UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ESTUDO DO COMPORTAMENTO DO TCP/IP EM ENLACES VIA SATÉLITE

IGOR PEREIRA MARCIANO DE OLIVEIRA
LEONARDO FERREIRA DE CASTRO

PROJETO FINAL DE GRADUAÇÃO EM
ENGENARIA DE REDES DE COMUNICAÇÃO

ORIENTADOR: SEBASTIÃO DO NASCIMENTO NETO

PUBLICAÇÃO: UnB.LabRedes.PFG12/2002

BRASÍLIA / DF: SETEMBRO/2002

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ESTUDO DO COMPORTAMENTO DO TCP/IP EM ENLACES VIA SATÉLITE

.

IGOR PEREIRA MARCIANO DE OLIVEIRA
LEONARDO FERREIRA DE CASTRO

PROJETO FINAL DE GRADUAÇÃO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE REDES DE COMUNICAÇÃO.

APROVADA POR:

SEBASTIÃO DO NASCIMENTO NETO, Mestre, UnB
(ORIENTADOR)

RAFAEL TIMÓTEO DE SOUZA JÚNIOR, Doutor, UnB
(EXAMINADOR INTERNO)

DATA: BRASÍLIA/DF, 04 DE SETEMBRO DE 2002.

FICHA CATALOGRÁFICA

CASTRO, LEONARDO FERREIRA DE.
OLIVEIRA, IGOR PEREIRA MARCIANO DE.
Estudo do Comportamento do TCP/IP em Enlaces Via Satélite [Distrito Federal] 2002.
xiv, 99p., 297 mm (ENE/FT/UnB, Engenheiro de Redes de Comunicação, Engenharia Elétrica, 2002).

Projeto Final de Graduação – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Satélite 2. TCP/IP
3. *Throughput*

I. ENE/FT/UnB. II. Estudo do Comportamento do TCP/IP em Enlaces Via Satélite

REFERÊNCIA BIBLIOGRÁFICA

OLIVEIRA, IGOR PEREIRA MARCIANO DE.; CASTRO, LEONARDO FERREIRA DE. (2002). Estudo do Comportamento do TCP/IP em Enlaces Via Satélite. Projeto Final de Graduação, Publicação G12/2002, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília-DF, 113p.

CESSÃO DE DIREITOS

NOME DOS AUTORES: IGOR PEREIRA MARCIANO DE OLIVEIRA
LEONARDO FERREIRA DE CASTRO
TÍTULO DA DISSERTAÇÃO: Estudo do Comportamento do TCP/IP em Enlaces Via Satélite
GRAU/ANO: Engenheiro de Redes de Comunicação/2002.

IGOR PEREIRA MARCIANO DE OLIVEIRA

SCLRN 714 BL. F, ENTRADA 40, AP. 101
CEP 70760-556 – Brasília-DF – Brasil

LEONARDO FERREIRA DE CASTRO

SQS 215 BL. F, AP. 507
CEP 70294-060 – Brasília-DF – Brasil

Dedicamos este trabalho a todas as pessoas que acreditam que o vencer está muito além do simples término da graduação.

Igor Pereira Marciano de Oliveira
&
Leonardo Ferreira de Castro

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter-me proporcionado boas condições para concluir o curso de Engenharia de Redes de Comunicação na Universidade de Brasília. Agradeço imensamente à minha mãe por ter-me presenteado com um veículo automotor, possibilitando que eu ficasse no laboratório até a madrugada realizando as várias etapas do projeto, sem correr riscos desnecessários. Agradeço a meu pai e irmão, por me apoiarem em todas as situações relacionadas à minha formação. Agradeço ao professor Sebastião do Nascimento Neto, que se mostrou ser amigo em todas as vezes que fez com que grandes problemas se transformassem em detalhes simplórios a serem resolvidos. Agradeço a ele também pelo empréstimo de seu Laptop, que infelizmente não chegou a funcionar conosco, mesmo depois de muito esforço. Agradeço aos demais professores do departamento, em especial aos do curso de Engenharia de Redes de Comunicação, Prof. Dr. Rafael Timóteo de Sousa Júnior e Prof. Claúdia Jacy Barenco, que mesmo em meio à vida corriqueira, souberam nos dar a atenção e palavra necessárias, acrescentando muito em nosso trabalho. Agradeço a Edimária Cerqueira, Fábio Buiati, Alexandre Vilarinho, Robson Albuquerque e Patrícia Mayumi, mestrandos de Engenharia de Redes e funcionários do LabRedes, que souberam, com muita paciência, compartilhar as instalações do laboratório conosco, nos ajudando no esclarecimento de algumas dúvidas ou simplesmente por estarem compartilhando conosco esta difícil etapa. Agradeço, em especial ao Alexandre, por nos emprestar seu LapTop, desta vez já funcionando... Agradeço, também, a todos os outros colegas, futuros Engenheiro de Redes de Comunicação, que, como nós, estão concluindo mais esta fase da vida. E como não poderia deixar de ser, agradeço ao meu grande amigo Leonardo Ferreira de Castro, companheiro neste trabalho, que com seu bom-humor, alegria e musicalidade, fez com que as horas árduas no laboratório fossem superadas com grande otimismo. Valeu, cumpadre... E a todos vocês, muito obrigado!
Igor Pereira Marciano de Oliveira

AGRADECIMENTOS

Agradeço ao meu paizão, Américo, minha amada mãe, Divina, meu irmão-amigo, Leandro, minha amada noiva, Priscila, os quais, com enorme paciência, compartilharam todos os momentos comigo, apoiando-me em cada um deles. Agradeço, também, ao pessoal do LabRedes, Edimária Rodrigues, Robson Albuquerque Fábio Buiati, Alexandre Vilarinho e Patrícia Mayumi, pela imensa colaboração; ao Prof. Dr. Rafael Timóteo de Sousa Júnior e Prof. Claúdia Jacy Barenco, pelas orientações. Agradeço, é claro, ao CNPq, que proporcionou boas condições para a realização do nosso trabalho, cedendo indiretamente várias máquinas do LabRedes para os alunos de projeto final. Não posso deixar de agradecer também ao Prof Edson Carvalho, do Departamento de Música, que, mesmo sem saber, contribuiu muito em nosso trabalho pelas aulas relaxantes de canto coral, as quais pudemos desfrutar neste semestre. Aos amigos da turma que me acompanharam por todo o curso, e que agora estão desfrutando da mesma alegria. Ao professor Sebastião do Nascimento Neto, pela sua disposição em nos ajudar e orientar neste projeto. Ao meu amigão, Engenheiro de Redes de Comunicação e músico Igor Pereira Marciano de Oliveira, pela grande alegria de ter trabalhado junto a uma pessoa bacana e de tanto humor e musicalidade, é claro. E o principal agradecimento, ao meu Querido Deus, Senhor da minha vida.

Enfim, a todos, deixo o meu Muito Obrigrado,

Leonardo Ferreira de Castro

RESUMO

O presente trabalho tem por objetivo estudar e avaliar alguns métodos de melhoramento da performance do TCP/IP sobre *links* de longo atraso, como é o caso dos satélites, levando em consideração todos os parâmetros envolvidos para uma transmissão de sucesso. Serão realizados vários testes comparativos com estudos já existentes, de modo a comprovar a prática do uso difundido desses mecanismos.

ABSTRACT

The present paper has for objective to study and to evaluate some methods of improvement of the performance of the TCP/IP over links of long delay, as it is the case of the satellites, taking into account all the involved parameters for a success transmission. Some comparative tests with existing studies will be carried through already, in order to prove the practical one of the spread out use of these mechanisms.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. INTRODUÇÃO

Quando em 1945, Artur Clarke, renomado escritor de ficção científica, escreveu um artigo sobre a possibilidade dos homens construírem um sistema de comunicação global utilizando satélites artificiais lançados no universo, ele antecipou uma possibilidade que em poucos anos tornou-se realidade entre os homens. No marco do lançamento do primeiro satélite artificial russo, o *Sputinik* em 1963, iniciava-se uma nova era de comunicação, que em pouco tempo inseriu milhões de pessoas num mesmo contexto de comunicação.

Com o advento da Internet e dos telefones celulares na década de 80, despertou-se a viabilidade da interação entre as pessoas de todo o mundo, consolidando a nova forma de comunicação e convivência. À medida em que essas tecnologias iam se desenvolvendo, juntamente com o aumento vertiginoso do número de pessoas capazes de acessar a Internet, novos serviços foram sendo criados, a fim de prover cada vez mais qualidade às comunicações. Dessa forma, alguns serviços multimídia e de banda larga, antes presentes somente em idéias, começaram a se tornar cada vez mais freqüentes nas aplicações que utilizam o TCP/IP como protocolos básicos de comunicação.

Entretanto, com o surgimento das novas aplicações e serviços multimídia, tornou-se óbvia a necessidade do uso de enlaces que proporcionassem largura de banda e confiabilidade suficientes para as transmissões. Atualmente, tecnologias de comunicação como a fibra óptica e os sistemas via satélite possibilitam cada vez mais rapidez e acessibilidade, além de uma largura de banda confiável. Situados entre uma variedade de opções para possibilitar comunicação de dados para os municípios, os satélites se destacam pelos totais alcance e rapidez na implantação do acesso e no oferecimento de alta velocidade para a Internet. Entretanto, neste último caso, existe um agravante importantíssimo que em hipótese alguma pode ser desconsiderado, já que define todos os parâmetros a serem ajustados para uma comunicação de sucesso: o atraso imposto pela distância entre o satélite e a superfície terrestre.

É dentro deste contexto que está inserido o presente trabalho, procurando contribuir na pesquisa das tendências do uso da Internet, através do entendimento dos protolocos TCP/IP e do uso da tecnologia de transmissão via satélite. Serão

apresentadas várias características inerentes à união das duas tecnologias, bem como a importância da existência de certos elementos para que uma comunicação seja realmente bem sucedida, levando-se em consideração os problemas envolvidos.

Importante ressaltar que grande parte do trabalho foi desenvolvida baseado na tese de mestrado de Fritz J. e Dolores H. Russ [14], membros da Escola de Engenharia e Tecnologia da Universidade de Ohio. Foram repetidos vários testes realizados por Fritz e Dolores, primeiramente, a fim de comprovarmos a necessidade do ajuste de certos parâmetros nos protocolos básicos utilizados.

Assim, contemplando esse assunto, o presente trabalho fica organizado da seguinte forma: no capítulo 2 serão abordadas as características gerais do TCP e do IP, bem como alguns detalhes importantes para o entendimento específico quando o ambiente dos satélites for ressaltado. O capítulo 3 tratar-se-á de explicar a Qualidade de Serviço sobre redes TCP/IP e seus parâmetros relevantes. O capítulo 4 estará responsável por fornecer as informações em nível físico necessárias a uma comunicação via satélite, tais como potência entregue à antena, freqüências de *uplink* e *downlink*, etc. No capítulo 5 serão apresentados os problemas intrínsecos ao ambiente dos satélites decorrentes do longo atraso imposto pelo tipo de comunicação. O capítulo 6 irá relatar os testes realizados por [14], de modo a fornecer uma referência para os nossos testes que serão relatados no capítulo 8. O capítulo 7 apresenta uma breve explicação sobre os *software*s de simulação e interpretação utilizados em nossos experimentos. Finalmente, o capítulo 9 é destinado à conclusão do trabalho, onde serão analisadas as importâncias das principais características observadas durante o trabalho.

2. TCP/IP – VISÃO GERAL

O *Transmission Control Protocol/Internet Protocol* (TCP/IP) é o conjunto de protocolos no qual a Internet e outros vários serviços estão baseados e amplamente difundidos atualmente. O TCP/IP, entretanto, foi desenvolvido sem levar em consideração seu desempenho sobre *links* de alta velocidade, como fibras ópticas, ou *links* de longo atraso, como os satélites, fazendo com que os esforços ficassem concentrados em remediar algumas limitações que se tornaram evidentes nos *links* cuja largura de banda é pouca aproveitada [42]. Atualmente, pouco se tem aproveitado da capacidade total disponível em satélites GEO, devido à incompleta utilização da largura de banda. Contudo, fazendo uso de intensificações convenientes para melhorar o desempenho do TCP/IP, taxas de transmissão acima de 500 Mbps podem ser alcançadas sobre os *links* de satélites geoestacionários.

A seguir, são apresentados detalhes importantes de ambos os protocolos de comunicação, de modo a facilitar o entendimento do processo quando a atenção estiver concentrada no ambiente dos satélites.

2.1 Internet Protocol

O IP (*Internet Protocol*) é um protocolo da camada de rede cuja função é permitir que o tráfego de dados flua diretamente entre diferentes tipos de mecanismos de transporte (Ethernet, ATM, Frame Relay, etc). O IP reside em dispositivos terminais (estações de trabalho, *hosts*) e em roteadores que funcionam como *switches* na rede, roteando datagramas[1] em direção a seu endereço de destino. A Figura 2.1 mostra a pilha de protocolos para uma rede TCP/IP.

[1] O datagrama é a unidade básica transferência do protocolo IP (ver Figura 2.2).

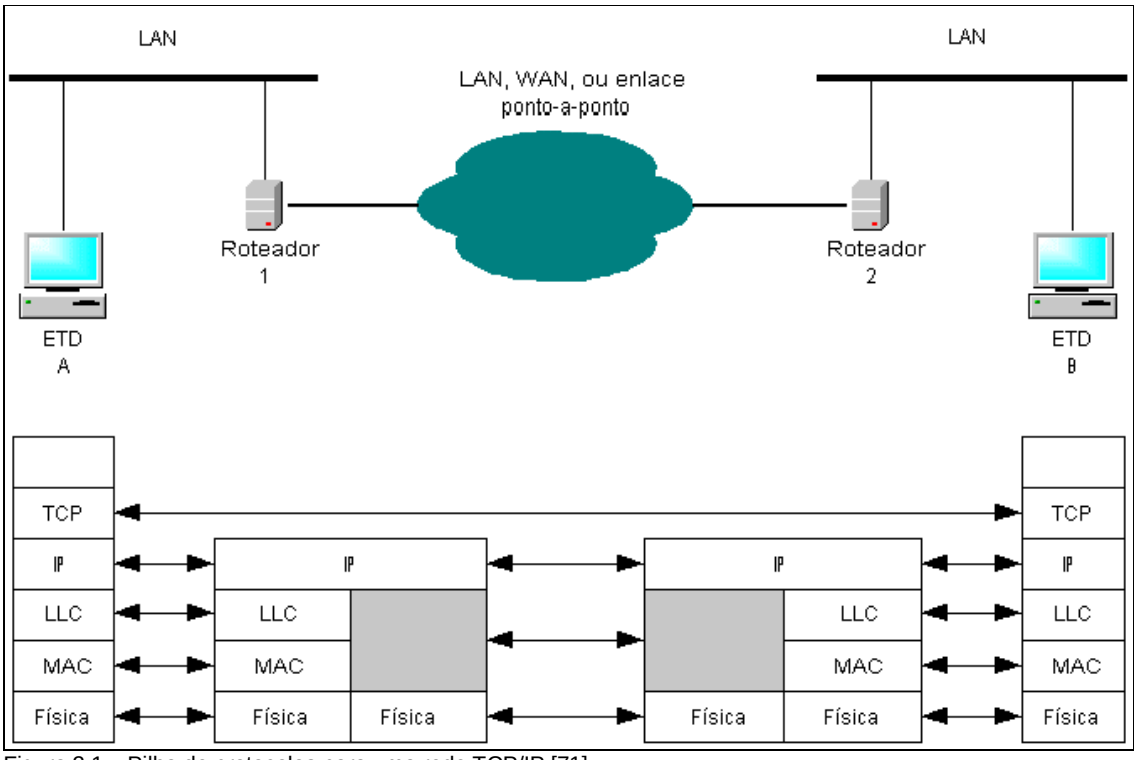


Figura 2.1 – Pilha de protocolos para uma rede TCP/IP [71].

Os roteadores na rede são necessários para interpretar diferentes esquemas de endereçamento. Por exemplo, LANs operando o padrão de IEEE 802 LAN, anexam dispositivos com endereços binários de 16 ou 48 bits. Uma rede pública de comutação de pacotes X.25, por outro lado, usa endereços decimais de 12 dígitos. O IP fornece um esquema de endereçamento global e um serviço de diretório. A versão atual do IP (IPv4) tem limitações na quantidade de endereços e, por isso, ameaçam inibir o crescimento da Internet. Já a nova versão (IPv6), apesar de ainda estar na fase de testes, apresenta melhoramentos no sentido de suprir a demanda de endereços [42] [71].

Especificamente, um endereço IP codifica a identificação da rede à qual um *host* se acopla. Podemos resumir que, a cada *host* de uma rede TCP/IP é atribuído um endereço único de 32 bits que é usado em todas as comunicações com aquele *host*.

Conceitualmente, cada endereço é um par (*netid, hostid*) em que *netid* identifica uma rede e *hostid* identifica uma conexão à rede por um *host*. Na prática, cada endereço IP deve ter uma das três primeiras formas mostradas na Figura 2.2.

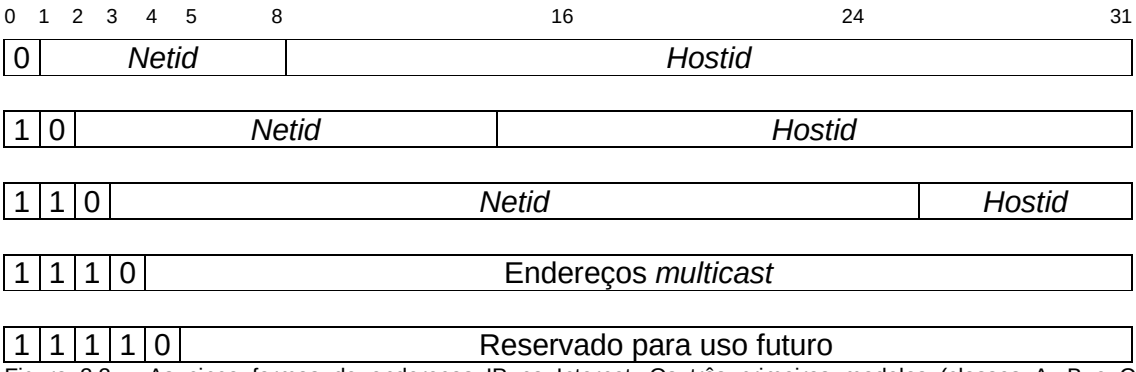


Figura 2.2 – As cinco formas de endereços IP na Internet. Os três primeiros modelos (classes A, B e C, respectivamente) podem ser diferenciados pelos três primeiros bits.

Dado um endereço IP, seu tipo pode ser determinado a partir dos três bits de mais alta ordem, sendo dois bits suficientes para distinguir entre as três classes principais:

- Endereços do tipo A, que dedicam 7 bits para *netid* e 24 bits para *hostid*.
- Endereços do tipo B, que dedicam 14 bits para o *netid* e 16 bits para o *hostid*.
- Endereços do tipo C, que dedicam 21 bits para o *netid* e 8 bits para o *hostid*.

Os endereços de rede podem ser usados tanto para se referir a redes, como também a *hosts* individuais. Por convenção, um endereço que tem todos os bits do *hostid* iguais a zero é reservado para se referir à própria rede. Além disso, os endereços IP podem ser usados para especificar difusão (*broadcast*) na rede; tais endereços mapeiam para endereços de *hardware* de difusão, se estiver disponível. Por convenção, um endereço de *broadcast* tem *hostid* com todos os bits ajustados a 1 [6].

2.1.1 Fragmentação

Roteadores também são necessários para tratar diferenças no tamanho dos pacotes que podem trafegar em redes distintas [42]. Cada tecnologia de transmissão de dados coloca um limite superior, fixo, no total de dados que podem ser transferidos em um quadro físico. Referimo-nos a esses limites como MTU (*Maximum Transfer Unit*). O tamanho da MTU pode ser bem pequeno; algumas tecnologias de *hardware* limitam a transferência para 128 octetos ou menos [6]. Por exemplo, redes X.25 comumente operam com pacotes tendo um tamanho máximo de 1.000 *bytes*, em contraste da Ethernet, que permite pacotes de até 1.500 *bytes*.

Para compensar essas diferenças, datagramas podem ser divididos em pacotes menores e reunidos novamente quando alcançarem seu destino: é o processo conhecimento como FRAGMENTAÇÃO. A fragmentação ocorre normalmente em um roteador situado em algum ponto ao longo do caminho entre a origem do datagrama e seu destino final. O roteador recebe um datagrama de uma rede com uma MTU grande, e precisa enviá-lo em uma rede para a qual a MTU seja menor do que o tamanho do datagrama, conforme mostra a Figura 2.3.

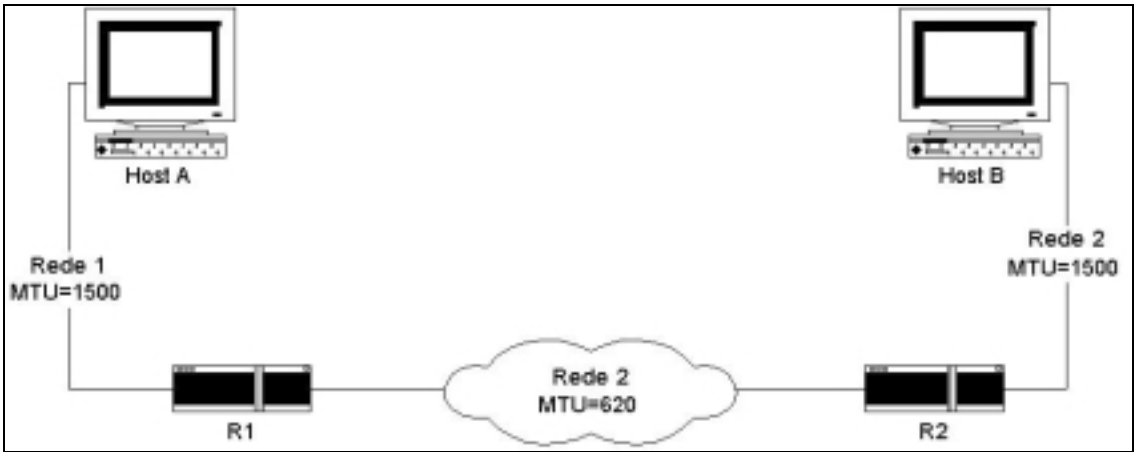


Figura 2.3 – Uma ilustração de onde ocorre a fragmentação. O roteador R1 fragmenta datagramas extensos enviados de A para B; R2 fragmenta datagramas extensos enviados de B para A.

Na figura, ambos os *hosts* conectam-se diretamente às Ethernets que possuam uma MTU de 1500 octetos. Desta forma, os dois *hosts* podem gerar e enviar datagramas de até 1500 octetos de comprimento. O caminho entre eles, entretanto, inclui uma rede com uma MTU de 620. Se o *host* A envia ao *host* B um datagrama maior que 620 octetos, o roteador R_1 fragmentará o datagrama. Da mesma forma, se B envia um datagrama grande a A, o roteador R_2 fragmentará o datagrama.

Fragmentar um datagrama significa dividi-lo em várias frações. Cada fragmento passa a conter um cabeçalho do datagrama original (exceto para um bit no campo *FLAGS* do datagrama IP (ver seção 2.1.2) que mostra que é um fragmento seguido por tantos dados quantos puderem ser transportados no fragmento), enquanto mantém o comprimento total menor que a MTU da rede na qual precisa trafegar.

Em uma rede TCP/IP, quando um datagrama tiver sido fragmentado, os fragmentos trafegam como datagramas isolados ao longo do percurso até o último

destino onde precisam ser remontados. A máquina receptora inicia um *temporizador de remontagem* quando recebe um fragmento inicial. Se o temporizador termina antes que todos os fragmentos cheguem, a máquina receptora descarta os fragmentos remanescentes sem processar o datagrama. Assim, a probabilidade de perda de datagrama cresce quando a fragmentação ocorre, porque a perda de um fragmento único resulta na perda do datagrama inteiro [6].

2.1.2 O Serviço IP

O IP tem seu serviço definido como um sistema de transmissão, não orientado à conexão, de melhor esforço (*best-effort*) e não-confiável. O serviço é conhecido como NÃO-CONFIÁVEL porque a entrega não é garantida. O datagrama pode ser perdido, reproduzido, atrasar-se ou ser entregue com problemas, mas o serviço não detectará tais condições, nem informará isso ao transmissor nem ao receptor. Ele é denominado NÃO ORIENTADO À CONEXÃO porque cada datagrama é independente dos outros. Uma seqüência de datagramas enviados de um computador a outro pode trafegar por caminhos diferentes, ou alguns podem ser perdidos enquanto outros são entregues. Finalmente, o serviço utiliza uma transmissão *BEST-EFFORT* porque o IP faz uma série de tentativas para entregar os datagramas. Isso significa que a rede não rejeita pacotes por simples capricho; a não-confiabilidade surge quando os recursos esgotam-se ou as redes básicas falham (por exemplo, as redes tornam-se congestionadas pela falta de memória dos *buffers* de roteadores intermediários [42]) [6].

Como um quadro[2] típico de rede física, um datagrama é dividido em cabeçalho (*header*) e áreas de dados (*payload*). Similar a um quadro, o cabeçalho de um datagrama contém os endereços de origem e destino e um tipo de campo que identifica o conteúdo do datagrama. A Figura 2.4 abaixo mostra o formato detalhado de um datagrama IP (IPv4).

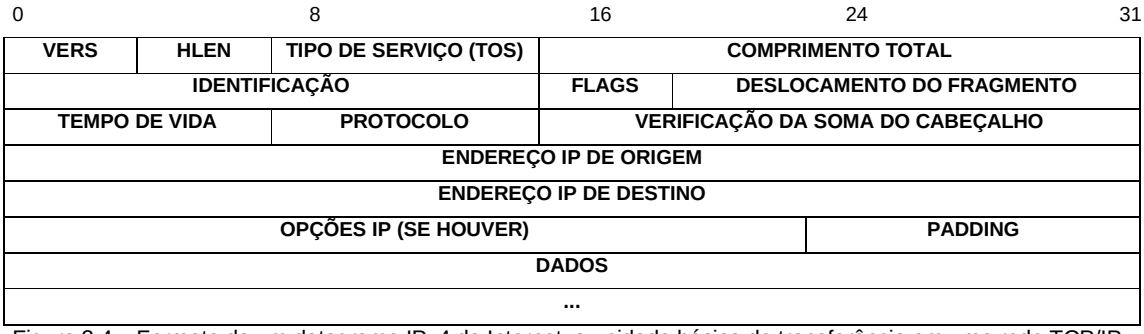


Figura 2.4 – Formato de um datagrama IPv4 da Internet, a unidade básica de transferência em uma rede TCP/IP.

Visto que o processamento de datagramas se dá em *softwares*, o conteúdo e o formato não são restringidos por quaisquer *hardwares*. O primeiro campo de quatro bits de um datagrama (VERS), por exemplo, contém a versão do protocolo IP utilizada para criar o datagrama. Ele é utilizado para verificar se o transmissor, o receptor e quaisquer roteadores existentes entre eles concordam quanto ao formato do datagrama. Todo *software* IP precisa verificar o campo de versão antes de processar um datagrama, para assegurar-se de que ele se adapta ao formato que o *software* espera. Se os padrões mudarem, as máquinas rejeitarão datagramas com versões de protocolo diferentes dos seus, impedindo que eles deturpem o conteúdo do datagrama com um formato desatualizado. A versão atual do protocolo IP é a quatro (IPv4).

O campo de comprimento do cabeçalho (HLEN), também de quatro bits, fornece o comprimento do cabeçalho do datagrama medido em palavras de 32 bits. Todos os campos do cabeçalho contêm um comprimento fixo, exceto o campo OPÇÕES IP e os campos correspondentes, PADDING. O cabeçalho mais básico, que não contém qualquer opção e nenhum preenchimento, mede 20 octetos e o campo de comprimento de cabeçalho é igual a 5 (cinco), como pode ser facilmente verificado pelo leitor.

O campo COMPRIMENTO TOTAL fornece o comprimento do datagrama IP medido em octetos, incluindo os octetos no cabeçalho e nos dados. O tamanho da área de dados pode ser calculado subtraindo-se de COMPRIMENTO TOTAL o comprimento do cabeçalho (HLEN). Já que o campo COMPRIMENTO TOTAL possui 16 bits de comprimento, o maior tamanho possível para um datagrama IP é de 2^{16} ou 65.535 octetos.

[2] Unidade básica de transferências dos protocolos da Camada de Enlace, levando-se em conta o MODELO OSI.

O campo TIPO DE SERVIÇO (*Type of Service – TOS*), de oito bits, especifica como o datagrama deve ser tratado e é fracionado em cinco subcampos, conforme mostra a figura abaixo.

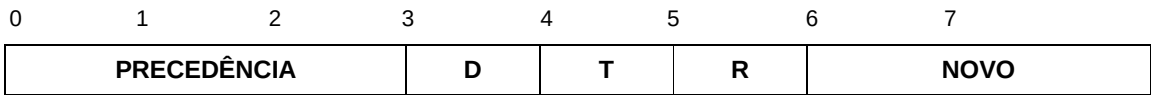


Figura 2.5 – Os cinco subcampos que compõem o campo SERVICE TYPE, de oito bits.

Três bits PRECEDÊNCIA especificam a precedência do datagrama com valores variando de zero (precedência normal) até sete (controle de rede), permitindo que os transmissores indiquem a importância de cada datagrama. Embora a maioria dos *softwares* que rodam em *hosts* ignore o tipo de serviço, trata-se de um conceito importante, porque fornece um mecanismo que pode permitir que informações de controle (ou voz) tenham precedência sobre dados. Se, por exemplo, todos os *hosts* e roteadores reconhecem a precedência, é possível implementar algoritmos de controle de congestionamentos que não sejam influenciados pelo congestionamento que estão tentando controlar. Além disso, como será visto mais adiante, esse campo é fundamental na identificação do tipo de QoS usado na transmissão.

Os bits D, T e R especificam o tipo de transporte que o datagrama deseja. Quando ajustado, o bit D solicita um intervalo baixo, o bit T solicita um *throughput* alto e o bit R solicita alta confiabilidade. É claro que não deve ser possível que a rede garanta o tipo de transporte solicitado (ou seja, pode acontecer que nenhum caminho para o destino tenha a propriedade solicitada). Assim, consideramos a solicitação de transporte como uma sugestão para os algoritmos de roteamento, e não uma exigência. Se um roteador realmente conhece mais de uma rota possível para determinado destino, ele pode utilizar o tipo de campo de transporte para selecionar aquela cujas características mais se aproximem das desejadas. Suponha, por exemplo, que um roteador possa selecionar entre uma linha alugada, de baixa capacidade, e uma conexão alta, de satélites de banda larga (mas de intervalo de tempo alto). O conjunto de bits D poderia solicitar aos datagramas que carregam toques no teclado de um usuário para um computador remoto que esses sejam entregues o mias rápido possível, enquanto um conjunto de bits T poderia solicitar

aos datagramas correspondentes uma transferência de arquivos que trafeguem nos *links* de alta capacidade de satélites.

Também é importante entender que os algoritmos de roteamento precisam escolher entre tecnologias de rede físicas básicas as quais possuem, cada uma, características de intervalo, *throughput* e confiabilidade. De uma maneira geral, algumas tecnologias representam um compromisso entre duas características. Desta forma, a idéia é apresentar uma sugestão ao algoritmo de roteamento sobre o que é mais importante, e raramente faz sentido especificar os três tipos de serviço. Com isso, considera-se o tipo de especificação de transporte uma sugestão ao algoritmo de roteamento para ajudá-lo a escolher entre os vários caminhos para um destino, com base em seu conhecimento das tecnologias de *hardware* disponíveis nesses caminhos. Uma interligação em redes não garante o tipo de transporte solicitado.

O campo TEMPO DE VIDA (*Time to Live - TTL*) especifica quanto tempo, em segundos, o datagrama pode permanecer no sistema. A idéia é simples e importante: toda vez que uma máquina injeta um datagrama na rede, ela estabelece um tempo máximo de vida para o datagrama. Os roteadores e os *hosts* que processam datagramas precisam decrementar o campo TTL à medida que o tempo passa e remover o datagrama da interligação em redes quando seu tempo expira. Sempre que um campo TTL alcança zero, o roteador descarta o datagrama e envia uma mensagem de erro de volta à origem.

O campo PROTOCOLO é análogo ao campo de TIPO em um quadro de rede. O valor no campo PROTOCOLO especifica qual protocolo de alto nível foi utilizado para criar a mensagem que está sendo transportada na área DADOS do datagrama. Na verdade, o valor PROTOCOLO especifica o formato da área DADOS (*payload*). O mapeamento entre um protocolo de alto nível e o valor de número central para garantir um consenso em toda a Internet.

O campo VERIFICAÇÃO DA SOMA DO CABEÇALHO assegura a integridade dos valores de cabeçalho. A verificação IP é formada com o tratamento do cabeçalho como uma seqüência de números inteiros de 1 bit (na ordem de *bytes* da rede), reunindo-os com uma aritmética de complemento de um, e a seguir considerando o complemento de um como o resultado. Para finalidade de calcular a soma de verificação, considera-se que o campo VERIFICAÇÃO DA SOMA DO CABEÇALHO contenha zero. É importante observar que a soma de verificação somente se aplica a valores do cabeçalho IP, e não aos dados.

