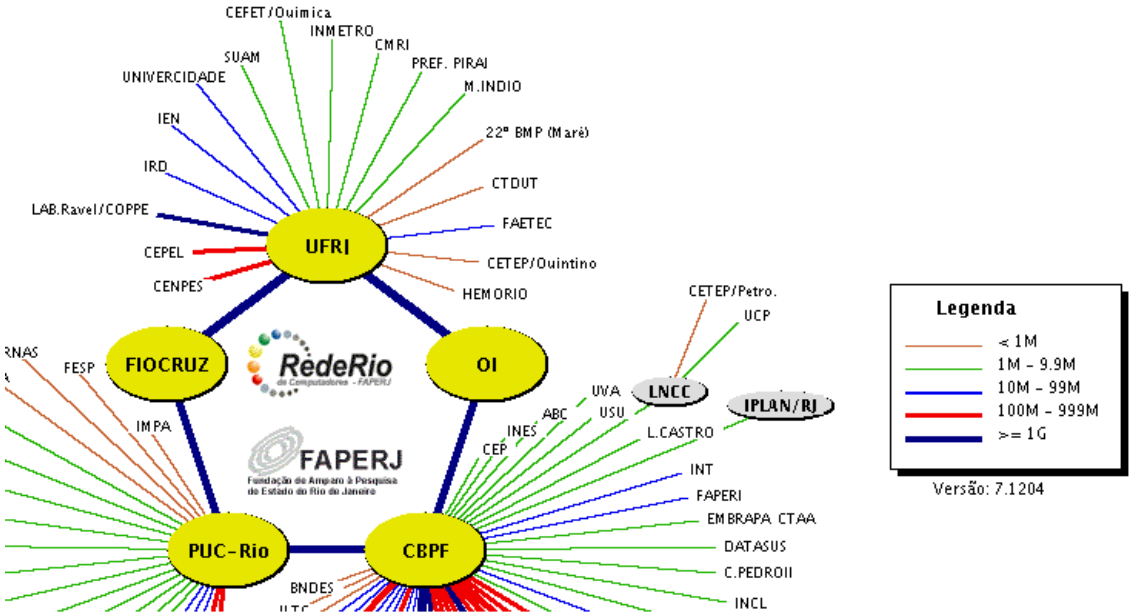

FIG. 1.2: Estrutura da Rede Rio.

Nesta rede imaginemos que dois clientes, um na OI e outro no HEMORIO estejam conectados a um servidor do CBPF. Na ausência de sobrecarga, o protocolo de transporte que rege a conexão entre este servidor e o terminal localizado na OI pode ser bem mais agressivo do que aquele presente na conexão estabelecida entre o CBPF e o HEMORIO, pois no primeiro caso dispõe-se de um enlace de capacidade maior ou igual a 1 Gbps, enquanto que no segundo a velocidade da conexão não chega a 100 Mbps.

Muitos trabalhos refletem a preocupação dos pesquisadores com a diversidade das redes de comunicação, introduzindo conceitos que representam a flexibilização dos mecanismos envolvidos na ligação entre dois terminais. Alguns destes trabalhos foram estudados durante a revisão bibliográfica. O resultado desta revisão compõe o conteúdo da seção a seguir.

1.3 TRABALHOS RELACIONADOS

1.3.1 PARTICIONANDO CONEXÕES TCP

Devido às diferenças significativas que os enlaces componentes de uma ligação fim-a-fim podem apresentar entre si (fibra ótica, ethernet, 802.11, Rádio, etc), estudos (REZENDE, 2008) estão propondo uma adaptação da camada de transporte através da criação de sessões TCP entre pares de nós que interconectam as máquinas transmissora e receptora. Estas sessões TCP objetivam emular uma conexão fim-a-fim mais eficiente. Os nós intermediários servem como comutadores da camada de transporte numa abordagem conhecida com "split approach" (LEE, 2008). O presente trabalho, porém, concentra seus esforços sob uma abordagem fim-a-fim, que trata a comunicação entre o receptor e o transmissor sem levar em consideração os nós intermediários entre eles.

1.3.2 ENLACES COM PRODUTO BANDA RETARDO ELEVADO

É inegável que a atual Internet tornou-se funcional com o uso dos algoritmos de congestionamento presentes nas versões tradicionais do TCP (Tahoe, Reno, New Reno). Entretanto, o TCP se mostra pouco escalável em redes com produto banda retardo (BDP) elevado, também chamadas de *fast long-distance networks*, isto é, elevada velocidade e longo RTT. Tanto o emprego cada vez maior dos enlaces com velocidade na ordem de Gigabits por segundo, quanto a utilização de ligações de alta latência (satélites, por exemplo), são suficientes para elevar o produto banda retardo a níveis nos quais o *throghput* alcançado pelas versões tradicionais do TCP será bem aquém da capacidade real do enlace. Para amenizar este problema, diversos autores têm proposto novas funções para atualização da janela de congestionamento, apresentando assim diversas variantes TCP para redes de alto BDP, tais como BIC TCP (XU, 2004), CUBIC (XU, 2005), Fast TCP (JIN, 2004), H-TCP (SHORTEN, 2004), High Speed TCP (FLOYD, 2003), abordadas com maiores detalhes no decorrer deste trabalho.

Infelizmente, a maioria dos trabalhos realizados para confrontar o desempenho das novas versões TCP são baseados em simulações, onde há maior chance de a influência de fatores importantes relacionados à transmissão de dados em ambientes reais ser tratada de maneira inadequada. Neste trabalho, adota-se uma abordagem prática, através de experimentos em laboratório, para avalição da escalabilidade de algumas destas versões.

Este trabalho não visa propor uma nova abordagem, mas fornecer critérios para es-

colher, entre as opções de transporte de dados hoje disponibilizadas, aquela que melhor se adequada ao estado momentâneo da rede.

1.3.3 TRANSPORTE SOBRE UDP

No outro extremo dos enlaces de grande capacidade, estão aqueles que oferecem pequenas taxas de transmissão da ordem de centenas de bits por segundo associada a elevada taxa de erros. Os enlaces rádio de alta freqüência (HF), enquadram-se nesta categoria. Por associar alta mobilidade a um custo relativamente baixo, a comunicação baseada em HF continuará exercendo por muito tempo ainda um importante papel nos sistemas de comunicação, principalmente em contextos operacionais. Só para ilustrar, podemos citar o que aconteceu durante as operações de salvamento após o ataque de 11 de setembro em Nova Iorque (JODALEN, 2004), onde a falta de energia elétrica, a destruição parcial das redes e o congestionamento causado por milhares de chamadas simultâneas produziram um colapso na infra-estrutura de telecomunicações. Durante o resgate das vítimas, a Guarda Nacional Americana dispunha apenas das redes HF para conduzir as operações.

O emprego de protocolos de transporte baseados em UDP em redes HF, com esquemas específicos para trabalhar sobre enlaces com quase nenhum *feed-back*, no lugar de protocolos de transportes reativos (TCP), pode reduzir o tempo de *download* de um arquivo de dezenas de horas para dezenas de minutos (ASENSTORFER, 1997).

Por este motivo, o trabalho também traz uma análise a respeito dos ganhos reais que o UDP pode trazer em relação ao TCP quando os dados são transmitidos por enlaces que se aproximem das características proporcionadas pelas transmissões digitais em HF.

Serão realizadas portanto, considerações a respeito de uma camada de transporte adaptativa sobre UDP. Estas considerações são para emprego em projetos especiais tais como o Projeto C2 em Combate, que prevê a possibilidade de se trabalhar com os mais diversos tipos de enlaces, incluindo o HF.

Há, além daquelas situações em que se empregam enlaces de baixa capacidade e elevadas taxas de erro, ocasiões em que não é interessante para as aplicações todos os serviços oferecidos por uma camada de transporte reativa. Dentre elas, destacam-se:

- Aplicações de tempo real: Para este tipo de aplicação não vale a pena perder tempo com o reenvio de pacotes perdidos em detrimento daqueles que estão sendo gerados durante suas atividades.

- Aplicações com necessidade de resposta rápida: O TCP, por exemplo, perde muito tempo com a abertura de conexão antes da troca efetiva de dados. Aplicações como DNS, no entanto, precisam de respostas rápidas às suas requisições, sem perder tempo com as preliminares próprias do TCP.

- Clientes leves: O transporte orientado a conexão requer o armazenamento de inúmeras variáveis de estado para viabilização das suas atividades. Com UDP, porém, o estado das conexões é ignorado, aumentando significativamente, assim, o número máximo de requisições que podem ser atendidas por um servidor ao mesmo tempo.

1.3.4 REDES AUTONÔMICAS

Existe uma forte tendência, devido ao grande número de usuários alcançados pelas redes de comunicação, em tornar esta rede cada vez mais independente do usuário final, ou seja, os sistemas computacionais devem interligar-se de forma auto configurável, encarregando os administradores das tarefas de elevada complexidade. As redes baseadas nesta filosofia são denominadas autonômicas (KEPHART, 2003). Uma das características das redes autonômicas é a capacidade de gerenciar seus nós e conexões sem a intervenção humana, "aprendendo" a modificar seu comportamento através da análise de eventos que ocorrem na rede. A tabela 1.1 reúne os principais vetores da computação autonômica e seus objetivos em relação às tecnologias tradicionais.

Como já foi dito, as redes atuais são extremamente complexas. Portanto, a existência de um mecanismo de transporte de dados que possa, em função das condições da rede, escolher a melhor função de atualização da janela de congestionamento de forma imperceptível para os usuários finais vai ao encontro dos objetivos perseguidos pela Auto configuração e Auto-otimização pregadas na computação autonômica.

1.4 VARIANDO O TRANSPORTE NAS DISTRIBUIÇÕES LINUX

A necessidade da adaptação da camada de transporte é tão evidente que, a partir do *kernel* 2.6.7, algumas distribuições linux já trazem a opção de carregamento de outros algoritmos de controle de congestionamento em tempo de execução através do comando **sysctl** (Cap. 12.2). A ativação dos módulos, entretanto, depende da intervenção do usuário, que deve carregá-los através da linha de comando. Além disso, dificilmente os usuários finais, interessados em uma computação cada vez mais autonômica, terão discernimento a ponto

Vetores da Computação Autonômica		
Conceito	Computação "Tradicional"	Computação Autonômica
Auto configuração	A instalação, configuração e interligação entre sistemas consome tempo e é altamente suscetível a erros.	Configuração dos sistemas em alto nível. Os outros ajustes são feitos de forma automática e transparente pelo próprio sistema.
Auto-otimização	Os sistemas possuem centenas de parâmetros, dificilmente ajustados de forma ótima. Esses parâmetros crescem a cada nova distribuição.	Os componentes do sistema procuram continuamente oportunidades para melhorar seu desempenho e eficiência.
Auto diagnóstico	A identificação de um problema em um sistema complexo pode consumir várias semanas da equipe de programadores.	O sistema automaticamente detecta, diagnostica e repara problemas localizados de *hardware* e *software*
Auto proteção	A detecção e recuperação de ataques ou falhas em cadeia é manual.	O sistema automaticamente se defende contra ataques maliciosos e usa um histórico para se antecipar e prever falhas que comprometam o sistema como um todo.

TAB. 1.1: Computação "Tradiconal" X Autonômica.

de escolherem a melhor abordagem de transporte para uma determinada rede. Outro ponto que deve ser destacado é que, após a mudança do algoritmo de transporte, somente as novas conexões seguirão as novas regras, ou seja, aquelas previamente estabelecidas continuarão regidas pelo módulo em vigor durante o seu início (SILVA, 2006).

Um *kernel* da família 2.6.7, dotado da possibilidade de alterar o "tipo" de TCP em tempo de execução, serviu de base para realização dos experimentos deste trabalho, cuja proposta visa o levantamneto de dados em topologias de redes reais.

1.5 OBJETIVO

Apesar de vários estudos estarem sendo realizados com o objetivo de tornar o transporte mais escalável possível, pouco se discute a respeito de uma camada adaptativa, isto é, um esquema de transporte que pudesse agregar várias maneiras de realizar a comunicação fim-a-fim e escolher uma delas dinamicamente, de acordo com as condições impostas pela rede, ou seja, uma camada de transporte conforme a camada de rede. Este trabalho visa a flexibilização da camada de transporte, adaptando-a convenientemente à diversidade própria das redes de comunicação hodiernas. Para isso, vamos estudar as vantagens de uma camada de transporte adaptativa e propor uma arquitetura para implementação de uma camada desse tipo.

Esta arquitetura é apresentada visando a construção de camadas de transporte adaptativas capazes de suportar ajustes em seus mecanismos, mesmo durante a troca de dados. Ou seja, ainda que uma determinada conexão já tenha sido estabelecida, a proposta deste trabalho visa dar a esta conexão a possibilidade de adequar-se aos novos valores dos parâmetros da rede, os quais, em geral, podem a qualquer momento assumir valores praticamente imprevisíveis.

O trabalho faz uma investigação da viabilidade de uma camada de transporte adaptativa e levanta, de forma prática, alguns critérios para definição da escolha entre abordagens de transporte diferentes. Para isso, uma tabela relacionará o protocolo de transporte (versão TCP), mais indicado para os valores do RTT e velocidade apresentados por uma ligação fim-a-fim. Esta tabela servirá de base para as camadas de transporte adaptativas adequarem suas atividades às condições impostas pela rede em um dado momento.

Visando atender situações mais específicas, avalia-se, através de cenários reais, os ganhos que o emprego de um transporte sem conexão pode produzir em relação a protocolos de transporte reativos em enlaces que se aproximem das características próprias da trans-

missão de dados em HF. À luz destes resultados, são feitas considerações de como se implementaria uma camada de transporte que precisasse alternar entre uma abordagem de transporte específica sobre UDP para uma filosofia mais tradicional.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

O trabalho está dividido em 10 capítulos. No Capítulo 2 serão apresentados os conceitos básicos da comunicação digital, necessários aos estudos realizados neste trabalho.

O Capítulo 3 traz um histórico da evolução do protocolo TCP desde suas versões mais remotas até aquelas projetadas em função das tecnologias de transmissão atuais.

Do capítulo 4 ao capítulo 8 tratam-se questões relativas a uma camada de transporte que ajuste suas atividades dentro das variantes TCP.

O capítulo 4 analisa viabilidade de uma camada de transporte que seja capaz de agregar diversas versões TCP e faz uma descrição da metodologia adotada pelo trabalho.

No capítulo 5, levantam-se os dados que vão servir de critério para seleção do controle de congestionamento mais adequado dentre as versões TCP.

No Capítulo 6, é feita uma análise geral dos resultados obtidos no capítulo 5, análise esta que serve de base para a determinação dos critérios de chaveamento entre as variantes TCP de melhor desempenho.

O capítulo 7 apresenta alguns estudos de casos, para destacar-se as vantagens que podem advir de uma camada de transporte flexível.

O capítulo 8 detalha a arquitetura para implementação de uma camada de transporte adaptativa entre as variantes TCP. Ainda neste capítulo, são feitas considerações sobre ferramentas que podem auxiliar o processo de desenvolvimento de uma camada TCP adaptativa.

O capítulo 9 agrupa a segunda fase do trabalho. São feitas considerações sobre as vantagens do UDP em relação ao TCP em enlaces com elevada taxa de erro e baixa velocidade. Além disso, é apresentado um modelo de transporte adaptativo sobre UDP.

Por fim, o capítulo 10 destaca as principais conclusões e algumas propostas para trabalhos futuros.

2 CONCEITOS BÁSICOS

2.1 SUITE TCP: UMA VISÃO GERAL

Todo e qualquer tipo de comunicação requer um conjunto mínimo de serviços que a torne possível. Nas comunicações digitais não é diferente; as mesmas precisam de canal, controle de erros, roteamento, controle de fluxo, controle de congestionamento, etc. A fim de facilitar o projeto e organização das redes de dados, a *International Organization for Standardization* desenvolveu o modelo de referência OSI (*Open System Interconnection*), que define como os vários serviços envolvidos numa comunicação digital devem interagir entre si. Estes serviços foram hierarquizados (STALLING, 1994) em sete camadas: aplicação, apresentação, sessão, transporte, rede, enlace e física. Cada camada se utiliza das tarefas das camadas adjacentes, sem acesso aos detalhes de implementação das mesmas.

O TCP/IP é uma suite de protocolos padronizada utilizada para conectar diferentes dispositivos de uma rede. Pode-se dizer que o TCP é uma simplificação do modelo OSI (FIG. 2.1). No topo da pilha TCP/IP encontramos a camada de aplicação que agrega

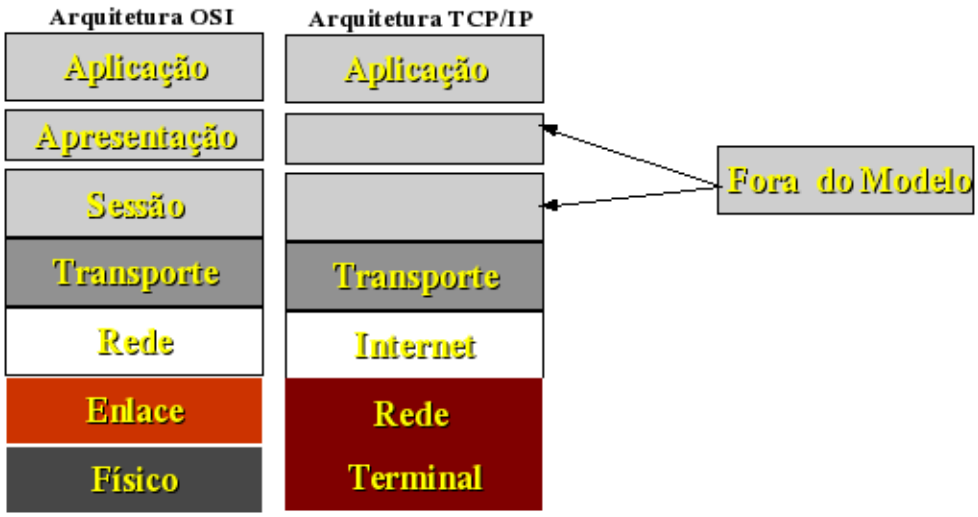

FIG. 2.1: Analogia Camadas OSI X Camadas TCP.

protocolos de alto nível tais como FTP, HTTP, SMTP, Telnet, DNS, etc.

A camada de transporte foi projetada para gerenciar a comunicação entre processos

instanciados nos terminais de uma rede. Dois mecanismos de transportes foram definidos nesta camada, o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*), que serão estudados com maiores detalhes na sessão a seguir. A camada Internet define um formato padronizado de pacotes chamado IP (*Internet Protocol*), e é encarregada da transferência de dados entre dispositivos associados a endereços IP distintos. Os dados oriundos da camada de transporte são, na camada Internet, distribuídos em um ou mais pedaços menores, cada qual com seu cabeçalho IP, e com seu próprio conjunto de *bytes* detetores de erros; a camada de transporte instanciada no computador remoto, ao receber os dados da camada Internet, checa a integridade deles; uma vez que não haja erros, eles são remontados e encaminhados ao programa que os espera.

A camada rede terminal não possui protocolos bem definidos por esta arquitetura, devendo entretanto ser capaz de encaminhar os pacotes IP que a sensibilizem.

Como este trabalho visa propor uma modificação na camada de transporte sobre IP, é feita a seguir uma investigação mais detalhada da camada de transporte na pilha TCP/IP e da sua interação com as camadas aplicação e Internet.

2.2 ANÁLISE DA CAMADA DE TRANSPORTE TCP/IP

A camada de transporte TCP/IP (TANENBAUM, 2003) (KUROSE, 2008) tem por finalidade promover a comunicação entre processos remotos de forma transparente, ou seja, se um processo gera uma mensagem para um outro, ele simplesmente entrega esta mensagem para a camada de transporte, que, com o auxílio das camadas inferiores, se encarregará de dar à mensagem o devido destino. A camada de transporte realiza, portanto, algumas atividades chaves, dentre as quais destacam-se:

- Garantia da confiabilidade: Os dados devem chegar ao destino da maneira como foram gerados pelo transmissor;
- Multiplexação: Vários processos podem estar injetando dados na camada de transporte ao mesmo tempo, o que exige que esta camada seja capaz agregar a comunicação ente diversos processos diferentes;
- Demultiplexação: Os dados oriundos das camadas inferiores devem ser despachados para os respectivos processos destino;

- Controle de Congestionamento: A camada de transporte deve ser capaz de aproveitar ao máximo a capacidade de escoamento de dados da rede sem sobrecarregá-la;
- Controle de Fluxo: A taxa com que os dados são transportados deve respeitar a capacidade de tratamento destes dados por parte do receptor.