O campo denominado PADDING depende das opções selecionadas no campo OPÇÕES IP, que é de tamanho variável. O campo PADDING representa bits contendo zero e que podem ser necessários para garantir que o cabeçalho do datagrama se estenda até o múltiplo exato de 32 bits [6].

Dessa forma, juntamente com o TCP, o IP fornece as bases para transmissão e endereçamento das mensagens sobre as mais variadas tecnologias e topologias de redes, permitindo adequação e funcionalidade.

2.2 Transmission Control Protocol

O *Transmission Control Protocol* (TCP) [26] [11], em contraste ao IP, é um protocolo de transporte seguro, orientado à conexão, e, como já foi dito, é usado pela maioria dos serviços de comunicação na Internet. Como exemplo, podem ser citados: SNMP [25], NNTP [3], HTTP [54] [63], FTP [24]. O TCP reside em terminais finais (estações de trabalho, *hosts*) para garantir a entrega apropriada de uma mensagem.

O TCP proporciona uma entrega segura dos segmentos de dados através de um mecanismo de confirmação de recepção. Cada segmento[3] de dados é transmitido contendo um número de seqüência de 32 bits, indicando a posição do dado na transmissão. O número de seqüência é determinado por um número de seqüência base, gerado pelo sistema operacional, e também pela posição relativa do primeiro octeto de dados na transmissão. Por exemplo, assumindo que um transmissor TCP com número de seqüência base X transmita segmentos de 100 *bytes*, o primeiro segmento conterá número de seqüência igual a X e o segundo segmento irá conter número de seqüência igual a X+100. Por simplicidade, este documento trata o número de seqüência como um “número de segmento” ao invés do já tipicamente conhecido “número de octeto“. A Figura 2.6 mostra um exemplo de como acontece a atribuição de números de seqüência aos cabeçalhos dos segmentos TCP.

[3] A unidade básica de transferência entre dois *hosts* em uma conexão TCP é chamada segmento ou pacote. Um segmento consiste de um cabeçalho (*header*) TCP associado a seu respectivo campo de dados (*payload*).

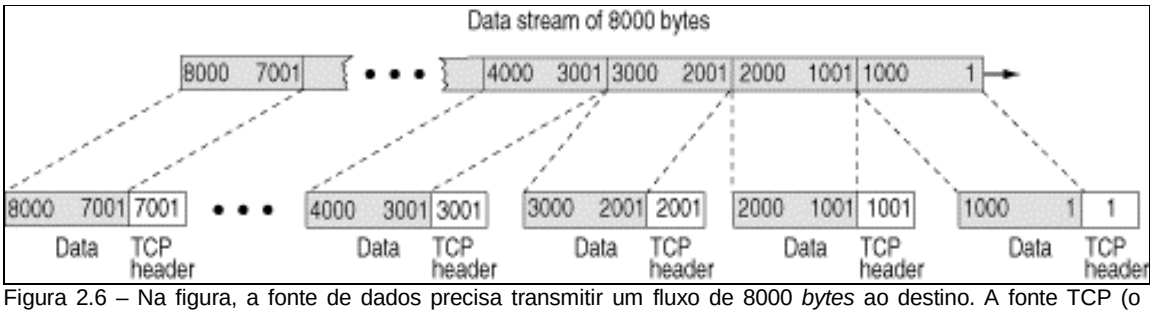


Figura 2.6 – Na figura, a fonte de dados precisa transmitir um fluxo de 8000 *bytes* ao destino. A fonte TCP (o transmissor) divide os 8000 *bytes* em grupos de 1000 *bytes* e atribui um cabeçalho TCP a cada grupo, criando 8 segmentos TCP. Note que o número de seqüência em cada cabeçalho TCP representa o número que o transmissor TCP designa ao primeiro *byte* de dados em cada segmento.

Além disso, através da atribuição de números de seqüência, o receptor é capaz de responder ao transmissor pelo envio de uma confirmação de recebimento (*ACK*, do inglês *acknowledgement*) pelos segmentos recebidos. Cada ACK contém o número de seqüência do próximo segmento "em ordem" que o receptor espera receber. A Figura 2.7 ilustra como é possível para o receptor responder pela recepção de segmentos através do envio de um ACK para cada segmento individualmente ou enviando um ACK para um conjunto de segmentos.

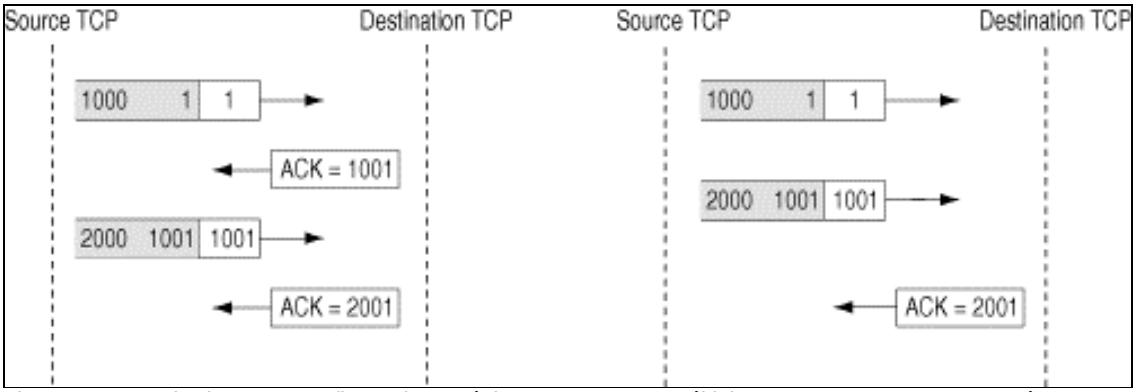


Figura 2.7 – Envio de ACKs confirmando um único segmento ou múltiplos segmentos. Note que o número ACK usado pelo receptor é o número do próximo *byte* do fluxo que o receptor espera receber. Se o receptor TCP envia o ACK de número 2001, isso informa à fonte que todos os segmentos até o *byte* de número 2000 foram recebidos corretamente.

Incluir números de seqüência dentro dos cabeçalhos dos segmentos permite o receptor reagrupar o fluxo de dados caso os segmentos cheguem fora de ordem. Na realidade, isso é bastante freqüente, já que o protocolo de rede responsável por encapsular os segmentos TCP é o IP (ver seção 2.1.2). Quando o destino TCP recebe um segmento fora de ordem, ele imediatamente envia um ACK duplicado ao transmissor, enfatizando a requisição do próximo segmento em ordem.

A Tabela 2.1 contém um exemplo do comportamento de ACKs em uma transmissão TCP.

Segmento Recebido	ACK Enviado
1	2
2	3
3	4
5	4
6	4
4	7

Tabela 2.1 – Exemplo de ACK cumulativo no TCP. Esta tabela mostra o comportamento da confirmação de recebimento do TCP do ponto de vista do receptor. Cada segmento recebido define a transmissão de um ACK cumulativo contendo o número de seqüência do próximo segmento "em ordem" esperado pelo receptor.

A tabela mostra que o segmento 1 gera um ACK contendo número de seqüência 2. Isso indica que o próximo segmento esperado pelo receptor deve conter número de seqüência igual a 2. A chegada dos segmentos 2 e 3 ao receptor geram ACKs da mesma forma. Note que o segmento 4 não chega ao receptor antes dos segmentos 5 e 6. O receptor, então, gera ACKs contendo número de seqüência igual a 4 em resposta a esses segmentos, já que o segmento 4 é o próximo segmento "em ordem" esperado pelo receptor. Quando o segmento 4 chega, o ACK enviado contém número de seqüência igual a 7, indicando que o próximo segmento esperado pelo receptor deve conter número de seqüência igual a 7. Se o transmissor não receber um ACK referente a um certo segmento dentro de um intervalo de tempo predefinido, o segmento é retransmitido. O intervalo de tempo que o transmissor espera por um ACK antes de retransmitir o segmento de dados é chamado de *Retransmited Timeout*[4] (RTO).

Para ilustrar melhor o envio e a recepção de ACKs duplicados, a Figura 2.8 é mostrada a seguir.

[4] O RTO é uma estimação do *Round Trip Time* (RTT) (duração do percurso completo (de ida e volta) na transmissão de um segmento) somado a alguma variação. Os detalhes exatos do cálculo do RTO são dados em [52] e [65].

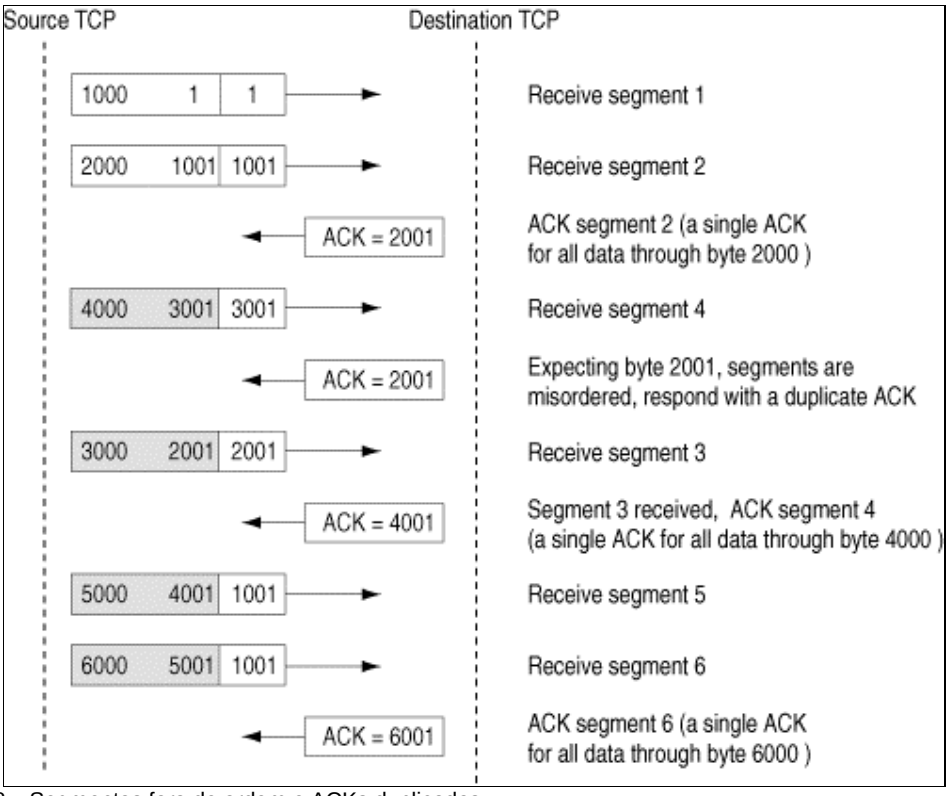


Figura 2.8 – Segmentos fora de ordem e ACKs duplicados.

2.2.1 Tamanho da Janela

Outro aspecto importantíssimo a respeito do TCP é o fato dele ser um protocolo de Janela Deslizante. Um protocolo de janela deslizante permite que o transmissor envie vários segmentos antes de receber um ACK. Quando um ACK é recebido pelo transmissor, a janela desliza (desloca-se) a fim de permitir que mais uma leva de segmentos seja transmitida. Cada segmento TCP enviado (segmento de dados e ACKs) contém uma Janela de Anúncio (ou Janela de Aviso) determinada no início da comunicação. O tamanho da janela de anúncio do receptor é o limite superior da janela deslizante do transmissor. A maior janela padrão que o TCP pode "anunciar" é de 65.535 *bytes* devido aos 16 bits alocados para isso no cabeçalho TCP. Isso limita o *throughput* a 2^{16} *bytes* divididos pelo tempo de resposta (*Round Trip Delay* ou *Round Trip Time* (RTT)) do circuito. Para um caminho GEO com um RTT de 600 ms, isso corresponde aproximadamente a 840 kbps [42]. A Figura 2.9 mostra o efeito do *Round Trip Time* (RTT) no *throughput* em função do tamanho da janela. Pode ser visto que um longo *link* terrestre de fibra, com um RTT de 200 ms,

estaria limitado a 2.6 Mbps com o mesmo tamanho máximo da janela TCP, 64 k*bytes*.

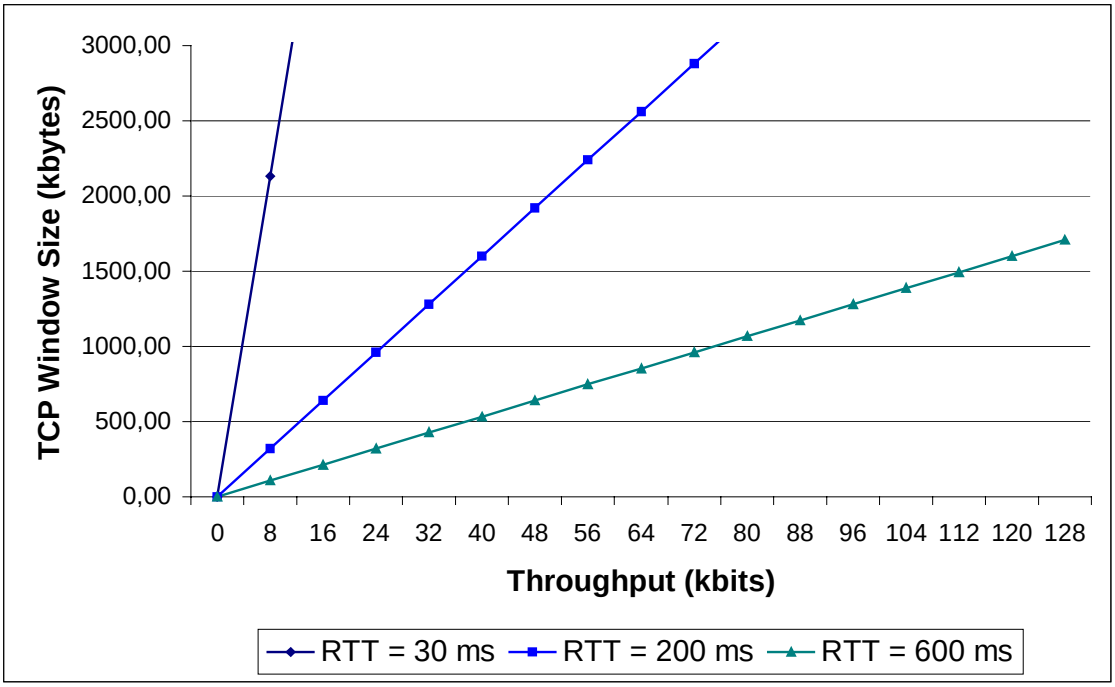


Figura 2.9 – Máximo *throughput* por uma simples conexão TCP em função do tamanho máximo da janela e do RTT.

2.2.2 Reconhecimentos Seletivos (SACKs)

O TCP assegura a entrega completa de dados sobre um *link* pela retransmissão de qualquer pacote que não tenha sido reconhecido (*ACKed*). Isto é, transmite-se tudo que foi enviado desde o último pacote reconhecido. Esse esquema é claramente ineficiente numa situação onde muitos *bytes* de um pacote foram recebidos corretamente e somente um ou dois chegaram corrompidos. Nessas situações, é preferível retransmitir somente a informação corrompida, isto é, fazer um reconhecimento seletivo. O TCP já foi modificado e amplamente desenvolvido com esta capacidade [14]. É, portanto, possível se implementar essa potencialidade nos *links* de satélite através da introdução de interfaces apropriadas nas estações terrestres em cada final de *link*.

Uma modificação relativamente direta do TCP que, de alguma maneira, remedia algumas de suas falhas atuais tem sido aprovada pelo IETF [19]. Isso permite o reconhecimento dos pacotes recebidos corretamente, mas fora de ordem.

Essa nova característica foi denominada Reconhecimento Seletivo (*Seletive Acknowledgment*, *SACK*) [42]. Os SACKs fornecem a informação sobre quais segmentos exatamente chegaram no receptor e permitem que o transmissor TCP retransmita somente aqueles segmentos que foram perdidos na rede. Um SACK é definido pelo TCP em [39]. Os SACKs têm sido provados com sucesso em muitos protocolos de pesquisas (por exemplo, VMTP [8],NETBLT [9], RDP [54]), bem como o TCP [14] [28] [36].

2.2.3 Controle de Congestionamento

Um ponto importante a ser considerado em qualquer explanação mais aprofundada sobre o TCP, é saber que ele usa um conjunto de algoritmos de Controle de Congestionamento que favorece o controle do processo de envio de segmentos TCP. Esses mecanismos são importantes porque asseguram que o TCP não transmitirá dados a uma taxa imprópria para os recursos de rede disponíveis. Se a taxa de transmissão do TCP for muito alta, os roteadores intermediários da rede podem ter seus *buffers* sobrecarregados. Caso um roteador intermediário receba segmentos de dados mais rapidamente do que é capaz de retransmitir, os segmentos serão enfileirados para posterior processamento. Se segmentos chegarem a um roteador que não é provido de memória suficiente para enfileirá-los, os segmentos serão descartados. Portanto, é importante que o TCP seja capaz de adaptar sua taxa de envio às condições da rede para evitar perda de segmentos. Quando muitas conexões TCP são estabelecidas a uma taxa demasiadamente alta (e, portanto, imprópria), a rede pode sofrer um Colapso de Congestionamento (*Congestive Collapse*). Colapso de Congestionamento é um estado em que os segmentos de dados estão sendo injetados na rede, porém está-se aproveitando muito pouco desse fluxo da entrada de dados; a maioria dos segmentos de dados e seus ACKs correspondentes são descartados por um dos roteadores intermediários da rede antes de alcançar seu destino. Isso acaba fazendo com que o transmissor retransmita os dados, favorecendo o agravamento do problema.

Os algoritmos de controle de congestionamento do TCP tentam prevenir esse colapso de congestionamento através da detecção da ocorrência do congestionamento e, conseqüentemente, pela redução da taxa de transmissão. Ao passo em que esses algoritmos são importantes, eles também podem ter um

impacto negativo na performance do TCP em canais de satélite, o que será explicado num momento apropriado. Os quatro algoritmos de controle do congestionamento do TCP são SLOW START, CONGESTION AVOIDANCE, FAST RETRANSMIT e FAST RECOVERY.

2.2.3.1 SLOW START E CONGESTION AVOINDANCE

Os algoritmos Slow Start e Congestion Avoidance permitem o TCP incrementar a taxa de transmissão de dados sem exceder a capacidade dos *buffers* dos roteadores intermediários. Para realizar isso, os transmissores TCP usam uma variável chamada *Congestion Window* (Janela de Congestionamento), *cwnd*. A janela de congestionamento (*cwnd*) do TCP é o tamanho da janela deslizante usada pelo transmissor e, portanto, não pode exceder o tamanho da janela de anúncio do receptor. Por isso, o TCP não pode injetar mais do que *cwnd* segmentos de dados sem reconhecimento[5] na rede.

Quando uma conexão TCP é estabelecida, a fonte TCP não transmite uma janela de anúncio do receptor completa; ao invés disso, o transmissor evita exceder a capacidade da rede no início e transmite poucos pacotes somente, aguardando os ACKs respectivos. Isso permite ao transmissor sondar a rede a fim de determinar a largura de banda para a conexão disponível no momento. Esse é o mecanismo de Slow Start. A Figura 2.10 mostra um diagrama de blocos ilustrando o funcionamento do TCP e os momentos de ativação do mecanismo Slow Start.

Como se pode ver na figura, o Slow Start é usado [27]:

- no início de cada nova conexão TCP;
- quando uma conexão TCP é reinicializada depois de longo período de ociosidade;
- quando uma conexão TCP é reinicializada depois de ocorrer o RTO (isto é, em certas instâncias depois de detectado o congestionamento).

[5] Sem reconhecimento (não reconhecidos): refere-se ao número de segmentos que são injetados na rede e que, ao chegarem no receptor, permitem o envio de um único ACK como confirmação de reconhecimento; é exatamente o tamanho da janela do transmissor.

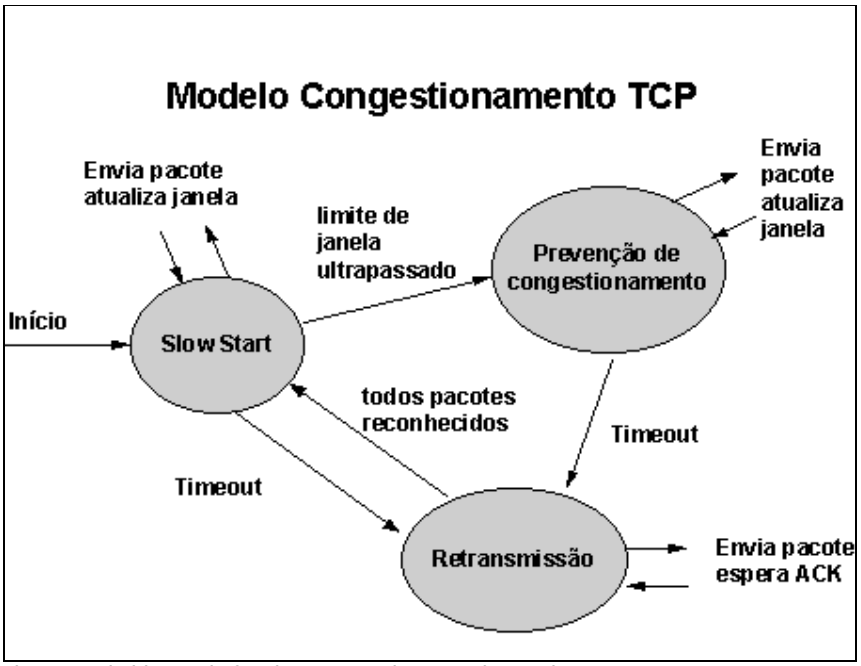


Figura 2.10 – Diagrama de blocos do funcionamento do mecanismo Slow Start.

O algoritmo começa pela inicialização de *cwnd* para 1 segmento[6]. O transmissor, então, envia o segmento, aguardando o ACK correspondente. Quando o ACK chega, a fonte incrementa *cwnd* de 1 para 2, e 2 segmentos são enviados. Quando esses dois segmentos são reconhecidos pelo transmissor, *cwnd* é incrementado de 2 para 4 segmentos, e 4 segmentos são enviados. Esse algoritmo proporciona um crescimento exponencial no tamanho da janela deslizante. O Slow Start continua até que o tamanho de *cwnd* alcance o "*slow start threshold*" (*ssthresh*) (limiar do Slow Start) ou até que seja detectada uma perda de segmento. O valor de *ssthresh* é inicializado para o tamanho da janela de anúncio do receptor, no início da conexão. Se o RTO do TCP expirar para um dado segmento, o TCP retransmite o segmento e aproveita para usar este mesmo segmento como uma indicação de congestionamento na rede. Esse procedimento do Slow Start será finalizado quando o valor de *cwnd* for igual ou superior ao *ssthresh* ou quando uma outra perda de dados for detectada. O novo valor de *ssthresh* coloca um limite superior no algoritmo Slow Start na metade da taxa de transmissão quando uma perda for detectada [14]. A Figura 2.11 e a Figura 2.12 tentam ilustrar o funcionamento do Slow Start e o incremento exponencial que esse mecanismo proporciona ao *cwnd*.

[6] Na prática, *cwnd* é medido em *bytes*; contudo, para simplificar a abordagem, vamos expressa-lo em termos de segmentos.

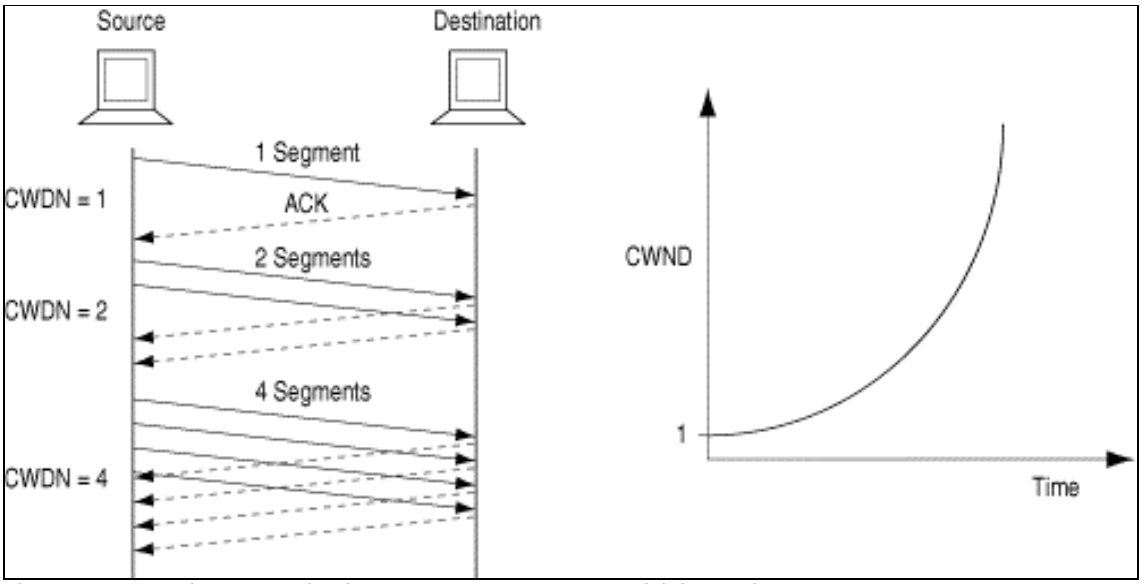


Figura 2.11 – Funcionamento do Slow Start e aumento exponencial do *cwnd*.

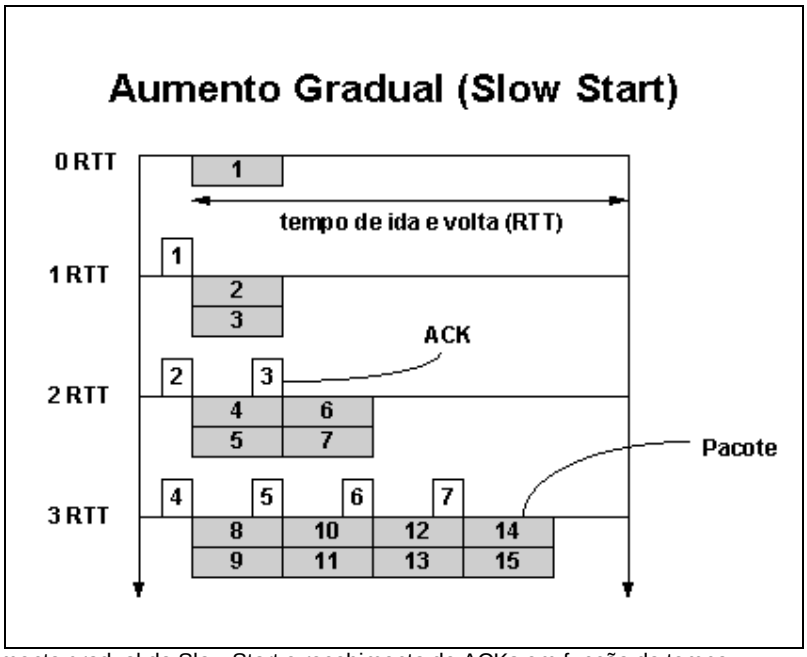


Figura 2.12 – Aumento gradual do Slow Start e recebimento de ACKs em função do tempo.

Com o Slow Start, o transmissor deve controlar uma janela de congestionamento (*cwnd*) que representa sua estimativa da quantidade de tráfego que a rede pode absorver sem entrar num Colapso de Congestionamento. Quando uma sessão TCP é estabelecida inicialmente, *cwnd* é inicializado para 1 único segmento anunciado pelo *host* de destino (receptor). O transmissor pode transmitir o mínimo de seu *cwnd* (representando o controle de fluxo imposto pelo transmissor) e

a janela de anúncio do receptor (representando o controle de fluxo imposto pelo receptor) [27].

O Congestion Avoidance é a fase seguinte ao Slow Start. Nessa fase, o valor de *cwnd* é maior ou igual ao *ssthresh*. Quando o transmissor TCP percebe que um pacote foi descartado ou quando ocorrer um *timeout*, o TCP reduz sua taxa de transmissão pelo ajuste do *ssthresh* para a metade do valor de *cwnd*, e, assim, ajusta o *cwnd* para 1 segmento novamente. Esse algoritmo incrementa o *cwnd* a uma taxa menor do que no Slow Start. Para cada segmento reconhecido durante o Congestion Avoidance, a janela de congestionamento é acrescida de 1/*cwnd* (a não ser que isso faça o valor de *cwnd* ficar maior do que a janela de anúncio do receptor). Isso dificilmente adiciona 1 segmento ao valor de *cwnd*. O algoritmo Congestion Avoidance fornece um incremento linear no tamanho da janela deslizante do TCP. Esse mecanismo é usado para sondar a rede para uma capacidade adicional num modo conservativo. A Figura 2.13 mostra os algoritmos Slow Start e Congestion Avoidance em função do tempo. Note que quando o valor de *cwnd* é menor ou igual ao *ssthresh*, a fonte TCP está operando o mecanismo Slow Start; quando *cwnd* é maior que *ssthresh*, está-se operando o Congestion Avoidance.

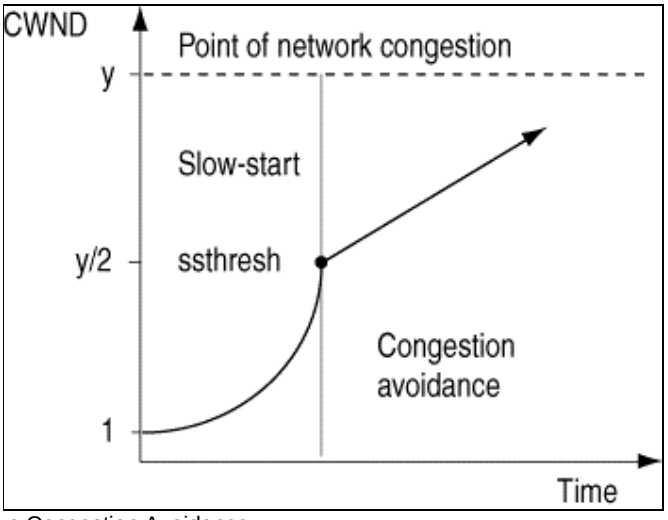


Figura 2.13 – Slow Start e Congestion Avoidance.

O Slow Start juntamente com o Congestion Avoidance causam uma redução à metade no valor de *cwnd*, cada vez que ocorre uma perda de pacote. Conseqüentemente, se o congestionamento durar por um período de tempo

contínuo, o volume de tráfego injetado na rede e a taxa de retransmissão pela fonte decrescem exponencialmente. Isso faz com que a os roteadores intermediários “esvaziem” suas filas de processamento.

2.2.3.2 FAST RETRANSMIT E FAST RECOVERY

Os algoritmos Fast Retransmit e Fast Recovery permitem o TCP detectar e recuperar segmentos descartados mais efetivamente do que somente confiando no RTO. Como definido no padrão TCP [26], o TCP pode retransmitir um segmento caso o RTO expire antes desse segmento ser reconhecido (*ACKed*), isto é, antes do ACK ser recebido. Esse mecanismo trabalha bem se a granulosidade do temporizador que o TCP usa for menor ou igual ao RTO. Contudo, a granulosidade do temporizador do S.O. BSD Unix é de 500 ms [15], o que é claramente insuficiente para gerar retransmissão na maioria das redes terrestres (já que têm RTTs menores do que 500 ms). A implementação do TCP está gratuitamente disponível e tem sido usada como base de muitas outras implementações de TCP. As investigações mostram que temporizadores com granulosidade de aproximadamente 500 ms estão prevalecendo.

O Fast Retransmit provê uma maneira de retransmitir um segmento antes do RTO expirar. Como discutido na seção 1.1, o TCP gera ACKs duplicados quando os segmentos chegam fora de ordem. Quando um número pequeno (geralmente 3) de ACKs duplicados chega ao transmissor, o TCP usa isso como uma indicação de que um segmento foi perdido, e retransmite o segmento apropriado. Junto a isso, o TCP usa o segmento perdido como uma indicação de congestionamento na rede e reduz a taxa de transmissão. Quando um segmento é retransmitido usando Fast Retransmit, o algoritmo Fast Recovery é acionado, como segue a explicação. O *cwnd* é reduzido à metade e o *ssthresh* é ajustado para o novo valor de *cwnd*. Cada ACK duplicado, recebido pelo transmissor, indica que um segmento chegou ao receptor que não está muito distante na rede. O TCP usa esse conhecimento para inferir que a rede pode absorver um outro segmento e artificialmente infla o *cwnd* de 1 segmento. Novos segmentos de dados podem ser enviados se o valor do *cwnd* for maior do que o número de segmentos não reconhecidos na rede. Sobre a recepção de um ACK não duplicado, o TCP reduz o *cwnd* à quantidade que foi inflado artificialmente (isto é, volta ao valor de *ssthresh*) e o algoritmo Congestion

Avoidance é usado como descrito na seção 2.2.3.1. Abaixo é mostrada a figura que ilustra os mecanismos de Fast Retransmit e Fast Recovery, levando em conta o tamanho de *cwnd*.