2.2.1 INTERAÇÃO ENTRE AS CAMADAS APLICAÇÃO E DE TRANSPORTE : "SOCKETS"

Tanto o TCP quanto o UDP utilizam como interface com a aplicação os *sockets*. Cada processo na máquina que deseja utilizar os serviços da camada de transporte da pilha IP deve estar associado a um *socket* que se encarrega de intermediar a troca de dados entre cada processo e a camada de transporte (FIG. 2.2).

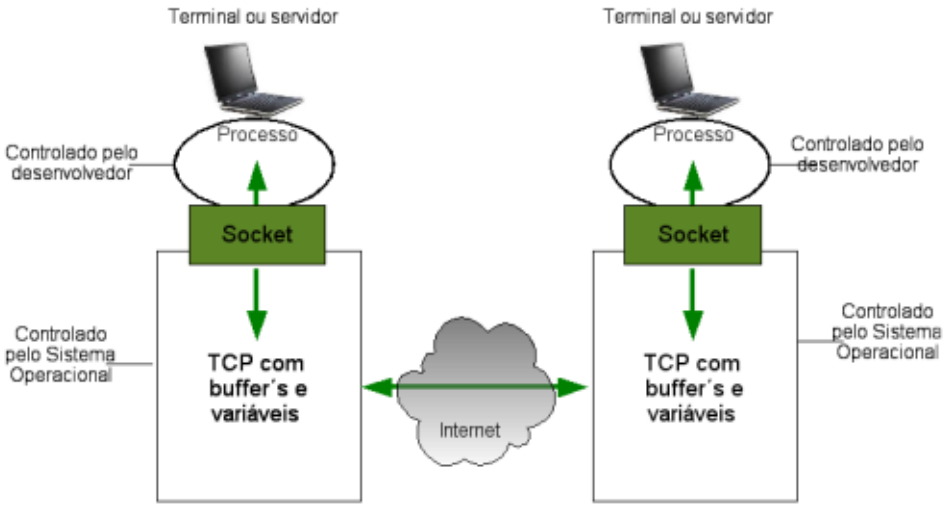

FIG. 2.2: O Papel dos Sockets.

2.2.2 UDP

Nesta modalidade de transporte (POSTEL, 1980), não há implementação de controle de congestionamento, a confiabilidade resume-se a verificar se os segmentos recebidos estão íntegros, sem se preocupar com a possibilidade de perda ou desordenação dos mesmos. A multiplexação/demultiplexação é feita principalmente através da porta destino, ou seja, pelo valor da porta destino o transporte UDP despacha os segmentos para o processo

ligado ao *socket* associado àquela porta, de forma que dois processos remotos podem atingir o mesmo *socket* desde que eles enviem para a mesma porta destino (FIG. 2.3).

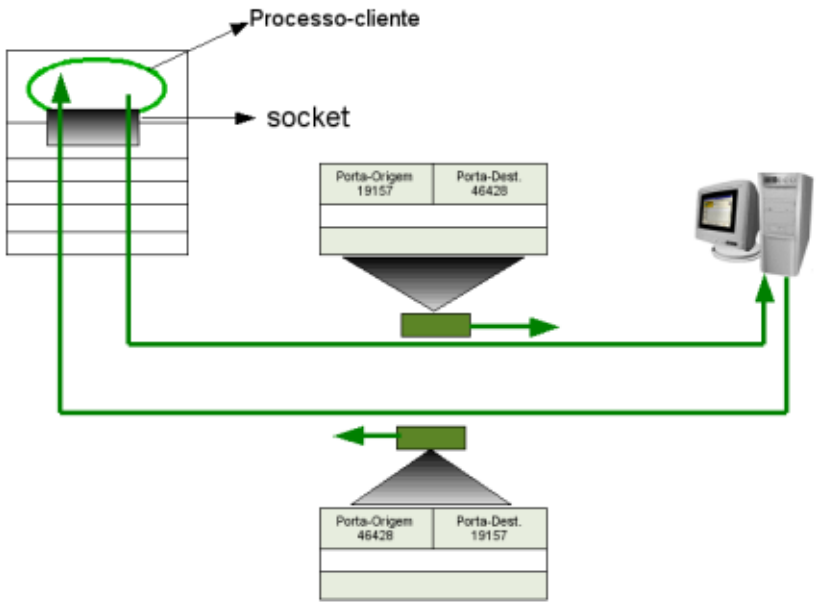

FIG. 2.3: Transporte UDP.

2.2.3 TRANSPORTE TCP

O transporte TCP trata a confiabilidade de forma muito mais ampla quando comparado ao UDP, já que o TCP, além de verificar a integridade dos segmentos, também realiza a recuperação de segmentos eventualmente perdidos bem como reordena os datagramas caso os mesmos cheguem fora de ordem. As conexões são estabelecidas da seguinte forma: há no servidor um *socket* encarregado de receber novas conexões; quando um *socket* no cliente informa a este *socket* o desejo de abrir uma conexão com o servidor, estes dois *socket*s realizam o famoso *three-way handshaking* (KUROSE, 2008); uma vez aceita a conexão, instancia-se um novo *socket* no servidor, agora responsável pela troca de dados efetivos entre este servidor e o cliente (FIG. 2.4).

Na multiplexação/demultiplexação utilizam-se as portas associadas aos processos interligados pela rede; cada par ordenado de número de portas identifica um determinado *socket*, como ilustrado na Figura 2.5.

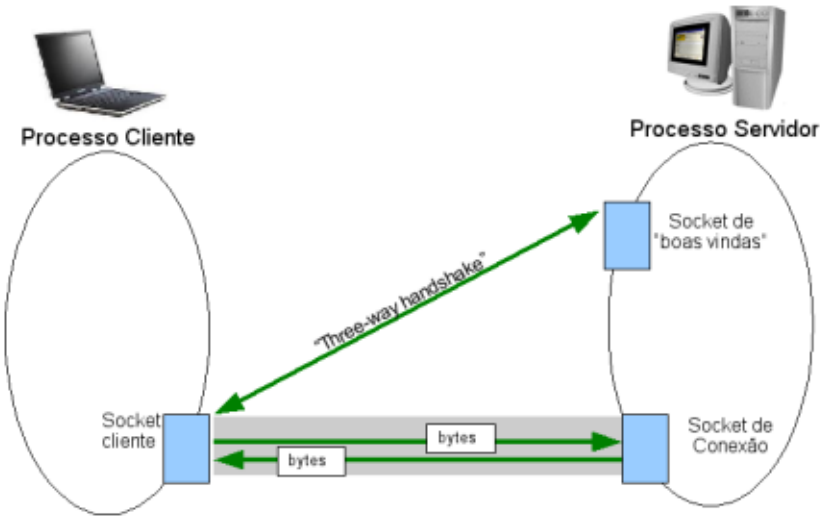

FIG. 2.4: Conexão TCP.

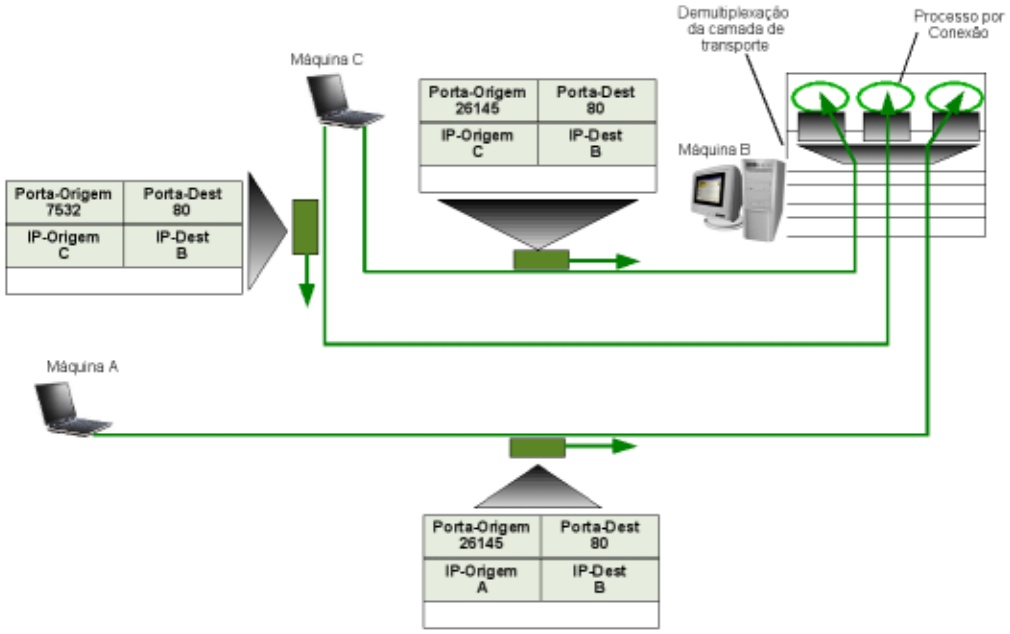

FIG. 2.5: Multiplexação/Demultiplexação TCP.

Quando os dados são recebidos pela camada de transporte eles são agrupados em segmentos, cujo tamanho deve variar de acordo com as características do enlace (MTU).

O controle de congestionamento pode ser resumidamente descrito como sendo o número máximo de segmentos que a camada de transporte passa às camadas inferiores a cada transmissão. Esta quantidade é influenciada por dois fatores: a capacidade do receptor (controle de fluxo), e a capacidade da rede (controle de congestionamento). O transmissor guarda consigo dois valores, o número máximo de *bytes* que podem ser tratados pelo receptor (janela de fluxo), e o número estimado de *bytes* que podem ser lançados na infraestrutura da rede sem congestioná-la (janela de congestionamento), denotada por cwnd. O menor entre estes dois valores será a quantidade de dados que a camada de transporte irá despejar na rede a cada tentativa de transmissão. Na próxima sessão analisaremos com mais detalhes as variáveis envolvidas no controle de congestionamneto TCP.

2.2.4 CONTROLE DE CONGESTIONAMENTO TCP

Durante o estabelecimento de uma conexão TCP o transmissor e o receptor negociam o valor inicial da janela de congestionamento (cwnd), chamado de *Maximum Segment Size* (MSS). Durante a transmissão a janela de congestionamento nas diversas variantes TCP tem seu valor, dado em MSS's, ditado por uma função cujos valores das variáveis de domínio são determinados de acordo com o número de segmentos que tiveram seu recebimento confimado pelo receptor. A confimação de cada porção transmitida é realizada através de pacotes ACK's durante toda a transmissão. Dependendo do perfil de chegada dos pacotes ACK's, o transporte TCP assume algum dos estados abaixo relacionados:

a) Slow Start

Inicialmente, a fonte TCP não possui qualquer informação atinente ao estado da rede. Em razão disso, o algoritmo Slow Start inicia a conexão com uma janela de congestionamento de 1 MSS, sendo aumentada exponencialmente no decorrer da transmissão. O TCP utiliza o Slow Start no início de cada conexão ou no caso de *timeout* (tempo máximo de espera por uma confirmação). O algoritmo é resumido a seguir:

1. Início da conexão ou após o timeout $\longrightarrow$ cwnd = 1;
2. Cada recebimento de ACK $\longrightarrow$ cwnd = cwnd + 1.

A ocorrência de *timeout* é interpretada pelo TCP como um indício de alto grau de congestionamento da rede. Desta forma, o algoritmo reduz bruscamente a janela de congestionamento e, conseqüentemente, a taxa de envio dos pacotes.

b) *Congestion Avoidance*

Após a ocorrência do primeiro *timeout*, o algoritmo *Congestion Avoidance* é acionado, visando incrementar a cwnd de maneira mais conservadora, aumentada de 1 MSS cada vez que todos os segmentos da janela atual são entregues com sucesso. Já o decremento da cwnd é mais agressivo, reduzida à metade caso algum segmento se perca durante uma rodada de transmissão. Esta filosofia de funcionamneto, que adota incremento aditivos (neste caso, 1 MSS a cada vez que todos os segmentos da cwnd são entregues com sucesso), e decréscimos multiplicativos (o valor da cwnd é aqui reduzido à metade), é conhecido como AIMD (*Additive Increase/Multiplicative-Decrease*). O *Congestion Avoidance* é apresentado a seguir:

1. Cada recebimento de ACK $\longrightarrow$ cwnd = cwnd + $\frac{1}{cwnd}$

Na realidade, os algoritmos Slow Start e Congestion Avoidance são implementados em conjunto, sob o controle de uma variável denominada ssthresh (Slow Start threshold) da seguinte maneira:

1. Início da conexão $\longrightarrow$ cwnd = 1 e ssthresh = 0 : Slow Start;
2. Ocorreu *timeout* $\longrightarrow$ ssthresh = $\frac{cwnd}{2}$;
3. Se cwnd = 1 : reinicia Slow Start;
4. Se cwnd $<$ ssthresh $\longrightarrow$ cwnd manipulada por Slow Start;
5. Se cwnd $>$ ssthresh $\longrightarrow$ cwnd manipulada por Congestion Avoidance;

A Figura abaixo mostra o *Slow Start* atuando em conjunto com o *Congestion Avoidance*

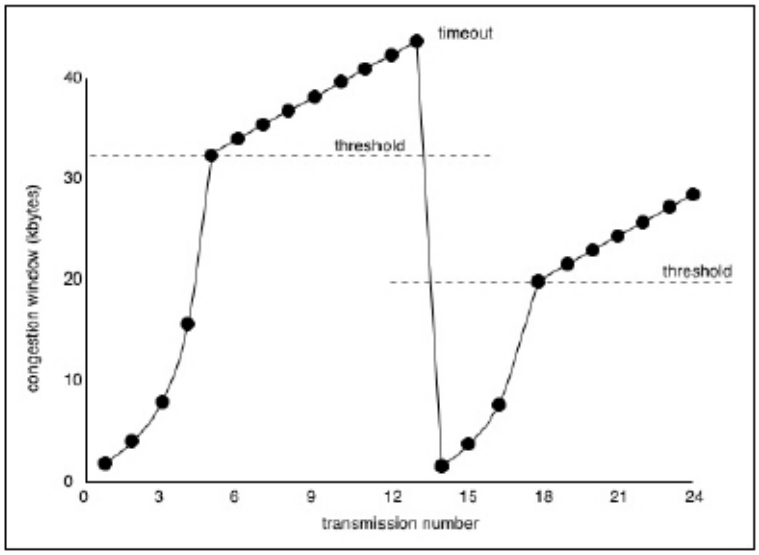

FIG. 2.6: Atualização da Janela de Congestionamneto TCP.

c) *Fast Retransmit*

A finalidade do *Fast Retransmit* é tornar mais eficiente o processo de retransmissão de pacotes. Este algoritmo estabelece que o recebimento de 3 (três) ACKs duplicados indica uma perda de pacote, devendo a fonte retransmiti-lo imediatamente. Desta forma, evita-se a longa espera pelo estouro do temporizador (*timeout*), agilizando a retransmissão.

d) *Fast Recovery*

O Fast Recovery é implementado em conjunto com o *Fast Retransmit*, visando a manipulação da cwnd em caso de congestionamento moderado. A descrição do algoritmo é resumida a seguir:

1. Ao receber 3 ACKs duplicados consecutivos ou ocorrer *timeout* $\longrightarrow$ ssthresh $= \frac{cwnd}{2}$;
2. Retransmite o pacote supostamente perdido: *Fast Retransmit*;
3. cwnd = ssthresh + 3;
4. A cada ACK duplicado recebido $\longrightarrow$ cwnd = cwnd + 1;
5. Ao receber o novo ACK do pacote retransmitido $\longrightarrow$ cwnd = ssthresh.

2.2.5 INTERAÇÃO ENTRE AS CAMADAS TRANSPORTE E INTERNET

A interação entre as camadas de transporte e Internet pode ser assim resumida: Quando a camada de transporte recebe um conjunto de dados da aplicação (mensagem), ela o particiona em segmentos. Cada segmento tem no máximo 64 Kbytes, mas, em geral, ele não chega a 1500 bytes (tamanho do quadro Ethernet). Na camada Internet, cada segmento entregue é então encapsulado em um ou mais datagramas IP. Na camada Internet da máquina destino o segmento original é remontado e entregue aos cuidados da respectiva camada de transporte, como ilustrado na Figura 2.7.

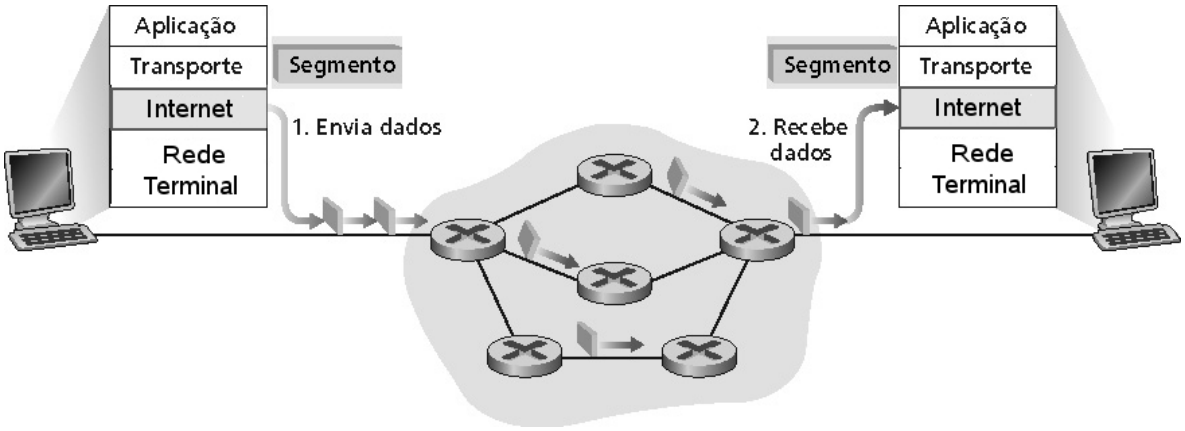

FIG. 2.7: Interação Camadas Transporte e Internet.

2.3 RESUMO

O controle de congestionamento TCP fica, como pôde ser observado, completamente transparente tanto para a aplicação, que utiliza como portal de acesso para as camadas inferiores os *sockets*, quanto para as atividades inerentes à camada Internet; é plenamente possível interferir de forma exclusiva no controle de congestionamento, mantendo as demais características da pilha TCP/IP constantes.

3 EVOLUÇÃO DO TCP

3.1 VISÃO GERAL

O TCP é um dos principais protocolos de transporte devido a sua predominância no conjunto de protocolos da Internet. A evolução do TCP perseguiu a maior escalabilidade deste protocolo. Para isso, as modificações propostas ao longo dos anos focaram o controle de congestionamento. Nas implementações mais antigas do TCP (POSTEL, 1981b), o emissor repassava para o canal de comunicação múltiplos segmentos até o limite da janela de fluxo anunciada pelo receptor. Posteriormente, surgiram três novas versões do TCP: TCP Tahoe (JACOBSON, 1988), dotado do *Congestion Avoidance* e *Slow Start*, o TCP Reno (JACOBSON, 1990), que trouxe o *Fast Retransmit* e introduziu o *Fast Recovery* no *Congestion Avoidance*, e o TCP New Reno (FLOYD, 2004), propondo uma otimização no *Fast Recovery*. Apesar de toda contribuição e utilidade das versões tradicionais do TCP, estudos comprovam (REZENDE, 2005) que o mesmo apresenta um desempenho muito restrito em redes de alta velocidade, nas quais o produto banda retardo (BDP) seja elevado, bem como em redes sem fio. Com o objetivo de aproveitar melhor os recursos destas redes, diversas variantes do TCP foram propostas. A seção a seguir apresenta um resumo dos principais representantes das variações TCP.

3.2 VARIANTES TCP

3.2.1 BIC (XU, 2004)

A idéia central do controle de congestionamento desta variante do TCP é a utilização do valor médio entre a janela de congestionamento imediatamente antes e depois de uma perda (w). Quando a "distância" da janela de congestionamento (cwnd) de w é maior do que um determinado valor, o incremento de cwnd é mais agressivo; quando cwnd está "perto" de w este incremento passa a ser mais suave. Além disso, para redes subutilizadas, onde o crescimento da janela de congestionamento passa a ser sistemático, esta variante implementa um algoritmo que aumenta esta janela mais agressivamente (exponencialmente).

3.2.2 CUBIC (XU, 2005)

Uma característica marcante nesta variante é a atualização da sua janela de congestionamento levando-se em consideração o tempo decorrido desde o último evento de perda, não privilegiando, assim, fluxos com RTT's maior ou menor. Sua função de crescimento da janela de congestionamento é semelhante a do BIC, possuindo, porém, um formato mais simples e maior cautela antes de tornar o incremento agressivo.