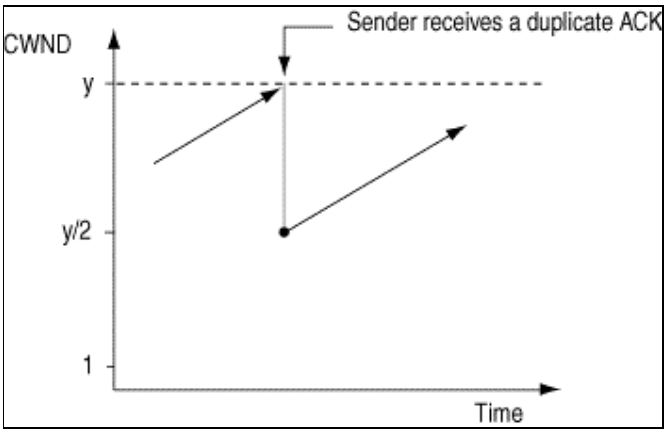


Figura 2.14 – Fast Recovery depois de acionado o Fast Retransmit.

O TCP reduz sua taxa de transmissão para cada segmento perdido que for detectado, mas a quantidade da redução depende da maneira em que os segmentos perdidos foram detectados. Quando a retransmissão é adequada à expiração do RTO, o TCP fica incapaz de inferir qualquer coisa sobre o estado da rede e, portanto, inicia o Slow Start (acima da metade da taxa de transmissão quando ocorrer perda de segmento). Quando a retransmissão é iniciada pelo mecanismo de Fast Retransmit, o TCP está recebendo ACKs duplicados indicando que os segmentos ainda estão seguindo entre o transmissor e o receptor. Portanto, a redução da taxa de transmissão não é tão brusca quanto quando o transmissor não recebe um *feedback*. Quando a transmissão é iniciada por ACKs duplicados, o TCP reduz a taxa de envio pela metade [14].

2.2.4 Exemplo de Uma Sessão TCP Típica

A fim de tentar explicar melhor o funcionamento dos algoritmos de controle de congestionamento usados pelo TCP, será abordado aqui um exemplo de *throughput* para uma sessão TCP típica. A Figura 2.15 mostra o comportamento dos mecanismos de controle, bem como de *cwnd* em função do tempo.

Quando uma conexão TCP é estabelecida inicialmente, a fonte TCP tenta evitar sobrecarga na rede assumindo que se trata de uma rede de pouca

capacidade. Como mostrado no *throughput* da curva A, o TCP inicia com o Slow Start, mas rapidamente incrementa sua taxa de transmissão para determinar a capacidade atual da rede.

Eventualmente o Slow Start alcança a taxa de transmissão onde a fonte TCP recebe ACKs duplicados indicando perda ou segmentos fora de ordem no meio da uma janela transmissão. Como mostrado no *throughput* da curva B, o transmissor TCP ajusta o *ssthresh* para a metade do valor de *cwnd* e então reinicia rapidamente com o Congestion Avoidance até se aproximar do valor anterior de *cwnd*, que resultou em perdas de pacotes.

Reiniciar rapidamente com o Congestion Avoidance eventualmente alcançará a taxa de transmissão em que a fonte recebe ACKs duplicados indicando perda ou segmentos fora de ordem no meio da uma janela transmissão. Conforme mostrado na curva C, o transmissor TCP ajusta o *ssthresh* para a metade do valor de *cwnd* e então reinicia rapidamente com o Congestion Avoidance.

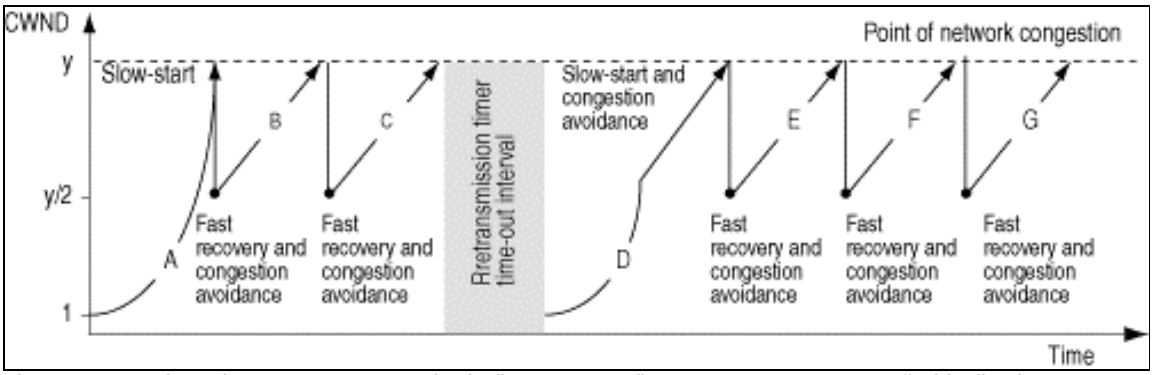


Figura 2.15 – *Throughput* para um exemplo de fluxo TCP. A figura apresenta uma versão idealizada para uma sessão TCP porque o valor de *cwnd* resulta em perda de pacotes constante.Numa rede de produção, *cwnd* é constantemente alterado, resultando numa curva de *throughput* TCP real (ver Figura 2.16).

Agora, assuma que o último segmento da janela foi descartado por um roteador intermediário durante a transmissão em função da sobrecarga de seus *buffers*. Isso faz com que o transmissor TCP aguarde pela expiração do tempo de retransmissão (RTO) antes que ele possa retransmitir o segmento perdido. Como mostrado no *throughput* da curva D, a fonte TCP ajusta o *ssthresh* para a metade do valor de *cwnd* e então reinicia rapidamente com o Congestion Avoidance. Eventualmente a fonte TCP alcança a taxa de transmissão onde ela pode receber ACKs duplicados indicando perda ou segmentos fora de ordem no meio da um fluxo de transmissão.

Analogamente, as curvas E, F e G mostram como a fonte TCP recebe periodicamente ACKs duplicados, indicando perda ou segmentos fora de ordem no meio da uma janela transmissão. O TCP responde pela ativação do mecanismo de Fast Recovery com Congestion Avoidance.

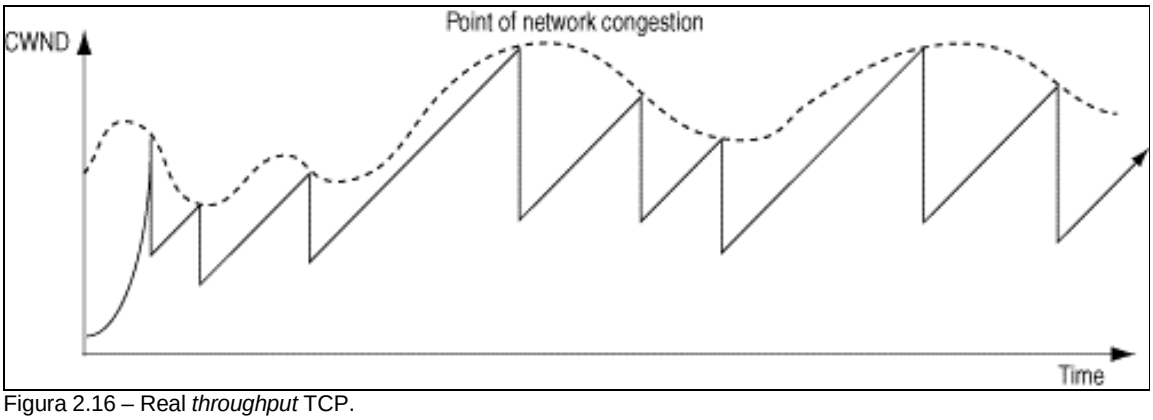


Figura 2.16 – Real *throughput* TCP.

3. QUALIDADE DE SERVIÇO EM REDES TCP/IP

O conceito de qualidade de serviço é muito amplo e complexo, e pode gerar uma série de desdobramentos, pois envolve os diversos elementos que compõe o cenário da computação distribuída.

A recomendação E.800, do ITU-T[7], define "qualidade de serviço" (*Quality of Service* - *QoS*), como sendo "o efeito coletivo da performance de um serviço na determinação do grau de satisfação do usuário deste serviço".

Para [30], "QoS é uma especificação qualitativa e quantitativa dos requisitos de uma aplicação que um sistema multimídia deveria satisfazer, a fim de obter a qualidade desejada".

"Qualidade de serviço (QoS) é um requisito da(s) aplicação(ões) para a qual exige-se que determinados parâmetros (atrasos, vazão, perdas) estejam dentro de limites bem definidos (valor máximo, valor mínimo)" [4].

Estas definições são subjetivas, focadas na percepção que o usuário tem da qualidade dos serviços oferecidos em um ambiente de computação distribuída. Do ponto de vista da rede, existem um conjunto de características que a sua infra-estrutura deve suportar. Estas características mensuráveis devem ser policiadas e controladas a fim de se garantir uma qualidade mínima dos serviços [10].

3.1 Parâmetros Relevantes para a Transmissão com QoS em Redes

3.1.1 Largura de Banda

Para muitos usuários a largura de banda é o principal parâmetro de QoS. Largura de banda é a capacidade do meio ou da aplicação em que se pode transmitir, podendo ser mensurada tanto em Hz [18] quanto em bps[8]. Na verdade, nem todas as aplicações necessitam de maiores requisitos de QoS. Aplicações de transferência de arquivos como no protocolo FTP, acesso a banco de dados relacionais (com informações meramente alfanuméricas), que ainda é a base das

[7] Até 1990, o ITU-T era conhecido como CCITT.
[8] Bps, bits por segundo.

aplicações informatizadas, requerem um tempo de resposta curto, que do ponto de vista da rede pode ser obtido com acréscimo de largura de banda (pelo implementação de tecnologias de maior velocidade, como Fast Ethernet, Gigabit Ethernet, ATM), pela redução de congestionamentos com a implementação de tecnologia de *Lan Switches*, isolando o trafego, pela segmentação da rede.

A Tabela 3.1 ilustra os requisito de largura de banda de algumas aplicações típicas.

Aplicação	Largura de banda Típica
Aplicações transacionais	1 kbps a 50 kbps
Quadro branco	10 kbps a 100 kbps
Voz	10 kbps a 120 kbps
Aplicações Web	10 a 500 kbps
Transferência de grandes	10 kbps a 1Mbps
Vídeo (streaming)	100 kbps a 1 Mbps
Vídeo MPEG	1 Mbps a 10 Mbps
Aplicações imagens	10 Mbps a 100 Mbps
Aplicação realidade virtual	80 Mbps a 150 Mbps

Tabela 3.1 – Largura de banda típica de aplicações em rede.

Porém, como em aplicações multimídia distribuídas, somente o atendimento destes requisitos pelo sistema não é suficiente para estabelecimento do nível mínimo de QoS; outros parâmetros, nem sempre tão facilmente disponibilizáveis, são exigidos [10].

3.1.2 Latência (Atraso)

Entende-se por latência como o somatório de todos os atrasos impostos por um sistema de comunicações desde o usuário final (ou aplicação) até a fonte de informação. Normalmente designado atraso “fim-a-fim”. Esta latência implica em alto tempo de resposta, que em aplicações em tempo real, podem impor fortes restrições a manutenção de uma QoS mínima.

A latência depende dos atrasos impostos pelo sistema de telecomunicações (relativos à tecnologia de transmissão e meio físico adotado) e dos atrasos intrínsecos dos equipamentos de conexão de rede. Assim, os principais fatores que influenciam a latência em uma rede são:

- de equipamentos: o processamento “fim-a-fim” é estabelecido, em geral, através de uma grande cadeia de equipamentos de interconexão de redes (roteadores, *switches*, *firewalls*), que possuem atrasos intrínsecos, relativos ao tempo de processamento interno. Atualmente, é uma preocupação constante dos fabricantes produzir equipamentos com a menor latência possível, para atender os exigentes requisitos impostos por aplicações multimídia distribuídas.
- Atraso de propagação: normalmente dependente da tecnologia empregada para transmissão da informação (sinal elétrico, sinal óptico, sinal de rádio freqüência) e da distância entre os nós de transmissão.
- Velocidade de transmissão: dependente em geral da tecnologia de redes adotada (Ethernet, Fast Ethernet, Gigabit Ethernet, ATM). Normalmente nas LANs (*Local Area Networks*) tende a atingir valores mais elevados. Nas WANs (*Wide Area Networks*) tende a ser um fator restritivo na manutenção da QoS, pela baixa velocidade. Atualmente a evolução e conseqüente redução de custos das redes ATM (que atendem as porções WAN e LAN das redes), tendem a possibilitar implementações de aplicações em redes de longas distâncias com atendimentos aos requisitos de velocidade [10].

3.1.3 Jitter (Variação de Atraso)

O *jitter* é um fenômeno que ocorre quando voz ou vídeo é transmitido em uma rede, cujos pacotes não chegam ao seu destino em ordem consecutiva ou em uma base temporal, ou seja, é uma variação da latência [15].

Em aplicações como Voz sobre IP, por exemplo, a variação do atraso provoca uma distorção da informação, o que prejudica a sua inteligibilidade.

Algumas soluções podem ser implementadas para controle da variação do atraso nestas aplicações em tempo real, como o mecanismo de “*buffering*”, onde uma parcela da informação é armazenada em um dispositivo temporário de armazenamento, antes de ser reproduzida [10].

3.1.4 Skew

O *skew* é a falta de sincronismo entre dados que apresentam uma relação temporal. Podemos dar como exemplo de *ske*w, a falta de sincronia entre o áudio e o vídeo em uma sessão de videoconferência [10].

3.1.5 Taxa de Perdas de Pacotes

Determinadas aplicações (como Voz sobre IP – VoIP), videoconferência, não suportam perdas de informação (pois não é possível admitir-se a retransmissão de uma informação “ao vivo”). A redução das perdas, pela conseqüente redução da taxa de erros na transmissão é um parâmetro que deve ser bem estabelecido na definição de um SLA [10].

3.1.6 Disponibilidade

A adoção do paradigma de aplicações distribuídas em ambiente corporativo tem tornado este requisito como peça importante na manutenção de QoS. As exigências de sistemas tolerantes à falha, com redundância, são exigidas para que uma mínima interrupção dos serviços ocorra [10].

3.2 O MPLS

3.2.1 Motivação para o MPLS

Como em todas as tecnologias emergentes, o esforço de pesquisa e desenvolvimento dos muitos fabricantes levou à criação de um grupo de trabalho para propor uma arquitetura de suporte a LBS[9], em plataforma aberta e interoperável. O IETF [19] tomou a iniciativa de criar um grupo de trabalho, numa

[9] A comutação baseada em etiquetas (*Label Based Switching* - LBS) é uma nova arquitetura de encaminhamento rápido e de alta prioridade de fluxo de dados, dentro de um paradigma de circuito virtual comutado, de forma mais eficiente que o roteamento convencional, onde são analisados os cabeçalhos pacote a pacote. A tecnologia LBS possibilita a utilização do que há de melhor das redes baseadas em pacotes (como as redes IP) e das redes orientadas a conexão (como as redes ATM). Dentro da sua arquitetura é possível a implementação de garantia de níveis de serviços em redes corporativas e de longa distância [10].

plataforma LBS, denominada *Multiprotocol Label Switching* – MPLS. A maior parte dos documentos gerados por este grupo de trabalho ainda se encontra em fase de "trabalho em progresso" sob forma de *Internet Draft* do IETF.

Esta plataforma se propõe a atingir os seguintes objetivos gerais [4]:

- Ser escalável em preço e performance para toda a rede;
- Ser independente da camada de ligação (ATM, Ethernet, etc.);
- Ser independente da camada de rede. O foco principal dos trabalhos atuais é o IP, mas deve operar com redes IPX, *Appletalk*, *DECnet*, etc..
- Coexistir com redes ATM nativas e esquemas de roteamento convencionais;
- Facilitar o suporte a redes multiserviço (dados, voz e vídeo), capazes de tratar diferenciadamente tráfegos de melhor esforço e sensíveis à QoS

Dentro deste enfoque, pode-se enumerar os principais benefícios da plataforma MPLS. Tais benefícios estão relacionados nos itens subseqüentes [19].

3.2.2 Roteamento Simplificado

Tecnologia baseada em etiquetas habilita o encaminhamento de pacotes a partir da análise de etiquetas de comprimento fixo e reduzido, que são anexadas ao pacote no ingresso da rede MPLS. Assim, os pacotes podem ser agrupados (de acordo com a informação contida nestas etiquetas) em classes de equivalência de encaminhamento (*Forward Equivalence Class* – FEC), num processo denominado roteamento explícito, efetuado pelos nós principais da infra-estrutura MPLS, denominados *Label Switching Routers* (LSR). Com isso, o encaminhamento dos pacotes é simplificado, se comparado com o roteamento convencional, onde os cabeçalhos dos pacotes são analisados pacote a pacote ("*hop by hop*").

3.2.3 Roteamento Explícito Eficiente

O roteamento explícito é uma técnica poderosa, podendo ser aplicada para vários propósitos. Para muitas aplicações, o roteamento implícito, baseado na análise dos datagramas, pacote a pacote, gera uma sobrecarga muitas vezes inaceitável. O MPLS permite que os pacotes sejam classificados, a partir de

etiquetas atribuídas na admissão dos nós MPLS, e encaminhados, dentro de uma mesma classe, num caminho virtual, sem a necessidade de ser analisado nó a nó.

3.2.4 Suporte a Engenharia de Tráfego

Entende-se por engenharia de tráfego o processo de seleção do caminho pelo qual o trafego de dados deve ser encaminhado, a fim de propiciar o balanceamento de carga entre vários *links* que interligam roteadores e *switches* na rede. A engenharia de tráfego é muito útil em redes complexas (como a Internet) onde existem diversos caminhos alternativos (paralelos) disponíveis no ambiente de rede.

Nas redes baseadas em datagramas, como as redes IP, a engenharia de tráfego é dificultada devido ao caráter de não orientação à conexão. Dentro deste enfoque, a arquitetura MPLS possibilita que fluxos de dados oriundos de um particular nó de entrada direcionado para um particular nó de saída possam ser identificados individualmente, como num circuito virtual. Além disso, dentro da estrutura de etiquetas, com o roteamento explícito, pode ser habilitado o encaminhamento destes fluxos a partir de um caminho preferencial (na arquitetura MPLS é denominado *label switching path* – LSP).

3.2.5 Suporte a QoS

A capacidade do MPLS em estabelecer uma orientação à conexão (no estabelecimento dos LSP) entre os nós de ingresso e saída preenche uma das principais lacunas das redes IP para implementação de serviços Integrados. Com o MPLS, o estabelecimento dos LSP e a incorporação do protocolo RSVP ao plano de controle do roteamento MPLS, facilita a incorporação do IntServ (ver seção 3.3.1) nos domínios MPLS.

Com o advento das FECs, e a possibilidade de associar estas classes de equivalência a classes de serviços, também alavanca a utilização desta arquitetura como suporte aos Serviços Diferenciados (DiffServ, seção 3.3.2) do IETF [19].

3.2.6 Requisitos Funcionais

Dentro do esforço de padronização do MPLS, o IETF estabelece os seguintes requisitos funcionais [19]:

Os mecanismos de encaminhamento do MPLS devem ser simples a fim de propiciar um encaminhamento rápido de pacotes e com baixo custo, bem como aprimorar a performance se confrontado com o esquema de roteamento convencional;

- O MPLS deve ser independente da tecnologia da camada de enlace do modelo OSI. Otimizações para uma dada mídia em particular, entretanto, podem ser consideradas;
- MPLS deve ser compatível com uma grande faixa de protocolos da camada de rede do modelo OSI (apesar do foco principal do grupo de trabalho estar centrado no protocolo IP);
- O MPLS, em conjunção com os protocolos de roteamento convencional, deve suprir mecanismos que possibilitem prevenção ou detecção de *loops* que possam vir a consumir recursos da rede;
- O MPLS deve permitir a identificação e seleção individual dos caminhos por onde trafegarão os fluxos de dados entre os extremos de ingresso e saída de um domínio;
- O MPLS deve manter compatibilidade retroativa com ferramentas de monitoramento de rede (como *traceroute*) pré-existentes;
- A arquitetura MPLS deve suportar tráfego "*unicast*" e "*multicast*";
- MPLS deve ser uma solução escalável;
- MPLS deve ser implementado como uma solução de *software*, sem exigir um *hardware* específico (simplesmente atualizando o *software* dos *switches* existentes a arquitetura deverá ser suportada). Otimizações específicas de uma dada implementação de *hardware* podem ser utilizadas.

O atendimento destes requisitos funcionais exige implementação de uma arquitetura MPLS, com elementos e protocolos constituintes, que serão discutidos na seção a seguir.

3.3 Garantias de QoS em Redes Baseadas no Protocolo IP

Para muitas pessoas, a qualidade de serviço é um dos principais fatores para alavancamento do MPLS. Entretanto, MPLS em si não é uma arquitetura de provimento de QoS. As suas características de encaminhamento rápido de pacotes baseados em etiquetas, com roteamento explícito, aliada a característica de orientação à conexão, provêem facilidades para implementação das arquiteturas de QoS em IP, tais como Serviços Integrados [45] e Serviços Diferenciados [10] [48].

Atualmente, existem dois modelos para implementar QoS em redes baseadas em IP (Figura 3.1): serviços integrados (IntServ) e serviços diferenciados (DiffServ). O IntServ é um modelo baseado em reserva de recursos, enquanto que, Serviços Diferenciados é uma proposta onde os pacotes são marcados de acordo com classes de serviços pré-determinadas [15].

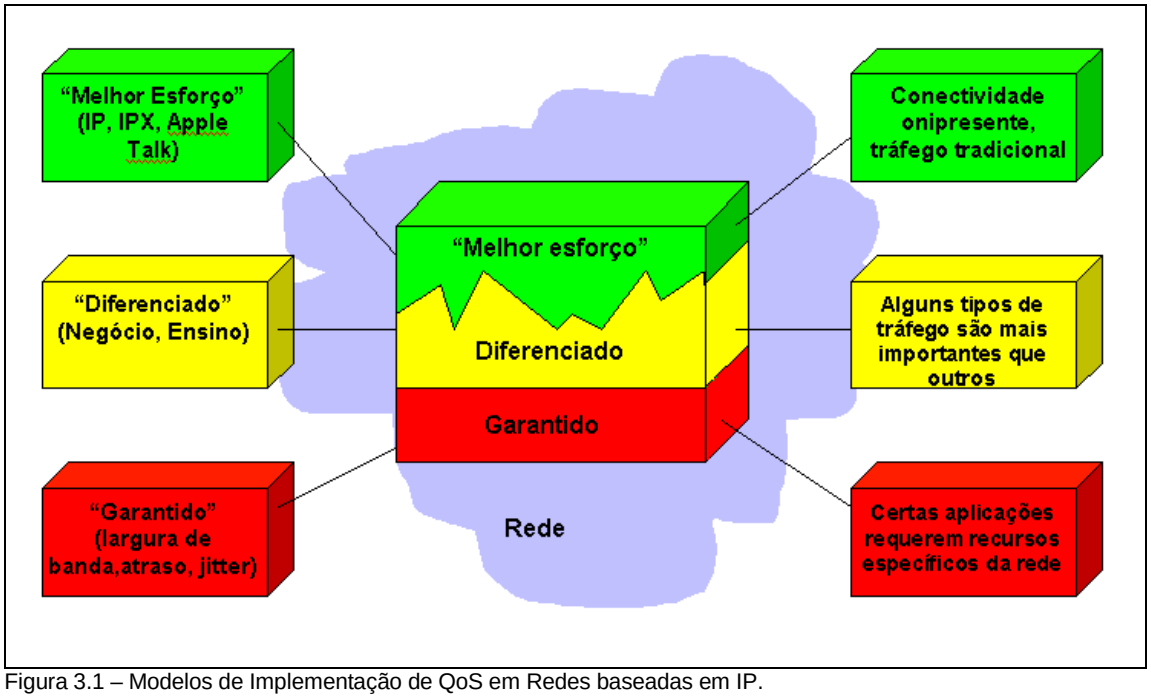


Figura 3.1 – Modelos de Implementação de QoS em Redes baseadas em IP.

Nas próximas seções, será mostrado um pouco desses dois modelos de QoS para redes IP, onde serão descritas características vantagens e desvantagens de cada modelo.

3.3.1 Serviços Integrados (IntServ)

O modelo de serviços integrados é caracterizado pela reserva de recursos. Antes de iniciar uma comunicação, o emissor solicita ao receptor a alocação de recursos necessários para definir uma boa qualidade na transmissão dos dados. O protocolo RSVP *(Resource Reservation Protocol)* [46] é utilizado, nesse modelo, para troca de mensagens. de controle de alocação dos recursos. A Figura 3.2 ilustra uma requisição RSVP.

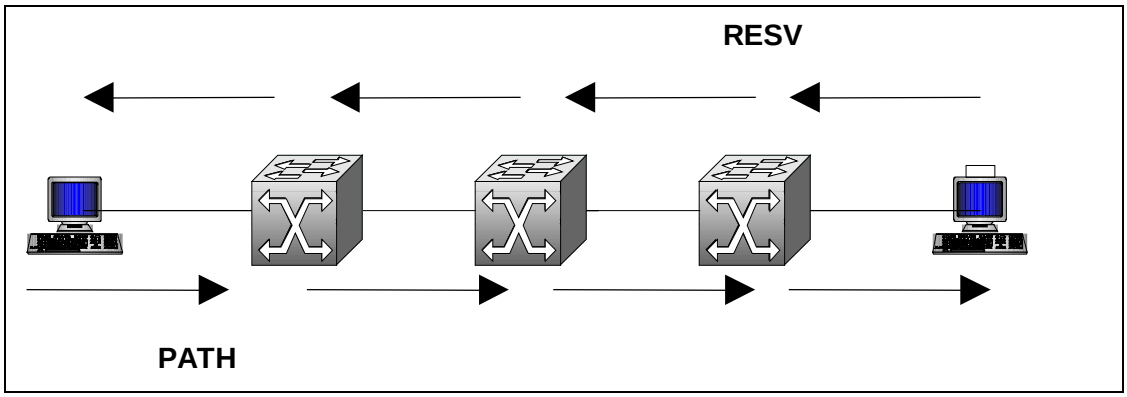


Figura 3.2 – Sinalização RSVP.

A alocação de recursos diz respeito à largura de banda e ao tempo em que será mantida a conexão. Neste período de tempo, o emissor daquele serviço tem uma faixa da largura de banda disponível para transmitir seus dados. O IntServ é caracterizado pela alocação de recursos para dois novos tipos de serviços que são os serviços garantidos para aplicações que necessitam de um atraso constante, e serviços de carga controlada para aplicações que requerem segurança e destacam o serviço de melhor esforço.

Na Figura 3.3 que se segue, suponha, que a máquina *jazz.empresa1.com.br* deseje fazer uma comunicação de voz através da Internet (Voz sobre IP) com a máquina *ciranda.empresa2.com.b*r. Esta aplicação requer baixo atraso e baixa variação do atraso *(jitte*r) para que sejam mantidos os requisitos de qualidade. Então, *jazz* envia uma mensagem especificando as características para o tráfego *(path*) para *ciranda*. Quando a mensagem de *path* chega a *ciranda*, inicia-se o procedimento de reserva de recursos *(resv*) por todo caminho entre este dois nós da rede. Todos os roteadores entre os dois pontos passam pelo processo de alocação de recursos e qualquer um deles pode rejeitar a solicitação, informando

para *jazz* que a solicitação não foi aceita. Caso todos os roteadores tenham condições de disponibilizar os recursos solicitados, é alocada a largura de banda e *buffer 4* necessários para a aplicação.

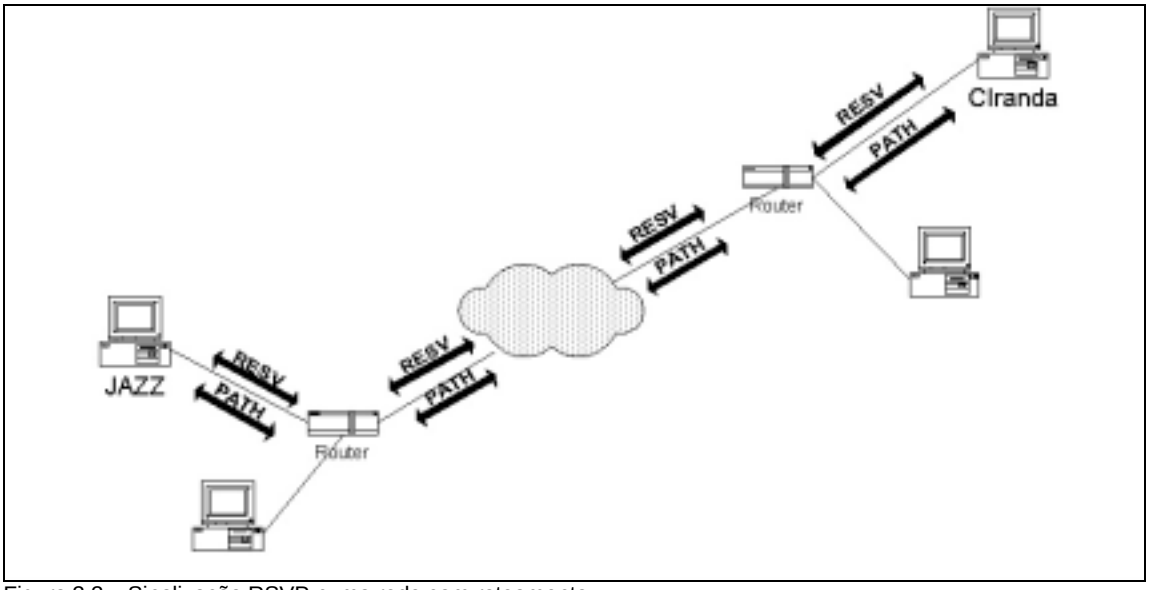


Figura 3.3 – Sinalização RSVP numa rede com roteamento.

Durante a transmissão dos pacotes, são feitas classificações nos roteadores para cada fluxo, colocando-os em filas específicas para a aplicação. Como o controle é feito basicamente nos roteadores, isso exige grande capacidade de processamento, armazenamento e bons algoritmos para tratamento de filas. Ou seja, há um aumento do grau de complexidade nos roteadores.

Resumidamente, pode-se dizer que o modelo de serviços integrados:

- Cria caminhos entre origem e destino, reservando banda nos nós intermediários;
- Todos os nós no caminho devem suportar esse tipo de reserva (serviços integrados);
- Necessita de *refreshes* periódicos;
- Utiliza outros protocolos para roteamento e transmissão (problema no IP atual - mudança de rotas) [15].

3.3.2 Serviços Diferenciados (DiffServ)

O modelo de serviços diferenciados implementa QoS com base na definição de tipos de serviços. No cabeçalho de um pacote IP, existe um campo chamado TOS *(Type of Servic*e) que representa o tipo do serviço. No entanto, serviços diferenciados ampliam a representação de serviços e o tratamento que pode ser dado para encaminhar um pacote, definindo um novo *layout* para o TOS, passando a chamá-lo de Campo DS *(Differentiated Service Field*). No Campo DS, são codificadas as classes para serviços diferenciados. O campo TOS já existia na definição do pacote IP, mas só recentemente se definiu uma utilização para o mesmo.

A arquitetura DiffServ parte do princípio que domínios adjacentes tenham um acordo sobre os serviços que serão disponibilizados entre os mesmos. Este acordo denomina-se *Service Level Agreement* (SLA). Um SLA determina as classes de serviços suportadas e a quantidade de tráfego na banda entre os domínios. Os domínios podem definir um SLA estático ou dinâmico, sendo que, neste último caso, um protocolo de sinalização e controle será necessário para o gerenciamento da banda.

Dois novos tipos de serviços especiais surgiram juntamente com o modelo de serviços diferenciados: serviços assegurados e os serviços especiais. Serviços assegurados são os serviços para clientes que precisam de segurança para seus provedores serviços no momento em que houver um congestionamento. E os serviços especiais são para aplicações que necessitam de baixo atraso e baixo *jitte*r.

Com DiffServ, os próprios clientes podem marcar seus Campos DS e enviar para o receptor. No entanto, não há como saber se há recursos disponíveis para a comunicação. Em geral, o pacote chega em um roteador que não provê QoS com o Campo DS marcado, é remarcado e passa a ser um pacote de um serviço de melhor esforço.

Existem tipos de serviços que não podem conviver com essa incerteza. Por isso, um componente mediador foi inserido nesse modelo para gerenciar recursos no domínio QoS. Este componente foi denominado *Bandwidth Broker* (BB). O BB trabalha como um gerente de recursos do domínio que tem com função básica controlar a largura de banda, as políticas e prioridades dentro e entre as organizações. Quando há uma solicitação de um fluxo, o BB é o componente que

verifica a disponibilidade de recursos e a autorização do cliente para conexão dentro do domínio QoS. Ele também se encarrega de fazer as alocações necessárias para a comunicação dentro do seu domínio e solicita ao BB do domínio adjacente, caso o pedido de conexão seja para fora do domínio, para fazer o mesmo. O processo de solicitação de alocação de recursos é continuado entre BBs adjacentes até que se chegue ao BB do domínio do receptor. O protocolo de sinalização para alocação de recursos entre os BBs pode ser o RSVP.

O modelo de Serviços diferenciados tem sido o mais utilizado para implementação de QoS. Ele exige menos dos roteadores, necessitando pouca atualização de *software* para prover bons métodos de classificação, policiamento, montagem e remarcação de pacotes.

Resumidamente, pode-se dizer que o modelo de serviços diferenciados:

- Utiliza o campo DS *(Differentiated Service*s) para determinar a prioridade do pacote;
- Altera o campo TOS *(Type of Servic*e) do IPv4 [50] ou o campo “classe de tráfego” do IPv6 [47].

Fazer QoS não é uma técnica extremamente necessária para *backbones* que não têm problemas de congestionamento, pois, desta forma, um pacote dificilmente seria descartado ou atrasado. Assim, só se faz necessário a implementação de QoS em situações em que importantes aplicações estarão disputando o enlace com todos os outros serviços, mas que não podem suportar a falta de disponibilidade de largura de banda e a grande variação de latência que ocorrem atualmente na Internet.

Conforme já discutido, existem outras técnicas para oferecer QoS como MPLS *(Multiprotocol Label Swirchin*g), engenharia de tráfego, roteamento QoS, assim como o próprio RSVP. A Internet é uma rede em constante evolução, e QoS surgiu com o intuito de suprir as novas necessidades de comunicação da mesma e resolver alguns problemas que impedem o desenvolvimento de algumas aplicações.

O QoS permitirá novas formas de negociação na Internet, liberando, por outro lado, o desenvolvimento de aplicações que antes necessitaram de maior qualidade e/ou segurança na transmissão de seus dados. No entanto, QoS investe um pouco na forma de uso da Internet, pois, com essa nova filosofia, terá mais direito à banda quem quiser e puder pagar mais. O prejuízo que QoS traz para

aplicações de melhor esforço ainda é uma realidade, mas já há discussões sobre o assunto, porém nada de concreto sobre como não degradar demais o que a Internet oferece hoje. No entanto, QoS avança na intenção pura e simples de evolução para a Internet e a diminuição do problema da escassez de recursos para serviços que realmente precisam de uma maior qualidade e segurança.

4. COMUNICAÇÃO VIA SATÉLITE [2]

4.1 Introdução

Os sistemas de satélites vêm desempenhando, nos últimos anos, um papel importante como meio alternativo para o transporte de sinais de telecomunicações. Isto se deve ao fato do satélite poder cobrir grandes áreas, ou mesmo chegar onde os outros meios, tais como cabos de fibra óptica e enlaces microondas, não podem chegar ou chegam com um custo elevado.

4.2 Classificação dos Sistemas

Estes sistemas podem ser classificados conforme o tipo de sua órbita, em sistemas geoestacionários e não-geoestacionários. Os satélites geoestacionários estão localizados a 36.000 km da superfície da terra, na linha do equador, e sua movimentação orbital é fixa (geoestacionária) em relação a um ponto localizado na superfície terrestre. Os satélites não-geoestacionários possuem órbitas mais baixas, e por isso, necessitam movimentar-se rapidamente para se manterem a uma certa distância da superfície terrestre. Estes sistemas não estão localizados na linha do equador, e geralmente, são constituídos por uma constelação de satélites[10]. Os sistemas LEO (baixa órbita terrestre) possuem suas órbitas localizadas de 700 a 2000 km de altura. Os sistemas de órbitas médias, MEO (média órbita terrestre), possuem suas órbitas posicionadas a 10.000 km de altura. A Figura 4.1 ilustra as três classificações dos satélites, conforme a sua distância da superfície terrestre.

Cada satélite possui a bordo um certo número de *transponders*[11] que recebem o sinal numa faixa de freqüências (enlace de subida ou Terra-para-espaço), amplifica e o transmite em outra faixa de freqüências (enlace de decida ou espaço-para-Terra). Tipicamente, um satélite contém 24 *transponders*, sendo que cada *transponder* possui uma banda de 36 MHz. Apesar de existirem outras bandas

[10] Uma constelação de satélite é composta por no mínimo 3 satélites.
[11] *Transponder*, receptor para transmissor.

de freqüências as mais usadas são a banda C (6/4 GHz) e banda Ku (14/12 GHz). A Figura 4.2 ilustra um enlace típico de satélite.

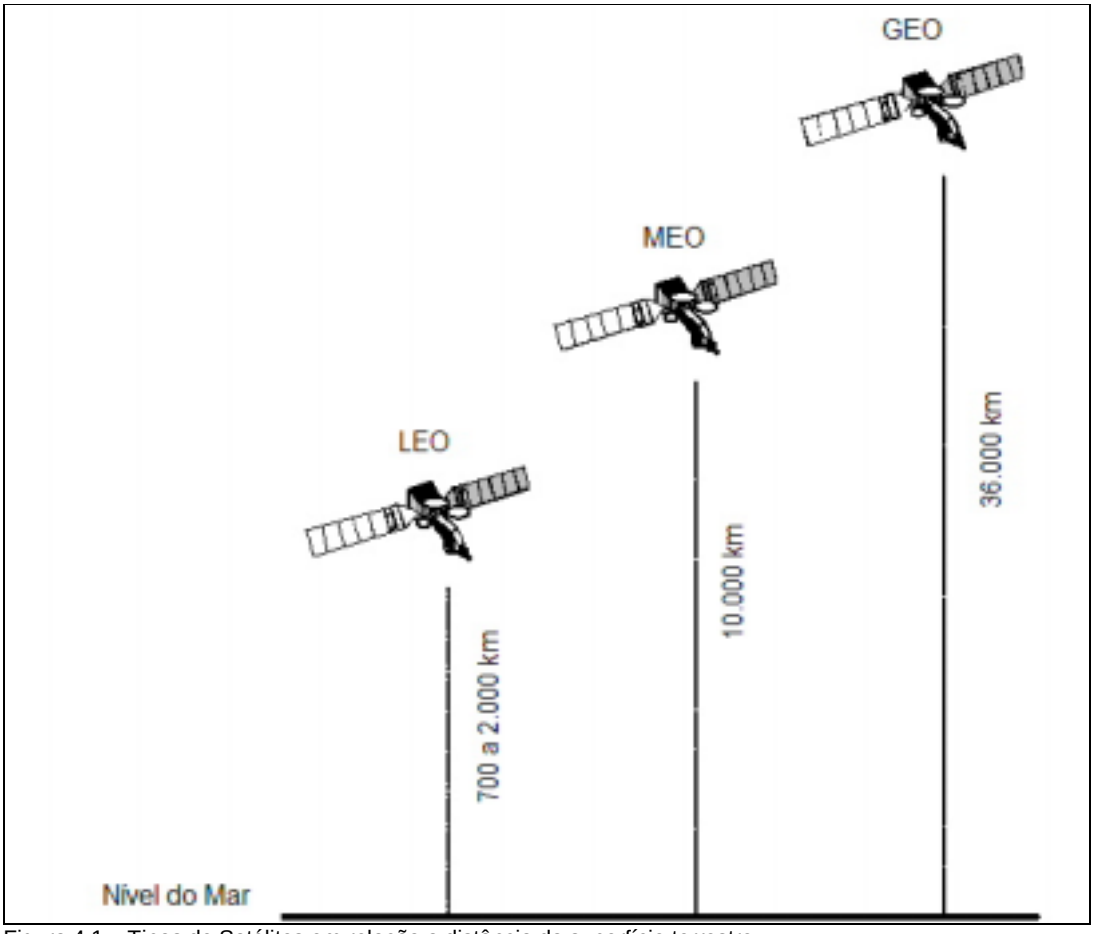


Figura 4.1 – Tipos de Satélites em relação a distância da superfície terrestre.

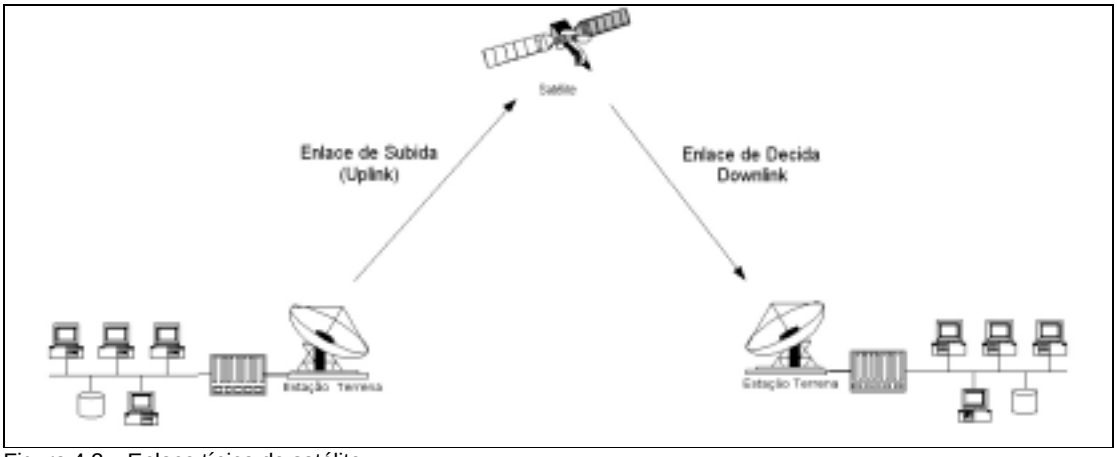


Figura 4.2 – Enlace típico de satélite.

4.3 Parâmetros Físicos da Comunicação Via Satélite

Comecemos esta seção pela ilustração da Figura 4.3, mostrando a largura de banda de um satélite com seus 24 *transponders*, 12 na polarização horizontal e outros 12 na vertical. A figura mostra implicitamente o reuso da freqüência. Cada canal tem uma largura de banda de 36 MHz e 4 MHz de banda de guarda, isto é, a banda mínima que garante a não interferência co-canal.

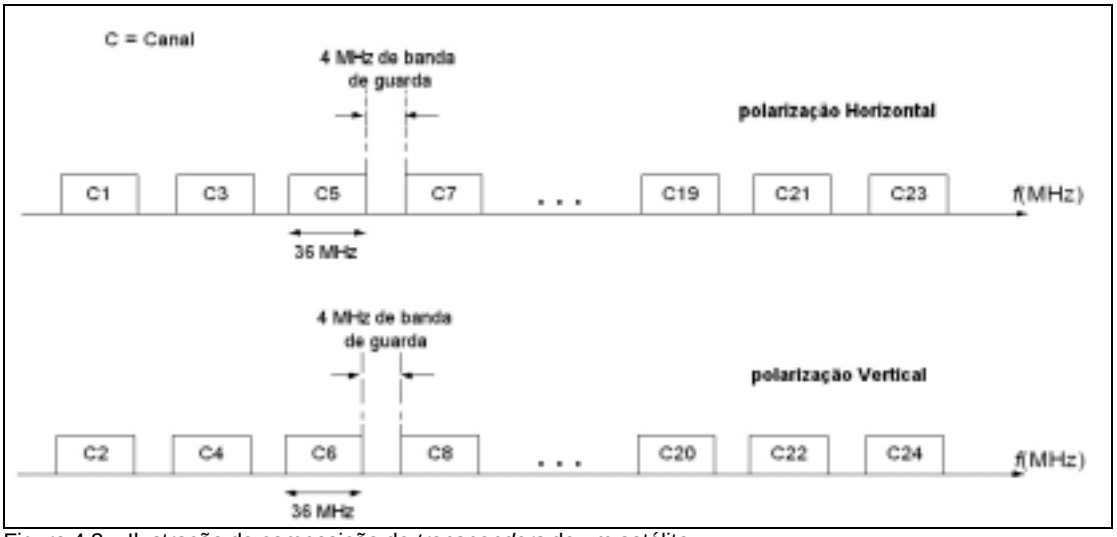
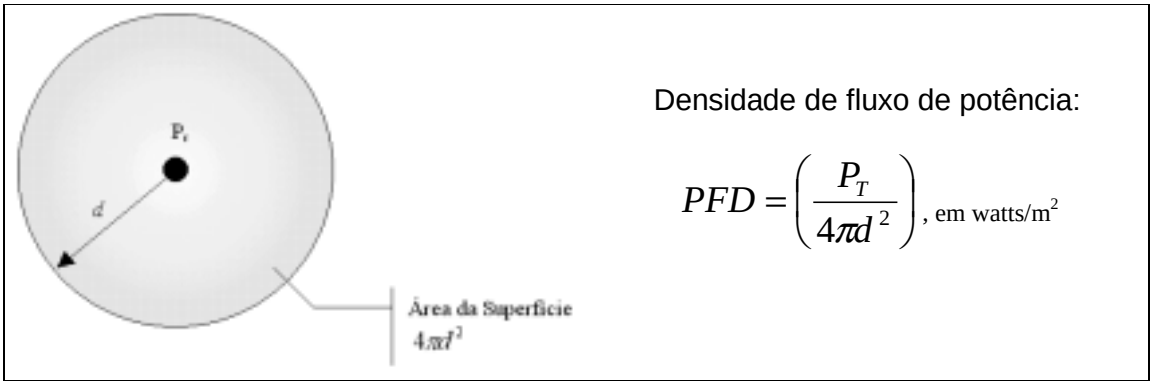
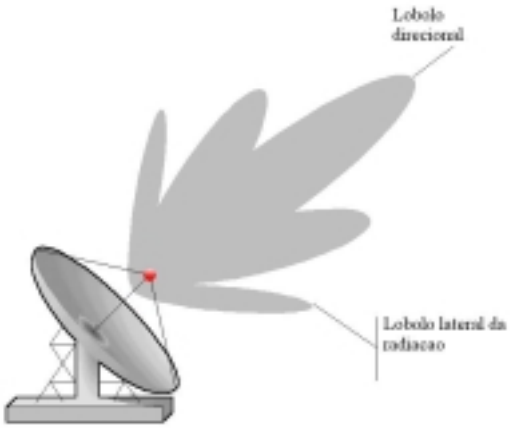


Figura 4.3 – Ilustração da composição de *transponders* de um satélite.

Considerando um transmissor isotrópico irradiando no espaço livre com uma potência total P_T watts em todas as direções uniformemente, tem-se que a densidade do fluxo de potência para esse caso é dada pela expressão abaixo.





G_T = Ganho de transmissão da antena

Com isso, $PFD = \left(\frac{P_T G_T}{4\pi d^2} \right)$ no lóbulo direcional.

Define a Potência isotrópica irradiada efetiva:

$$EIRP = P_T G_T$$

$$\therefore PFD = \left(\frac{EIRP}{4\pi d^2} \right)$$

Para uma antena receptora ideal com uma área física A (m^2), a potência recebida pode ser dada por $P_R = (PFD)A$. Na prática, alguma energia é perdida. Façamos η representado a eficiência da antena, cujos valores típicos estão entre 0,5 e 0,9.

$$\therefore P_R = \left(\frac{EIRP}{4\pi d^2} \right) A\eta$$

O ganho da antena recebida pode ser mostrado como dado em $G_R = \frac{4\pi}{\lambda^2}(A\eta)$, onde λ é o comprimento de onda da onda eletromagnética.

$$\therefore (A\eta) = \frac{G_R \lambda^2}{4\pi}$$

Então, $P_R = \left(\frac{EIRP}{4\pi d^2} \right) \left(\frac{G_R \lambda^2}{4\pi} \right)$.

Dessa forma, chegamos à expressão da perda no espaço livre, conforme é mostrado abaixo.

$$L_S = \left(\frac{4\pi d}{\lambda} \right)^2$$

Assim, $P_R = \left(\frac{EIRP}{L_S} \right) G_R$

Expressando esta equação em decibéis, temos que

$$10\log_{10}(P_R) = 10\log_{10}\left(\frac{EIRP}{L_S} \right) + 10\log_{10}(G_R)$$

$$P_R(dBW) = EIRP(dBW) + G_R(dB) - L_S(dB)$$

Levando-se em consideração outras perdas como atenuação atmosférica, desvanecimento pelas chuvas, perda das pontas, etc., como $L_0(dB)$, representando todas as outras perdas.

$$P_R(dBW) = EIRP(dBW) + G_R(dB) - L_S(dB) - L_0(dB)$$

Ou equivalentemente,

$$P_R = \left(\frac{EIRP}{L_S L_0} \right) G_R$$

4.4 Conclusão

Considerando as distâncias envolvidas e somando-se os percursos do enlace de subida com o de descida, podemos concluir que o sinal que está sendo transportado está sujeito a grandes retardos, o que pode ser prejudicial para determinadas aplicações em tempo real, como, por exemplo, aplicações de voz.

Dessa forma, fica claro que um enlace via satélite apresenta algumas características que podem comprometer o bom desempenho de protocolos de transporte, como é o caso do TCP, se não for levado em consideração a distância do satélite à Terra, visto que é essa distância que irá determinar o atraso físico do enlace e, conseqüentemente, o nível de utilização da capacidade aproveitado.

5. PROBLEMAS DO TCP EM SATÉLITES

Como já dito na introdução deste trabalho, o TCP trabalha muito bem em ambientes terrestres, porém não consegue utilizar toda a largura de banda disponível do canal nas suas transmissões, quando se trata de *links* de longo atraso. Tendo isso em mente, e já tendo observado, no capítulo 4, os parâmetros físicos que compõem um ambiente de satélite e suas principais relevâncias, foi discriminado este breve capítulo a fim de explicar o porquê disso ocorrer.

Os problemas de performance do TCP em satélites geoestacionários têm sido uma preocupação recente, sendo amplamente discutidos com o aumento do uso de satélites como meio alternativo para transporte de informação em telecomunicações. Podem ser divididos em duas grandes categorias: problemas que tratam do aumento do tamanho da janela deslizante e problemas no comportamento da estabilidade (*steady-state*).

5.1 Problemas que Tratam do Aumento da Janela Deslizante

Como discutido na seção 2.2.3, o TCP usa dois algoritmos para incrementar o tamanho da janela deslizante. As duas seções seguintes discutirão os problemas do Slow Start e Congestion Avoidance no ambiente dos satélites.

5.1.1 Slow Start em Canais de Satélite

Kruse [16] mediu o RTT no satélite ACTS [53] da NASA com valor aproximado de 560 ms [14]. Por comparação, o RTT na Internet terrestre foi medido entre a Universidade de Ohio e a Universidade da Califórnia, em Berkley, com valor de aproximadamente 80 ms. A equação 5.1 fornece o tempo que o algoritmo Slow Start leva para alcançar um tamanho de janela de *W* segmentos numa rede com um RTT de *R* como definido por [65].

$$tempo \quad de \quad slow \quad start = R \log_2 W \tag{5.1}$$

Assumindo segmentos de 512 *bytes*, um tamanho de janela de 128 segmentos (máxima janela do TCP, 128 segmentos x 512 *bytes* = 65.535 *bytes*) e os RTTs dados acima, o TCP operando em redes de satélite leva 3,92 segundos para incrementar o *cwnd* até a completa janela de anúncio (*Advertised Window*). O TCP rodando em rede terrestre leva 560 ms para alcançar o mesmo tamanho de janela. Ao se usar o algoritmo Slow Start, o TCP pode desperdiçar largura de banda disponível. A Figura 5.1 mostra um modelo matemático da quantidade máxima de dados enviados sobre ambas as redes, terrestre e de satélite, em função do tempo. Quando o protocolo o TCP é usado para mostrar a janela de aviso (128 segmentos) na rede de satélite, o TCP está apto a mandar aproximadamente 22 vezes mais dados sobre a rede terrestre.

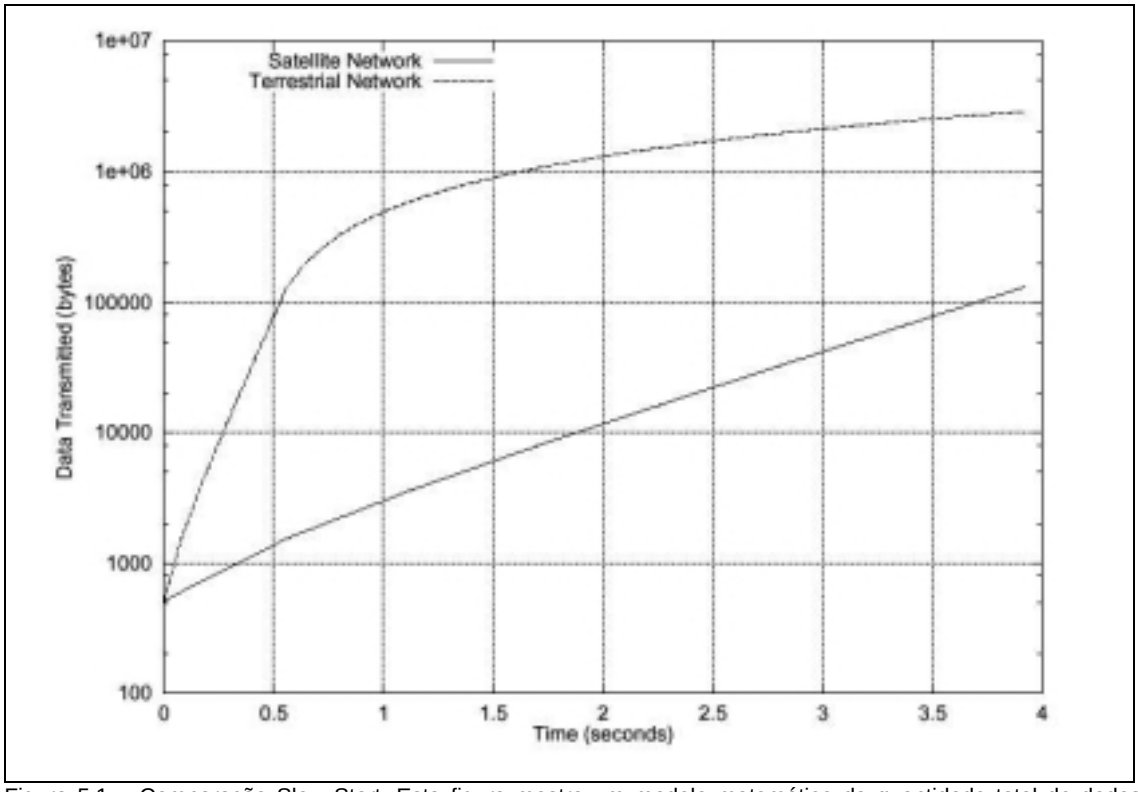


Figura 5.1 – Comparação Slow Start. Esta figura mostra um modelo matemático da quantidade total de dados enviados pelo TCP sobre uma rede de satélite e uma rede terrestre em função do tempo. Ambas as transmissões iniciam com um *cwnd* de 1 segmento e incrementam o tamanho de *cwnd* usando o algoritmo Slow Start. A quantidade de tempo mostrada representa o tempo que o TCP leva sobre a rede de satélite para alcançar a janela de anúncio (128 segmentos, neste caso) usando o Slow Start.

5.1.2 Congestion Avoidance sobre Canais de Satélite

O TCP usa o Congestion Avoidance para sondar se a rede está preparada para uma capacidade adicional após a perda de segmentos. Os canais de satélite inserem um atraso no tempo que o Congestion Avoidance leva para incrementar *cwnd* quando comparado com *link*s terrestres. Por exemplo, quando uma conexão TCP utilizando a máxima janela possível (128 segmentos, contendo 512 *bytes* cada) perde um único segmento, o valor de *cwnd* é reduzido para 64 segmentos. Para cada RTT transmitido com sucesso, o valor de *cwnd* é acrescido de aproximadamente 1 segmento. Sobre a rede terrestre descrita acima, leva-se 5,12 segundos para incrementar o valor de *cwnd* de 64 segmentos para 128 segmentos usando Congestion Avoidance. Sobre o canal do satélite ACTS o mesmo incremento leva 35,84 segundos. Cada RTT que o TCP não consegue usar completamente a largura de banda disponível representa perda de *throughput*. A Figura 5.2 apresenta um modelo matemático ilustrando a quantidade total de dados que o algoritmo Congestion Avoindance é capaz de enviar sobre ambas as redes, a terrestre e a de satélites, em função do tempo. A figura mostra que quando o TCP está operando sobre a rede terrestre o incremento para a janela de aviso é mais rápido, e então o TCP é capaz de enviar aproximadamente 9 vezes mais dados sobre a rede terrestre na mesma quantidade de tempo quando comparado à rede de satélites.

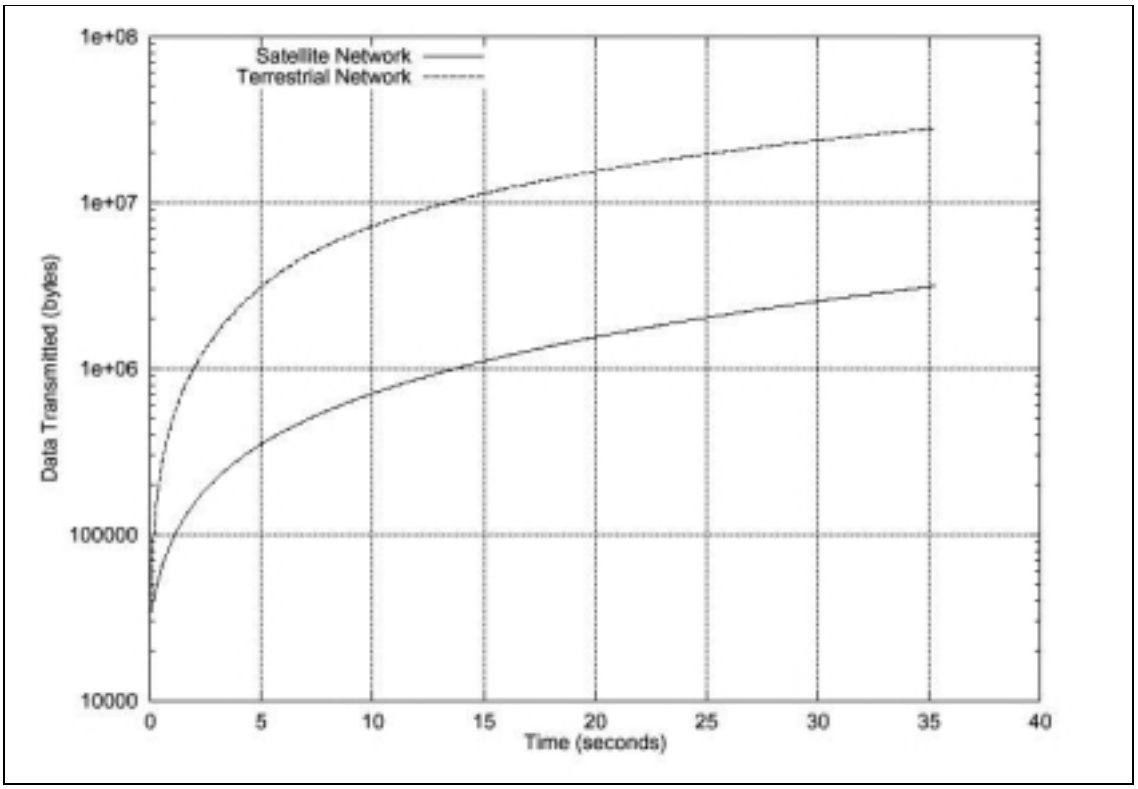


Figura 5.2 – Comparação do Congestion Avoidance. Esta figura mostra um modelo matemático da quantidade total de dados enviados pelo TCP sobre a rede de satélites e a rede terrestre em função do tempo. Ambas as transmissões iniciam com um *cwnd* de 64 segmentos e incrementam o tamanho de *cwnd* usando o Congestion Avoidance até *cwnd* alcançar a janela de anúncio (128 segmentos). A quantidade de tempo mostrada representa o tempo que o TCP leva sobre a rede de satélite para alcançar a janela de anúncio usando o Congestion Avoidance.

5.2 Problemas com o Comportamento da Estabilidade do TCP

O canal de satélite estudado neste documento tem 1,536 Mbits/s (ou 192.000 *bytes*/s) de capacidade (velocidade T1) [14]. Um limite superior para o *throughput* TCP é dado na equação 5.2. Esta equação assume uma rede sem perdas e um TCP que não usa qualquer algoritmo de controle de congestionamento descrito na seção 2.2.3.

$$\max \quad throughput = \frac{tamanho \quad da \quad janela \quad receptora}{round \quad trip \quad time} \tag{5.2}$$

Portanto, quando se usa a janela máxima de recepção TCP, 65.535 *bytes*, sobre um canal de satélite (RTT de 560 ms), o limite superior do *throughput* é dado pela equação 5.3.

$$\max \quad throughput = \frac{65535 bytes}{560 ms} \approx 117.027 \, bytes/segundo \tag{5.3}$$

Claramente, esse limite superior no *throughput* TCP garante que o TCP será incapaz de utilizar completamente a largura de banda disponível por canais de satélite T1.

5.3 Soluções Possíveis

Os problemas esboçados acima podem ser solucionados fazendo uso de algumas alterações. Uma mudança que tem sido estudada é romper a conexão TCP no roteador (*gateway*, terminal terrestre do satélite) antes do canal de *uplink*. Usando esse mecanismo, o roteador antes do canal de satélite aceita e reconhece todos os segmentos enviados para uma máquina através do *link* de satélite. Do ponto de vista do transmissor, a transferência torna-se completa quando o roteador reconhece todos os segmentos. O roteador então envia os dados para o *host* através do canal de satélite. Isso faz a transferência parecer mais rápida do ponto de vista do transmissor. Contudo, os ACKs recebidos pelo transmissor não indicam que o dado foi recebido pelo destino final e, portanto, esse mecanismo acaba com a semântica fim-a-fim do TCP.

A Figura 5.3 ilustra esse mecanismo, onde o roteador, terminal de satélite, reconhece os segmentos antes do pacote TCP chegar ao destino final. Note que no momento 2, o roteador terminal de satélite envia o ACK ao transmissor e ao mesmo tempo repassa o pacote TCP entrante, fazendo com que a fonte transmissora acredite que os pacotes estão chegando normalmente ao receptor, o que não é verdade.

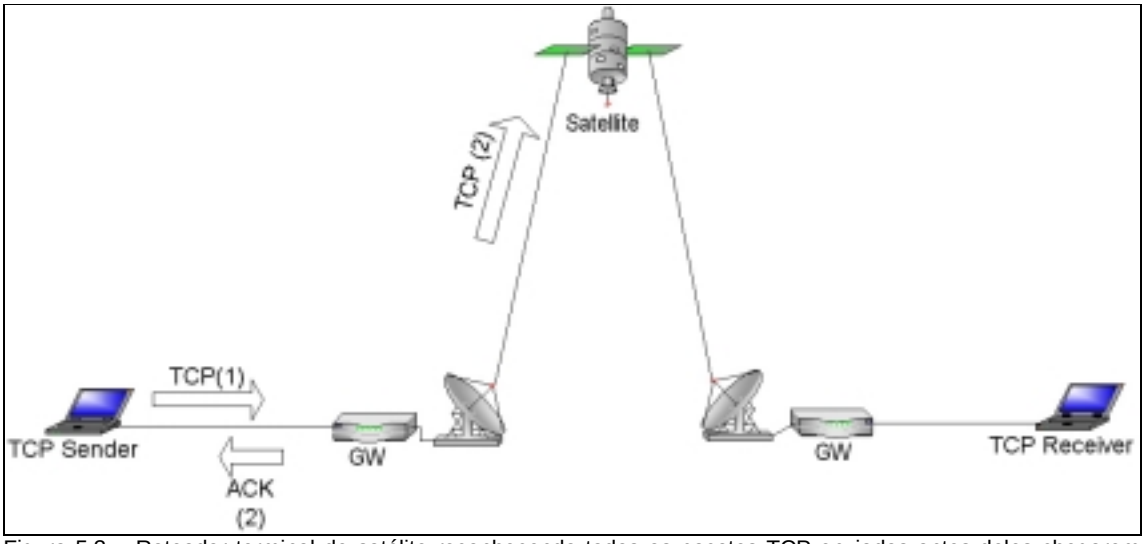


Figura 5.3 – Roteador terminal de satélite reconhecendo todos os pacotes TCP enviados antes deles chegarem efetivamente ao receptor.

Uma idéia similar é fazer o roteador fazer retransmissões no lugar do transmissor [17]. Fazer o roteador reparar todas as perdas sem notificar o transmissor TCP permite retransmitir sem que o transmissor TCP reduza a taxa de transmissão. Pesquisadores também têm investigado compressão de cabeçalhos TCP/IP num esforço de reduzir o *overhead*, deixando mais largura de banda para dados. Finalmente, novos mecanismos de enfileiramento, como o *Random Early Detection*, partem da premissa que roteadores intermediários possam informar ao transmissor TCP que um congestionamento está ocorrendo antes do transmissor TCP enviar uma quantidade de dados imprópria. Por dar essa rápida indicação de congestionamento, o roteador pode forçar o TCP a reduzir a taxa de transmissão antes que o transmissor TCP transmita uma taxa de dados imprópria, forçando o roteador a descartar um grande número de segmentos. Em uso potencial, esses mecanismos estão fora do escopo de nosso estudo [14].