3.2.3 FAST TCP (JIN, 2004)

Utiliza uma sistemática baseada em quatro componentes: **controle de estimativa**, que fornece as informações necessárias às demais componentes (por exemplo, estima o RTTmin e RTTmed, integrantes da função de atualização da janela de congestionamento, calculada pela componente **controle de janela**); **controle de dados**, que seleciona quais segmentos, dentre os ainda não reconhecidamente transmitidos, devem ser enviados, e **controle de rajada** que regula a quantidades de dados a serem transmitidos a fim de evitar grandes rajadas de dados, as quais podem criar longas filas e aumentar a probabilidade de erros em massa.

3.2.4 HIGH SPEED TCP (HSTCP) (FLOYD, 2003)

Para altas taxas de perdas de pacotes, este protocolo se comporta de forma idêntica ao TCP-Reno; uma vez que os congestionamentos ocorram com menor freqüência, porém, a atualização do valor da janela de congestionamento passa a ser mais agressiva. Em suma, pode-se dizer que na fase de *congestion avoidance* (quando cwnd > ssthresh), o algoritmo AIMD é modificado de tal forma que o incremento e o decremento da janela de congestionamento variem de acordo com o tamanho da mesma.

3.2.5 SCALABLE TCP (STCP) (KELLY, 2003)

Seu mecanismo permite que a atualização da janela de congestionamento esteja em função apenas do RTT da conexão, sem levar em consideração o tamanho desta janela no momento da perda.

3.2.6 TCP HYBLA (CAINI, 2004)

Baseia o controle da janela de congestionamento em um RTT referência para conexão, eliminando a dependência entre o número de segmentos presentes nesta janela e o RTT real do enlace. Este protocolo simula conexões TCP em RTT´s pequenos.

3.2.7 TCP-ILLINOIS (LIU, 2006)

Todas as características do TCP New Reno são mantidas com exceção do AIMD, ou seja, no *congestion avoidance*, o acréscimo na janela de congestionamento será tanto maior quanto menor for o atraso estimado na fila dos roteadores, enquanto que o decréscimo será tanto mais agressivo quanto maior for este atraso.

3.2.8 TCP VEGAS (BRAKMO, 1994)

A idéia básica deste algoritmo é a utilização do RTT como indicador da iminência ou não de um evento de perda de pacotes. Quanto maior o RTT, maior é, em princípio, a carga nos roteadores, o que requer adaptações na atualização da janela de congestionamento. O HSTCP é considerado uma evolução deste algoritmo.

3.2.9 H-TCP (SHORTEN, 2004)

A janela de congestionamento deste algoritmo sofre influência tanto do número de perdas de pacotes ocorridas quanto do tempo decorrido desde o último evento de congestionamento. Sendo assim, quanto melhor for a resposta da rede ao fluxo de dados (longo período sem perdas), mais agressivo será o aumento da janela de congestionamento.

3.2.10 COMPOUND TCP (TAN, 2006)

Incorpora ao incremento do *congestion avoidance* do TCP tradicional uma componente baseada no atraso (RTT). Desta forma, agrega as vantagens dos algoritmos baseados em atraso (agressividade em redes subutilizadas) junto com aquelas inerentes aos baseados em perdas, apropriados para redes congestionadas.

3.2.11 TCP WESTWOOD (CASETTI, 2001)

Monitora continuamente os pacotes ACK´s para estimar o ERE (*Eligible Rate Estimation*), fundamental na atualização da janela de congestionamento e cuja técnica de

estimação está em aberto. Como ele usa a taxa estimada para atualizar a janela de congestionamento, este protocolo se mostra extremamente eficiente em redes *wireless*, onde perdas de pacotes, geralmente causadas por ruídos, requerem, na maioria das vezes, um aumento na quantidade de retransmissão e não uma redução do número de segmentos inseridos no meio.

3.2.12 TCP WESTWOOD+ (DELL'AERA, 2004)

Difere do Anterior, principalmente, por basear a realização de novas estimativas da ERE a cada RTT e não em recebimento de pacotes ACK´s apenas.

3.2.13 RESUMO DAS VARIANTES TCP

Segue abaixo uma tabela com as situações mais indicadas para aplicação de cada um destes protocolos e uma breve explicação de por que se espera uma boa resposta destes protocolos nestas situações:

Protocolo	Indicação	Explicação
BIC TCP	Fluxos com RTT´s diferentes e não TCP.	Atualização da janela de congestionamento baseada no tamanho da mesma. Agressivo com fluxos TCP.
CUBIC	Fluxos com longos RTT´s e diferentes entre si.	Janela atualizada conforme o tempo desde o último evento de perda.
Fast TCP	Redes com RTT's estáveis.	Fluxos com RTT's muito variável dificultam as estimativas próprias deste protocolo.
H-TCP	Redes não congestionadas	Se há muitas perdas, este protocolo se comporta conforme o TCP.
High Speed TCP	Fluxos com RTT's próximos	Os fluxos com RTT's mais curtos são privilegiados, deixando pouca banda para os de RTT's mais longos
Scalable TCP	Conexões com RTT homogêneo.	O tamanho da janela depende da frequência com que chegam os ACK's.
TCP Hybla	Próprio para enlaces de longo RTT.	A atualização independe do RTT.
TCP-Illinois	Redes de pequeno *jitter*.	Facilita a estimativa do atraso nos roteadores.
TCP Vegas	Redes de pequeno *jitter*.	Grandes oscilações do RTT dificultam a detecção da iminência de um congestionamento.
Compound TCP	Conexões com RTT's diferentes.	Insere uma componente de atraso no aumento da janela de congestionamento.
Westwood Westwood+	Redes *wireless*.	Pouco agressivo na redução da janela de congestionamento

TAB. 3.1: Variantes TCP e suas indicações.

4 ANÁLISE DA VIABILIDADE E METODOLOGIA PARA CONSTRUÇÃO DE UMA CAMADA DE TRANSPORTE ADAPTATIVA

4.1 VIABILIDADE

4.1.1 CONSIDERAÇÕES INICIAIS

Como já foi mencionado acima, a própria evolução da camada de transporte TCP buscou adaptar-se (infelizmente de maneira isolada), às novas demandas da rede. Antes de propor como seria uma camada de transporte adaptativa dentro das variantes TCP, analisemos mais uma vez a função de atualização da janela de congestionamento do TCP Reno (FIG.2.6): O gráfico desta função é linear acima da *threshold* e exponencial abaixo dela. Esta variação na forma da função de atualização da janela de congestionamento, ou seja, linear acima da *threshold* e exponencial abaixo dela, forma a base para o estudo da viabilidade de uma camada de transporte adaptativa entre as variantes TCP.

Como pôde ser observado, a função de atualização da janela de congestionamento do TCP Reno pressupõe uma adaptação da mesma de acordo com o *feedback* que a rede lhe proporciona. Do acima exposto, podemos concluir que o chaveamento entre funções de atualização da janela de congestionamento é algo inerente às versões tradicionais do TCP. Além disso, vários dos algoritmos que procuraram dar ao TCP uma maior escalabilidade concentram seus esforços na forma de atualização da janela de congestionamento; e não poderia ser diferente, pois é esta a função que vai determinar a quantidade de datagramas inseridos a cada rodada de transmissão na rede.

Fica claro, portanto, que podemos disponibilizar na camada de transporte um conjunto de funções que possam ser, de acordo com as condições pontuais da rede, escolhidas durante a troca de dados. Sendo assim, para uma camada de transporte TCP adaptativa, precisaríamos apenas trabalhar no controle de congestionamento da camada em questão.

Em suma, a alteração mais significativa para que as camadas de transporte tradicionais se tornem adaptativas está basicamente vinculada ao controle de congestionamento, totalmente transparente para as aplicações, o que torna possíveis implementações (veja detalhes no capítulo 8) do transporte adaptativo plenamente amigáveis às aplicações hoje existentes.

4.2 METODOLOGIA

Como foi visto, a crescente diversidade nas redes de comunicação requer protocolos distintos para melhor se realizar a troca de informações. Neste contexto, cabe um estudo para se verificar a viabilidade e as vantagens de uma camada de transporte adaptativa, capaz de adaptar-se às variações que podem ocorrer na rede durante a troca de dados. Este estudo será dividido em duas fases distintas: adaptação entre as variantes TCP e a adaptação quando da necessidade de emprego de um protocolo de transporte baseado em UDP.

A adaptação entre as variantes TCP é vista como uma opção muito mais abrangente, pois a variação do protocolo de transporte dentro das versões TCP não requer modificação alguma nos aplicativos, para os quais as atividades relativas ao transporte das mensagens continuará transparente. Testes práticos envolvendo diferentes algoritmos de transporte serão realizados para avaliação das vantagens de um algoritmo de transporte para outro, bem como para especificação de critérios que justifiquem o chaveamento entre estes algoritmos. Para ressaltar as vantagens de uma camada de transporte adaptativa, são apresentados alguns estudos de casos, nos quais utilizam-se as conclusões extraídas dos experimentos. Diante das vantagens apontadas pelos experimentos, apresentam-se os elementos necessários à implementação de uma camada de transporte que possa variar entre as variantes TCP.

Na segunda fase do trabalho, trata-se a adaptação do transporte baseada em UDP objetivando contemplar projetos específicos, em cuja gênese já haja a preocupação com a possibilidade de se trabalhar com protocolos de transporte otimizados para situações especiais (ex. Programa C2 em combate).

Os testes práticos se limitarão aos algoritmos de transporte presentes no 2.6.25 do linux, no qual estão disponíveis algumas das variante apresentadas na seção 3.2 (Reno, BIC, CUBIC, STCP, HSTCP , HTCP, VEGAS e WESTWOOD+); como o HSTCP é considerado uma evolução do VEGAS e o WESTWOOD+ é indicado para redes *wireless*, os testes englobarão apenas as versões Reno, BIC, CUBIC, STCP, HSTCP , HTCP do TCP. No *kernel* em questão o TCP Reno é na verdade uma implementação do New Reno. Daqui em diante, a menos que haja alguma ressalva, todas as vezes que nos referirmos ao TCP Reno estaremos nos reportando ao TCP New Reno.

5 DESEMPENHO DAS VARIANTES TCP

Para uma avaliação prática das versões TCP de alta velocidade foram realizados *downloads* de arquivos, cujos tamanhos produzem um tempo de transferência de quatro a seis minutos através do TCP Reno, variando-se o protocolo de transporte e a capacidade do enlace. Para cada *clock rate* ajustado, acrescentou-se, através do Netem (Apêndice 1), atrasos (200, 400, 600, 800 e 1000 ms), ao RTT. Fixados o RTT e o protocolo de transporte (Reno, BIC, CUBIC, STCP, HSTCP e HTCP), mediu-se o tempo de download. O download foi realizado de um servidor apache versão 2.0, utilizando-se como cliente o FireFox 3.0.1. A partir destes testes podemos escolher qual dos protocolos se comportou melhor, isto é, proporcionou menor tempo de *download*. Os testes foram realizados no laboratório de redes do IME. As máquinas utilizadas seguem a seguinte configuração:

- Hardware: Intel(R) Pentium(R) 4 HT 3.00 GHz;
- SO: Suse 11, kernel 2.6.25.11-0.1-pae;
- Memória: 512 MB;
- Placa de Rede: ASUSTeK 82540EM Gigabit Ethernet;
- HD: Samsung(R) SP 0802N 7200 rpm IDE ATA 133;
- Partição de Swap: 1 Gb.

Alguns outros parâmetros foram alterados através do comando sysctl (Apêndice 2), de acordo com (MICHEL, 2008).

5.1 EXPERIMENTO COM ENLACES DE 1200 BITS POR SEGUNDO A 1 MEGABIT POR SEGUNDO

A flexibilidade na velocidade do enlace foi garantida pela infra-estrutura da Figura 5.1: Duas máquinas, grecia01, na rede 4 e grecia11, na rede 6, se comunicavam, de forma exclusiva, através de um enlace serial, que fazia a ponte entre os roteadores Mykonos (Cisco(R)2611) e Santorini (Cisco(R)2621), e cuja velocidade foi configurada para os valores 1.2, 56, 512 e 1000 Kbps.

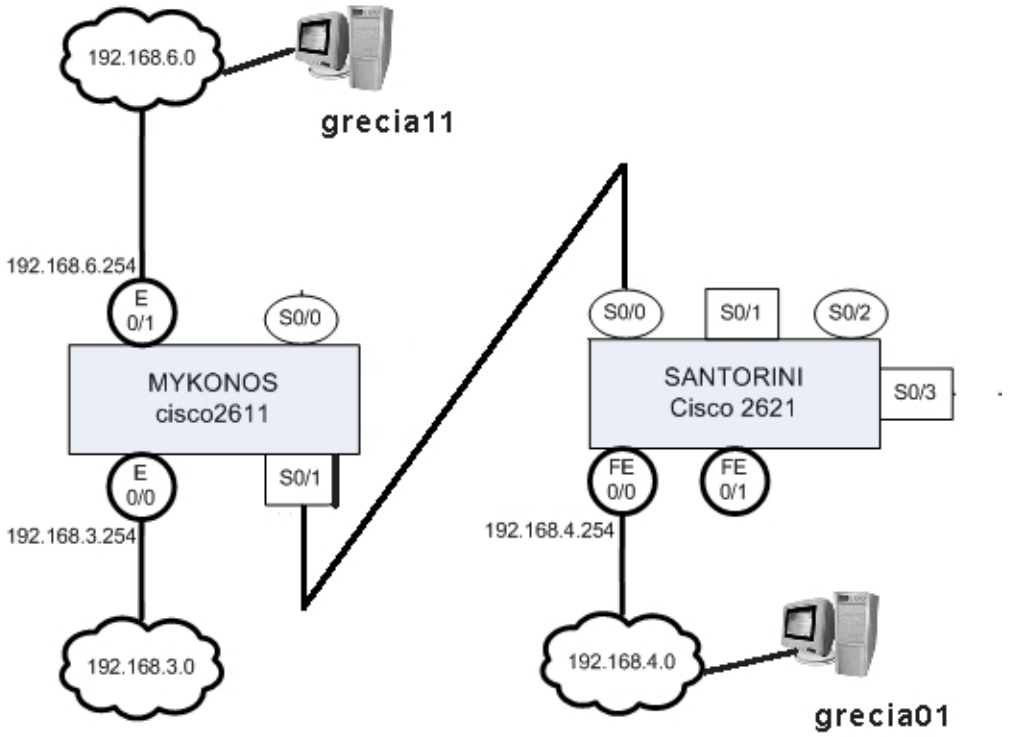

FIG. 5.1: Estrutura para Enlaces de 1200bps a 1Mbps.

- **Enlace de 1200 bps**

  Com a velocidade do enlace serial ajustada para 1200 bps, foi levantado o tempo médio entre três *downloads* de um arquivo de 40 Kbytes para cada um dos acréscimos no RTT considerados, chegando-se ao gráfico da Figura 5.2. Como se pode observar, nestas condições o TCP Reno se comportou de forma extremamente satisfatória, conseguindo em alguns casos um comportamento melhor do que as outras abordagens. Para o menor RTT (1290 ms), o Reno alcançou um tempo de *download* menor do que aquele obtido com os outros protocolos, principalmente em relação ao CUBIC. Conforme aumentamos o RTT, os protocolos, com exceção do HTCP, praticamente se igualaram durante os testes, com ligeira vantagem para o Reno. O HTCP começa a se destacar negativamente de um RTT de 2090 ms em diante.

- **Enlace de 56 Kbps**

  Agora os *downloads* realizados transferiram um arquivo de 2.7 Mbytes. Os tempos de *download* para cada RTT estão sumarizados no gráfico da Figura 5.3. Mais uma vez o TCP Reno se mostra uma excelente escolha para este tipo de configuração. Até

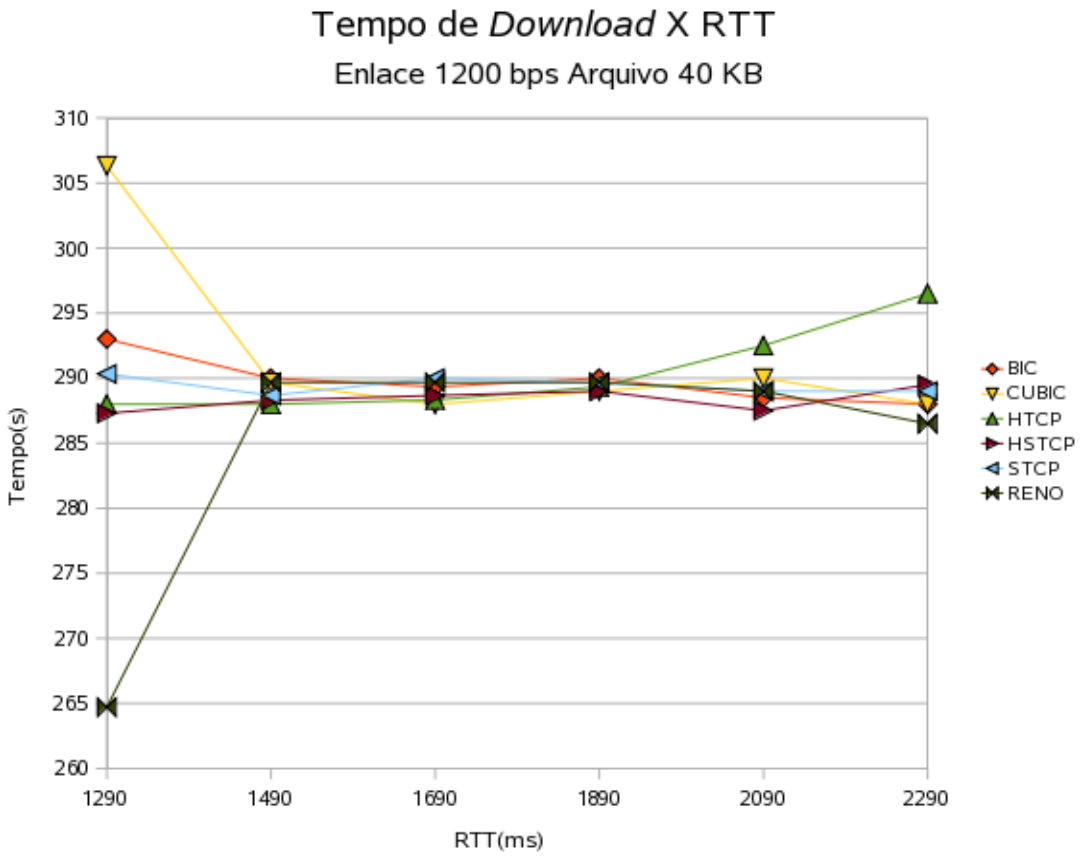

FIG. 5.2: Desempenho dos Protocolos: 1200 bps.

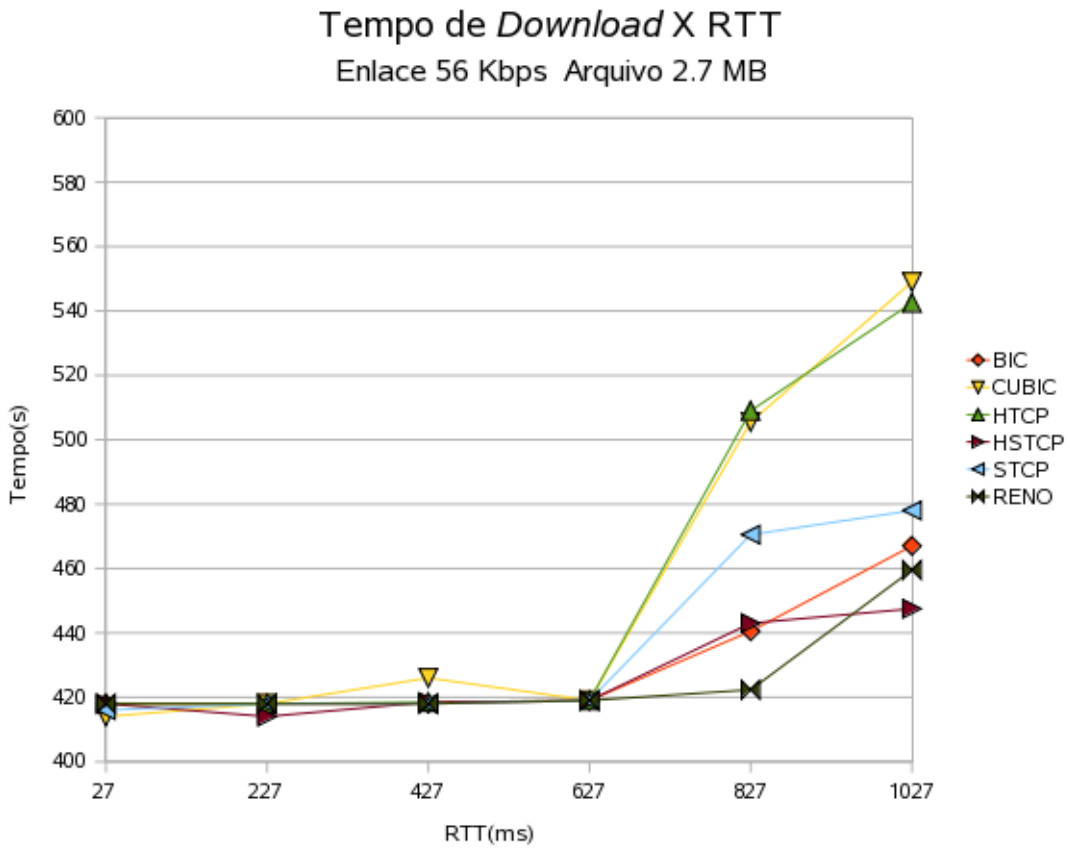

FIG. 5.3: Desempenho dos Protocolos: 56 Kbps.

o RTT de 627 ms os protocolos se equivalem; a partir daí, observa-se uma grande vantagem dos protocolos BIC, HSTCP e Reno em relação aos protocolos STCP, HTCP e CUBIC estes dois últimos com desempenhos semelhantes, fornecendo os maiores tempos de *download*. O desempenho global mostra ligeira vantagem para o Reno nesta fase dos testes.