6. RESULTADOS DE REFERÊNCIA

Neste capítulo serão apresentadas soluções, modificações, ambientes de teste e resultados encontrados por [14] em sua tese a fim de servir como parâmetro de referência para os testes que foram por nós realizados e que serão apresentados no capítulo 8. Tais modificações e testes apresentam certo embasamento teórico que ajuda no entendimento do funcionamento das plataformas de testes utilizadas, bem como de seus resultados práticos.

6.1 Uma Solução em Nível de Aplicação

A primeira providência a ser tomada para melhor utilizar a largura de banda disponível em *link*s de satélite foi empregar múltiplas conexões de dados TCP em paralelo. Alterou-se um cliente e servidor FTP para usar tais conexões a fim de transmitir um arquivo simples. Chamou-se MFTP a versão modificada do FTP. As mudanças para o FTP são discutidas em [33]. Esta seção está organizada da seguinte forma: a subseção 6.1.1 esboça a aplicação em nível de aplicação. A subseção 6.1.2 descreve vários ambientes experimentais que foram usados para testar o MFTP. A subseção 6.1.3 apresenta o resultado desses testes. Finalmente, a seção 6.1.4 relata as lições deixadas pelos mecanismos do MFTP que poderiam ser usadas no TCP para ajudá-lo a utilizar melhor a capacidade disponível nos canais de satélite.

6.1.1 Teoria do MFTP

Como descrito na equação 5.3, o TCP é limitado a um *throughput* de aproximadamente 117.027 *bytes*/segundo sobre o canal de satélite usado no experimento. Esse limite vem do máximo tamanho da janela TCP (65.535 *bytes*) e do atraso imposto pela comunicação via satélite (560 ms de RTT). Quando múltiplas conexões TCP são usadas em paralelo, uma aplicação pode utilizar uma maior janela EFETIVA TCP. A equação 6.1 mostra que uma aplicação que usa 2 conexões

TCP em paralelo, cada uma usando a janela máxima de 65.535 *bytes*, consegue utilizar totalmente um *link* T1 dum satélite (192 k*Bytes*/s).

$$\max \quad throughput = \frac{2(65535bytes)}{560ms} \approx 234.054\, bytes/segundo \tag{6.1}$$

O estudo de Fritz e Dolores [14] achou que os cliente e servidor FTP padrão SunOS 4.1 usam uma janela de anúncio de 24 kB[12], ao invés do máximo tamanho de janela de 65.535 *bytes*. A equação 6.2 mostra que essa janela, menor, reduz ainda mais o *throughput* que o FTP é capaz de alcançar em canais de satélite. Até que se diga o contrário, o tamanho da janela em todo o nosso experimento também será de 24 kB, fazendo com que comparações válidas com o padrão SunOs e também com o trabalho de [14] possam ser realizadas.

$$\max \quad throughput = \frac{24576 \quad bytes}{560ms} \approx 43.886\, bytes/segundo \tag{6.2}$$

Re-arranjando a equação 6.2, chega-se à equação 6.3, que define o tamanho da janela necessária para utilizar completamente um canal com uma dada largura de banda e um dado RTT.

$$Tamanho \quad da \quad janela = \left(L\arg ura \quad de \quad banda \right)(RTT) \tag{6.3}$$

Assim, para uma utilização completa da capacidade do circuito de satélite usado em nosso estudo, uma janela TCP de 107.520 *bytes* é necessária de acordo com a equação 6.4.

$$Tamanho \quad da \quad janela = \left(192.000 \quad bytes/segundo \right) \cdot (560ms) = 107.520 \quad bytes \tag{6.4}$$

[12] Neste documento, 1 kB = 1024 *bytes*.

De acordo com a equação 6.5, o MFTP deve usar no mínimo 5 conexões de dados paralelas, cada uma com uma janela de 24 kB, para utilizar totalmente um canal de satélite T1.

$$\left\lceil \frac{janela\ \ efetiva\ \ necessária}{tamanho\ \ de\ \ janela\ \ para\ \ 1\ \ conexão} \right\rceil = \left\lceil \frac{107.520\ \ bytes}{24576\ \ bytes} \right\rceil = 5\ \ conexões \quad (6.5)$$

6.1.2 Ambiente Experimental

O MFTP (e as subseqüentes modificações TCP) foi testado em 3 ambientes. O ambiente de satélite usado para cada pesquisa foi o Sistema de Satélite NASA ACTS, que está na órbita geoestacionária. Foi usado, também, um emulador de *software* construído na Universidade de Ohio em conjunto com essa pesquisa, e também um emulador de *hardware* comercialmente disponível para o modelo do Sistema ACTS. Um fator muito importante a ser ressaltado nos testes de [14] foi que nenhuma das três plataformas de teste foi colocada na disputa de tráfego na rede. Cada um desses ambientes foi discutido no detalhe abaixo, como mostra a Figura 6.1.

6.1.2.1 NASA ACTS

O *layout* da rede da plataforma de teste do NASA ACTS é mostrado na Figura 6.1. Cada rede física contém um roteador Cisco 2514. Os dois roteadores estão conectados *via satélite* ACTS. Nesse ambiente, o cliente e servidor são estações Sun IPX rodando SunOS 4.1.3. Foi verificado que o RTT foi de aproximadamente 560 ms. Os roteadores Cisco usados em nosso experimento empregaram enfileiramento *drop-tail.* Por esse tipo de enfileiramento, quando um roteador recebe um segmento para ser retransmitido, mas o processo de transmissão de outro segmento já está ocorrendo, o segmento entrante é colocado em fila para um posterior processamento. Um roteador que usa enfileiramento *drop-*

tail descarta segmentos entrantes quando o roteador consome toda a memória disponível.

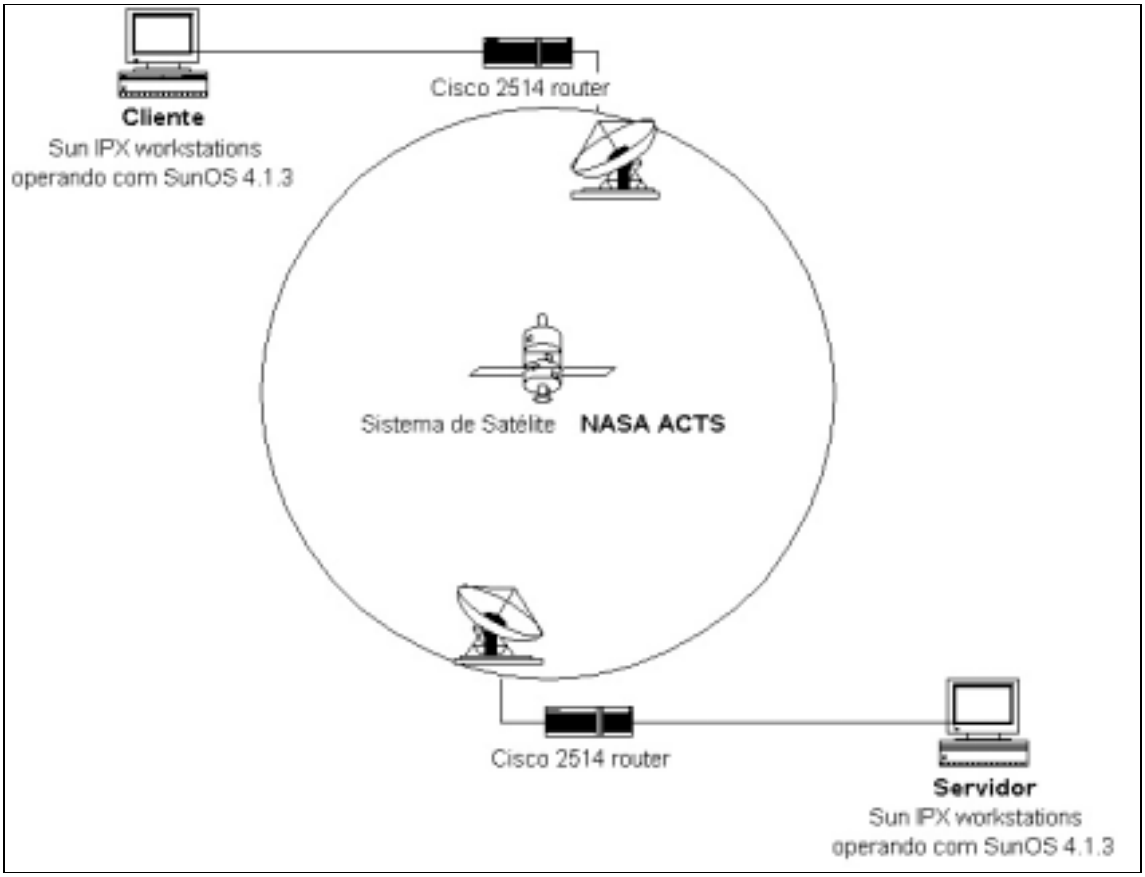


Figura 6.1 – Sistema de Satélite NASA ACTS. Esta figura mostra o *layout* da rede para os testes deste estudo envolvendo o Sistema de Satélite NASA ACTS.

6.1.2.2 EMULADOR DE SOFTWARE

O emulador de *software* usado em nossos experimentos é o *Ohio Network Emulator* (*ONE*). O *layout* da rede para os experimentos que usam ONE é mostrado na Figura 6.2. O emulador roda numa estação Sun que tem o Solaris como sistema operacional. Os pontos finais na figura são uma mistura de máquinas Sparc IPC e Intel 486 rodando NetBSD 1.1. O emulador de *software* também usa uma estratégia de enfileiramento *drop-tail*.

Figura 6.2 – Configuração do Emulador de *Software ONE (Ohio Network Emulator)*. Esta figura mostra o *layout* da rede para todos os testes envolvendo o emulador de *software ONE*.

O ONE passa segmentos entre as duas redes físicas nas quais ele está conectado baseando-se num número de parâmetros configurados pelo usuário. Cada segmento, que passa através do emulador de *software*, é submetido a 3 atrasos:

- Atraso de transmissão: é o tempo que se leva para transmitir o segmento numa rede com uma largura de banda configurada pelo usuário.
- Atraso de propagação: é o tempo gasto por um segmento viajar pelo comprimento do canal emulado.
- Atraso de enfileiramento: este atraso é determinado pelo ONE. Se um segmento chega enquanto outros estão sendo servidos, o segmento entrante deve ser enfileirado e aguardar seu momento para ser transmitido. O tamanho da fila determina o atraso de enfileiramento inserido pelo ONE.

O ONE foi configurado para modelar o Sistema ACTS e o mesmo conjunto de experimentos rodaram em ambos os ambientes. A Figura 6.3 mostra o ONE modelando precisamente o canal de satélite ACTS.

6.1.2.3 EMULADOR DE HARDWARE

O emulador de *hardware* usado em nossos experimentos foi o *TesteLink Data Link Simulator*. O emulador roda numa máquina Intel 486, equipada com duas placas especiais de expansão que foram desenhadas para emular a rede configurada pelo usuário.

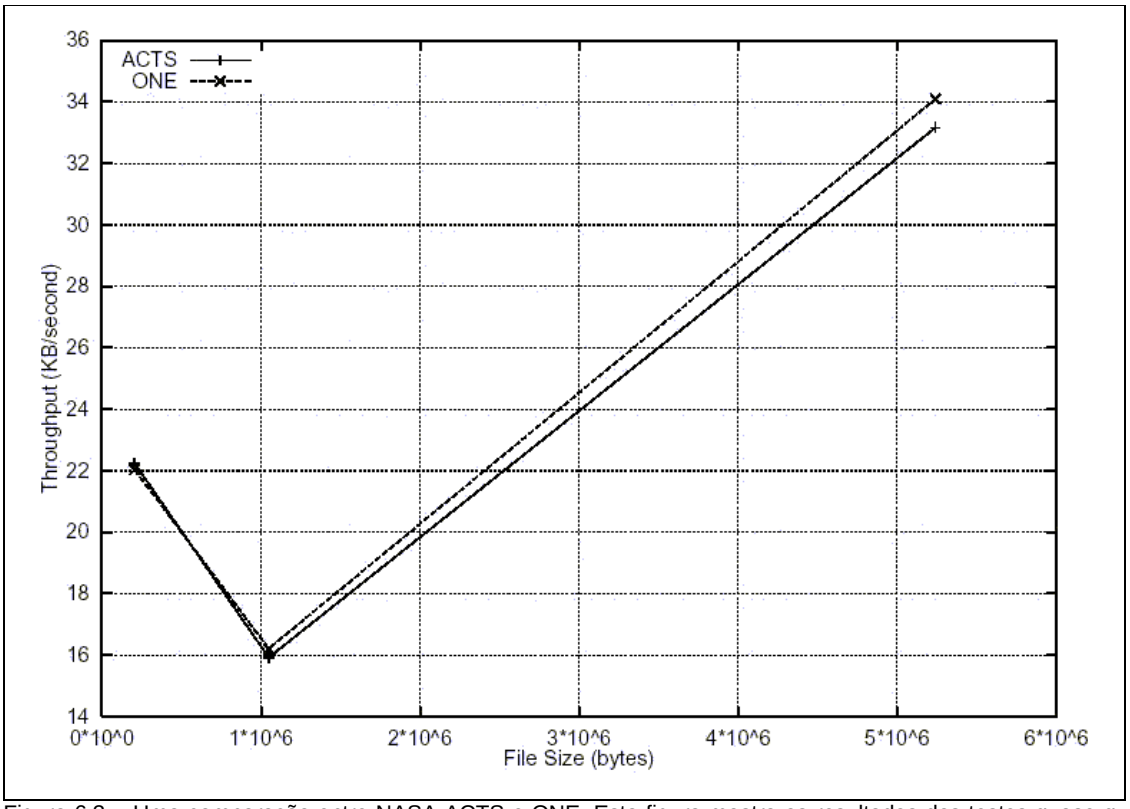


Figura 6.3 – Uma comparação entre NASA ACTS e ONE. Esta figura mostra os resultados dos testes quase que idênticos realizados no Sistema de Satélite NASA ACTS e no emulador de *software* ONE.

O emulador é configurado usando um programa sucinto. O *layout* da rede usado nos experimentos de [14] é ilustrado na Figura 6.4. Como no experimento NASA ACTS, foram usados os roteadores Cisco 2514 empregando enfileiramento *drop-tail*. Além disso, a mistura de máquinas Sun IPC e Intel 486 NetBSD 1.1 usadas na plataforma descrita acima foi novamente usada como pontos finais para experimentos usando o emulador de *hardware*.

O emulador de *hardware* limita a largura de banda entre os roteadores por fornecer um pulso de *clock* apropriado para o canal baseado na largura de banda configurada pelo usuário. O pulso de *clock* determina a taxa em que os roteadores podem transmitir seus segmentos. O atraso de enfileiramento é inserido pelos roteadores, assim como no sistema ACTS. O atraso de propagação ajustado pelo usuário é injetado pelo emulador atrasando os segmentos para o tempo ajustado pelo usuário antes deles transmitidos.

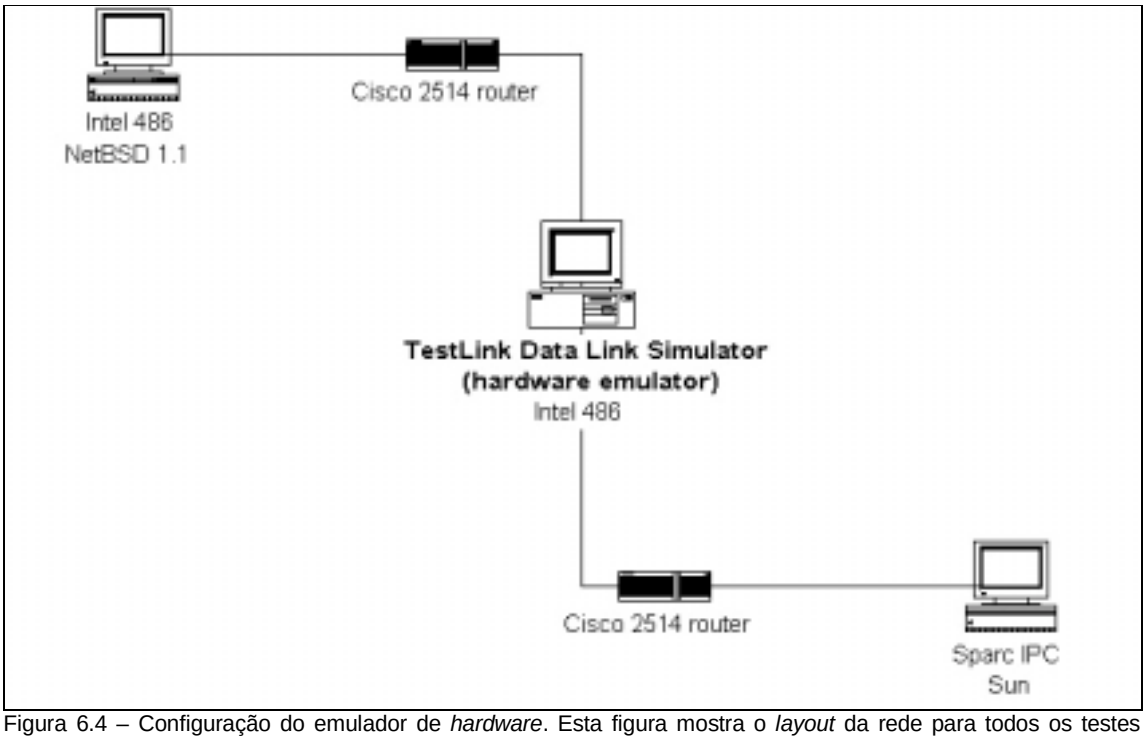


Figura 6.4 – Configuração do emulador de *hardware*. Esta figura mostra o *layout* da rede para todos os testes envolvendo o emulador de *hardware*.

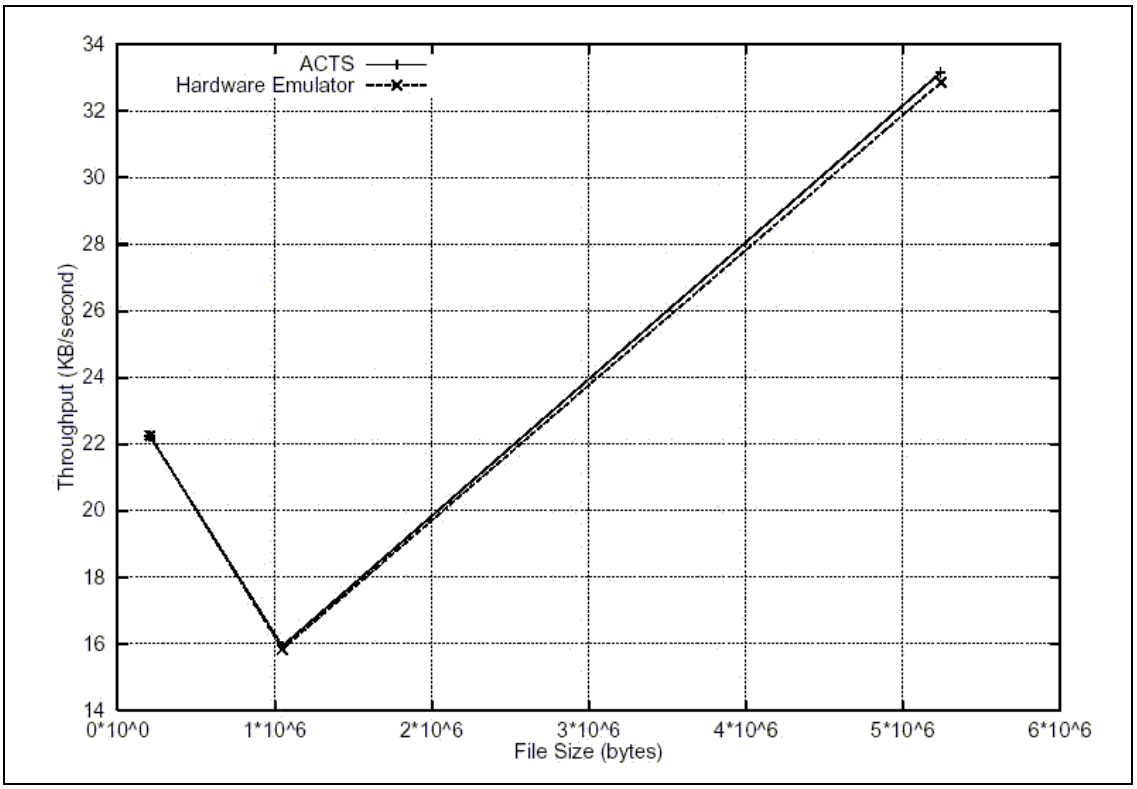


Figura 6.5 – Uma comparação do NASA ACTS e Emulador de *Hardware*. Esta figura mostra os resultados dos testes praticamente idênticos realizados nos Sistema de Satélite NASA ACTS e emulador de *hardware*.

Foi configurada a plataforma de teste de *hardware* para modelar o ambiente ACTS e rodar os mesmos experimentos em ambos os ambientes. A Figura 6.5 mostra que o emulador de *hardware* modela corretamente o canal de satélite ACTS.

6.1.3 Resultados Experimentais

O primeiro experimento de Fritz-Dolores [14] mostrou que o MFTP foi capaz de utilizar 84% da largura de banda disponível sobre o *link* de satélite através do número de conexões de dados usadas. A Figura 6.6 apresenta o resultado da transferência de um arquivo de 5 MB (5.242.880 bytes) em função do número de conexões de dados usadas. Esta figura apresenta resultados interessantes, como é explicado abaixo.

Como descrito em [16] e já citado na equação 6.2, usar uma única conexão TCP para transferir um arquivo não utiliza completamente a largura de banda disponível. A diferença entre o *throughput* da equação 6.2 e o *throughput* atual (Figura 6.6) pode ser explicada pelo *overhead* no cabeçalho do segmento TCP/IP (aproximadamente 8%).

Usar 4 conexões de dados em paralelo dá ao MFTP uma janela efetiva de 98.304 *bytes*. Isso é aproximadamente 91% do tamanho de janela necessária para se utilizar completamente a largura de banda disponível (de acordo com a equação 6.4). Usando 4 conexões de dados, o MFTP foi capaz de utilizar 87% da capacidade disponível, levando-se em conta o cabeçalho TCP.

Como indicado na equação 6.5, usar 4 conexões de dados não provê uma janela efetiva grande o suficiente para usar a capacidade disponível. Entretanto, 8 conexões de dados provêem uma janela efetiva de 196.608 *bytes*, que é muito maior do que os 107.520 *bytes* necessários. Entretanto, usando 8 conexões, tem-se um rendimento de uso da largura de banda de 84% somente. E mesmo quando o *overhead* do cabeçalho é levado em consideração, a utilização ainda é somente de 92%.

Embora o tamanho requerido da janela (107.520 *bytes*, equação 6.4) tenha sido excedido, a utilização continuou a melhorar as conexões que foram adicionadas pelas 8 conexões. Contudo, quando 10 e 12 conexões são de dados foram empregadas, a utilização despencou. O exame dos traços da rede indicou

uma grande perda no final do Slow Start na maioria das conexões TCP. Essa perda explica a redução no *throughput*, já que o TCP deve, primeiramente, esperar o tempo certo para retransmitir o dado perdido e então reduzir a taxa de transmissão.

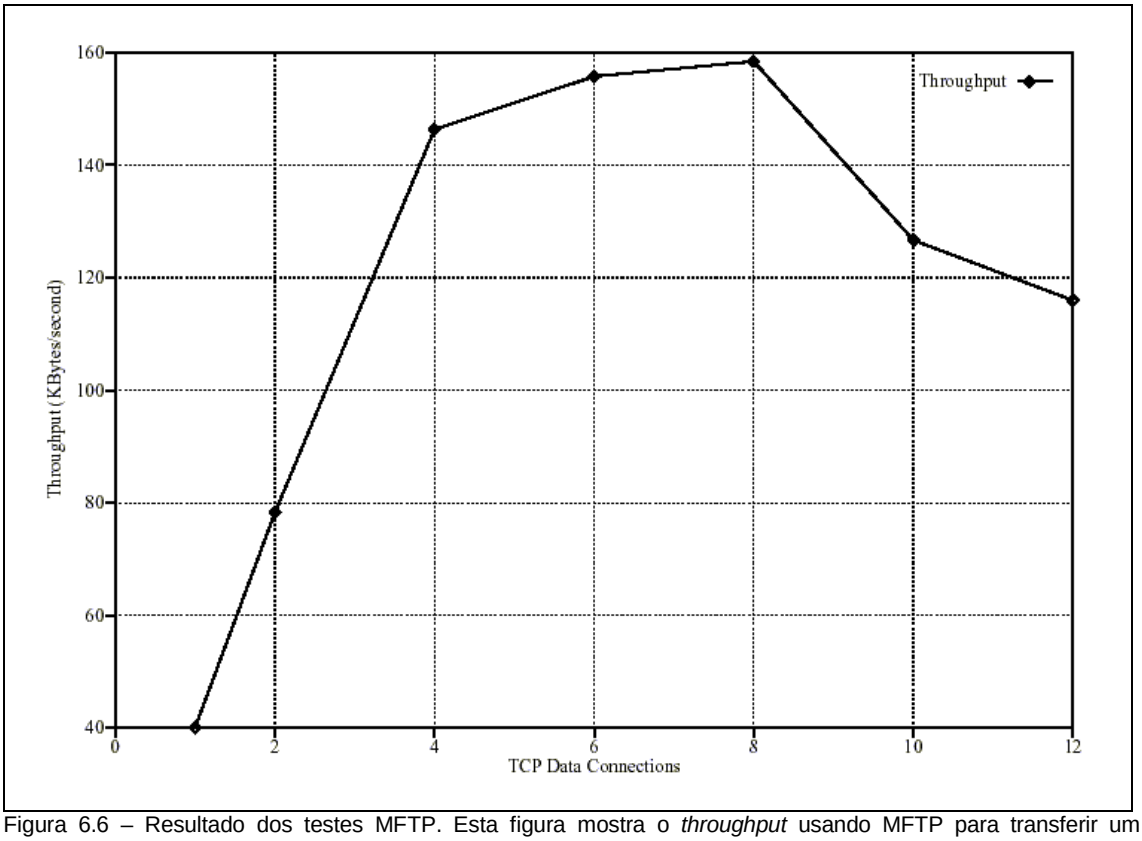


Figura 6.6 – Resultado dos testes MFTP. Esta figura mostra o *throughput* usando MFTP para transferir um arquivo de 5 MB utilizando o sistema de satélite NASA ACTS plotado em função do número de conexões de dados em paralelo empregadas.

A causa da perda pode ser diretamente explicada pelo algoritmo Slow Start. Cada conexão de dados é iniciada pelo incremento exponencial da taxa de transmissão de cada RTT até ele alcançar a janela recebida ou alguma perda ser detectada. Com múltiplas conexões de dados, dobrando-se suas taxas de transmissão, o roteador intermediário fica facilmente sobrecarregado pelas rajadas de segmentos, e deve, portanto, descartar os segmentos que não é capaz de enfileirar. No resultado do experimento mostrado pela Figura 6.6, o roteador foi capaz de enfileirar bastante tráfego evitando que perdas acontecessem quando até 8 conexões foram usadas pelo MFTP. Contudo, quando 10 conexões foram empregadas, a taxa de transmissão agregada sobrecarregou os *buffers* do roteador e uma perda maciça de dados ocorreu.

Depois desses experimentos com o ACTS, Fritz-Dolores [14] não foram capazes de explicar porque o MFTP utilizou somente 92% da capacidade do canal quando 8 conexões foram empregadas, levando-se em conta o cabeçalho. Analisando os cliente e servidor MFTP descritos, observou-se que as transmissão e recepção foram desnecessariamente lentas. O MFTP escreve e lê os dados em blocos de 8 kB. A versão do MFTP descrita acima escreveria 8 kB num momento específico, mesmo se a rede fosse incapaz de aceitar 8 kB completos. Quando o sistema operacional esteve incapaz de escrever todo o bloco, o MFTP foi forçado a aguardar até que a rede pudesse aceitar o resto. Um problema similar foi encontrado no lado do receptor. Depois da primeira bateria de testes, o MFTP foi modificado para usar chamadas de leitura e escrita assíncronas. A versão modificada do MFTP transmite a quantidade que for possível do bloco de 8 kB e coloca o resto em fila para posterior transmissão. Isso permite ao MFTP transmitir dados em uma outra conexão de dados ao invés de esperar que todo o bloco de 8 kB seja transmitido. Novamente, um mecanismo similar é empregado no lado do receptor. Neste ponto, o MFTP está sempre escrevendo ou lendo (a não ser que todas as conexões estejam indisponíveis). Essa otimização rende uma utilização melhor da rede, como mostrado na Figura 6.7. Usando de 6 a 8 conexões, essa versão do MFTP foi capaz de utilizar aproximadamente 90% da capacidade disponível. Levando em conta o *overhead*, o MFTP utiliza aproximadamente 98% da capacidade do canal, como pode ser observado.

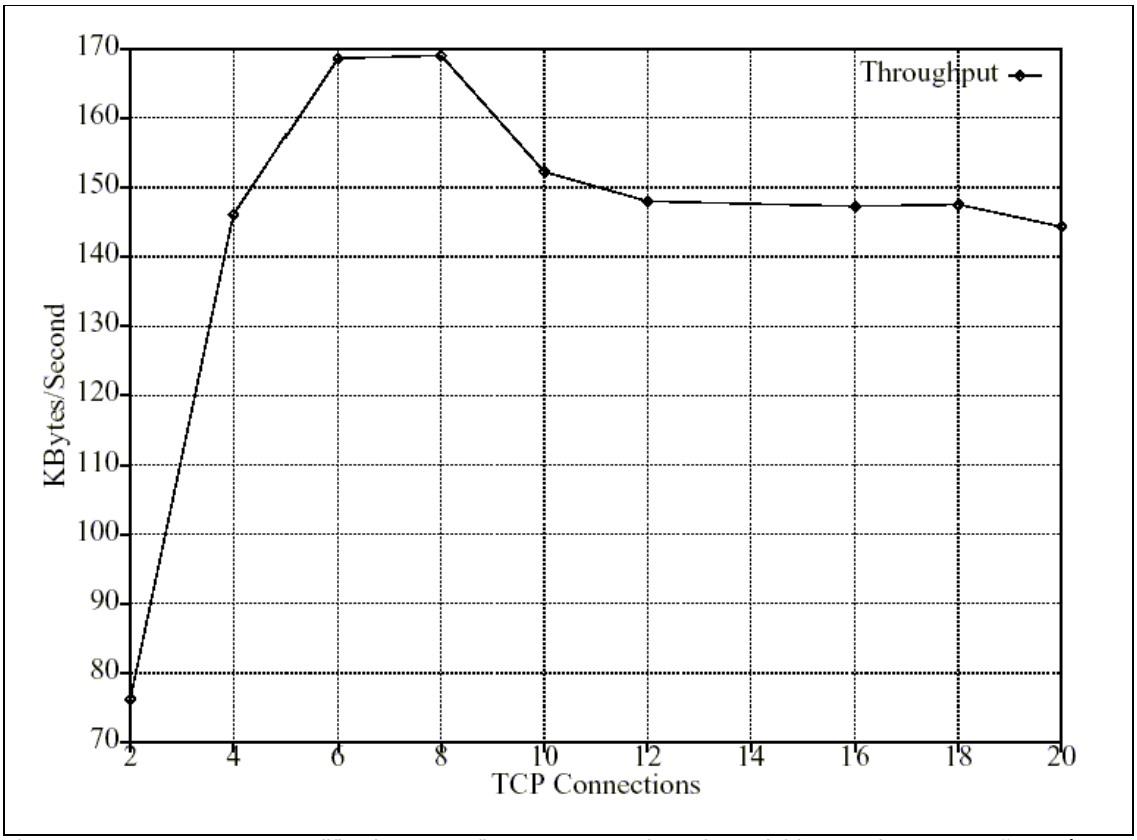


Figura 6.7 – Testes MFTP modificados. Essa figura mostra o *throughput* obtido usando uma versão assíncrona do MFTP para transferir um arquivo de 5 MB através do emulador de *hardware* plotado em função do número de conexões paralelas empregadas.