- **Enlace de 512 Kbps**

  O tamanho de arquivo considerado nesta rodada de teste foi de 17 MB, o que resultou no gráfico da Figura 5.4. Novamente destaque positivo para o desempenho

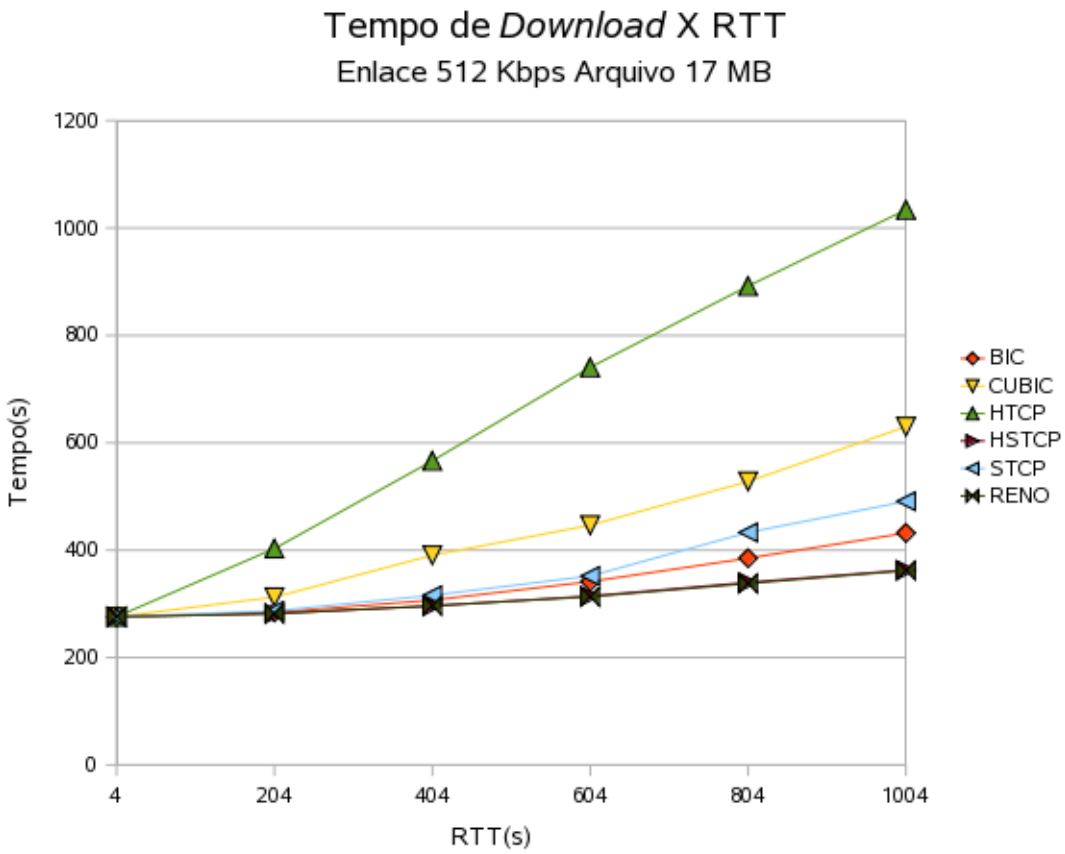

FIG. 5.4: Desempenho dos Protocolos: 512 Kbps.

do TCP Reno. Para esta fase dos testes o HTCP impôs um tempo de *download* muito acima dos outros. Os protocolos de melhor comportamento (praticamente idênticos) são o Reno e o HSTCP. Os demais protocolos podem ser ordenados de forma crescente em relação ao tempo de *download* para cada um dos valores de RTT considerados da seguinte forma: BIC, STCP e CUBIC.

- **Enlace de 1 Mbps**

  Para um arquivo de 30 MB, chega-se ao gráfico da Figura 5.5. Outra vez o TCP Reno

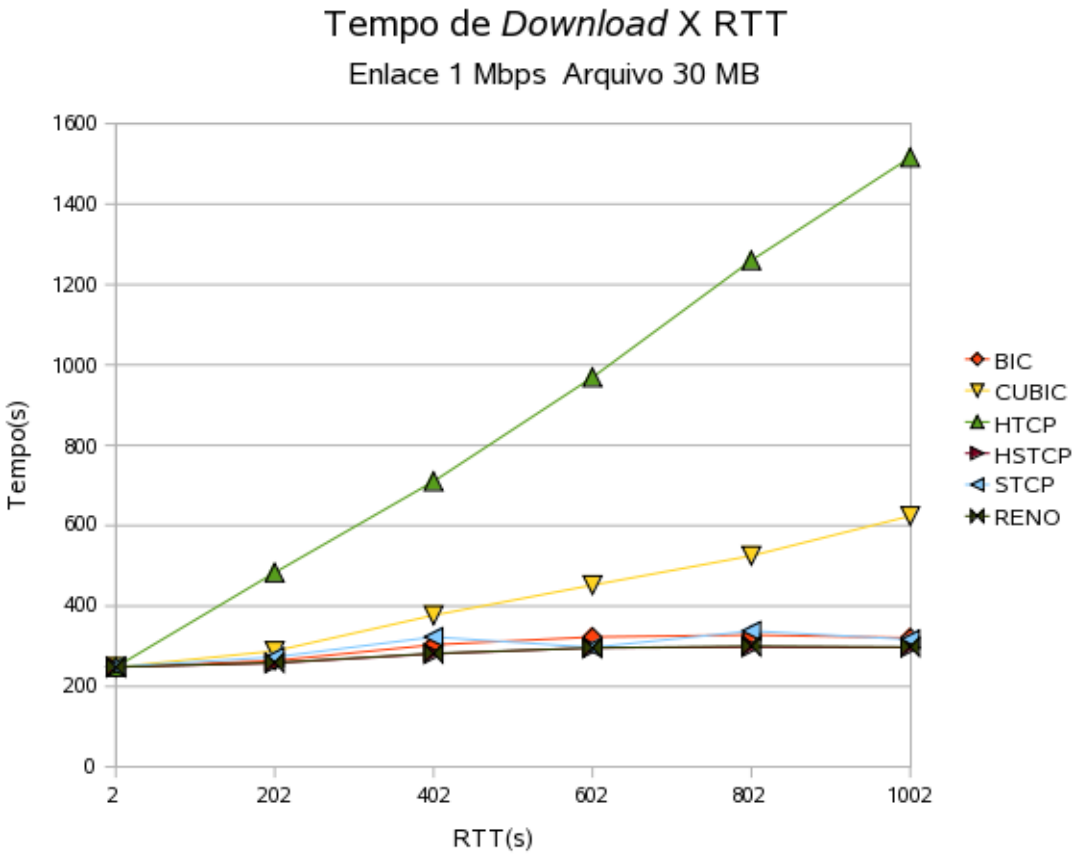

FIG. 5.5: Desempenho dos Protocolos: 1 Mbps.

constitui uma excelente opção. Como no caso anterior, o HTCP atinge os maiores tempos de *download*; em seguida vem o CUBIC, mas com tempos de transferência bem menores. Os protocolos Reno, STCP, HSTCP e BIC tiveram desempenhos muito semelhantes, proporcionando os menores tempos de *download*.

5.2 EXPERIMENTO COM ENLACE DE 10 MEGABITS POR SEGUNDO

Para ligar duas máquinas a 10 Mbps, utilizou-se a configuração ilustrada na Figura 5.6. Como pode-se observar nesta figura, as redes 3 e 6 se interligam por interfaces Ethernet

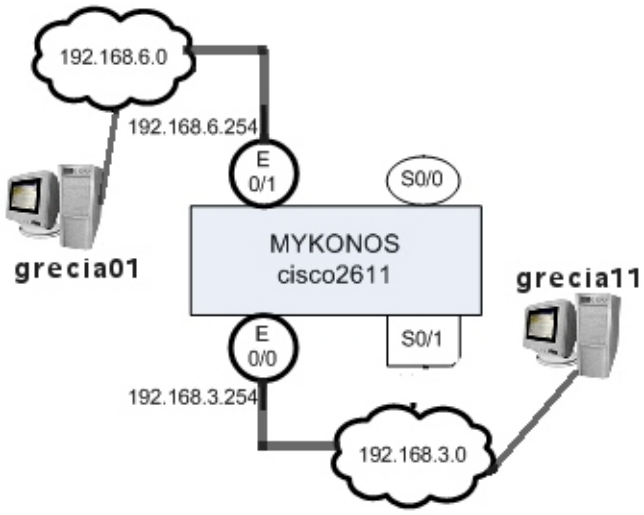

FIG. 5.6: Estrutura para Enlace de 10 Mbps.

(velocidade de 10 Mbps) do roteador Mykonos, dedicado ao experimento. Durante esta fase um arquivo de 300 MB foi transferido da máquina grecia01, na rede 6, à máquina grecia11, na rede 3, para o levantamento dos pontos do gráfico da Figura 5.7. Como vemos, o TCP Reno continua entre os protocolos de melhor desempenho. Mais uma vez o HTCP é o destaque negativo nesta configuração, seguido pelo CUBIC que, apesar disto, mostra-se bem mais eficiente do que o HTCP. Os protocolos STCP e BIC demonstram um comportamento intermediário entre os protocolos considerados. Os menores tempos de *download* estão associados aos protocolos Reno e HSTCP, os quais mantêm suas respostas praticamente idênticas durante todas as situações consideradas nesta fase.

5.3 EXPERIMENTO COM ENLACE DE 100 MEGABITS POR SEGUNDO

Como estamos trabalhando com placas FastEthernet concentradas em um *switch* Catalyst(R) 2950, através de cabos par trançado da categoria 5e, colocamos duas máquinas em uma mesma sub-rede para alcançarmos a velocidade de 100Mbps. Um arquivo de 1 GB, trocado entre as máquinas grecia01 e grecia 11, ambas na rede 6, serviu de base para a construção do gráfico da Figura 5.8. Não se pode deixar de destacar que, nesse enlace, a partir de um acréscimo de 600 ms no RTT, o TCP Reno passa a ter um desempenho abaixo de todas as outras variantes TCP consideradas. Destaque positivo para o CUBIC-TCP. Até o RTT de 400ms os desempenhos dos protocolos são idênticos, mas, para RTT´s acima deste valor o Reno é apontado pelos testes como a pior opção, seguido pelo HSTCP. Agora o HTCP, junto com o CUBIC e o BIC, constituem as melhores opções, seguidos

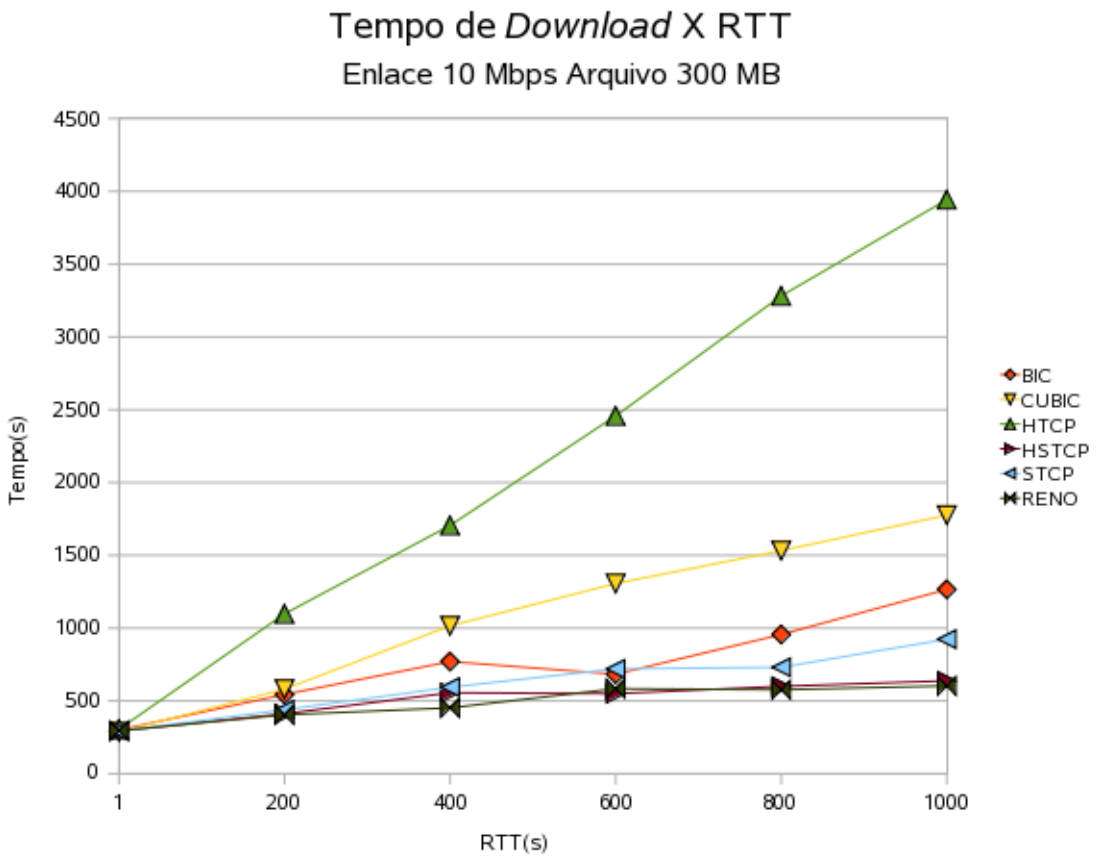

FIG. 5.7: Desempenho dos Protocolos: 10 Mbps.

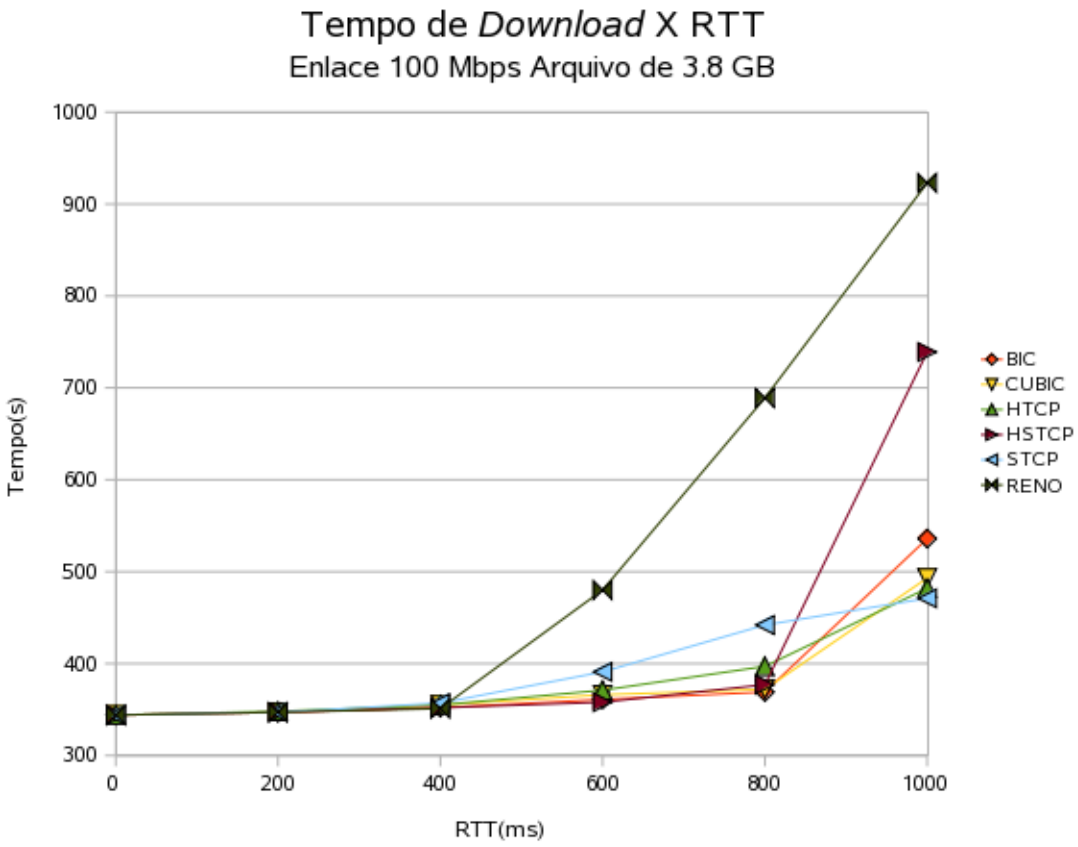

FIG. 5.8: Desempenho dos Protocolos: 100 Mbps.

de perto pelo STCP. Entretanto, o CUBIC obtem uma ligeira vantagem média quando considerados todos os pontos.

5.4 EXPERIMENTO COM ENLACE DE 1 GIGABIT POR SEGUNDO

As máquinas grecia01 e grecia11 foram diretamente ligadas por um cabo par trançado da categoria 5e, pois duas placas FastEthernet conectadas por um cabo desse tipo trabalham a uma velocidade de 1 Gbps (FIG.5.9). Um arquivo de 5 GB foi transportado

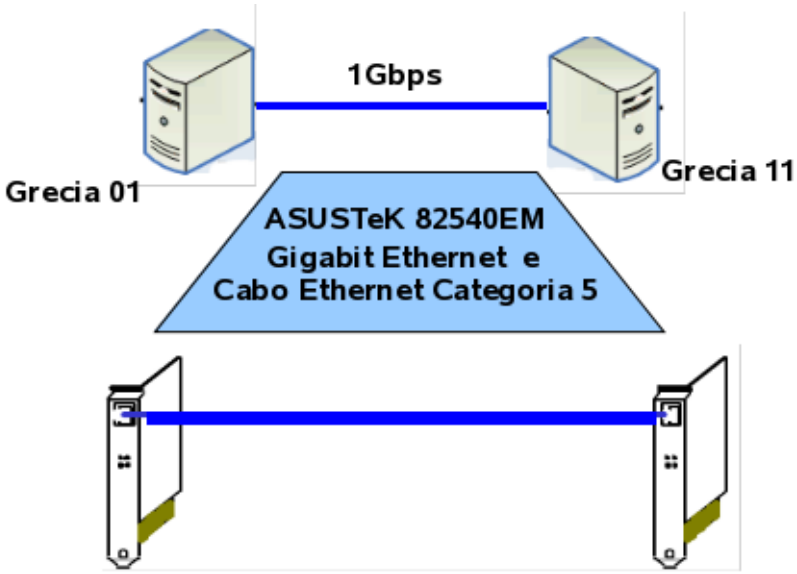

FIG. 5.9: Estrutura para Enlace 1 Gbps.

por cada um dos protocolos a cada acréscimo no RTT, de acordo com o gráfico da Figura 5.10. A partir de um acréscimo de 400 ms no RTT, o desempenho do TCP Reno começa a se degradar. Destaque positivo para o BIC TCP nesta configuração. Os menores tempos de *downloads* nesta configuração estão associados ao BIC e ao STCP, com o BIC conseguindo alcançar tempos um pouco menores. Entre 400 e 800 ms os protocolos em ordem decrescente em relação ao tempo de *download* são Reno, HTCP, HSTCP e CUBIC. Em 1000ms, porém, a ordem passa a ser HTCP, CUBIC, Reno e HSTCP.

5.5 PARALELO COM A LITERATURA

Verdadeiramente, a adoção de um protocolo de transporte preparado para redes de alto produto banda retardo pode aproveitar melhor a capacidade fornecida pelo enlace. Entretanto, não há um consenso entre os estudiosos sobre qual seria a abordagem de transporte

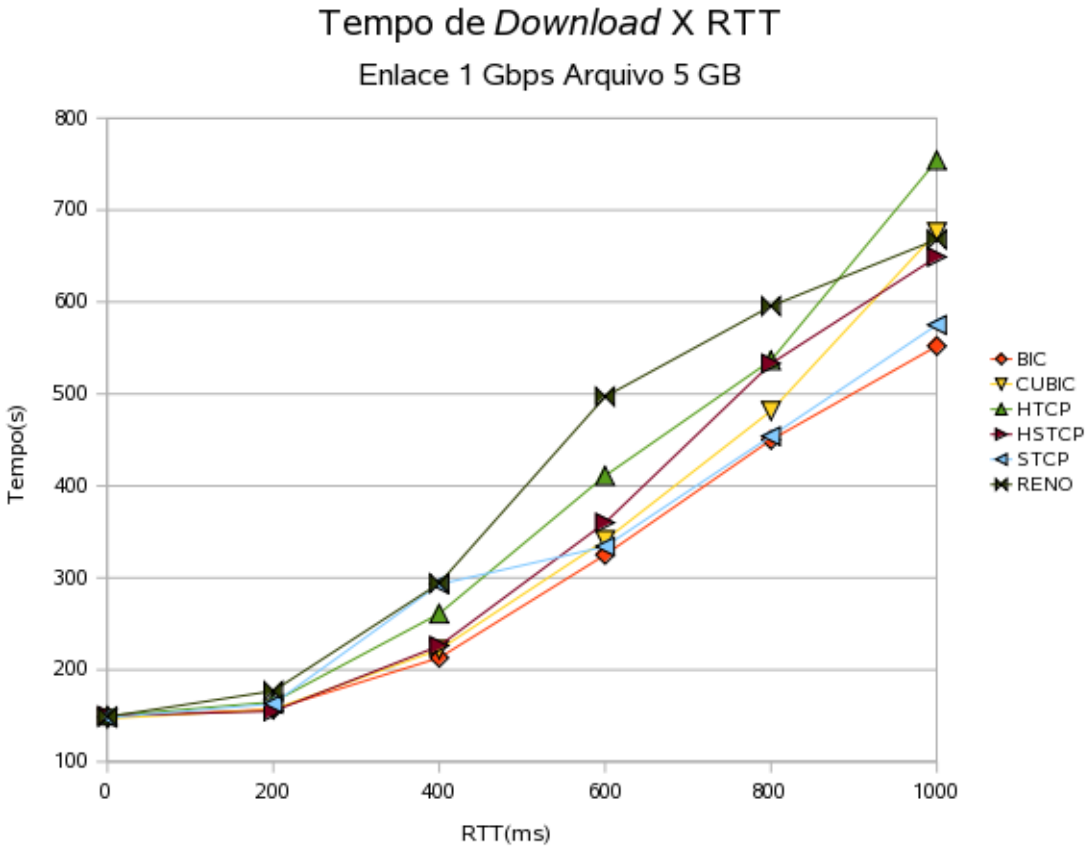

FIG. 5.10: Desempenho dos Protocolos: 1 Gbps.

mais escalável no que diz respeito ao aproveitamento do enlace. Em (MICHEL, 2007), por exemplo, o HTCP é tido como protocolo altamente escalável; o CUBIC e o STCP são destacados negativamente sob este aspecto no referido trabalho. Já o *Advanced Computing for Science Department* (MICHEL, 2008) sugere o HTCP e o CUBIC como as melhores opções em redes com BDP elevado. Na prática, porém, para um enlace de 1 Gbps, o HTCP demonstrou um desempenho muito aquém de algumas outras variantes, inclusive do próprio CUBIC, cuja performance está abaixo do BIC e do STCP, por exemplo, estes dois últimos apontados como protocolos de excelente escalabilidade por (XU, 2004), fato comprovado pelos experimentos.

6 CRITÉRIOS DE ADAPTAÇÃO ENTRE AS VARIANTES TCP

Neste capítulo é feita uma análise mais detalhada dos dados obtidos nos experimentos. Baseados nesta análise, apresenta-se de forma mais concreta a sistemática de adaptação entre as variantes TCP, baseando nossas decisões no RTT e velocidade do enlace.

6.1 ANÁLISE DOS EXPERIMENTOS COM ENLACES DE ATÉ 10MEGABITS POR SEGUNDO

Os dados extraídos dos experimentos deixam claro que, para enlaces de velocidade até 10 Mbps, a versão Reno constitui uma excelente opção, independente do RTT. Esta conclusão era esperada, pois os enlaces em questão constituem o "nicho" do Reno, ou seja, estamos nos enlaces em função dos quais o Reno baseou toda sua evolução; a utilização de uma outra abordagem nestes casos pode, inclusive, comprometer o desempenho da transmissão, basta observarmos o desempenho do HTCP nos gráficos correspondentes. Como segunda opção em termos de desempenho, temos o BIC TCP que, em geral, se comportou melhor do que as outras abordagens estando, porém, abaixo do TCP-Reno.

6.2 ANÁLISE DO EXPERIMENTO COM ENLACE DE 100 MEGABITS POR SEGUNDO

Para enlaces de velocidade de 100 Mbps o Reno manteve-se entre as abordagens de melhor desempenho até um acréscimo de 600 ms no RTT, a partir do qual o CUBIC-TCP e o STCP passaram a ser os protocolos de transporte mais indicados, com ligeira vantagem para o CUBIC, quando considerados todos os demais pontos até 1000 ms. Os gráficos mostram que o tempo de *download* pode ser reduzido em até 46,7%, escolhendo-se o protocolo de transporte adequado (CUBIC em vez do Reno). Na Tabela 6.2 os diversos ganhos percentuais proporcionados pelo protocolo CUBIC em relação ao Reno ($\frac{(T_{Reno} - T_{CUBIC}) * 100}{T_{Reno}}$), onde T_X representa o tempo de *download* utilizando-se o protocolo X durante os experimentos para o enlace em questão), são sumarizados a seguir:

RTT(ms)	Ganho Percentual (CUBIC sobre Reno)
600	27,7
800	46,0
1000	46,7

TAB. 6.1: Ganhos Percentuais do CUBIC em Relação ao Reno - Enlace de 100 Mbps.

6.3 ENLACE DE 1 GIGABIT POR SEGUNDO

Agora, a partir de um acréscimo de 400 ms o Reno é superado pelos protocolos BIC e STCP. Um fato interessante neste tipo de enlace é que o Reno conseguiu um desempenho melhor ou igual ao de algumas variantes (CUBIC, HTCP e HSTCP) que, teoricamente, deveriam superá-lo neste tipo de cenário. Como o BIC se destacou nesta fase, vamos sumarizar os ganhos percentuais que ele proporcionou em relação ao Reno na Tabela 6.3

RTT(ms)	Ganho Percentual (BIC sobre Reno)
400	27,5
600	34,6
800	24,4
1000	17,3

TAB. 6.2: Ganhos Percentuais do BIC em Relação ao Reno - Enlace de 1 Gbps.

6.4 ESCOLHENDO A MELHOR ABORDAGEM

Sendo assim, baseados nos nossos testes, apresentamos a Tabela 6.3, que sintetiza de forma prática e objetiva a análise dos dados levantados nos testes. Esta tabela serve de parâmetro para o chaveamento da melhor função de atualização da janela de congestionamento durante as atividades de uma camada de transporte TCP adaptativa.

Velocidade do Enlace	RTT	Protocolo Indicado
Até 10 Mbps	Qualquer	Reno
100 Mbps	$\leq$ 400 ms	Reno
	$>$ 400 ms	CUBIC
1Gbps	$\leq$ 200 ms	Reno
	$>$ 200 ms	BIC

TAB. 6.3: Condições de emprego dos protocolos.

7 ESTUDO DE CASOS

Agora, para avaliarmos de forma mais precisa as vantagens de uma camada de transporte adaptativa vamos montar cenários (baseado em situações reais), nos quais poderemos ver de forma mais nítida os benefícios que um transporte adaptativo pode nos trazer na prática.

7.1 CENÁRIO OPERACIONAL

Suponha que durante uma operação militar, onde o Módulo de Telemática está sendo empregado, o sistema "olhos da águia" (sistema de reconhecimento aéreo com captura de imagem), gere um filme (3,3 GB) que retrate a situação de uma determinada área de interesse. Há uma solicitação de que o tal filme seja transmitido do local da operação (suponhamos Manaus) para Brasília (Comando Geral das Operações). Existem duas possibilidades de enlace entre Manaus e Brasília: um limitado por um cabo Ethernet (banda 10 Mbps e RTT de 30 ms) e outro estabelecido por um satélite GEO (AI3, 2009) (banda da ordem de 100 Mbps e RTT de 600 ms), extremamente instáveis devido às dificuldades impostas pela natureza da operação. Sendo assim, quanto mais rápida uma informação for transmitida melhor, pois a qualquer momento pode-se perder a ligação entre os postos considerados. Como foi visto nos testes anteriores, uma das melhores opções de protocolo de transporte para o enlace Ethernet é o Reno. Já para um enlace de 100 Mbps com RTT da ordem de 600 ms, os mesmos experimentos revelaram que o CUBIC é a melhor opção. Para obtenção de dados mais concretos foram feitos, no laboratório de redes do IME, *downloads* de dois arquivos, um de 300 MB e outro de de 3 GB variando-se os enlaces (ora Ethernet com 30 ms de RTT, ora FastEthernet com 600 ms de RTT), chegando-se então à Tabela 7.1. Suponhamos ainda que o comportamento dos protocolos nos enlaces de satélite será semelhante ao que eles apresentaram no FastEthernet (100 Mbps/RTT de 600 ms), já que os satélites GEO também possuem velocidade de transmissão da ordem de 100 Mbps e RTT's de 600 ms. Assumiremos que, logo após a central de comando em Brasília receber os primeiros 300 MB do arquivo solicitado pela ligação Ethernet, o enlace por satélite torna-se, por acaso, disponível e que, após 10 minutos do início do *download*, ambos os enlaces saem do ar.

Enlace	Tamanho do Arquivo	Protocolo	Tempo(s)
Ethernet (10 Mbps/RTT 30 ms)	300 MB	Reno	300
		CUBIC	329
FastEthernet (100 Mbps/RTT 600 ms)	3 GB	Reno	346
		CUBIC	277

TAB. 7.1: Desempenho esperado dos protocolos CUBIC e Reno em enlaces Ethernet e satélite (GEO).

Vamos analisar o *download* dos 3,3 GB caso fosse feito com o protocolo de transporte estático e caso houvesse possibilidade de adaptação:

- Somente Reno

  Pelos dados da tabela, o *download* duraria 646 segundos (300 segundos para os 300 MB e 346 segundos para os 3 GB restantes), isto é, 10 minutos e 46 segundos de *download*. A missão, portanto, não teria sido cumprida.

- Somente CUBIC

  Da mesma forma, mantendo-se o CUBIC TCP, o *download* levaria 606 segundos (329 segundos mais 277 segundos), isto é, 10 minutos e 6 segundos. Novamente o objetivo não teria sido atingido.

- Chaveando entre Reno e CUBIC

  Agora vamos supor que durante os 300 MB iniciais tenhamos utilizado o Reno e que a camada de transporte alinhe seu controle de congestionamento com o CUBIC quando o enlace por satélite torna-se disponível. Nesse caso, o arquivo seria entregue em 577 segundos (300 mais 277 segundos), que equivalem a 9 minutos e 37 segundos, garantindo assim o êxito da missão.

7.2 REALIZAÇÃO DE *BACKUPS*

Vamos imaginar agora uma grande multinacional, localizada em Belém do Pará que realize espelhamento de toda a sua massa de dados (3 GB), para uma máquina localizada em seu país sede (*backup* completo), por um enlace satélite com as mesmas características mencionadas anteriormente. Para outra máquina localizada em São Paulo, cujo enlace

é limitado por uma ligação 10BaseT com RTT médio de 30 ms, deve ser realizado um *backup* parcial, ou seja, procede-se o espelhamento dos arquivos modificados nas últimas duas semanas (em média 300 MB). Dez *backup* de cada tipo devem ser realizados por dia. É desejável que as operações de *backup* sejam realizadas o mais rápido possível, pois as mesmas consomem grande parte dos recursos da rede em detrimento das atividades próprias do negócio da empresa. Segundo os experimentos, para os *backups* completos o CUBIC seria a melhor opção de transporte, já os arquivos modificados durante as duas últimas semanas devem ser transportados para São Paulo pelo Reno. Vamos analisar de forma prática os ganhos, caso a estação responsável pelo espelhamento pudesse adaptar seu modo de transportar os dados; para isso, tomaremos por base os mesmos dados da tabela 7.1.

- Somente Reno

  A tabela 7.2 nos dá uma perspectiva do tempo total que seria gasto para atender os 20 *backups* Verifica-se que o tempo total para realizar-se os espelhamentos seria

Tipo de *backup*	Tamanho do Arquivo	Duração (s)
10 parciais	300 MB	3000
10 completos	3 GB	3460
Total		6460

TAB. 7.2: Duração do processo de *backup* com uso exclusivo do Reno.

  de 6460 segundos, que totalizam 1 hora 47 minutos e 40 segundos.

- Somente CUBIC

  Tomando o mesmo procedimento para o CUBIC chegamos a Tabela 7.3: Sendo

Tipo de *backup*	Tamanho do Arquivo	Duração(s)
10 parciais	300 MB	3290
10 completos	3 GB	2770
Total		6060

TAB. 7.3: Duração do processo de *backup* com uso exclusivo do CUBIC.

  assim, mantendo-se o protocolo CUBIC estático, o tempo gasto é de 1 hora e 41 minutos.

- Chaveamento entre Reno e CUBIC

  Se nas duplicações de dados parciais o protocolo utilizado fosse o Reno e nas completas o CUBIC, os tempos de atendimento seriam conforme os constantes da Tabela 7.4: Perfazendo um tempo de 1 hora 36 minutos e 10 segundos. Como podemos

Tipo de *backup*	Tamanho do Arquivo	Duração(s)
10 parciais	300 MB	3000
10 completos	3 GB	2770
Total		5770

TAB. 7.4: Duração do processo de *backup* com adaptação entre o Reno e o CUBIC.

observar, com o transporte adaptativo teríamos os recursos da rede totalmente dedicados aos negócios da empresa durante praticamente 5 minutos a mais (tempo suficiente para atender um *backup* completo), em relação a um protocolo de transporte que se mantivesse apenas no CUBIC TCP. Este tempo livre é ainda mais expressivo (praticamente 11 minutos!), quando tomado em relação a um transporte de dados que insistisse no Reno. Portanto, a cada processo diário de *backup*, economiza-se 5 minutos em relação a um protocolo de transporte estático no CUBIC. Em uma semana as atividades próprias do negócio da empresa teriam 35 minutos de exclusividade na sua infra-estrutura de dados. Em um mês esse tempo saltaria para 2 horas e 30 minutos. Durante um ano de exercício, 1 dia 6 horas e 25 minutos! Em relação a uma camada de transporte TCP Reno os valores seriam: 11 minutos a mais para cada processo diário de *backup*; 1 hora e 17 minutos a cada semana; 5 horas e 30 minutos a cada mês e 2 dias e 18 horas e 55 minutos por ano! Este último estudo de casos chama atenção também em relação ao processamento das máquinas transmissoras, pois as mesmas tarefas podem ser realizadas com uma considerável economia de tempo, proporcionando uma menor sobrecarga nas estações, as quais podem alocar seu poder de processamento para outras tarefas quando do término das transmissões.

8 ARQUITETURA E IMPLEMENTAÇÃO

Agora que os testes revelaram que podemos obter grandes vantagens com o emprego de uma camada de transporte adaptativa, modelaremos uma camada de transporte com um módulo responsável pelo chaveamento entre as abordagens TCP. Estudaremos o padrão *observer* e *state* (GAMMA, 1995) e qual a utilidade destes para o referido módulo. Apresentaremos também uma implementação externa, através de *script*, da camada de transporte adaptativa que serve de modelo conceitual para a compilação de sistemas operacionais que desejem disponibilizar no seu *kernel* uma camada deste tipo.

8.1 PADRÕES DE PROJETO: VISÃO GERAL

Um Padrão de projeto é uma forma de solução que permite aos programadores resolverem problemas semelhantes. A prática de se documentar padrões para resolução de problemas de mesma natureza é muito comum, principalmente nas atividades de engenharia. No desenvolvimento de *softwares* não é diferente; nele encontramos a todo instante problemas, cujas características são comuns a algum outro previamente solucionado. O emprego de soluções padronizadas dá maior velocidade ao desenvolvimento, favorece o reuso e facilita a comunicação entre os profissionais da área. Este trabalho não tem por objetivo fazer uma análise aprofundada dos padrões de projeto já catalogados, caso o leitor queira tais informações, poderá encontrá-las em (GAMMA, 1995).

8.2 O PADRÃO *OBSERVER*

O propósito deste padrão é definir uma dependência um-para-muitos entre objetos para, quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente. A estrutura de classes deste padrão é ilustrada pela Figura 8.1. Tanto o sujeito como os objetos estão associados a uma interface. A interface do sujeito informa, através do método *Notificar*, as mudanças nas variáveis de estado contidas no *Sujeito_Concreto*. Acionados pela notificação, os observadores atualizam seus estados (método *Atualizar*), de acordo com os novos valores dos atributos do *Sujeito_Concreto*, obtidos pelo método *ObterEstado*.

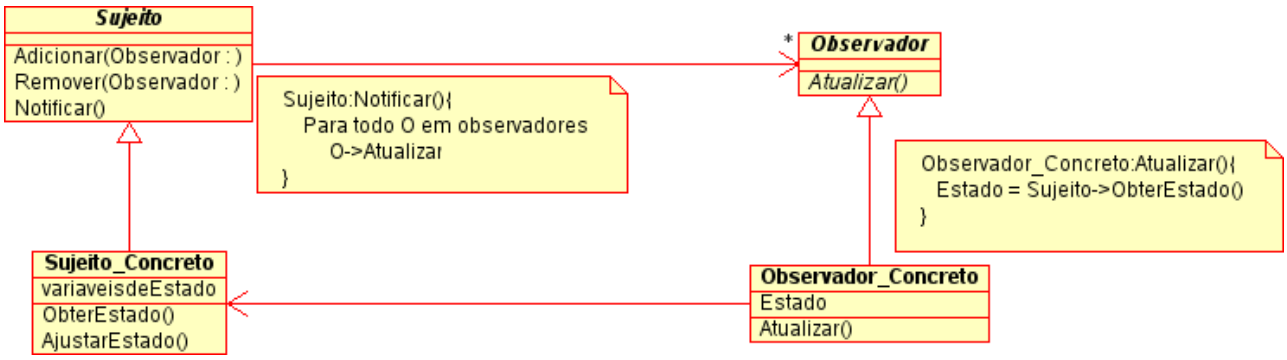

FIG. 8.1: Padrão *Observer*.

8.3 PADRÃO *STATE*

Quando desejamos que um objeto altere o seu comportamento, a partir da mudança nos valores dos seus atributos, devemos empregar o padrão *state*. O objetivo deste padrão compreende a utilização de objetos para representarem estados e polimorfismo para tornar transparente a execução de tarefas dependentes destes estados, como sugere a estrutura da Figura 8.2:

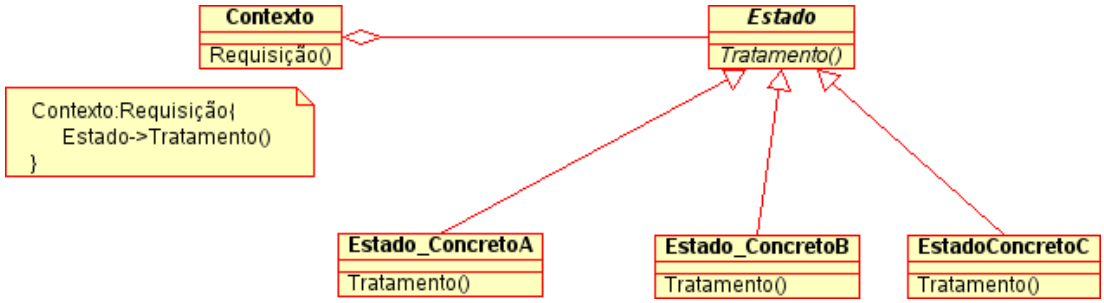

FIG. 8.2: Padrão *State*.