6.1.4 Lições do MFTP

Estudar o comportamento dos testes realizados por Fritz-Dolores [14] sobre o MFTP tem proporcionado compreensão com relação às modificações que podem melhorar a performance sobre canais de satélite. O uso de *N* conexões de dados MFTP o torna *N* vezes mais agressivo, diferentemente de quando se trata de apenas uma conexão TCP. Por exemplo, o Slow Start efetivamente começa pelo envio de *N* segmentos ao invés de 1 segmento. Analogamente, o Congestion Avoidance efetivamente adiciona *N* segmentos por RTT ao invés de 1 único segmento. Além disso, o MFTP é mais agressivo em face dos segmentos perdidos. Por exemplo, se cada uma das *N* conexões de dados estiver usando um *cwnd* de *m* segmentos e um simples segmento é perdido em uma das conexões, o MFTP irá reduzir o *cwnd* para o valor dado na equação 6.6 ao invés de reduzir o *cwnd* à metade como uma simples conexão TCP faria. Essa redução representa uma das *N*

conexões de dados que reduz seu *cwnd* pela metade, enquanto as outras *N* – 1 continuam usando o mesmo tamanho de janela.

$$\left(\frac{2N-1}{2N}\right)m \qquad 6.6)$$

Na medida em que o MFTP é mais agressivo do que uma simples conexão TCP, ele pode proporcionar certo entendimento através de novos mecanismos que ajudarão o TCP a utilizar melhor os canais de satélite. As lições proporcionadas pelo MFTP incluem a necessidade de janelas maiores, reconhecimentos seletivos e a possibilidade da necessidade de tornar os algoritmos de controle de congestionamento padrão mais agressivos.

6.1.4.1 JANELAS MAIORES

O caso de janelas TCP maiores é dado na seção 2.2.3.1 Além disso, os experimentos MFTP apresentaram em 6.1.1 que uma maior janela TCP efetiva pode utilizar totalmente a capacidade dos canais de satélite. As opções do TCP para proporcionar janelas maiores têm sido definidas em [56] [66] [67]. Essas extensões têm sido proveitosamente provadas em redes com alta latência e [36] e em redes de grande largura de banda [7].

6.1.4.2 RECONHECIMENTOS SELETIVOS

O MFTP mostra que os erros podem ser mais bem reparados se o TCP der uma idéia melhor sobre quais segmentos tiverem sido perdidos. Usar N conexões de dados permite o TCP detectar N perdas simultaneamente. O TCP é capaz de reparar as N perdas usando o Fast Retransmit, ao contrário de apenas 1 segmento perdido quando usa uma simples conexão TCP. Isso sugere que o transmissor TCP precise de mais informação sobre quais segmentos chegaram no receptor. Um mecanismo de RECONHECIMENTO SELETIVO (*Seletive Acknowledgment,* SACK) provê informação sobre quais segmentos exatamente chegaram no receptor. Os SACKs permitem que o transmissor TCP retransmita somente aqueles segmentos que foram perdidos na rede. Um SACK é definido pelo

TCP em [39]. Os SACKs têm sido proveitosamente provados em muitos protocolos de pesquisas (por exemplo, VMTP [8],NETBLT [9], RDP [VHS84]), bem como o TCP [36] [28].

7. SOFTWARES UTILIZADOS

Como não dispúnhamos de qualquer plataforma emuladora de *hardware* (que pudesse emular o satélite GEO de nosso estudo), ao contrário de [14], e já que foi necessário refazer os testes realizados primeiramente por eles em sua tese de mestrado, tornou-se necessário o uso de alguns *softwares*: um de simulação (*Network Simulator*) [43], onde pudemos descrever nosso ambiente de estudo; e outro (*Trace Graph*) [31], que apenas interpretou os arquivos gerados pelo primeiro, a fim de comprovar nossos estudos. A seguir serão discriminados ambos os *softwares*, a fim de que se possa entender os procedimentos adotados na realização dos nossos testes comparativos.

7.1 Network Simulator – NS

O *Network Simulator* é um simulador de eventos discretos centrado em pesquisas de redes com a finalidade de fornecer suporte a simulações para as mais variadas tecnologias. Trata-se de um *software* derivado do *Real Network Simulator,* surgido em 1989, tendo, hoje, seu desenvolvimento focado num trabalho de cooperação com o projeto VINT (*Virtual InterNetwork Testambémed*) formado por várias entidades educacionais e de pesquisa [43].

A abrangência das funcionalidades presentes no NS é enorme. Através do NS é possível simular os mais diversos ambientes de rede, que podem ir desde uma simples rede local de 10 Mbps até uma malha de telefonia celular, por exemplo. Com o NS também é possível fazer diversas modificações nos protocolos já existentes, como complemento de estudos didáticos mais aprofundados. Também existe a possibilidade de se inserir funções que simulam erros e atrasos físicos e de processamento, de modo que fique o mais próximo de uma situação realista.

O NS é um *software* de arquitetura modular, cujo núcleo foi construído através do uso a linguagem C++, orientada a objeto, com o objetivo de fornecer suporte à interface OTcl[13], responsável pela interpretação dos *scripts*[14] de

[13] Linguagem Tcl orientada a objeto.
[14] *Script*, algoritmo a ser interpretado por um certo interpretador de comandos; código fonte de um programa escrito para uma determinada linguagem.

simulação. Tais *scripts* são escritos em linguagem Tcl. Basicamente, a interface OTcl tem as funções de configuração e ajustes da simulação, ao passo que a interface C++ é responsável por ações referentes a cada pacote de um fluxo isoladamente. Esses *scripts*, por sua vez, são capazes de gerar os chamados arquivos *trace*[15], a fim de que toda a simulação fique registrada e possa ser analisada posteriormente, geralmente por um outro *software*, como será explicado mais adiante. Esses arquivos também podem ser usados para a animação da simulação[16], através do módulo NAM (*Network Animator*); análise gráfica de utilização dos *links,* através do módulo Xgraph*;* e cálculos estatísticos referentes aos pacotes, filas e enlaces da rede.

O NS foi desenvolvido primeiramente para a plataforma UNIX [64], abrangendo suas várias distribuições, incluindo, é claro, a plataforma Linux [29]. Contudo, é possível também executar o programa em ambiente *Microsoft Windows* [40], sendo necessário, para isso, a instalação dos pacotes em separado, e principalmente, a instalação de um compilador C para *Windows*, para que o programa possa ser compilado e executado corretamente. Para isso, geralmente é usado o *Borland C++* [5]. A fim de facilitar o processo de simulação e querendo-se evitar maiores problemas de execução, a plataforma Linux [29], na sua distribuição Red Hat 7.1, foi escolhida como base das nossas simulações NS.

7.1.1 Simulação

Como explicado na seção 7.1, seu funcionamento está baseado na interpretação de um *script* de simulação escrito na linguagem Tcl. Os processos que compõem uma simulação são mostrados na Figura 7.1 a seguir. A primeira etapa é construir um *script* onde serão listadas todas as configurações referentes à simulação. Esse arquivo será, então, interpretado pelo *Network Simulator* que, por sua vez, disponibilizará os chamados arquivos *trace*. Dependendo do tipo do arquivo *trace* que o programador escolheu para ser gerado durante a compilação, este pode ser usado para animação (exemplo.nam) ou usado para a análise estatística da simulação (exemplo1.tr). Deste último, podem ser obtidos diversos tipos de análise, incluindo gráficos de utilização de *links*, perdas de pacotes, etc.

[15] A extensão para os arquivos *trace* é “tr”.
[16] A extensão para os arquivos de animação é “nam”.

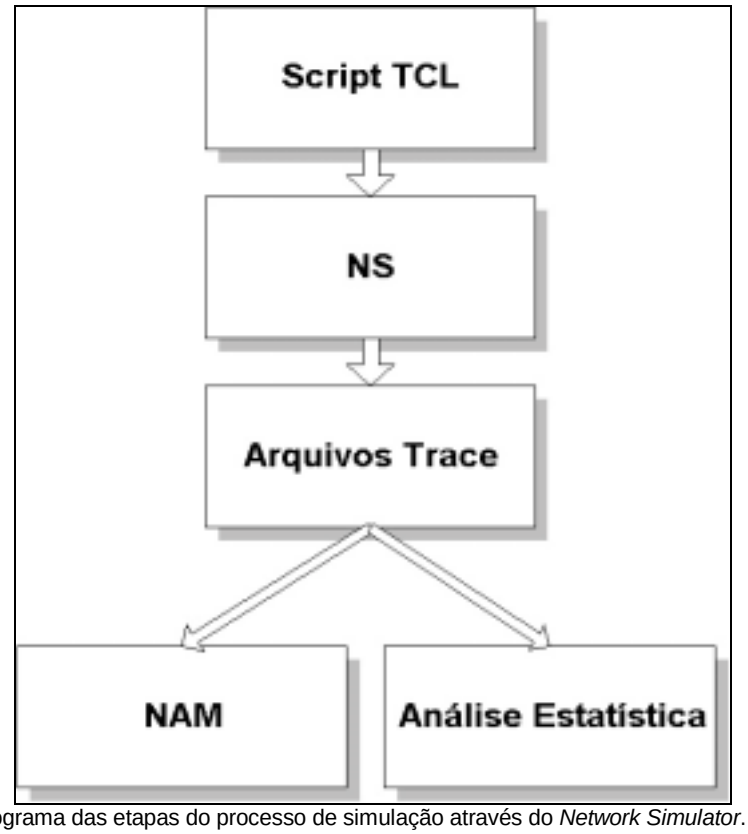


Figura 7.1 – Organograma das etapas do processo de simulação através do *Network Simulator*.

7.1.2 Criando um Script Tcl-NS

Geralmente, um *script* é iniciado pela criação de uma instância de simulação onde são chamados vários métodos para criação de nós, topologias de rede, escalonador de eventos e configuração de outros aspectos da simulação. Abaixo estão listados os métodos e suas respectivas descrições que compõem um arquivo Tcl.

- Métodos de escalonamento de eventos: usados para ativar e desativar eventos tendo como base uma escala temporal.
- Métodos de topologia: usados para configuração dos nós. Essas configurações são referentes a *buffers*, cores, formatos dos nós, posicionamento, entre outros.
- Métodos de *links*: usados para configuração dos *links*. Abrange banda passante, *delay*, tipo de tráfego e etc.
- Métodos de *tracing*: usados para geração de arquivos *trace*.

- Métodos de agentes: usados para inclusão e controle de agentes. Esses agentes são usados na implementação de protocolos de várias camadas. São eles que geram tráfego em um enlace. Exemplos de agentes são o TCP e o UDP.

A Figura 7.2 abaixo mostra um exemplo de um código NS de um dos programas usados para a realização de nossos testes. Note que o escopo do código contém as mesmas estruturas de uma linguagem de programação comum, sendo necessário declarar variáveis globais, criar laços de repetição e, habitualmente, fórmulas para a geração de cálculos.

```
#####################################################################################################
#Universidade de Brasilia - UnB                                                                      #
#Faculdade de Tecnologia - FT                                                                       #
#Departamento de Engenharia Eletrica – ENE
#Projeto Final de Graduacao em Engenharia de Redes de Comunicacoes - 1o./2002
#Graduandos:   Igor Pereira Marciano de Oliveira  -  97/22815
#              Leonardo Ferreira de Castro        -  97/22831
#
# Arquivo TCL a ser interpretado pelo NetWork Simulator - NS (www.isi.edu/nsnam/)
#
# Sobre este programa: Este arquivo simula o ambiente de satelite de nosso estudo com uma
# conexao TCP para transmissao de 1 arquivo de 5MBytes com a janela inicializada em 10
# pacotes.                                                                                           #
#####################################################################################################

# Criando um novo simulador de objeto
set ns [new Simulator]

# Abrindo os arquivos tr e nam
set animacao [open nam5Mj10.nam w]  ;#arquivo nam (animacao)
$ns namtrace-all $animacao
set dados [open dados5Mj10.tr w]    ;#arquivo tr (trace graph)
$ns trace-all $dados

set f1 [open 5Mj10_B.tr w]
set f2 [open 5Mj10_Mbps.tr w]

# Rotina de encerramento
proc finish {} {
        global ns animacao dados f1 f2
        $ns flush-trace
        close $animacao
        close $dados
        close $f1
        close $f2
        exec xgraph -m 5Mj10_B.tr –geometry 800x400 &
        exec xgraph -m 5Mj10_Mbps.tr –geometry 800x400 &
        exit 0
}

#########################################################################
#Criacao de um agente que ficara acima do sink do recepetor capaz de
#monitorar o numero de bytes recebidos e o instante respectivo

#########################################################################
Class TraceApp -superclass Application              ;#definicao da classe
#########################################################################

# Criando o objeto que monitora

TraceApp instproc init {args} {
        $self instvar bytes_
```

```
        $self instvar tempbytes
        $self set bytes_ 0
        $self set tempbytes 0
        eval $self next $args
}

TraceApp instproc recv {byte} {
        Global ns f1 f2 ftp
        $self instvar tempbytes
        set now [$ns now]
        $self instvar bytes_
        set bytes_ [expr $bytes_ + $byte]
        set tempbytes [$ns now]
        puts "TAM do pacote: $byte    Tempo de chegada: $tempbytes  Bytes acumulados no recp:
$bytes_"
        puts $f2 "$now [expr $bytes_*8/$tempbytes/1000000]"
        puts $f1 "$now [expr $bytes_/$tempbytes]"

        if "$bytes_ > 5242880" { ;# este if controla quando chegar 5M bytes no recetor,
encerra a conexao
                for {set a 1} {$a < 2} {incr a} {
                        $ns at $now "$ftp($a) stop"
                }
        }
        return $bytes_
}
#####################################################################

# Criando a topologia da rede

# Criando os 5 nos da nossa rede
set n1 [$ns node] ;# estacao terrestre comum
set n2 [$ns node] ;# estacao terrestre terminal de satelite
set n3 [$ns node] ;# este no eh o no satelite
set n4 [$ns node] ;# estacao terrestre terminal de satelite
set n5 [$ns node] ;# estacao terrestre comum

# Criando os links entre os nos.
# Atente para os links entre os terminais de sat (nos 2 e 4) e o proprio sat.
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.536Mb 130ms DropTail
$ns duplex-link $n3 $n4 1.536Mb 130ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail

# Melhorando a disposicao visual da rede
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient right-down
$ns duplex-link-op $n4 $n5 orient right

#####################################################################

# Gerar o trafego na rede

# Criar um agente TCP e atacha-lo ao no 1

for {set i 1} {$i < 2} {incr i} {
        set tcp($i) [new Agent/TCP]
        $ns attach-agent $n1 $tcp($i)   ;# o atachamos em n1;
        $TCP($i) set window_ 48         ;# configuramos a maxima janela para 48 pacotes;
        $TCP($i) set packetSize_ 512    ;# o tamanho dos pacotes eh igual a 512 bytes;
        $TCP($i) set ecn_ true
        $TCP($i) set rtt_ 0.56          ;# Round TripTime = 560 ms;
        $TCP($i) set srtt_ 0.56
        $TCP($i) set ssthresh_ 48       ;# slow-stat threshold (48 pacotes);
        $TCP($i) set bugFix_ true
        $TCP($i) set MWS 48
        $TCP($i) set windowInit_ 10     ;# inicializa o valor do cwnd para 10 pacotes;
        $TCP($i) set windowOption_ 1
        $TCP($i) set slow_start_restart_ true
        $TCP($i) set tcpTick_ 0.5

        # Criar o terminador de trafego
        set sink($i) [new Agent/TCPSink] ;# criamos o "esgotador de trafego" TCP
        $ns attach-agent $n5 $sink($i)   ;# o colocamos em n5
        $ns connect $tcp($i) $sink($i)   ;# conectamos os 2 agentes
```

```
#####################################################################

        # Criar uma aplicacao ftp e atacha-la ao tcp1
        set ftp($i) [new Application/FTP] ;# nova aplicacao FTP
        $ftp($i) attach-agent $tcp($i)    ;# associamos ao agente TCP criado anteriormente
        $ns at 0.0 "$ftp($i) start"
        $TCP($i) set class_ ($i)
}

set appover_n5 [new TraceApp]
for {set i 1} {$i < 2} {incr i} {
        $appover_n5 attach-agent $sink($i)
}
$ns at 0.0 "$appover_n5 start"
$ns at 130.0 "finish"          ;# duracao maxima da simulacao = 130 segundos
$ns run
```

Figura 7.2 – Exemplo de um código fonte escrito para ser interpretado pelo NS.

7.1.3 Arquivos Trace

Os arquivos *trace* são gerados depois de executada a simulação, ou seja, depois que o NS interpretou o *script* Tcl-NS. Para isso, torna-se necessário que dentro do *script* haja uma discriminação prévia de algum método de *trace* para que esses arquivos sejam gerados corretamente. São várias as finalidades de um arquivo *trace*, entre elas: análise do tráfego em um determinado *link*, comportamento de filas, estatística de perdas de pacotes, animação da simulação, entre outras.

O formato de um arquivo *trace* geral é mostrado na figura abaixo. Cada linha da figura determina um registro sobre a situação de um determinado pacote:

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
+	1.84375	0	2	cbr	210	-------	0	0.0	3.1	225	610
-	1.84375	0	2	cbr	210	-------	0	0.0	3.1	225	610
r	1.84471	2	1	cbr	210	-------	1	3.0	1.0	195	600
r	1.84566	2	0	ack	40	-------	2	3.2	0.1	82	602
+	1.84566	0	2	tcp	1000	-------	2	0.1	3.2	102	611
-	1.84566	0	2	tcp	1000	-------	2	0.1	3.2	102	611

Figura 7.3 – Exemplo de um arquivo trace gerado a partir de um código em OTcl.

- Coluna 1: descreve o evento. Opções: "+" o pacote foi colocado na fila; "-" o pacote foi retirado da fila; "r" pacote recebido; "d" pacote descartado.
- Coluna 2: o tempo em que ocorreu o evento, em segundos.
- Colunas 3 e 4: o enlace que ocorreu o evento.
- Coluna 5: tipo de pacote.
- Coluna 6: tamanho do pacote em *bytes*.

- Coluna 7: descreve os *flags*.
- Coluna 8: identificador de fluxo (*flow identifier*) do IPv6.
- Colunas 9 e 10: endereço de origem e destino respectivamente, no formato “NÓ.PORTA”.
- Coluna 11: número de seqüência do pacote do protocolo de camada de rede.
- Coluna 12: identificação única do pacote.

Através da manipulação desse arquivo é que se obtém as estatísticas requeridas. Normalmente o número de linhas é bastante grande sendo necessária filtragem ou interpretação automática desses dados através de algum *software* adicional, como o *Trace Graph* [31].

7.1.4 NAM - Network Animator

O NAM é o módulo do NS responsável por gerar as animações de rede, buscando essas informações no arquivo *trace*. O NAM foi construído com a capacidade de ler uma grande quantidade de dados de um arquivo *trace* e interpretar os comandos de configuração ali contidos.

A primeira etapa para tornar possível o uso do animador é gerar um arquivo *trace* próprio para a animação (como por exemplo, “teste.nam”). Seu formato não nos interessa já que só será usado pelo animador, o que dispensa o uso de algum filtro para coletar estatísticas. Esse arquivo contém informações de topologia como nós, enlaces e informações de *layout*, assim como informações de temporização de envio e recebimento de pacotes. Depois de gerado esse arquivo, ele está pronto para ser animado pelo *software*. De acordo com a Figura 7.4 a seguir, podemos observar um exemplo de uma animação no NAM.

Entretanto, apesar do NS ser, indiscutivelmente, uma ótima ferramenta de simulação e teste de tecnologias diversas, ele não provê suporte de animação para redes que envolvem satélites. Em outras palavras, o NAM não suporta animações que contenham elementos de satélite em sua topologia. Como será explicado no momento apropriado, isso foi, de certa forma, um tanto prejudicial na parte prática de nosso trabalho, já que quisemos ilustrar o que se passava numa transmissão.

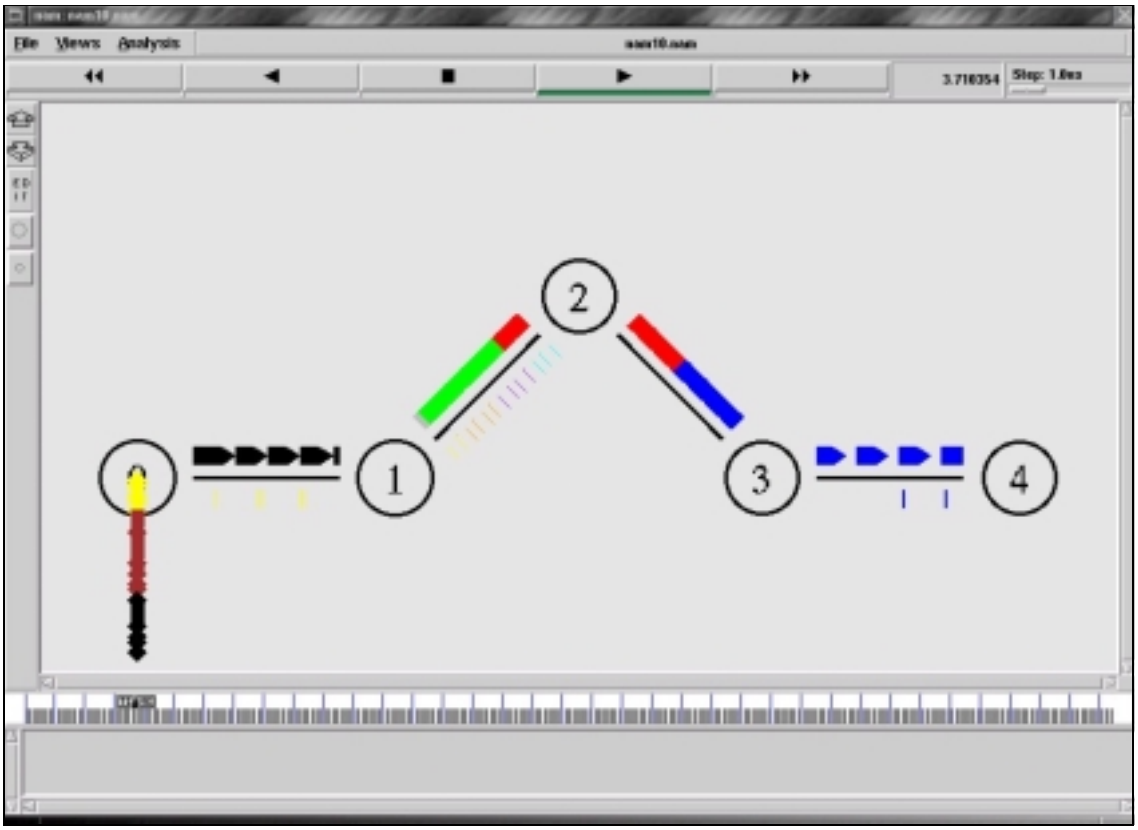


Figura 7.4– Exemplo de uma animação no NAM.

7.1.5 Xgraph

O Xgraph é uma ferramenta-módulo do NS capaz de plotar gráficos criados a partir de cálculos matemáticos que foram referenciados dentro do código Tcl. O Xgraph também funciona a partir de uma interpretação das fórmula usada no código e dessa forma, consegue gerar o gráfico envolvendo as variáveis utilizadas.

A Figura 7.5 que se segue mostra um gráfico gerado a partir de um código Tcl de nosso trabalho.

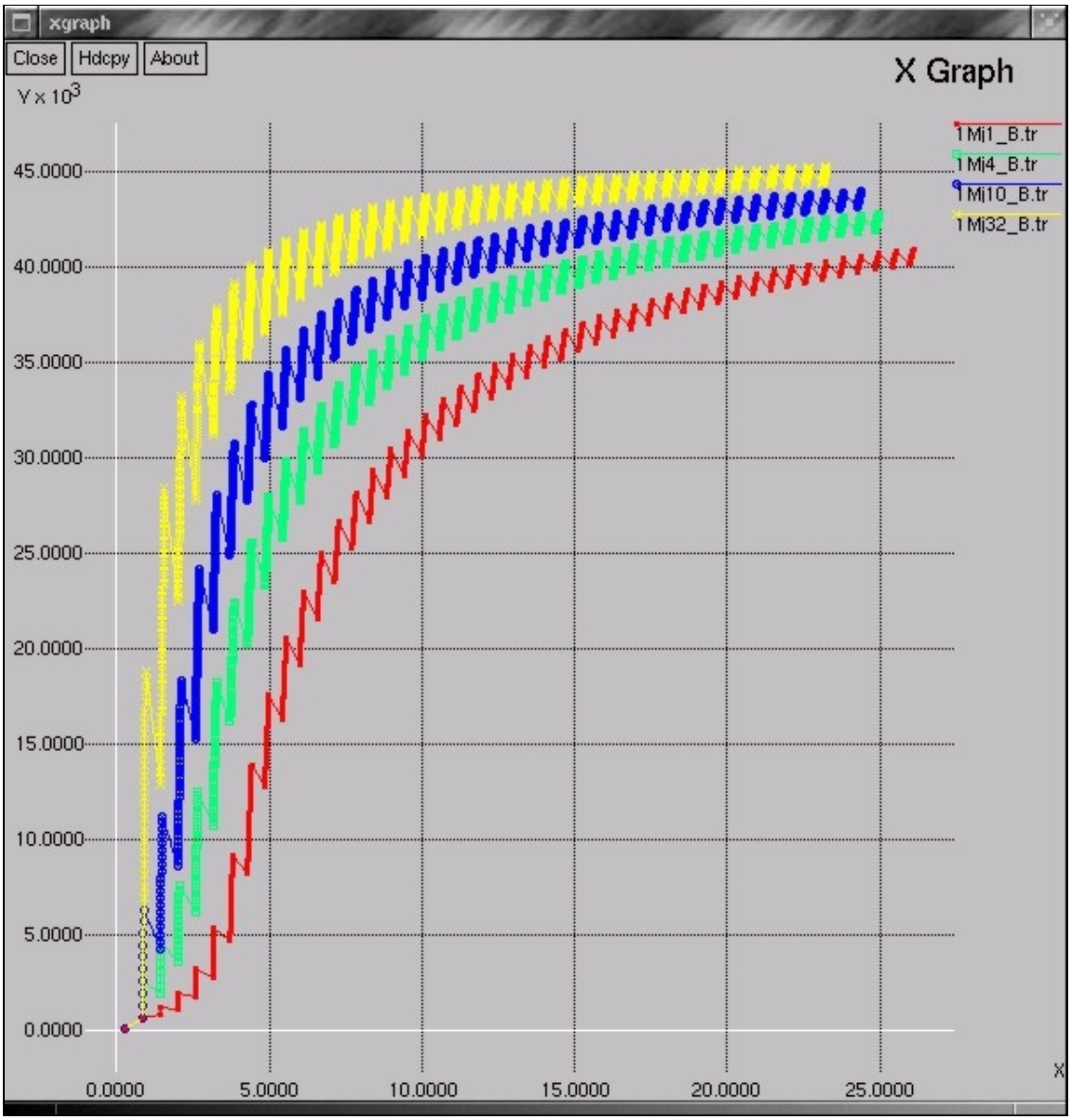


Figura 7.5 – Gráfico confeccionado pelo Xgraph.

7.2 Trace Graph

Como dito no início deste capítulo, o *Trace Graph* é um analisador de arquivos *trace*, gerados pelo NS, e foi desenvolvido por [31], estudante de Ciência da Computação na Universidade Técnica de Wroclaw, Polônia.

O programa opera através de algumas bibliotecas do *Matlab* [62], sendo, portanto, necessária sua instalação antes da execução do programa.

O *Trace Graph*, assim como o NS, foi desenvolvido primeiramente para a plataforma Linux [29], sendo compilado pela versão 2.96 (ou superior) do GCC[17]. Contudo, atualmente já existe sua versão compatível com *Microsoft Windows* [40], na qual tivemos analisados nossos resultados.

O programa é capaz de mostrar todas as informações ocorridas na simulação NS, plotando gráficos em 2D e 3D e salvando todos os resultados de análise. A Figura 7.6 abaixo ilustra um gráfico 3D criado pelo *Trace Graph* através da interpretação de um arquivo *trace* gerado no NS.

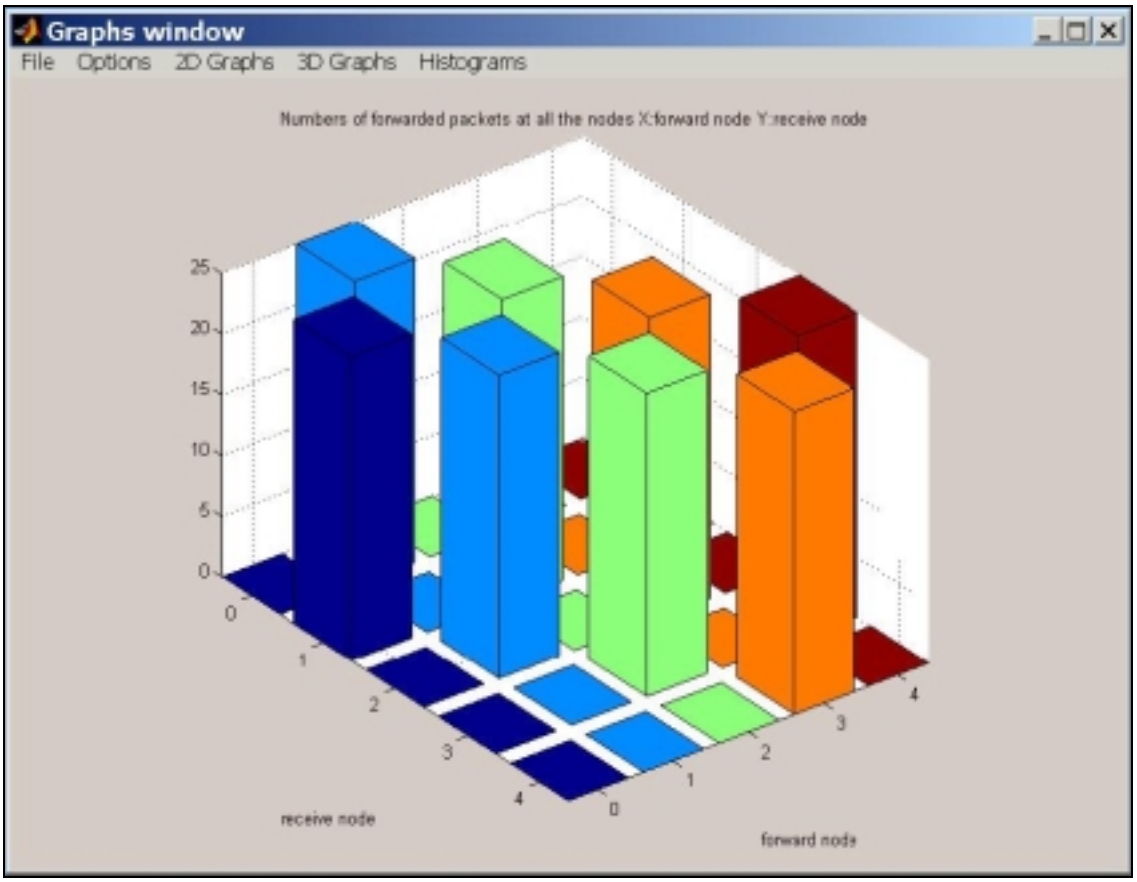


Figura 7.6 – Um exemplo de um gráfico 3D, gerado no *Trace Graph*, a partir dos dados de um arquivo *trace*.

Como já foi dito, o programa analisador de arquivos *trace* também é capaz de analisar tudo o que ocorreu na simulação, fornecendo informações valiosas como *throughput* alcançado, quantidade de segmentos perdidos, número de pacotes transmitidos, etc. A Figura 7.7 a seguir mostra de que o *Trace* Graph fornece essas informações.

[17] Compilador C presente no Linux [29].

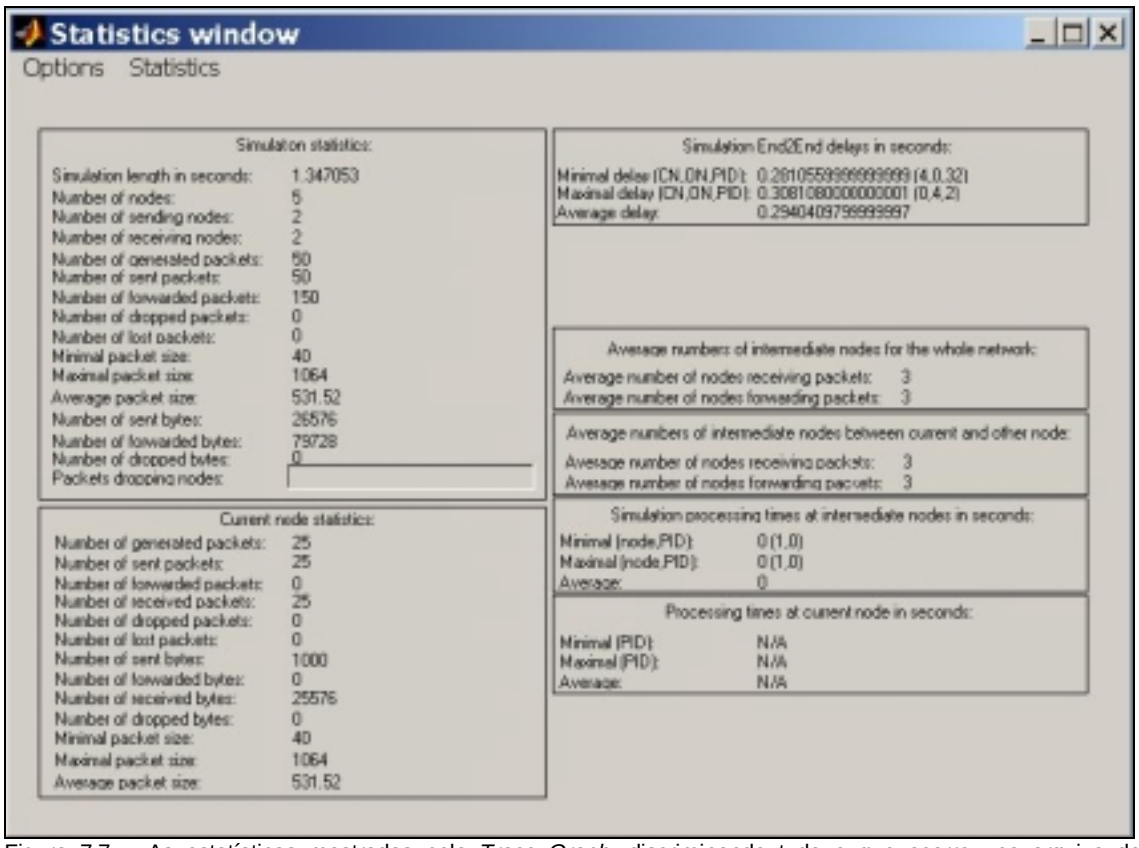


Figura 7.7 – As estatísticas mostradas pelo *Trace Graph*, discriminando tudo o que ocorreu no arquivo de simulação.