O objeto *Contexto* possui uma classe abstrata *Estado* que é concretizada em algum dos estados (*Estado_ConcretoA*, *Estado_ConcretoB*, *Estado_ConcretoC*, etc). De acordo com a situação, isto é, quando há uma mudança na situação de interesse, o objeto *Contexto* seleciona uma das implementações da classe *Estado* preparada para esta situação específica. Sendo assim, cada vez que o objeto contexto é acionado, ele reage com o método *Requisição*, que invoca o método *Tratamento*, do *Estado* concretizado pelo próprio *Contexto*.

8.4 ARQUITETURA DA CAMADA DE TRANSPORTE ADAPTATIVA

Basicamente, esta camada deve manter a mesma interface com a aplicação (*socket* no caso da pilha IP); por esta interface os dados seriam, como normalmente se faz, "bufferizados", aguardando a sua inclusão na janela de congestionamento para transmissão. Exatamente neste ponto, ou seja, na função que define a quantidade de segmentos que devem compor a janela de congestionamento, deve haver um conjunto de abordagem que estarão à disposição da camada de transporte para serem utilizadas no controle de congestionamento (FIG. 8.3).

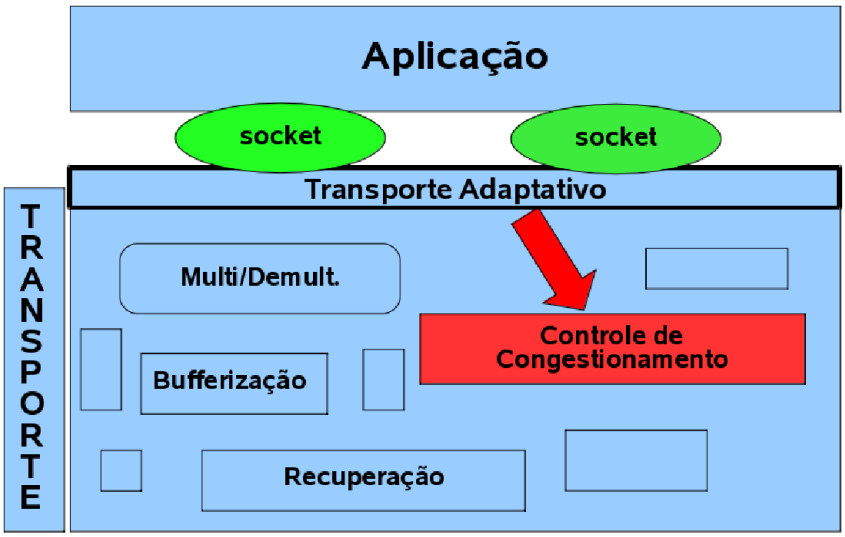

FIG. 8.3: Transporte adaptativo.

Uma visão mais detalhada do funcionamento desta camada é ilustrado na Figura 8.4.

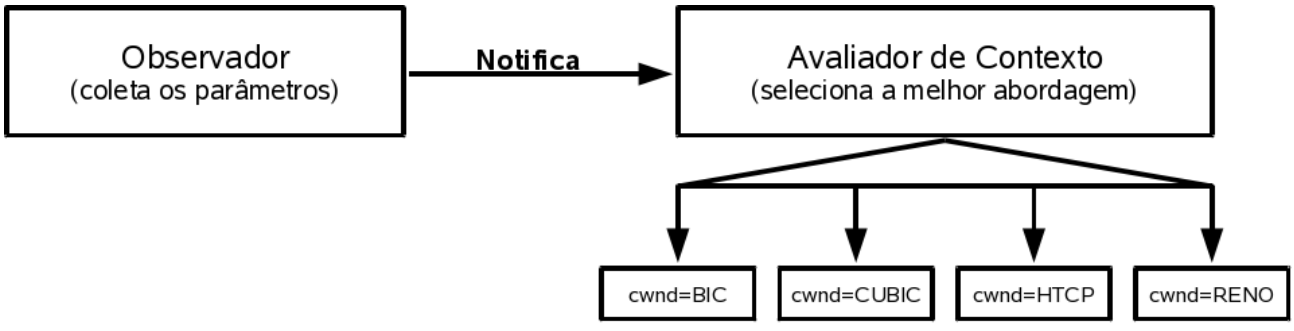

FIG. 8.4: Transporte adaptativo: Visão Detalhada.

O *Observador* deve se inteirar dos parâmetros de interesse da rede e informar ao *Avaliador de contexto* sempre que ocorrer uma mudança em algum desses parâmetros. O

Avaliador de contexto, a partir das notificações emitidas pelo *Observador*, avalia qual a variante TCP indicada, em função dos novos valores dos parâmetros levantados pelo observador. Com o intuito de embasar futuras implementações de uma camada como esta, vamos modelar seus componentes utilizando os padrões *state* e *observer*. O observador da figura 8.4 será implementado por um padrão *observer*, pois no mesmo deve haver uma classe capaz de encapsular os parâmetros da rede e notificar ao avaliador de contexto sempre que houver uma mudança em alguma variável de interesse (banda, retardo, *jitter*, etc). Já no módulo avaliador, empregaremos o padrão *state*: uma vez notificado pelo observador, o avaliador deve perceber o contexto vivido pela rede naquele momento e alinhar a função de atualização da janela de congestionamento de acordo com este contexto. A Figura 8.5 apresenta o detalhamento da arquitetura para implementação de uma camada de transporte adaptativa.

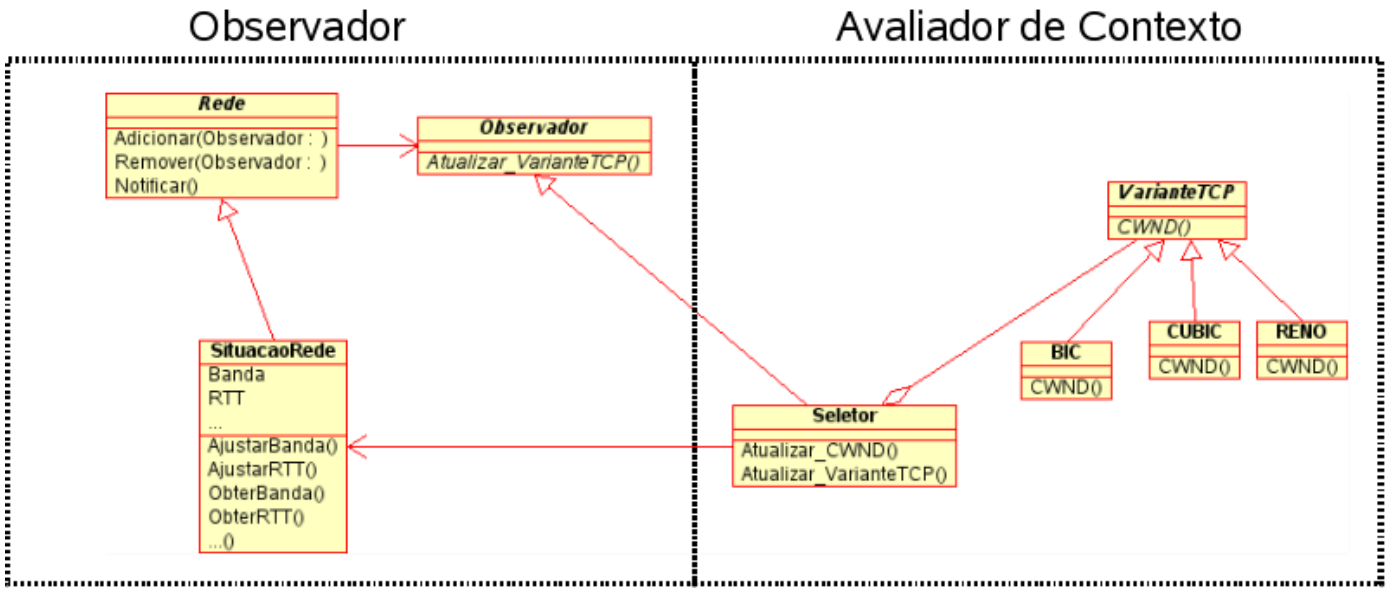

FIG. 8.5: Arquitetura da Camada de Transporte Adaptativa.

Supondo que as variáveis de interesse para a adaptação do transporte sejam apenas o RTT e a banda disponível, cada vez que os métodos *AjustarBanda* ou *AjustarRTT* forem acionados a classe abstrata *Rede* acionará o *Seletor* através do método *Notificar*, no qual haverá uma chamada para o método *Atualizar_VarianteTCP()*, concretizado neste *Seletor*. O *Seletor*, por sua vez, requisitará o RTT e a banda através dos métodos *ObterRTT* e *ObterBanda*, respectivamente. A partir destes valores, o *Seletor* concretizará a classe abstrata *VarianteTCP* de acordo com a melhor abordagem (BIC,CUBIC ou RENO) para

o momento vivido pela rede. O método *Atualizar_CWND* do *Seletor* chama o método virtual *CWND*, concretizado nas classes filhas de *VarianteTCP*, garantindo assim que a janela de congestionamento seja atualizada de acordo com a função de interesse. Ressaltamos ainda que esta arquitetura pode ser facilmente estendida para trabalhar-se com mais parâmetros ou mais variantes TCP.

8.5 IMPLEMENTAÇÃO

8.5.1 COLHENDO OS PARÂMETROS

Todo esforço envolvendo as propostas de variantes TCP baseia-se fundamentalmente nos parâmetros banda e retardo, como já foi discutido anteriormente. Para implementação de uma camada de transporte adaptativa faz-se necessário, portanto, algumas observações a respeito de como esses parâmetros serão levantados. As técnicas de medições em redes podem ser classificadas em ativas, em que há inserção de pacotes que visam única e exclusivamente levantar dados a respeito da conexão, ou passivas, em que um ou mais pontos da rede são encarregados da coleta de informações a repeito da malha da qual fazem parte. As medições passivas requerem, para sua maior eficiência, equipamentos dedicados e é indicada quando deseja-se observar com precisão uma grande quantidade de variáveis da rede como um todo. Como o foco do presente trabalho baseia-se no estudo das conexões fim-a-fim, apresentaremos técnicas ativas para medição dos parâmetros de interesse (apenas dois: banda e retardo), já que as mesmas são mais simples e, de acordo com a Tabela 6.3, não precisamos de medidas extremamente precisas para selecionarmos a melhor abordagem no transporte de dados.

8.5.1.1 RTT

O RTT (Round Trip Time) é o tempo de ida e volta da fonte ao destino. É quase impossível dissociar a medida do RTT da mais popular das ferramentas de medição: o ping (POSTEL, 1981a), baseado em troca de mensagens ICMP periódicas. Esta ferramenta na grande maioria dos casos (redes que não apresentam comportamento periódico (PAXSON, 1997)), se mostra eficiente na obtenção do RTT fim-a-fim.

8.5.1.2 BANDA DISPONÍVEL

As principais técnicas para medição de capacidade de enlace são: sondagem de pacotes de tamanho variável (JACOBSON, 1997), que determina a capacidade dos enlaces individualmente, e dispersão de par de pacotes (SAROIU, 2002), que mede a capacidade fim-a-fim. A dispersão de par de pacotes se baseia no fato de que, caso não haja tráfego concorrente, após um par de pacotes atravessar o "gargalo" de uma ligação, o intervalo entre os recebimentos do primeiro e segundo pacotes é proporcional à largura de banda deste "gargalo" (FIG. 8.6). Para que essa técnica funcione, o tamanho dos

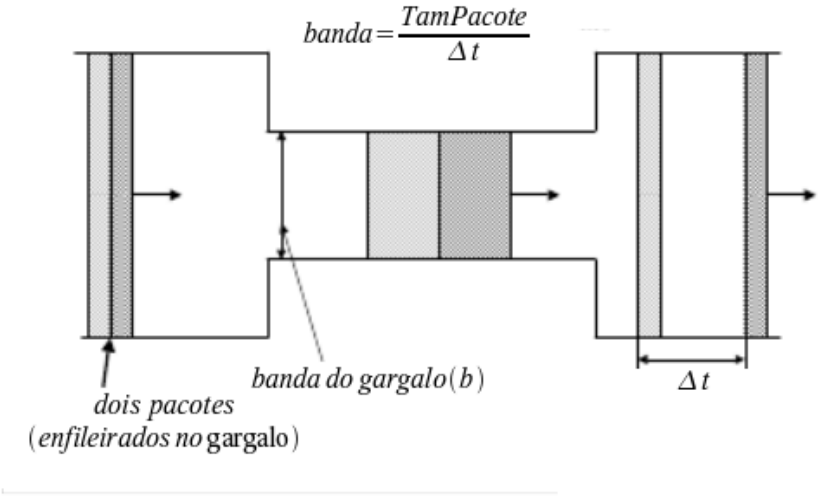

FIG. 8.6: Técnica de medição de banda par de pacote.

pacotes deve ser suficientemente grande para causar um enfileiramento no menor "gargalo". Infelizmente, para a maioria das situações há presença de tráfego concorrente durante as medições. Para contornar este problema, diversos trabalhos (DOVROLIS, 2001), (CARTER, 1996), (DOVROLIS, 2004), foram desenvolvidos, dentre os quais destacamos (DOVROLIS, 2004), que, para o cálculo da banda disponível, propõe um tratamento estatístico dos dados colhidos a partir do envio de múltiplos pares de pacotes durante as medições. Na tabela 8.1 as principais ferramentas de medição da banda fim-a-fim e suas respectivas referências, que podem ser utilizadas para alimentar a camada de transporte adaptativa com os parâmetros necessários ao seu funcionamento, ou mesmo servir de base para implementação de um novo coletor de parâmetros.

Aplicativo	Referência
bprobe	(CARTER, 1996)
nettimer	(LAI, 2000)
pathrate	(DOVROLIS, 2004)
sprobe	(SAROIU, 2002)

TAB. 8.1: Principais ferramentas de medição da banda fim-a-fim.

8.6 PROGRAMA

Como já foi dito, o pathrate é um aplicativo para estimar a capacidade do canal fim-a-fim. Resumidamente, este aplicativo avalia a capacidade do canal entre dois endereços IP e produz um arquivo no qual encontramos o RTT entre as máquinas e um intervalo de velocidades estimado para capacidade do enlace entre eles. Maiores detalhes a respeito da instalação, forma de funcionamento e emprego do pathrate se encontram em (PATHRATE, 2008). O programa, desenvolvido para adaptar a camada de transporte, cria, através do comando *fork*, um processo para executar o pathrate. Após a execução do pathrate, o referido programa lê o arquivo de saída para extrair a velocidade e o RTT. De posse destes dados, o programa, baseado na Tabela 6.3, utiliza o comando sysctl para ativar a variante TCP de melhor desempenho para o RTT e banda estimados (FIG.8.7). O código do programa se encontra no Apêndice 3. Destacamos ainda que um

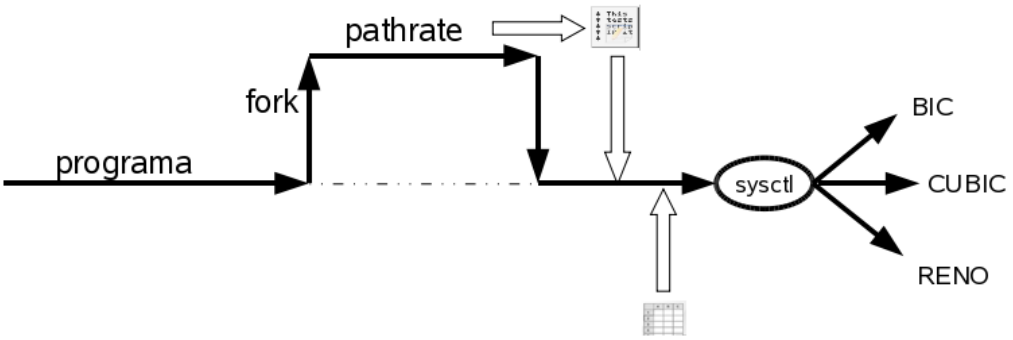

FIG. 8.7: Implementação de uma Camada de Transporte Adaptativa.

administrador pode de tempos em tempos, através do crontab, por exemplo, chamar o programa desenvolvido para adaptar seu transporte às condições da rede.

9 ADAPTAÇÃO SOBRE UDP

9.1 PRELIMINARES

Esta é a segunda fase do trabalho, cuja finalidade é mostrar como deve ser uma camada de transporte adaptativa que também tenha de operar com protocolos desenvolvidos para redes com características incomuns. Lembramos que no módulo de telemática, por exemplo, pode haver chaveamento de uma conexão "boa" (baixas taxas de erro e velocidades da ordem de Mbps, ex.: Ethernet, fibra optica), para um enlace HF. Em (ISODE, 2008) encontramos alguns motivos pelos quais os mecanismos TCP não são aconselhados no caso de utilização de enlaces HF, dentre as quais destacam-se o fato de as versões TCP terem sido projetadas para redes com velocidades muito acima daquelas proporcionadas pelos enlaces HF e o grande número de retransmissões desnecessárias produzidas pelos mecanismos TCP enquanto este protocolo não "descobrir" o RTT do enlace. A camada de transporte deve, portanto, adequar-se a tais condições.

Visando esta adequação, faz-se uma análise do desempenho do TCP e do UDP em enlaces restritos com o objetivo de dar uma visão mais concreta a respeito dos ganhos que podem ser alcançados pelo UDP em relação ao TCP em enlaces com baixa velocidade e elevada taxa de erro.

Posteriormente, é apresentado um esquema de adaptação quando da necessicade de empregar-se ora protocolos especiais, otimizados para situações específicas (geralmente baseados em UDP), ora protocolos alinhados com esquemas mais tradicionais.

9.2 ANÁLISE DO TCP E DO UDP EM ENLACES RESTRITOS

9.2.1 AMBIENTE DE TESTE

Os testes foram realizados no laboratório de redes do IME. As máquinas utilizadas seguem a configuração apresentada nao capítulo 5. Para avaliarmos a eficiência entre o transporte TCP e UDP, foram realizados transferências de dados na estrutura ilustrada pela Figura 9.1.

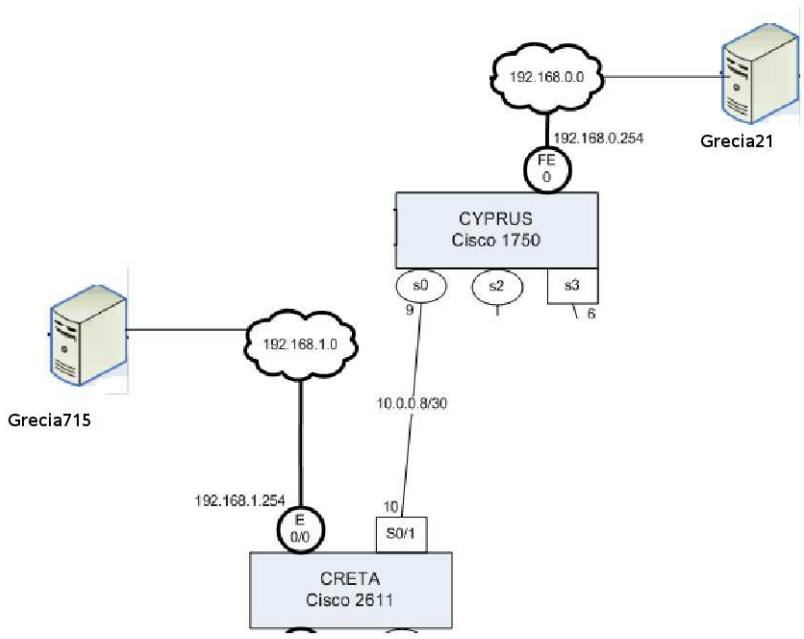

FIG. 9.1: Ambiente de testes - TCP X UDP.

9.2.2 EXPERIMENTOS

Como sugere a Figura 9.1, as máquinas Grecia21 e Grecia715 se interligam por dois roteadores: CRETA (CISCO(R) 2611) e CYPRUS (CISCO(R) 1750), que se conectam diretamente por um enlace serial. Na máquina Grecia21, um programa desenvolvido enviava dados (500 Bytes a cada 0.5 segundos), durante um determinado intervalo de tempo (15, 30, 60 e 120 segundos), para a máquina Grecia715 utilizando o enlace serial (s0-s0/1), ajustado a velocidade de 1200 bps, ora através do UDP, ora através do TCP. A taxa de erro a cada rodada de teste foi repectivamente 10, 25 e 50 %; estas taxas foram emuladas pelo Netem (Apêndice 1). O *goodput* proporcionado por ambos os protocolos para as taxas de erro utilizadas estão condensadas nos gráficos das Figuras 9.2, 9.3 e 9.4

9.2.3 ANALISANDO OS DESEMPENHOS

Como pode ser observado, o UDP proporciona uma transferência bruta de dados cada vez maior à medida que se aumenta a taxa de erros. Um comentário especial cabe na situação na qual a taxa de erros foi ajustada para 50% (configuração esta que traduz enlaces HF). Os resultados confirmam que a manutenção de protocolos alinhados à filosofia TCP podem comprometer as atividades de comunicação quando elas utilizam enlaces cujas características se aproximam daquelas proporcionadas pelas transmissões de dados

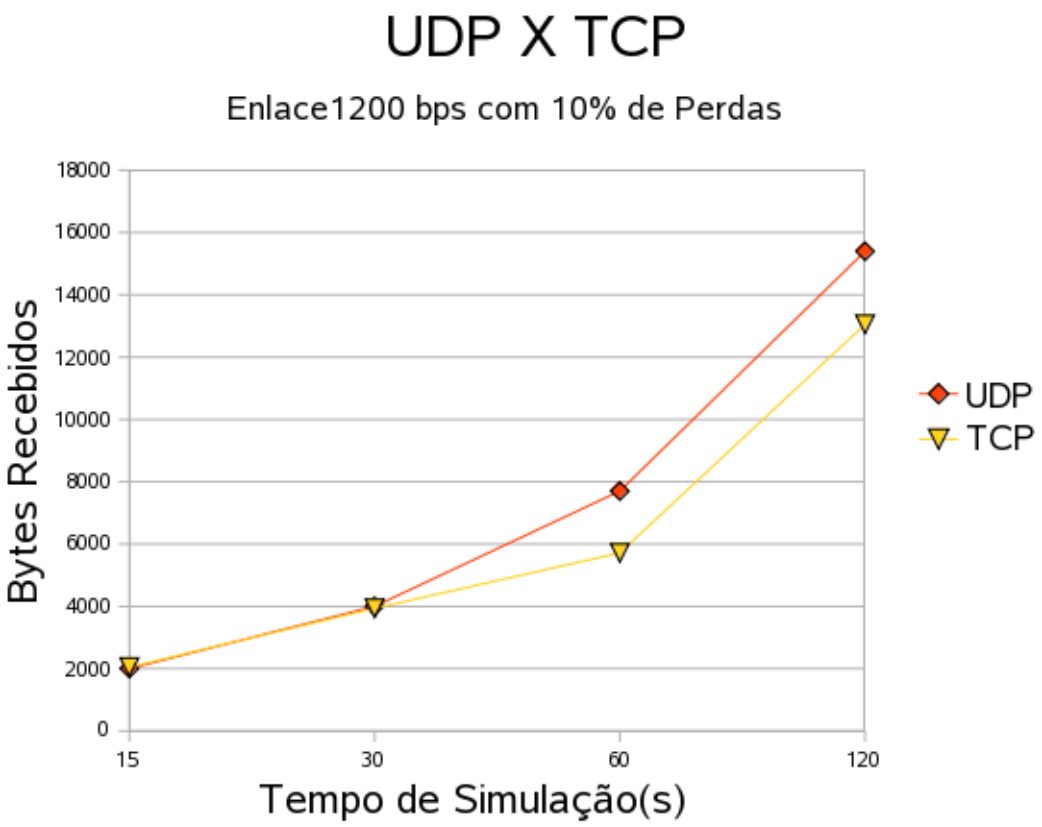

FIG. 9.2: TCP X UDP 10%.

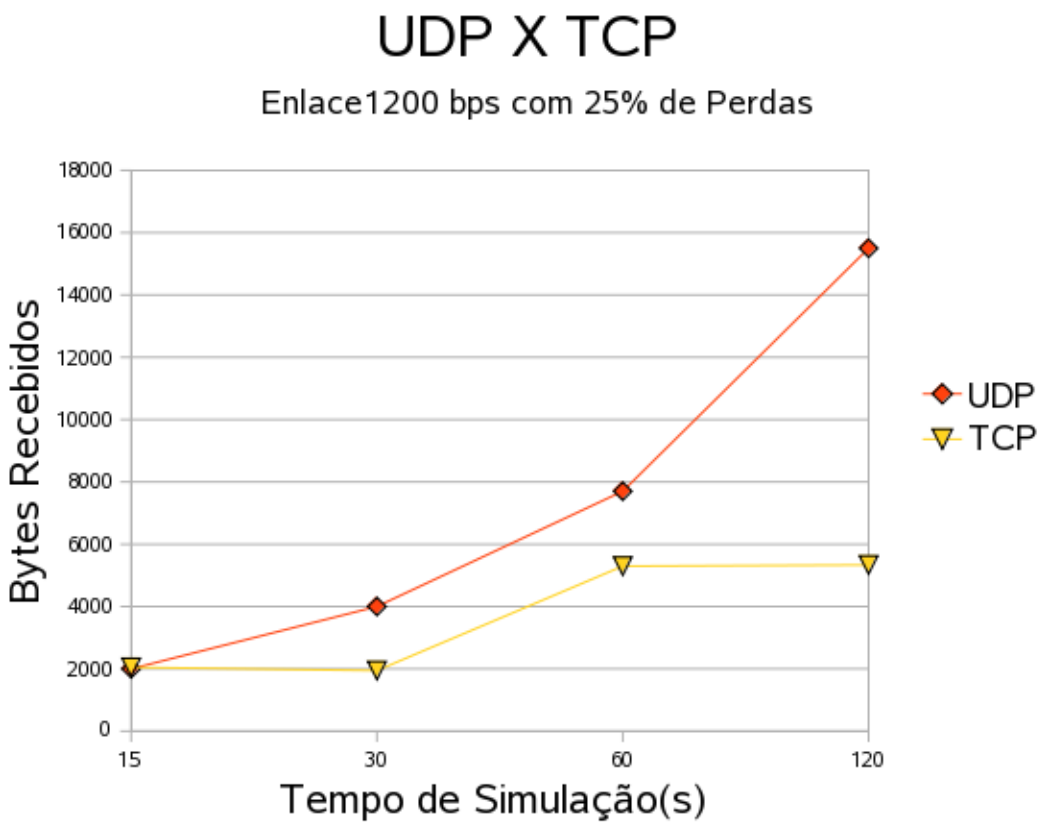

FIG. 9.3: TCP X UDP 25%.

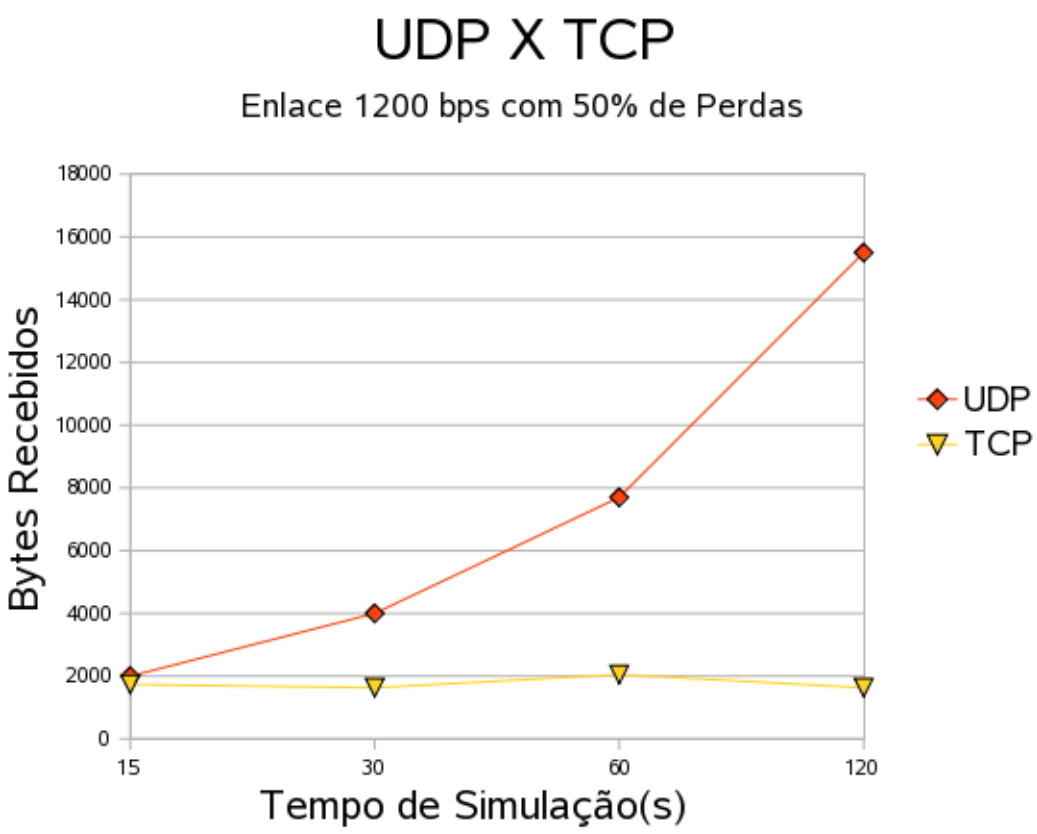

FIG. 9.4: TCP X UDP 50%.

em HF.

Os experimentos podem ser utilizados para avaliação de quão promissor pode ser o investimento em mecanismos confiáveis para transmissão de dados sobre UDP. No caso das transferências realizadas com um canal de taxa de erro em 50%, o *goodput* do UDP chega a ser praticamnete 700% maior do que aquele alcançado pelo TCP. Sendo assim, mesmo que a garantia da confiabilidade durante a transmissão UDP no terceiro experimento representasse uma redução de quase 75% no *goodput* do tráfego UDP, ele ainda seria o dobro do alcançado pelo TCP.

9.3 ESTRUTURANDO A ADAPTAÇÃO

Infelizmente a multiplexação/demultiplexação do UDP e do TCP seguem, como foi visto anteriormente, padrões distintos: o UDP utiliza somente a porta destino para alcançar o processo destinatário das informações enquanto que o TCP utiliza o par de portas para despachar os dados encaminhados. Sendo assim, passar do UDP para o TCP, ou vice-versa. requer uma estrutura mais elaborada do que aquela sugerida na adaptação entre as variantes TCP.

Uma forma de possibilitar a adaptação entre um protocolo de transporte específico sobre o UDP e o TCP seria acumular as atividades de transporte na aplicação. Quando em enlaces HF, por exemplo, o módulo da aplicação responsável pelo transporte de dados passaria a trabalhar de forma otimizada para o enlace em questão, como em (KENNINGTON, 1997). Caso um enlace reconhecidamente amigável ao TCP passe a reger a conexão, o TCP seria então emulado sobre o UDP (FIG.9.5). A emulação do TCP sobre o UDP já é adotada em diversas aplicações práticas, principalmente naquelas que realizam transmissão de dados em enlaces rádio. O programa C2 em combate é um exemplo de sucesso desta abordagem (RONDON, 2006). Outro exemplo de projeto que segue esta filosofia é o RakNet (RAKNET, 2008).

É importante ressaltar ainda que todas as considerações feitas sobre o projeto de camada de transporte adaptativa para o chaveamento entre as variante TCP valem para a implementação de um transporte adaptativo na aplicação sobre o UDP. Obviamente, neste caso, haverá necessidade de se implementar cada função de atualização da janela de congestionamento (Reno, BIC, etc), que sirva de opção para a camada de transporte adaptativa sobre o UDP.

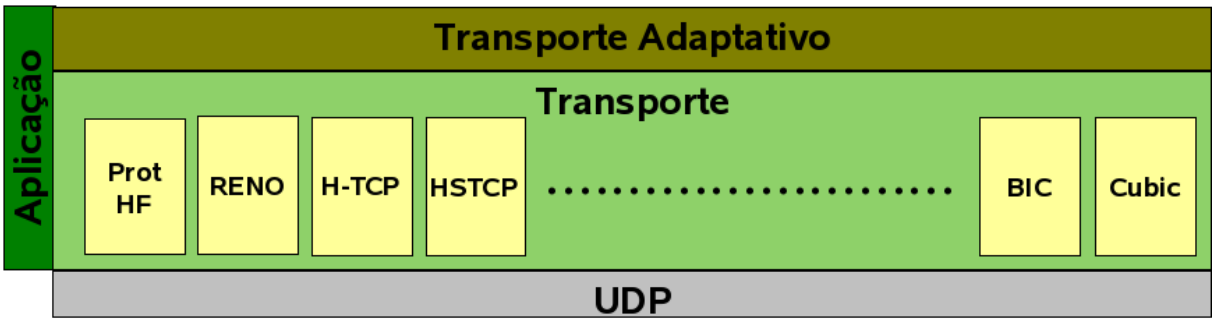

FIG. 9.5: Adaptação Sobre UDP.

10 CONSIDERAÇÕES FINAIS

10.1 CONCLUSÕES

No desenvolvimento deste trabalho, ficou provada a importância de uma camada de transporte adaptativa para o sucesso das comunicações digitais, principalmente devido à imprevisibilidade das redes de comunicações atuais. Os resultados comprovaram que uma grande economia, tanto de tempo quanto de recursos, pode ser atingida quando o transporte se adequa ao momento vivido pela infra-estrutura de comunicação. Felizmente, para uma boa escolha no que diz respeito à função de atualização da janela de congestionamento, não precisamos de medidas extremamente precisas, o que torna as implementações de camadas de transporte adaptativas independentes de sondagens extremamente eficientes, as quais tornariam tais implementações bem mais difíceis de serem concretizadas.

Quanto à viabilidade de uma camada de transporte flexível, vimos que a adaptação na função de atualização da janela de congestionamento é algo inerente às mais remotas versões do TCP e que bastaria disponibilizarmos outras funções de atualização (CUBIC, BIC, etc), que seriam selecionadas de acordo com os parâmetros levantados nos experimentos.

A arquitetura da camada de transporte adaptativa apresentada reforça ainda mais a aplicabilidade desta nova filosofia, pois esta arquitetura e os padrões de projetos a partir dos quais a mesma foi estruturada, são amplamente conhecidos e representam de forma simples e precisa toda a documentação necessária aos possíveis desenvolvedores desta camada.

Mesmo que os sistemas operacionais não sejam dotados de uma camada de transporte adaptativa, é possível emularmos um transporte adaptativo como ilustrado na implementação (seção 8.6), feita utilizando-se o comando sysctl e os módulos disponíveis nas versões mais modernas de *kernel*. Além disso, ainda que não haja no *kernel* os referidos módulos, o transporte de dados pode ter sua flexibilização garantida caso as aplicações, associadas a *sockets* UDP, se encarreguem das atividades de atualização da cwnd, como ilustrado no capítulo 9 na adaptação sobre o UDP.

Outro fato bem interessante é o bom desempenho do TCP Reno, como opção de transporte de dados para um grande número de situações, principalmente naquelas mais tradicionais (velocidades até 10 Mbps), e até em alguns enlaces mais robustos, como observado nos enlaces de 100 Mbps, para RTT's abaixo de 400 ms e ligações de 1 GB com RTT inferior a 200 ms.

Deve-se destacar também que a vantagem do emprego do UDP em relação ao TCP em canais HF está intimamente relacionada com a taxa de erros e nem tanto com a velocidade do enlace. Os experimentos apresentados no capítulo 9 mostram que, para uma taxa de erros de 10%, a diferença entre as quantidades de dados transportados pelas duas abordagens não é tão expressiva. Esta diferença poderia até ser tolerada diante da confiabilidade oferecida pelo TCP. Contudo, conforme se intensificavam as perdas no canal, o desempenho do UDP se mantinha estável enquanto que o TCP diminuía de forma drástica a quantidade de dados transportados a cada rodada de testes. Nestas situações é interessante o investimento em mecanismos eficientes, a serem implementados sobre o UDP, para garantia da confiabilidade na troca de informações.

10.2 TRABALHOS FUTUROS

Outras questões podem ser exploradas no contexto deste trabalho, dentre as quais destacamos:

- **Latência**

  Durante todo o trabalho, baseamos nossas conclusões, como geralmente é feito em trabalhos científicos, em um modelo simplificado, isto é, assumimos que a camada de transporte adaptativa é capaz de chavear entre as abordagens disponíveis de forma instantânea, sem fazermos qualquer consideração sobre a latência que pode existir durante este chaveamento, principalmente no que diz respeito ao ajuste das variáveis de contexto para realização das atividades da camada de transporte e como aproveitarmos as informações presentes nestas variáveis para alimentarmos aquelas requeridas pela nova abordagem de transporte ora instanciada.

  Deve ser levantado se é possível ou não a implementação de uma camada de transporte adaptativa com tempo de chaveamento desprezível entre os algoritmos de atualização da janela de congestionamento. Caso esta latência não seja desprezível, é preciso medir quais os impactos desta latência nos resultados ora apresentados.

- **Outras Variáveis**

  Como a maioria dos trabalhos na literatura justifica as propostas de novas versões TCP levando-se em consideração o BDP, no caso dos enlaces confiáveis, e na taxa de erros, no caso dos enlaces suscetíveis a interferências, este trabalho concentrou seus esforços nestes parâmetros. Entretanto, existem outros fatores que podem ser considerados como importantes na hora de se optar por esta ou aquela abordagem de transporte. Só para ilustrar, o desempenho do TCP tradicional é reconhecidamente prejudicado em redes de grande *jitter*. Até que ponto este *jitter* realmente degrada as mais novas versões do TCP e suas variantes, bem como se os valores deste *jitter* representam alguma situação prática, deve ser objeto de investigação para o aperfeiçoamento das implementações das camadas de transporte adaptativas.

- **Duração da Conexão**

  Um outro fator que deve ser estudado é qual duração mínima uma conexão deve ter para ser contemplada com a possibilidade de adaptar seu mecanismo de transporte. Obviamente, para conexões muito curtas não valeria a pena dotar a camada de transporte com a flexibilidade proposta por este trabalho.

11 REFERÊNCIAS BIBLIOGRÁFICAS

AI3. **Asian Internet Interconnection Initiatives**, 2009. URL `http://ai3.asti.dost.gov.ph/sat/chara.html`. *Visitado em visitado em 29/01/2009*.

ASENSTORFER, P. e SCHOLZ, J. B. Longfish - a hf radio network testbed. ***7th Int. Conf. on HF Radio Systems and Techniques***, 1997.

BRAKMO, L., O'MALLEY, S. e PETERSON, L. Tcp vegas: New techniques for congestion detection and avoidance. ***SIGCOMM 94 Symposium***, 1994.

CAINI, C. e FIRRINCIELI, R. Tcp hybla: a tcp enhancement for heterogeneous networks. ***Int. J. Satellite Commun. Netw.***, 2004.

CARTER, R. L. e CROVELLA, M. E. Measuring bottleneck link speed in packet-switched networks. Technical report, **Department of Computer Science, Boston University**, http://www.cs.bu.edu/faculty/crovella/papers.html, 1996.

CASETTI, C., GERLA, M., MASCOLO, S., SANADIDI, M. Y. e WANG, R. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. ***ACM Mobicom 2001***, 2001.

COTER. Principais aspectos do sistema de comando e controle do exército e da força terrestre. **Ministério da Defesa - Exército Brasileiro - Comando de Operações Terrestres**, 2002.

DELL'AERA, A., GRIECO, L. A. e MASCOLO, S. Linux 2.4 implementation of westwood+ tcp with rate-halving: A performance evaluation over the internet. ***IEEE International conference on Communication (ICC 2004), June 2004, Paris***, 2004.

DOVROLIS, C., RAMANATHAN, P. e MOORE, D. What do packet dispersion techniques measure? ***IEEE INFOCOM'2001, Anchorage, AK, EUA.***, 2001.

DOVROLIS, C., RAMANATHAN, P. e MOORE, D. Packet dispersion techniques and a capacity estimation methodology. ***IEEE/ACM Transactions on Networking***, 2004.

FLOYD, S. Highspeed tcp for large congestion windows - rfc 3649. *Experimental*, 2003. URL `http://www.icir.org/floyd/hstcp.html`.

FLOYD, S., HENDERSON, T. e GURTOV, A. The newreno modification to tcp's fast recovery algorithm - rfc 2582. ***IETF***, 2004.

GAMMA, E., HELM, R., JOHNSON, R. e VLISSIDES, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series)*. **Addison-Wesley**, 1995.

HEMMINGER, S. Network emulation with netem. ***In Proceedings of LCA***, 2005.

ISODE. Why ip over hf radio should be avoided. ***Isode's whitepapers***, 2008.

JACOBSON, V. Congestion avoidance and control. ***SIGCOMM '88 Conf., ACM***, 1988.

JACOBSON, V. Berkeley tcp evolution from 4.3-tahoe to 4.3-reno. ***Eighteenth Internet Engineering Task Force - University of British Columbia, Vancouver,B.C***, 1990.

JACOBSON, V. Pathchar: A tool to infer characteristics of internet paths. ***Presented at the Mathematical Sciences Research Institute (MSRI),April 1997.***, 1997. URL `http://www.caida.org/tools/utilities/others/pathchar/`.