Dessa forma, através da ajuda proporcionada por ambos os *softwares*, conseguimos plotar os gráficos e realizar as análises necessárias para nossos experimentos comparativos. Todos esses experimentos e testes comparativos que fizemos tendo por base a tese de [14] são apresentados no capítulo seguinte.

8. COMPARAÇÃO DOS TESTES E RESULTADOS

Como dito na introdução deste documento, este capítulo foi reservado para a apresentação de nossos testes e resultados, a fim de compará-los com os obtidos por [14], comprovando o sucesso da utilização de múltiplas conexões TCP paralelas na transmissão de arquivos utilizando uma aplicação como o FTP [50].

Os testes tiveram a mesma topologia física dos testes realizados por [14] que é novamente mostrada na Figura 8.1 abaixo.

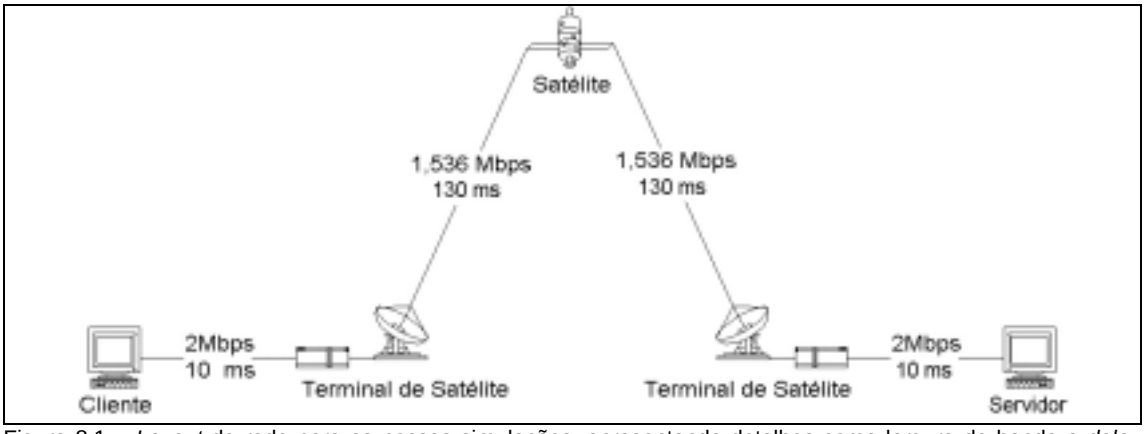


Figura 8.1 – *Layout* da rede para as nossas simulações, apresentando detalhes como largura de banda e *delay* dos enlaces envolvidos.

Contudo, conforme explicado no capítulo 7, não tivemos à nossa disposição uma plataforma emuladora de *hardware* que substituísse o satélite GEO de nosso estudo, e, portanto, fizemos uso do *Network Simulator* [43] para simular toda nossa rede, inclusive o próprio satélite. Dessa forma, optamos por ilustrar a nossa topologia de rede através de *links* de longo atraso, já que, como já foi dito na seção 7.1.1.3, o NS não provê animações para redes, cujas topologias apresentam elementos de satélite. Caso tivéssemos optado por utilizar as simulações de redes de satélites predefinidas, não teríamos como visualizar a rede através do NAM [43], o que comprometeria, certamente, o entendimento detalhado dos algoritmos de controle de congestionamento usados pelo TCP.

Analisando a Figura 8.1, observa-se a presença dos valores para a largura de banda definidos para cada enlace em nossa rede. Note que, entre os terminais de satélite e o próprio satélite, estabelecemos um *link* de 1,536 Mbps, o mesmo mostrado por [14] em seus experimentos. Note, ainda, que o atraso de cada

enlace foi devidamente calculado de modo a definir um RTT[18] de 560 ms, o mesmo observado no trabalho de [14].

Cada cabeçalho TCP, no NS [43], tem um tamanho fixo de 40 *bytes*, não podendo, portanto, ser ajustado pelo programador. Dessa forma, em todos os nossos testes, foi definido um tamanho de janela de anúncio máximo (ver seção 2.2.1) de 48 pacotes, tendo cada pacote um tamanho total de 552 *bytes*[19], de forma a deixar o cabeçalho TCP sendo aproximadamente 8% do tamanho total do pacote, conforme mostrado por [14] ser este o percentual comum da divisão de um pacote TCP. Dessa forma, utilizando 48 pacotes de 552 *bytes* cada, chegou-se a um tamanho de janela máximo de aproximadamente 24 kB, o tamanho usual da janela utilizado pelo FTP e o mesmo usado por [14] em seus experimentos.

8.1 Throughput e Número de Conexões Paralelas

Da mesma forma que fizeram [14] em sua tese, iniciamos nossos experimentos pela transferência de um arquivo de 5 MB, fazendo, primeiramente, uso de apenas 1 conexão TCP, e utilizando o FTP como aplicação. Os gráficos do *throughput* em função do tempo foram gerados em *bytes*/*segundo*, de forma a fazer a melhor comparação possível com o estudo de [14]. A Figura 8.2 a seguir mostra o *throughput* obtido para 1 conexão TCP.

Note que, utilizando uma conexão TCP apenas, obteve-se um *throughput* de 45.177 *bytes*/s, praticamente o mesmo do que o ilustrado na Figura 6.6, se desconsideramos os 8% de *overhead* (ver seção 6.1.3). Ao contrário de [14], por não terem revelado o tempo de transferência para cada número de conexões usadas em seus experimentos, percebe-se, pela Figura 8.2, que o tempo para a transmissão ser efetivamente completada foi de 116,31 segundos. Isto é, após aproximadamente 117 segundos de transmissão, o receptor já tinha recebido os 5 MB (na realidade, o receptor chegou a receber 5.254.528 *bytes*).

[18] O RTT, como já foi ilustrado, é definido como o tempo de percurso de ida e volta. Dessa forma, 2 x (10 + 130 + 130 + 10) ms = 560 ms.
[19] Tamanho total significa *header* mais *payload*. Isto é, 512 *bytes* de *payload* + 40 *bytes* de cabeçalho.

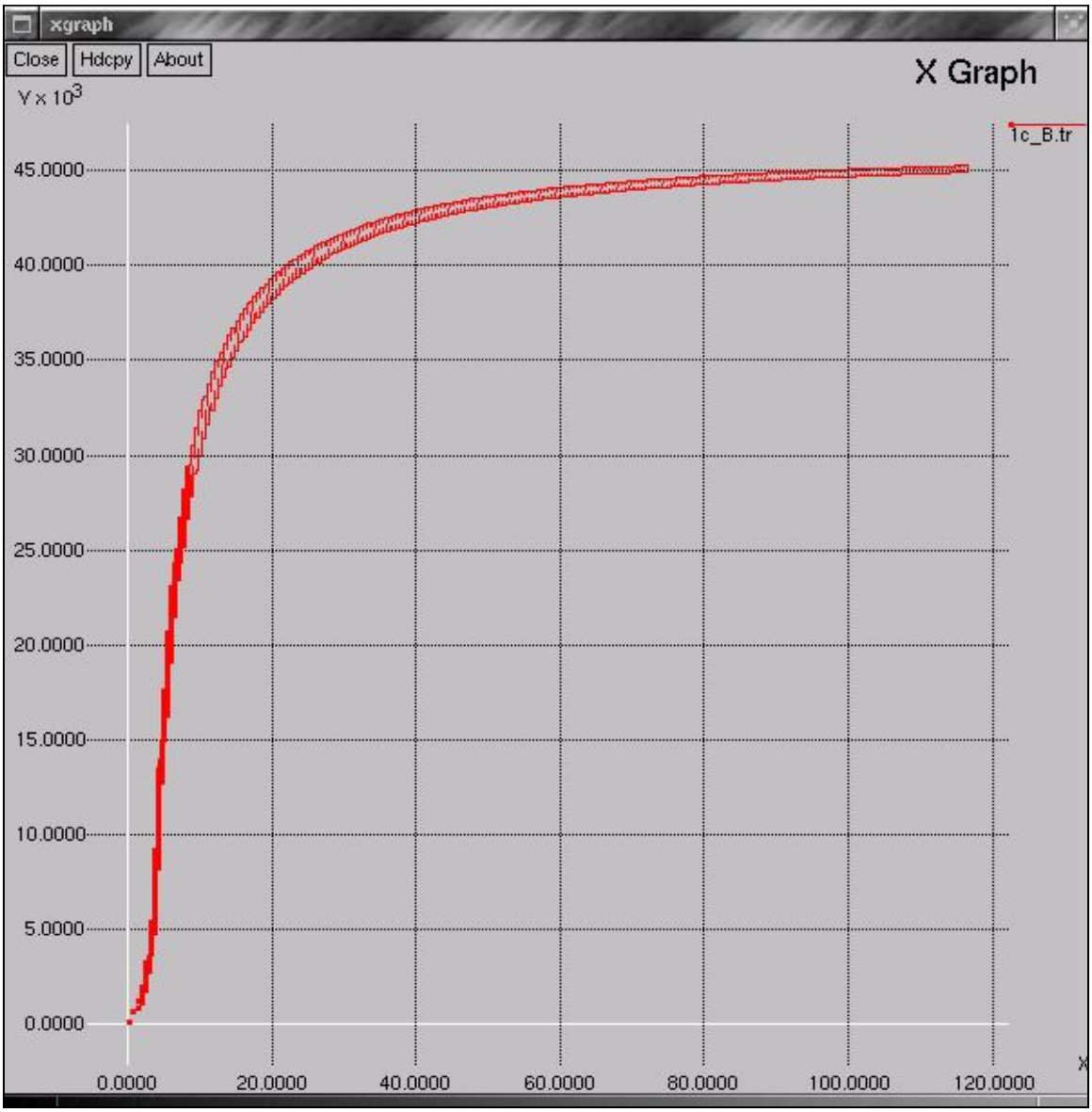


Figura 8.2 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando uma conexão TCP apenas.

A seguir, foi feito o mesmo para duas conexões. O que se obteve está mostrado na Figura 8.3, apresentando o resultado do *throughput* em *bytes*/s. Note que já com 2 conexões TCP paralelas, o tempo para os 5 MB serem completamente transmitidos caiu para 59,93 segundos, praticamente a metade do tempo de transmissão para uma conexão, e o *throughput* teve um aumento de quase 100%. Isso já comprova, de certa forma, o sucesso no uso das conexões TCP paralelas.

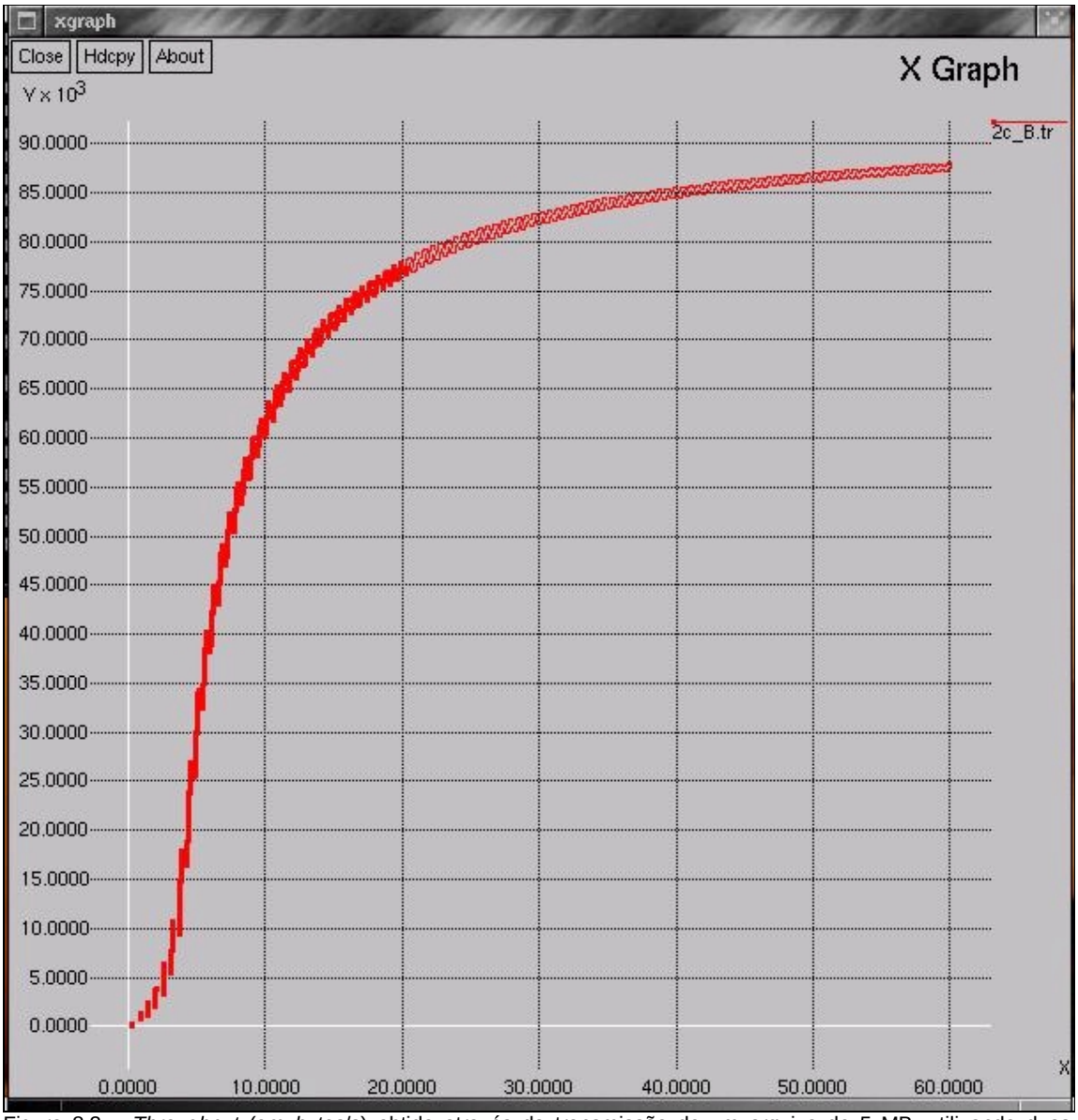


Figura 8.3 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando duas conexões TCP.

O próximo passo foi executar o programa para a análise de *throughput* para 4 conexões. O resultado é apresentado pela Figura 8.4, a seguir. O tempo de transmissão foi de 32,05 segundos e o receptor conseguiu receber 5.298.256 *bytes*, ou seja, os 5 MB transmitidos pela fonte. O *throughput* alcançado para 4 conexões foi, portanto, de 165.305 *bytes*/s.

Já para 6 conexões, obtivemos um *throughput* de 170.776 *bytes*/s, em 31,08 segundos. Neste caso, já se percebe que o aumento do número de conexões não influencia tanto no tempo de transmissão, o que já nos leva a supor que existe um limiar para o *throughput* na transmissão de nosso arquivo de 5 MB.

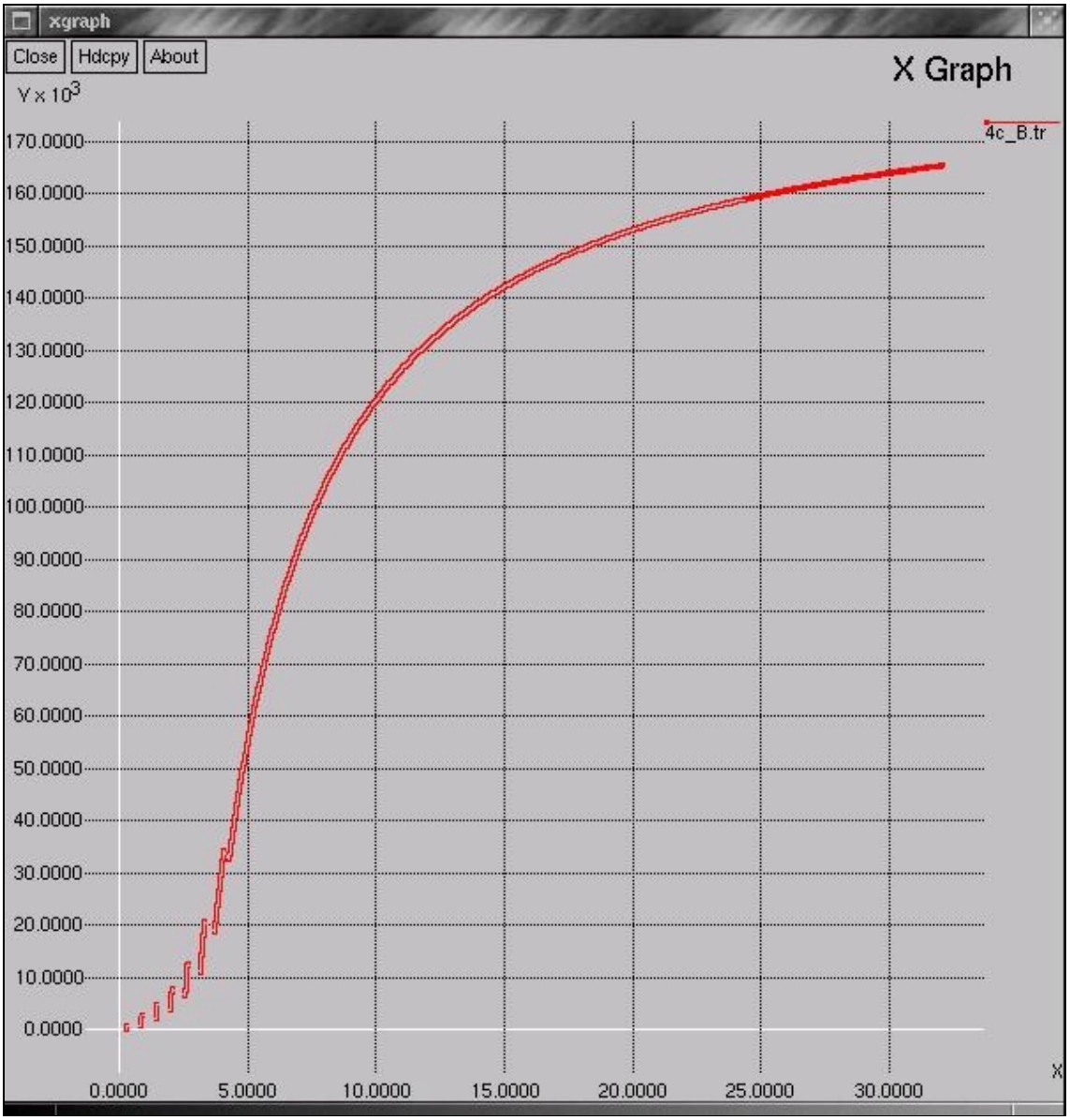


Figura 8.4 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando quatro conexões TCP.

Percebe-se que, desta forma, o tempo de transmissão para 6 conexões não é muito diferente do de 4 conexões, mesmo tendo aumentado em 2 unidades.

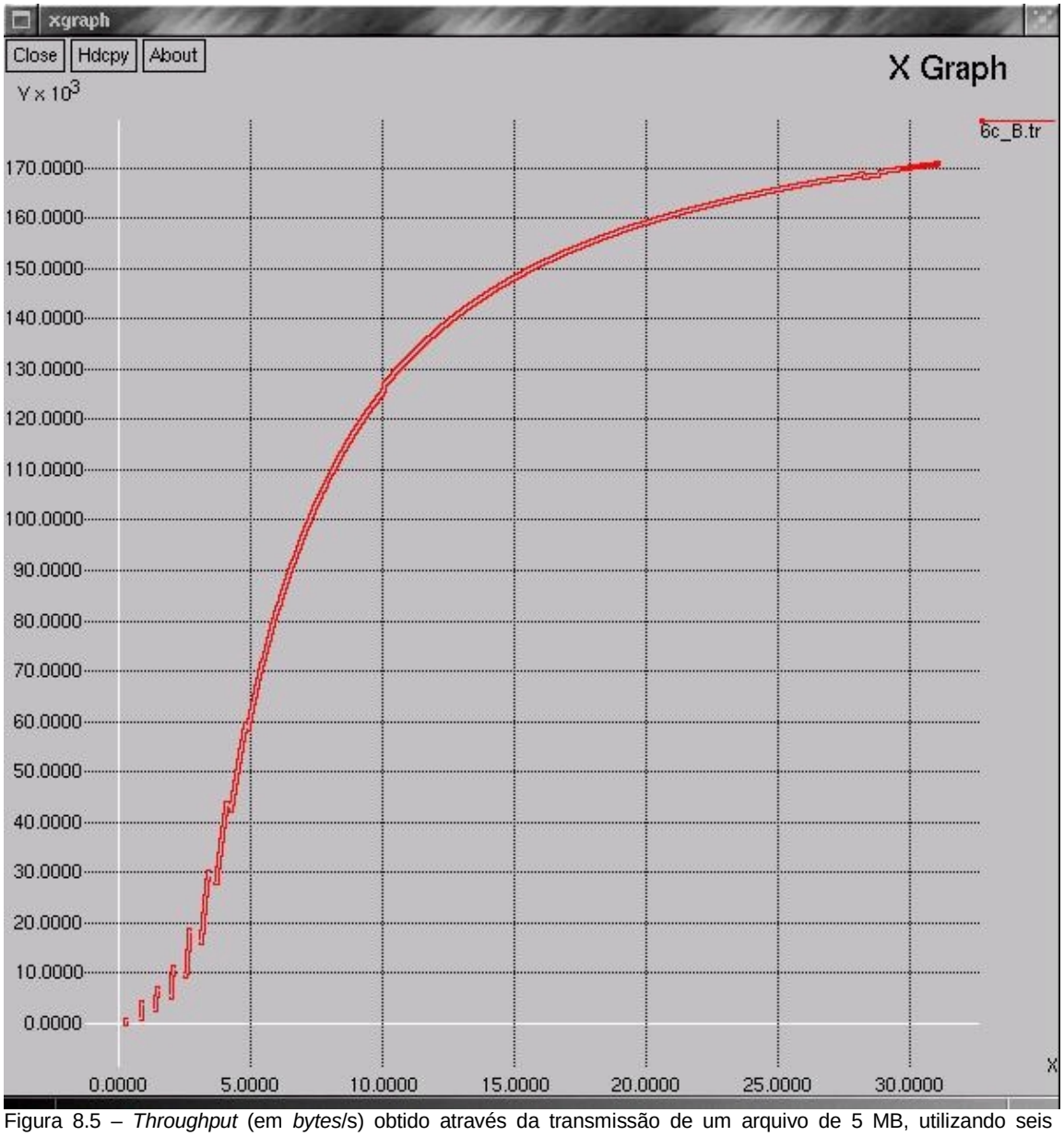


Figura 8.5 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando seis conexões TCP.

Neste ponto, apresentamos o resultados a partir da análise proporcionada pelo *software Trace Graph* [31], como dito no capítulo 7, comprovando estatisticamente alguns detalhes (como número de pacotes perdidos, número de pacotes recebidos, etc) observados durante a simulação e que valem a pena ser ressaltados neste momento.

A Figura 8.6 mostra as estatísticas analisadas para o caso de 4 conexões paralelas. É notado que não houve perda de pacotes neste caso, o que já era previsto, já que estava abaixo das 5 conexões, o número ótimo de conexões conforme calculado anteriormente (ver seção 6.1.1).

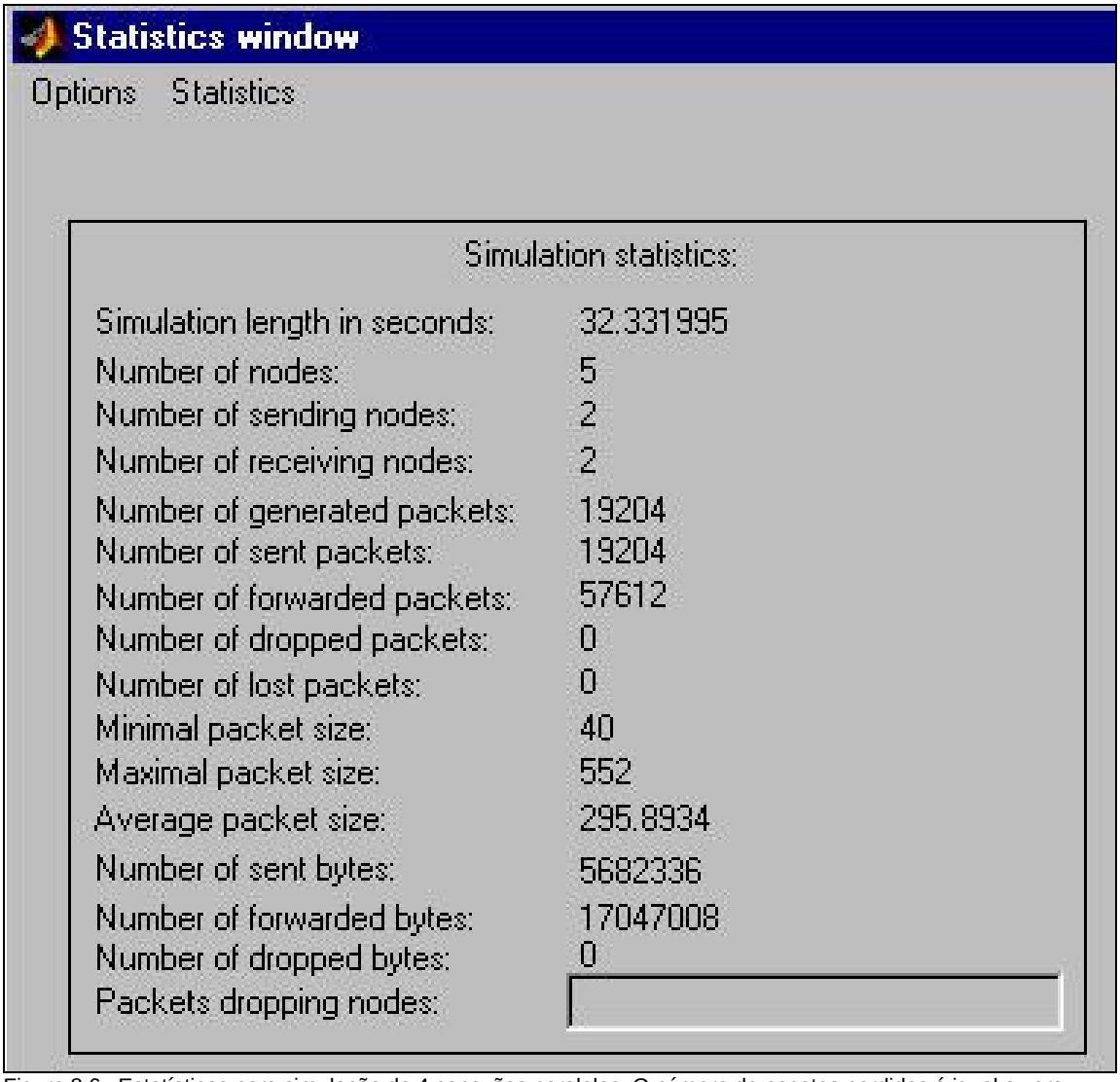


Figura 8.6– Estatísticas para simulação de 4 conexões paralelas. O número de pacotes perdidos é igual a zero.

Seguindo os mesmos procedimentos para a análise de uma, duas, quatro e seis conexões, o mesmo foi feito para 8, 10 e 12 conexões paralelas, obtendo-se *throughputs* mostrados pelas figuras 8.7, 8.8 e 8.9, respectivamente.

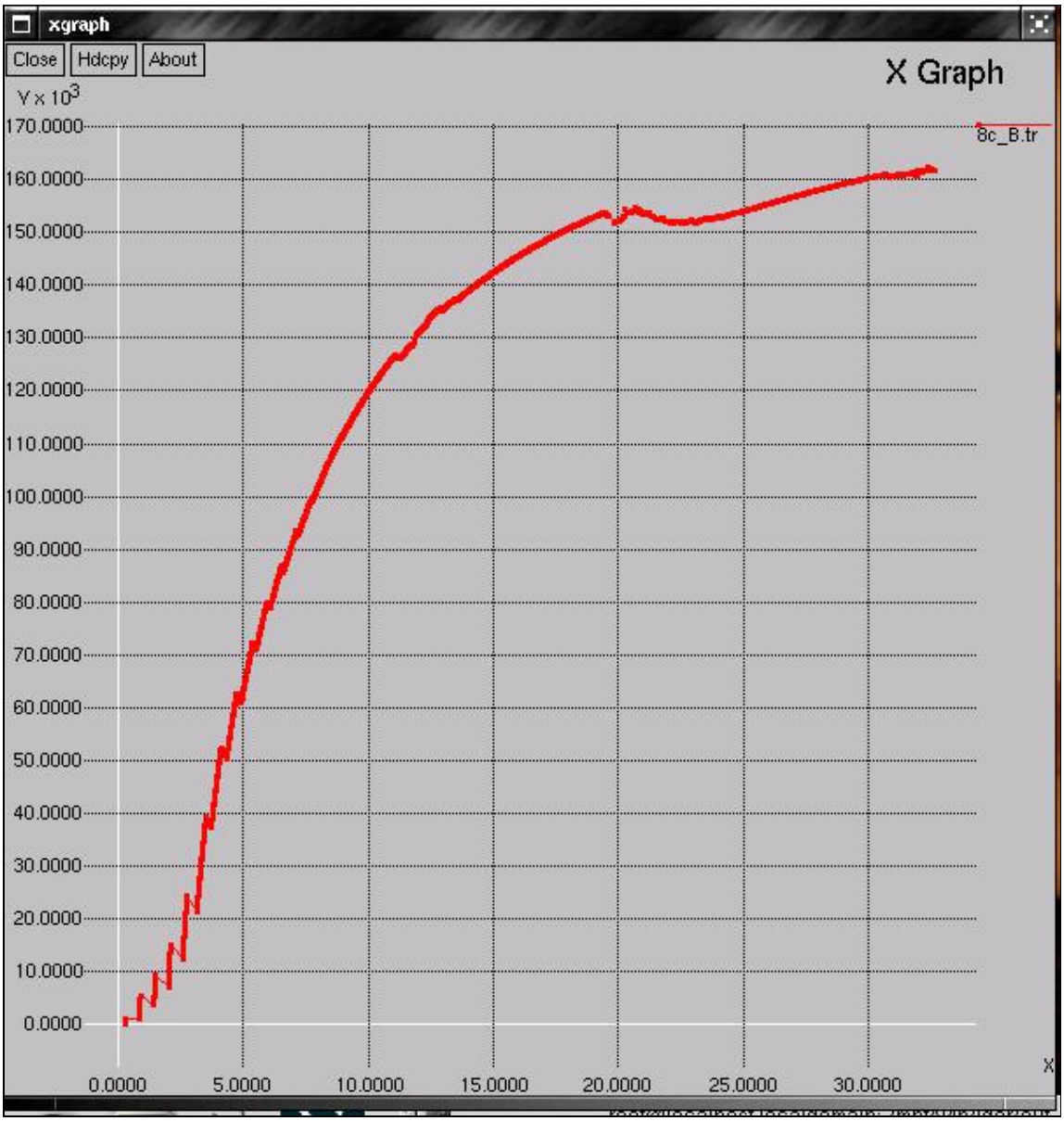


Figura 8.7 –*Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando oito conexões TCP.