JIN, C., WEI, D. X. e LOW, S. H. Fast tcp: Motivation, architecture, algorithms, performance. Em ***IEEE INFOCOM***, 2004.

JODALEN, V., EGGEN, A., SOLBERG, B. e GRONNERUD, O. Military messaging in ip networks using hf links. ***Communications Magazine, IEEE***, 2004.

JOHNSON, E. E. Hf radio in the international information infrastructure. Em ***Nordic Shortwave Conference , HF95, Faro, Sweden***, 1995.

KELLY, T. Scalable tcp: Improving performance in high-speed widearea networks. ***ACM SIGCOMM Computer Communication Review***, 2003.

KENNINGTON, A. The fitfeel transmission protocol. ***Int. Conf. on Telecommunications (ICT97), Melbourne, Australia.***, 1997.

KEPHART, J. e CHESS, D. The vision of autonomic computing. ***IEEE Computer***, 2003.

KESSLER, G. C. e TRAIN, D. A. *Metropolitan Area Networks - Concepts, Standards and Services*. **McGraw-Hill Inc**, 1992. ISBN: 0070342431.

KUROSE, J. F. e ROSS, K. *Computer Networking: A Top-Down Approach, 4/E.* **Addison-Wesley**, 2008. ISBN: 0321497708.

LAI, K. e BAKER, M. Measuring link bandwidths using a deterministic model of packet delay. ***ACM SIGCOMM'2000, Estocolmo, Suécia***, 2000.

LEE, C., LEE, D., KOO, J., BANERJEE, S. e CHUNG, J. Cli-based tcp: an end-to-end proactive approach robust to trafic load. ***Elsevier***, 2008.

LIU, S., BASAR, T. e SRIKANT, R. Tcp illinois: A loss and delay-based congestion control algorithm for high-speed networks. 1^{st} ***Int. Conf. on Performance Evaluation Methodologies and Tools***, 2006.

MICHEL, N. F. e FONSECA, N. L. S. Uma investigação sobre a escalabilidade de variantes do protocolo tcp para redes de alta velocidade. ***Wperformance***, 2007.

MICHEL, N. F. e FONSECA, N. L. S. Tcp tunning guide. ***Advanced Computing for Science***, 2008.

PATHRATE. 2008. URL `http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate_tutorial.html`. *Visitado em visitado em 29/01/2009.*

PAXSON, V. Measurement and analysis of end-to-end internet dynamics. ***University of California, Berkeley***, 1997.

POSTEL, J. User datagram protocol, rfc 768. ***7th Int. Conf. on HF Radio Systems and Techniques***, 1980.

POSTEL, J. Internet control message protocol. rfc 792. ***Internet Engineering Task Force (IETF)***, 1981a.

POSTEL, J. Internet protocol, std 5, rfc 791. ***USC/Information Sciences Institute***, 1981b.

RAKNET. 2008. URL `http://www.jenkinssoftware.com/raknet/manual/reliabilitytypes.html`. *Visitado em visitado em 29/01/2009.*

REZENDE, J. F., COSTA, L. H. M. K. e RUBINSTEIN, M. G. Avaliação experimental e simulação do protocolo tcp em redes de alta velocidade. ***XXII SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES - SBrT'05***, 2005.

REZENDE, J. F., DA SILVA, M. W. R., CARDOSO, K. V., MENDES, A. C., GUEDES, R. M. e AUGUSTO, C. H. P. Segmentação de conexões tcp para a transferência fim-a-fim em alta velocidade. ***SBC***, 2008.

RONDON. Programa c2 em combate "mecanismo de replicação e difusão de objetos nodais(rondon). Technical report, **Ministério da Defesa - Exército Brasileiro - Departamneto de Ciência e Tecnologia - Centro Integrado de de Guerra Eletrônica (CIGE)**, 2006.

SAROIU, S., GUMMADI, P. e GRIBBLE, S. Sprobe: A fast technique for measuring bottleneck bandwidth in uncooperative environments. ***IEEE INFOCOM***, 2002.

SHORTEN, R. e LEITH, D. H-tcp: Tcp for high-speed and longdistance networks. ***Second International Workshop on Protocols for Fast Long-Distance Networks, February 16-17, 2004, Argonne, Illinois USA***, 2004.

SILVA, L. A. F. Análise de desempenho de protocolos de transporte para redes de alta velocidade. Dissertação de Mestrado, **COPPE/UFRJ, M.Sc.,Engenharia Elétrica - Universidade Federal do Rio de Janeiro**, 2006.

STALLING, W. *Data and Computer Communications*. **Macmilan Publishing Company**, 1994. ISBN-0070342431.

TAN, K., SONG, J., ZHANG, Q. e SRIDHARAN, M. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. ***PFLDnet***, 2006.

TANENBAUM, A. S. *Computer Networks - Fourth Edition*. **Prentice Hall**, 2003. ISBN: 0130661023.

XU, L., HARFOUSH, K. e RHEE, I. Binary increase congestion control (bic) for fast long-distance networks. ***IEEE INFOCOM***, 2004.

XU, L. e RHEE, I. Cubic: A new tcp-friendly high-speed tcp variant. ***Workshop on Protocols for Fast Long Distance Networks***, 2005.

12 APÊNDICES

12.1 NETEM

12.1.1 VISÃO GERAL

O módulo Netem (Network Emulation) é um recurso recentemente agregado às versões Linux, cujas funcionalidades permitem alterar as propriedades de uma conexão fim-a-fim. As versões atuais alteram os atrasos (*jitter* e RTT), perda, duplicação e reordenação de pacotes. Vários trabalhos (REZENDE, 2008) (HEMMINGER, 2005) atestam a eficiência desta ferramenta na avaliação de protocolos de transmissão. Nas distribuições 2.6 Fedora, OpenSuse, Gentoo, Debian, Mandriva, Ubuntu, este módulo já faz parte do *kernel*. O Netem é acionado pelo comando "tc" (*trafic control*), empregado para controlar os parâmetros utilizados no controle de tráfego no Linux. Um exemplo destes parâmetros é a disciplina de fila (qdisc), que regula o envio de pacotes de acordo com as suas configurações; sendo assim, sempre que um pacote deve ser enviado por uma interface, ele é primeiro tratado pela qdisc associada a ela (FIG. 12.1). O Netem atua alternando a disciplina associada à fila de uma interface, como veremos nos exemplos a seguir.

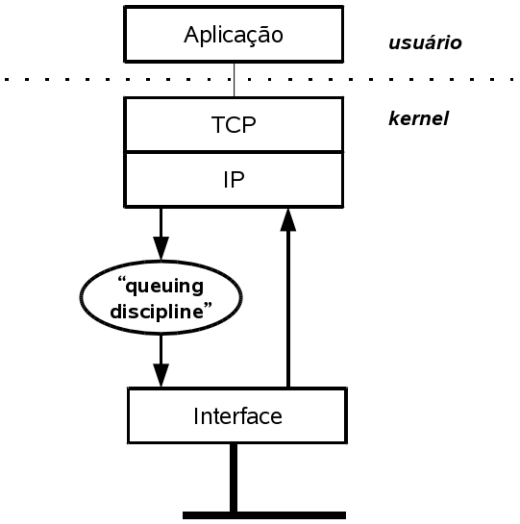

FIG. 12.1: Disciplina de Fila e o Netem.

12.1.2 EXEMPLOS

- Alterando Atrasos

  O comando a seguir adiciona um atraso fixo em todos os pacotes despachados pela interface eth0:

#tc qdisc add dev eth0 root netem delay 100ms

Este comando pode ser traduzido da seguinte forma: "Alerte ao controle de tráfego que antes de um pacote sair da fila associada à intervace eth0 (disciplina da fila - qdisc), ele deve esperar 100 ms". As alterações produzidas por este comando podem ser verificadas por simples testes, até mesmo através do comando ping. É possível acrescentarmos variações no atraso (*jitter*) da seguinte forma:

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

Observe que agora foi utilizado o parâmetro *change*, que simplesmente muda a disciplina da fila sem precisarmos recarregá-la. Este comando faz com que o atraso seja de 100±10ms. Se não estivermos interessados em uma variação puramente randômica (distribuída uniformemente) deste *jitter* devemos inserir o comando com o formato:

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

que produz um delay de 100± 10ms com o próximo elemento randômico dependendo 25% do anterior. Obviamente estes cálculos são resultados de aproximações estatísticas. Tipicamente, a distribuição do *jitter* não é uniforme, por isso podemos associar a variação do atraso a uma distribuição normal:

tc qdisc change dev eth0 root netem delay 100ms 20ms distribution normal

A princípio, qualquer distribuição pode ser utilizada, basta gerar o arquivo .dist correspondente e copiá-lo para /usr/lib/tc. As distribuições normal, pareto e paretonormal já estão disponíveis.

- Perda de Pacotes

  A perda de pacotes pode ser especificada através do comando tc de acordo com o exemplo a seguir:

  # tc qdisc change dev eth0 root netem loss 0.1%

  este comando causa uma perda randômica de 1 pacote a cada 1000. Existe a possibilidade de se tornar as perdas menos randômicas se utilizarmos o comando da seguinte maneira:

  # tc qdisc change dev eth0 root netem loss 0.3% 25%

  Assim, haverá 0,3% de perdas e cada probabilidade sucessiva depende de um quarto da anterior, como sugere a seguinte equação:

  $P_n = 0.25 * P_{n-1} + 0.75 random.$

- Duplicação de Pacotes

  A duplicação de pacotes é especificada da mesma forma empregada na Perda de pacotes

  # tc qdisc change dev eth0 root netem duplicate 1%

- Corrupção de Pacotes

  Para emularmos a corrupção de pacotes basta executarmos o seguinte comando:

  # tc qdisc change dev eth0 root netem corrupt 0.1%

  1 a cada 1000 pacotes terão um dos seus bits corrompidos.

- Reordenação de Pacotes

  Existem duas formas de proceder a reordenação de pacotes: O método *gap* e o *reorder*. No *gap*, uma seqüência lógica de reordenação é estabelecida, por exemplo

  # tc qdisc change dev eth0 root netem gap 5 delay 10ms

  Este comando produz o envio automático de todos os pacotes múltiplos de 5 (quinto, décimo, décimo quinto...) e mantém os demais pacotes na fila por 10ms. Para exemplificar, suponha que depois de executada a linha de comando anterior, 10 pacotes cheguem na fila: o pacote 1 no tempo t_1, o 2 em t_2 e assim sucessivamente até o décimo pacote no tempo t_{10}. Nestas condições, os pacotes 5 e 10 serão enviados em t_5 e t_{10} respectivamente, enquanto que os demais pacotes seguirão a seguinte regra: o pacote 1 será despachado em $t_1 + 10$ ms, o 2 em $t_2 + 10$ ms, o 3 em $t_3 + 10$ ms e assim por diante até o nono em $t_9 + 10$ ms.

  Com o parâmetro *reorder*, produzimos uma reordenação de uma determinada porcentagem de pacotes:

  # tc qdisc change dev eth0 root netem delay 10ms reorder 25% 50%

  Neste exemplo, 25% dos pacotes com uma correlação de 50% ($P_n = 0.50 * P_{n-1} + 0.50random$), serão enviados imediatamente, os outros permanecerão na fila por 10ms. Existe ainda uma forma indireta de se produzir a reordenação, através da variação do *jitter*:

  # tc qdisc change dev eth0 root netem delay 100ms 75ms

  Se ao primeiro pacote é associado um atraso de 100 ms e ao segundo pacote um

atraso de 50 ms (100 ms - 50 ms), o segundo pacote será enviado antes do primeiro; isto porque os pacotes na fila são dispostos segundo o momento de envio.

Existem ainda outras formas de se combinar o Netem com outros recursos para alcançarmos outros parâmetros da rede. Há também uma lista de discussão para que os usuários desta ferramenta possam sanar suas dúvidas em
http://www.linuxfoundation.org/en/Net:Netem

12.2 SYSCTL

12.2.1 VISÃO GERAL

O comando sysctl (System Control), permite configurar parâmetros do *kernel* em tempo de execução. O arquivo de configuração do sysctl é o /etc/sysctl.conf. Seguem alguns exemplos de comandos sysctl básicos:

- *sysctl -a*: exibe todos os valores de todos os parâmetros.
- *sysctl -w variavel=valor*: ajusta o parâmetro variável para valor
- *sysctl -p*: carrega as configuações do arquivo /etc/sysctl.conf

12.2.2 REFINANDO OS PARÂMETROS

Segundo (MICHEL, 2008), deve-se fazer as seguintes alterações para incrementar o desempenho do TCP em redes de alta velocidade:
net.core.rmem_max = 16777216[1]
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

Caso utilizemos o comando sysctl -p para ajustarmos estes parâmetros, devemos acrescentar estas linhas no arquivo /etc/sysctl.conf e então executarmos o comando sysctl -p
Se utilizarmos o sysctl -w para cada parâmetro devemos fazer:
sysctl -w net.core.rmem_max = 16777216
sysctl -w net.core.wmem_max= 16777216

[1]Os valores representam quantidades em bytes

net.core.rmem_max	Ajusta o valor, do *buffer* recepção máximo para cada conexão.
net.core.wmem_max	Ajusta o *buffer* de envio máximo para cada conexão
net.ipv4.tcp_rmem	O primeiro valor especifica a quantidade mínima de memória para o *buffer* de recepção que o kernel alocará para cada conexão TCP, mesmo em situações de sobrecarga do sistema. O segundo representa o valor default para o referido *buffer*. O terceiro valor especifica o tamanho máximo do *buffer* de recepção para cada conexão TCP.
net.ipv4.tcp_wmem	Análogo ao anterior, atuando agora sobre o *buffer* de envio de cada conexão TCP.

TAB. 12.1: Variáveis que devem ser ajustadas para enlaces de alta velocidade.

sysctl -w net.ipv4.tcp_rmem = 4096 87380 16777216
sysctl -w net.ipv4.tcp_wmem = 4096 65536 16777216

Segue abaixo uma tabela contendo o significado prático de cada um dos parâmetros mencionados:

É possível alterarmos também o protocolo de transporte utilizado pelo *kernel* da seguinte maneira:
sysctl -w net.ipv4.tcp_congestion_control=bic
As possibilidades de valores para net.ipv4.tcp_congestion_control se encontram, no caso do SUSE11, 2.6.25.11-0.1-pae, em /lib/modules/2.6.25.11-0.1-pae/kernel/net/ipv4. Neste diretório há arquivos cujos nomes começam por tcp_. Os valores possíveis para net.ipv4.tcp_congestion_control correspondem ao nome dos arquivos sem o tcp_ e sem a extensão; por exemplo, se quisermos ajustar o transporte para TCP CUBIC, basta verificarmos que no diretório /lib/modules/2.6.25.11-0.1-pae/kernel/net/ipv4 há o arquivo "tcp_cubic.ko". Retirando-se o tcp_ e a extensão do nome do respectivo arquivo teremos o parâmetro esperado; basta então executarmos o comando:
sysctl -w net.ipv4.tcp_congestion_control=cubic
Destacamos ainda que, após a mudança no protocolo de transporte, somente as novas conexões passarão a ser regidas pelo módulo carregado. As conexões anteriores permanecem no protocolo configurado quando do seu estabelecimento.

Existe uma enorme gama de parâmetros que podem ser dinamicamente configurados

pelo sysctl, e a tendência natural é que esta lista cresça cada vez mais, pois a cada nova versão de *kernel* mais parâmetros passam a estar sob a possibilidade de alteração em tempo de execução.

12.3 PROGRAMA

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include <stdlib.h>
#include <string.h>
#include <time.h>

void error(char *);
void espere(int);

int ExtrairValo(char *parStr,int *parValor);
int EstimarBandaRTT(int *parBanda, int *parRTT, char *parUnidade);

int main(int argc, char *argv[])
{
   int pid;
   int banda, RTT;
   char unidade;

   if (argc != 2) {
      fprintf(stderr,"Usar: servidor \n");
      exit(0);
   }

   int status;
   char senderPathrate[30];
   strcpy(senderPathrate,"-s");
   strcpy(senderPathrate+2,argv[1]);
   char outPathRate[30];
   strcpy(outPathRate,"-oteste.out");
```

```
printf("%s",senderPathrate );
pid = fork();
if (pid == 0) /* child process */
{
   printf("\n I'm the child!");
   execl("./pathrate_rcv","",senderPathrate,"-v",outPathRate,NULL);
   exit(0);
}
else /* parent process */
{
      wait(&status);
      printf("\n I'm the parent!");
      printf("\n Child returned: %d\n", status);
      EstimarBandaRTT(&banda,&RTT,&unidade);
      if(unidade == 'M')
      {
               if(abs(banda-1000) < abs(banda-100)) //caso Giga
               {
                     if(RTT > 200)
                     {
                        pid = fork();
                        if (pid == 0) /* child process */
                        {
                           printf("\n I'm the child!");
                           execl("/bin/sh","","./bic.sc",NULL);
                           exit(0);
                        }
                        else /* parent process */
                           wait(&status);
                     }

                  else
                  {
                        pid = fork();
                        if (pid == 0) /* child process */
                        {
                           printf("\n I'm the child!");
                           execl("/bin/sh","","./reno.sc",NULL);
                           exit(0);
                        }
```

```
                else /* parent process */
                    wait(&status);
            }
        }

    else if(abs(banda-10) > abs(banda-100)) //caso 100M
    {
        if(RTT > 400)
        {
            pid = fork();
            if (pid == 0) /* child process */
            {
                printf("\n I'm the child!");
                execl("/bin/sh","","./cubic.sc",NULL);
                exit(0);
            }
            else /* parent process */
                wait(&status);

            }
        else
        {
            pid = fork();
            if (pid == 0) /* child process */
            {
                printf("\n I'm the child!");
                execl("/bin/sh","","./reno.sc",NULL);
                exit(0);
            }
            else /* parent process */
                wait(&status);

        }


    }

    else //caso 10M
    {
        pid = fork();
```

```
                if (pid == 0) /* child process */
                {
                    printf("\n I'm the child!");
                    execl("/bin/sh","","./reno.sc",NULL);
                    exit(0);
                }
                else /* parent process */
                    wait(&status);

            }


        }

        else if(unidade == 'G')
        {
            if(RTT > 200)
            {
                pid = fork();
                if (pid == 0) /* child process */
                {
                    printf("\n I'm the child!");
                    execl("/bin/sh","","./bic.sc",NULL);
                    exit(0);
                }
                else /* parent process */
                    wait(&status);
            }
            else
            {
                pid = fork();
                if (pid == 0) /* child process */
                {
                    printf("\n I'm the child!");
                    execl("/bin/sh","","./reno.sc",NULL);
                    exit(0);
                }
                else /* parent process */
                    wait(&status);
```

```
                }
155         }
            else//default
            {
                pid = fork();
                if (pid == 0) /* child process */
160             {
                    printf("\n I'm the child!");
                    execl("/bin/sh","","./reno.sc",NULL);
                    exit(0);
                }
165             else /* parent process */
                    wait(&status);

            }

170     }
                return 0;

    }

175 int EstimarBandaRTT(int *parBanda, int *parRTT, char *parUnidade)
    {
            FILE *f= fopen("./teste.out", "r");
            size_t len= 200; // valor arbitrario
            char *linha= malloc(len);
180         int i=0;
            if (!f)
            {
                perror("./teste.out");
                exit(1);
185         }
            while (getline(&linha, &len, f) > 0)
            {
                printf("%s", linha);
                    i++;
190
            if(strstr(linha,"--> Average round-trip time:"))
            {
                ExtrairValor(linha,parRTT);
```

```
            }
            else  if(strstr(linha,"Final capacity estimate :"))
            {
                  ExtrairValor(linha, parBanda);

            if(strstr(linha,"Mbps"))
                  *parUnidade = 'M';
            else if(strstr(linha,"Gbps"))
                  *parUnidade = 'G';
            else
                  *parUnidade = 'k';


            }



            }
            if (linha)
               free(linha);
            fclose(f);
            printf("RTT: %d\nBanda: %d%cbps\n",*parRTT,*parBanda,*parUnidade);
            return 0;


}

int ExtrairValor(char *parStr,int *parValor)
{

   int i,j;
   int done=0;
   *parValor = 0;
   i = strlen(parStr);
   for(j=0; j < i && !done; j++)
   {
      if(parStr[j] >='0'&& parStr[j] <='9')
      {
         printf("%c\n",parStr[j]);
         while(parStr[j] >='0'&& parStr[j] <='9')
         {
            *parValor=*parValor*10+(parStr[j]-'0');
```

```
                j++;
235             printf("%d\n",*parValor);
            }
            done=1;
        }
    }
240 return 0;
}
```