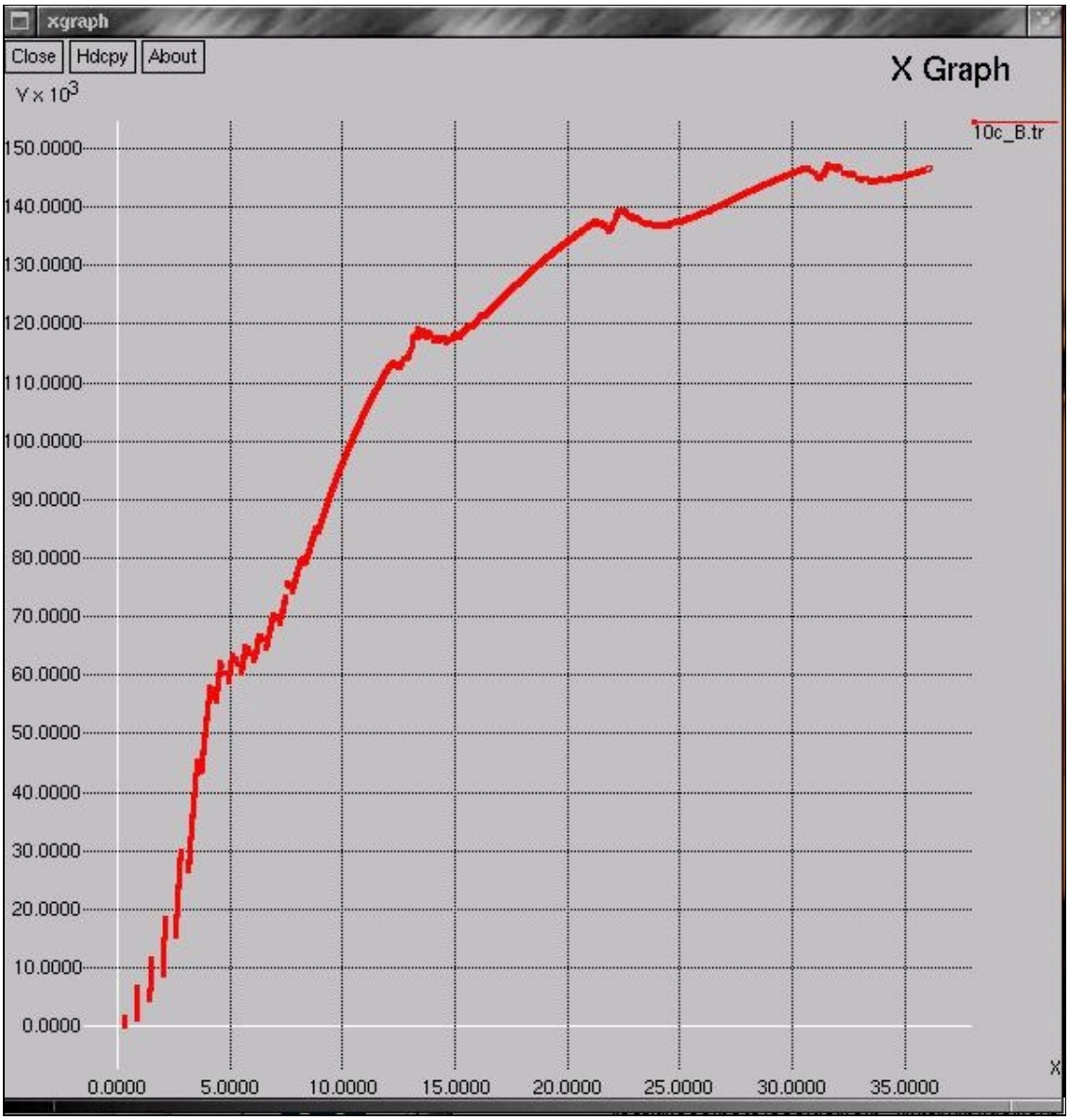


Figura 8.8 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando dez conexões TCP.

Observe que, ao utilizar 8, 10 ou 12 conexões paralelas (ver figuras 8.7, 8.8 e 8.9), o gráfico do *throughput* apresenta certas “irregularidades” em seu traçado. Estas irregularidades representam perdas maciças de pacotes em algum nó intermediário da rede (roteador), em função da sobrecarga no consumo de memória para enfileiramento de pacotes, já que a rede apresenta-se congestionada devido ao grande número de pacotes que estão sendo transmitidos e reconhecidos (*ACKed*).

Figura 8.9 – *Throughput* (em *bytes*/s) obtido através da transmissão de um arquivo de 5 MB, utilizando doze conexões TCP.

Novamente, utilizando o Trace Graph [31] foi utilizado para a comprovação da análise, a fim de termos quantitativamente esse valor das perdas, apresentamos as figuras 8.10 e 8.11, mostrando as estatísticas da transmissão para as transferências de 8 e 12 conexões, respectivamente. Note que, no caso de 8 conexões, o número de pacotes perdidos foi 40.

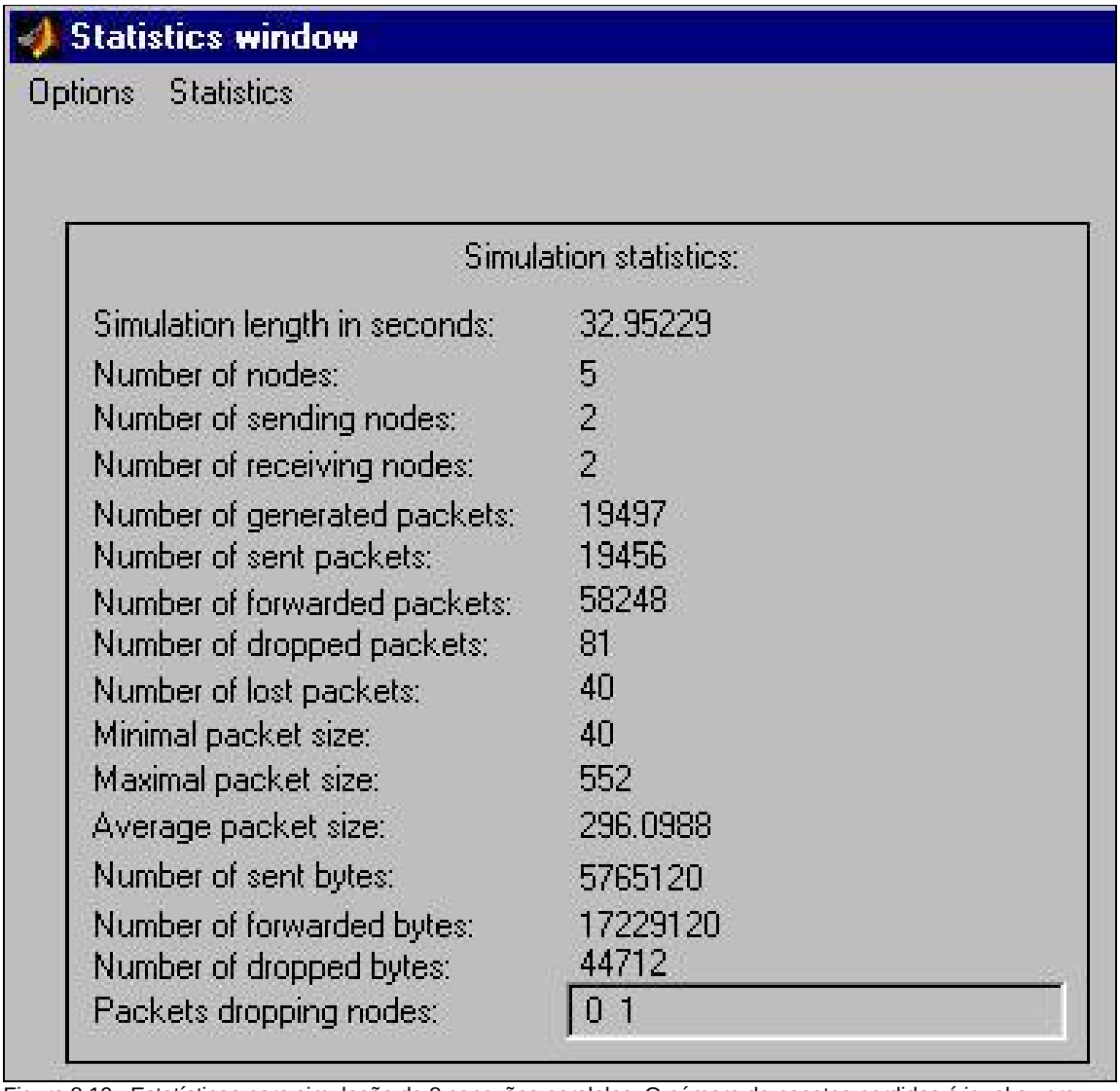


Figura 8.10– Estatísticas para simulação de 8 conexões paralelas. O número de pacotes perdidos é igual a zero

Já no caso de 12 conexões (ver Figura 8.11), o número de pacotes perdidos aumentou para 109 e o número de pacotes descartados pelos roteadores intermediários da rede foi para 177. Com isso, acrescentando algumas conexões, percebe-se uma maior perda de pacotes, por causa do algoritmo do Slow Start, conforme já dito em 6.1.3 .

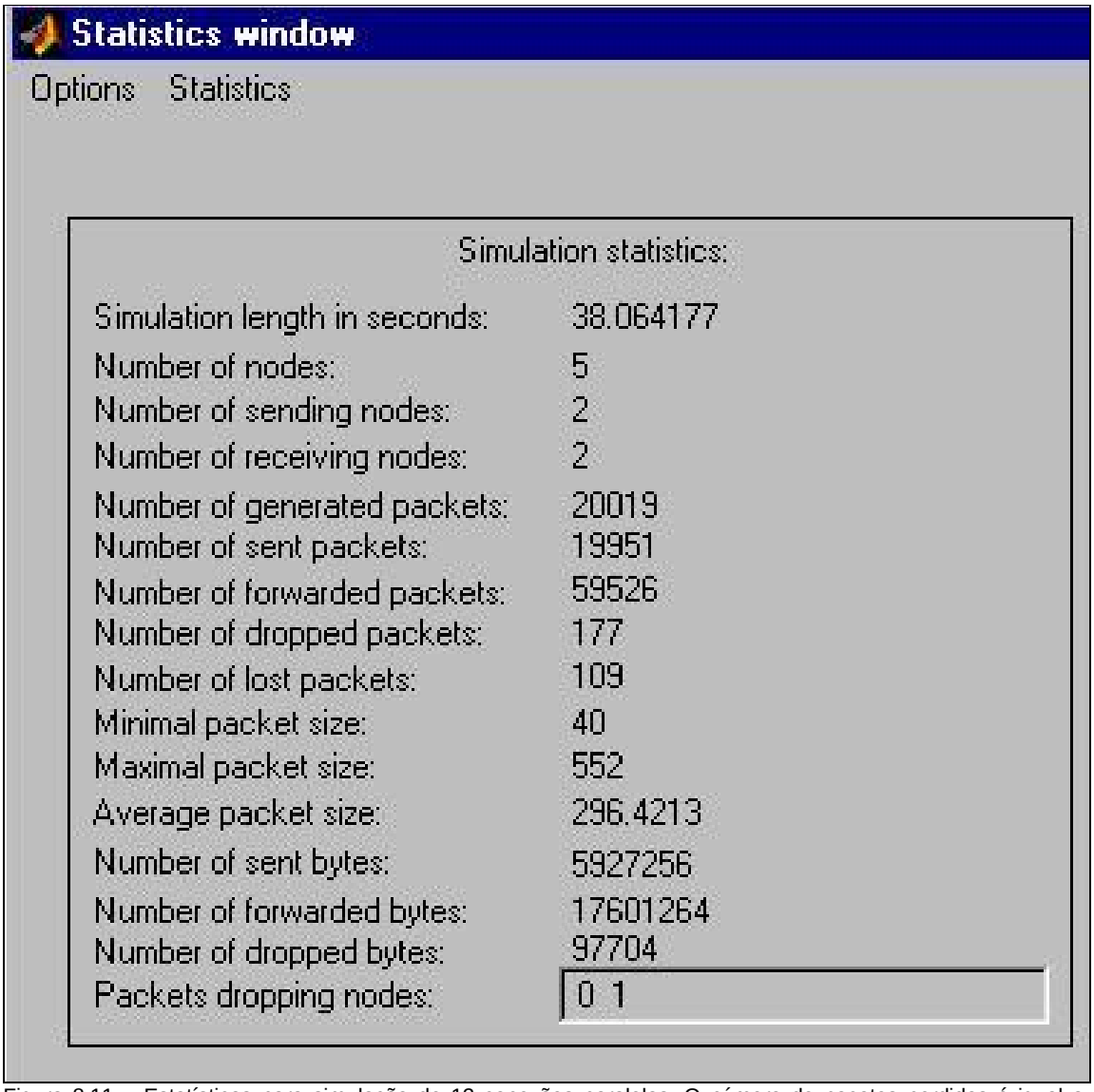


Figura 8.11 – Estatísticas para simulação de 12 conexões paralelas. O número de pacotes perdidos é igual a 109.

Na medida em que temos mostrado os gráficos do *throughput* para cada número de conexões em separado, já podemos mostrar um gráfico conjunto do *throughput* (em kB/s) em função do número de conexões utilizadas para transferir nosso arquivo de 5 MB, a fim de apresentarmos algo nas mesmas proporções do mostrado nas Figura 6.6 e Figura 6.7. A Figura 8.12 faz um resumo de todos os nossos resultados.

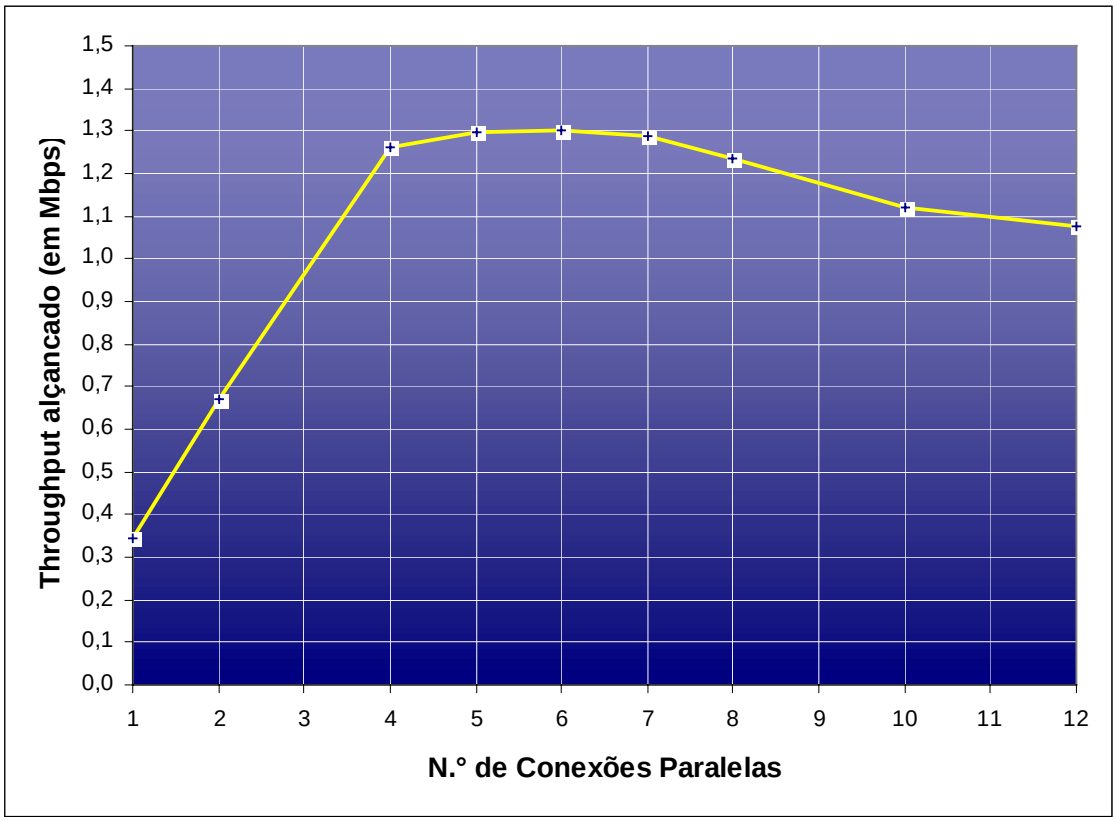


Figura 8.12 – *Throughput* (em Mbps/s) em função do número de conexões utilizadas na transferência de um arquivo de 5 MB, utilizando o *layout* de rede mostrado na Figura 8.1.

A figura acima, ao ser comparada com a Figura 6.7, mostra alguns detalhes interessantes. No estudo experimental de [14] (ver Figura 6.7), observa-se que o máximo *throughput* na transmissão de um arquivo de 5 MB foi alcançado utilizando 8 conexões paralelas, sendo que com 10 conexões, observou-se uma grande perda de segmentos e o *throughput* caiu vertiginosamente. Em nossos experimentos, entretanto, e de acordo com a Figura 8.13, obtivemos um máximo *throughput* ao utilizarmos 6 conexões. Além disso, através nossos resultados, fica claro que as perdas de pacotes nos roteadores intermediários vão se tornado maciças já para 8 conexões utilizadas, diferentemente dos resultados de [14]. O que se observa é que nossos resultados apresentaram-se mais precoces dos que os obtidos por [14]. Novamente, tem-se um máximo de *throughput* para 6 conexões, ao contrário das 8 conexões encontradas por [14], e observa-se um descaimento do *throughput* para 8 conexões paralelas, ao passo que [14] observaram esse descaimento a partir de 10 conexões.

Isso pode ser facilmente explicado pelo fato de não termos utilizado exatamente os 24 kB (24.576 *bytes*) para a janela do FTP. Conforme já explicado no

início deste capítulo, utilizamos uma janela de 48 pacotes, contendo 552 *bytes* cada um. Isso gera uma janela completa de 26.496 *bytes*, 7% maior do que a janela utilizada teoricamente. Dessa forma, como existem mais *bytes* sendo transmitidos na rede (isto é, um tráfego maior sendo gerado), espera-se que se chegue à máxima capacidade de transmissão mais rapidamente, da mesma forma que se espera uma queda significativa no *throughput* para um número de conexões menor do que o observado por [14].

Isso também explica o fato de termos alcançado *throughputs* numericamente maiores do que os obtidos por Fritz-Dolores em seu trabalho [14]. Como temos mais *bytes* trafegando na rede, presume-se que teremos valores maiores para as máximas utilizações conseguidas durante as transferências. Além disso, como visto na seção 6.1.1, utilizando 5 conexões alcançaríamos a utilização máxima dos links de nosso satélite. Contudo, o número ótimo encontrado para o máximo *throughput* foi de 6 conexões. Essa diferença é devida à capacidade de enfileiramento dos roteadores intermediários que, com isso, conseguem armazenar vários pacotes para posterior transmissão. Entretanto, com 8 conexões, observa-se uma sobrecarga dessa capacidade dos roteadores e uma conseqüente perda de segmentos.

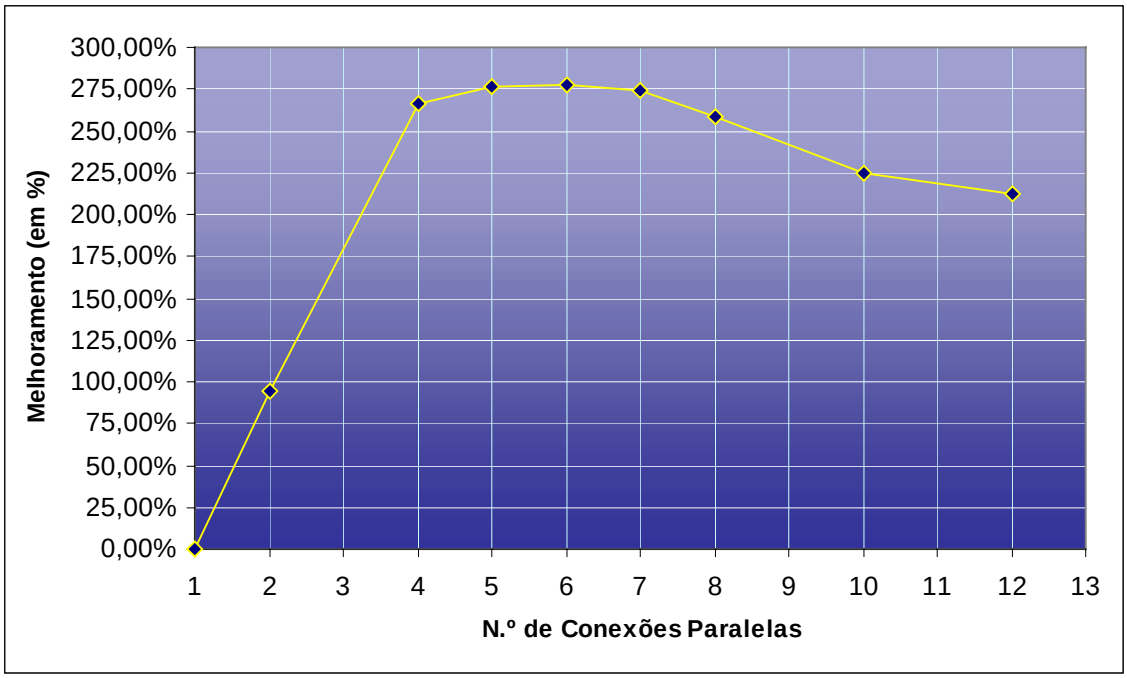


Figura 8.13 – Melhoramento do *throughput* (em %) obtido em função do número de conexões utilizadas na transferência de um arquivo de 5 MB, em comparação com a transferência usando 1 conexão.

Apresentamos, também, a Figura 8.13, que leva em consideração o percentual de melhoramento no aumento do número de conexões obtido em comparação com a transferência utilizando 1 conexão apenas.

A Figura 8.13 mostra que, tomando-se por base o *throughput* obtido para se transferir nosso arquivo através de uma conexão, ao utilizar duas conexões paralelas, temos um melhoramento de praticamente 100%. Com quatro conexões, obteve-se um melhoramento de mais de 250%, isto é, o *throughput* é melhorado por mais de 3,5 vezes. Com dez e com doze conexões, entretanto, o *throughput* é melhorado em pouco mais de 200% somente, comprovando o fato de que o aumento do número de conexões não é diretamente proporcional à taxa de utilização obtida.

Agora, já que apresentamos nossos resultados, fixando o tamanho do arquivo em 5 MB e variando o número de conexões, podemos apresentar uma tabela contendo os valores detalhados em cada transmissão, a fim de termos uma ferramenta para melhor comparação. A Tabela 8.1 mostra todos os resultados obtidos.

Nº de Conexões	***Bytes* recebidos**	**Duração (segundos)**	***Throughput* (k*bytes*/seg)**
1	5254528	116,3091	45,1773
2	5262848	59,9277	87,8199
4	5292856	32,0513	165,3057
5	5114304	31,2648	169,9772
6	5307168	31,0766	170,7766
7	5285544	31,3157	168,9102
8	5285720	32,6715	161,7835
10	5301256	36,1483	146,65276
12	5332248	37,7834	141,1266

Tabela 8.1 – A tabela acima mostra os resultados conjuntos levando-se em conta todos os números de conexões paralelas utilizadas.

8.2 Transferência de Arquivos de Tamanhos Diferentes

Além dos experimentos de análise do *throughput* em função do número de conexões paralelas empregadas na transferência de nosso arquivo de 5 MB, foram realizados também outros testes, a fim de verificar qual o comportamento do *throughput* na transmissão de arquivos de tamanhos variados em função do tamanho da janela inicial empregada, fixando o número de conexões para 1.

Esses testes foram baseados numa alteração no mecanismo de controle de congestionamento Slow Start (ver seção 2.2.3.1), que, ao invés de iniciar *cwnd* (*Congestion Window*) com 1 segmento, o ajustamos para começar a partir de um valor específico, maior que 1, a fim de comprovarmos que o uso de uma janela inicial maior beneficia o *throughput* alcançado na transmissão.

Dessa forma, através da mesma topologia de rede (ver Figura 8.1), foram transmitidos arquivos de 30 kB, 100 kB, 200 kB, 1 MB e 5 MB, e as janelas iniciais foram ajustadas para 1, 4, 10 e 32 pacotes para cada tamanho de arquivo transmitido. Não podíamos utilizar uma janela inicial que fosse superior ao tamanho da janela de anúncio do receptor, que foi de 48 pacotes, conforme já explicado, para evitar que houvesse truncamento e os resultados saíssem diferentes dos esperados. Dessa forma, utilizamos um tamanho de janela inicial para *cwnd* de no máximo 32 pacotes, acreditando que isso já nos trouxesse bons resultados.

As simulações foram realizadas, é claro, separadamente uma das outras, mas nos concentraremos em mostrar o resultado conjunto das transmissões de modo a facilitar o entendimento da comparação a ser feita pelo leitor. A Figura 8.14 mostra esses resultados para as transferências dos arquivos de tamanhos distintos, apresentando o *throughput* de cada transferência em função do tamanho inicial de *cwnd*.

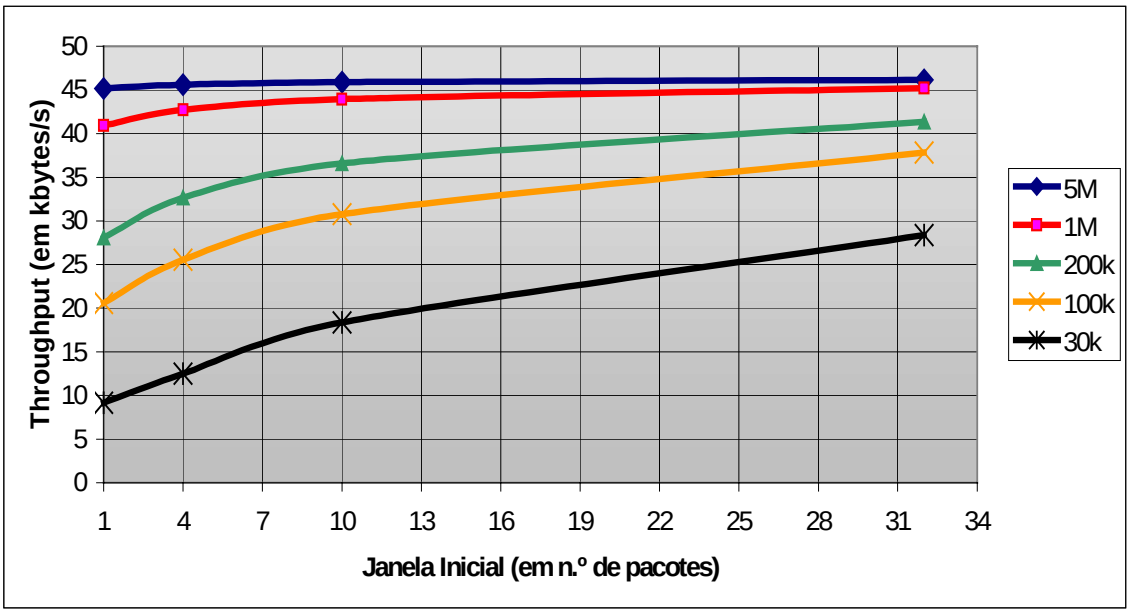


Figura 8.14 – *Throughput* obtido em função do tamanho da janela inicial ajustada para as transferências de arquivos de tamanhos distintos.

De acordo com a Figura 8.14 fica claro que as transferências menores, isto é, de arquivos de tamanho menor, sofrem um benefício maior quando do aumento do tamanho da janela inicial. A transferência do arquivo de 30 kB, por exemplo, começa com um *throughput* de 10 kB para uma janela inicial de 1 segmento[20] e chega a aproximadamente 28 kB, quando a janela inicial é de 32 segmentos, um aumento de 150% no *throughput*. O mesmo não ocorre com a transmissão de arquivos grandes, como o de 5 MB. O *throughput* para uma janela inicial de 1 segmento praticamente não se altera quando a janela inicial é ajustada para 32 pacotes.

Isso pode ser explicado facilmente pelo fato de que quando se têm arquivos de tamanho consideravelmente grandes, a transmissão é longa o suficiente para que o valor de *cwnd* chegue no seu limite (valor da janela de anúncio do receptor), fazendo com que o TCP reduza sua taxa de transmissão e inicie novamente o valor da janela inicial para 1 segmento. No caso da transferência de um arquivo de tamanho relativamente pequeno, o tempo de transmissão é reduzido, fazendo com que *cwnd* aumente sem alcançar o seu limite, permitindo um alto crescimento no *throughput* da transmissão.

Da mesma forma que na seção 8.1, mostraremos também o resultado desses testes em percentual, de modo a deixar ainda mais claro qual transferência mais se beneficiou com o aumento do valor inicial da janela deslizante (ou *cwnd*).

A Figura 8.15 mostra esse o melhoramento do *throughput* (em %) em função do valor inicial (em pacotes) da janela inicial utilizada.

[20] 1 segmento = 1 pacote.

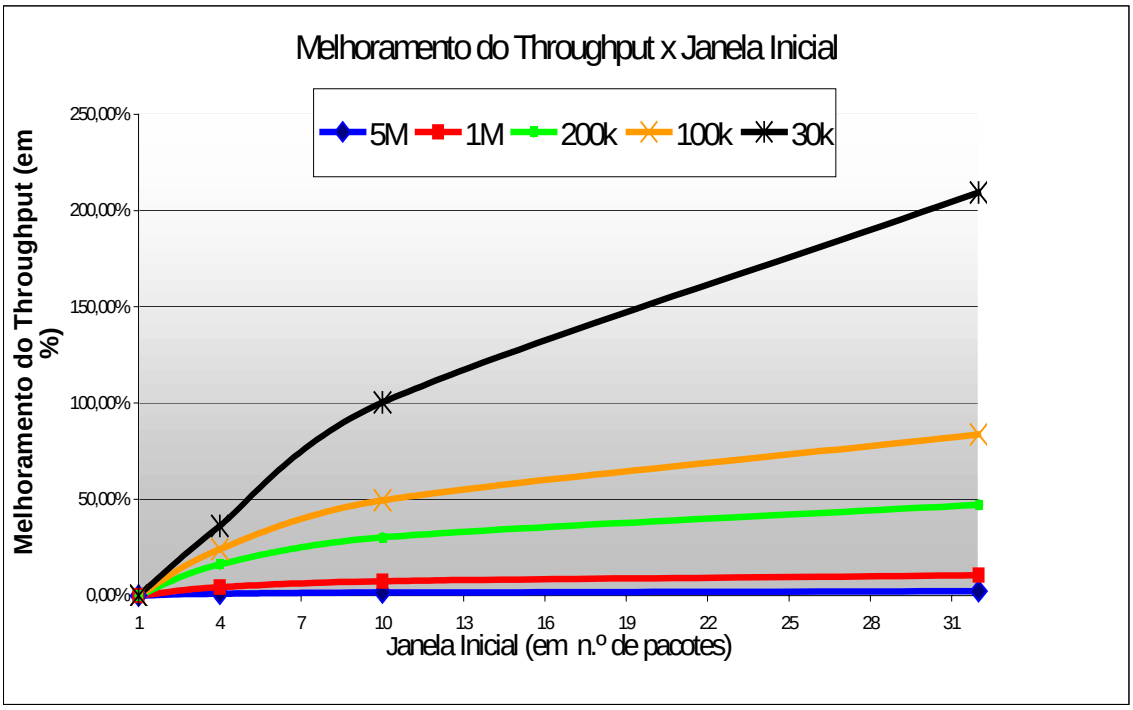


Figura 8.15 – Melhoramento do *throughput* (em %) em função do tamanho da janela inicial utilizada.

Através da Figura 8.15 fica claro que, como dito na análise do gráfico da Figura 8.14, a transferência que mais se beneficiou com o aumento do tamanho da janela inicial foi a do arquivo de 30 kB, em oposição ao arquivo de 5 MB que continuou praticamente com o mesmo *throughput* mesmo com o aumento do tamanho da janela inicial.

Através dos resultados apresentados neste capítulo, já se pode tirar algumas conclusões quanto à utilização de múltiplas conexões TCP e quanto ao ajuste do tamanho da janela inicial utilizada para um determinado tamanho de arquivo a ser transmitido. Essas conclusões são detalhadas no capítulo seguinte, de forma a fazermos um desfecho consistente do que foi apresentado neste documento.

9. CONCLUSÃO

Neste trabalho foi mostrada uma versão modificada do FTP que melhora a utilização dos canais de satélite. Contudo, nós sentimos que a solução em nível de aplicação, o MFTP, é muito agressiva para ser usada em redes de propósito geral. Além disso, tal solução requer que modificações sejam feitas em todas aplicações e serviços para conseguir uma completa utilização sobre os *link*s de satélite. Contudo, os experimentos MFTP apresentados nos capítulo 6 e nossa comprovação mostrada no capítulo 8 deram boas noções a respeito das modificações no TCP que ajudam a utilizar melhor os circuitos de satélite.

9.1 Janelas Largas

Além disso, foi mostrado neste documento (seções 1.2.2, 2.1 e 2.4.1) que para o TCP utilizar completamente a capacidade disponível dos canais de satélite, extensões do TCP para janelas maiores fazem-se necessárias [67], já que o valor de *cwnd* limita a quantidade de dados não reconhecidos que um transmissor TCP pode injetar na rede, anunciando uma janela mais larga do que a permitida pelo padrão TCP, não devendo ser prejudicial à rede. Janelas de anúncio maiores têm sido mostradas com bons resultados em redes de satélite [36] e sobre redes terrestres de longas áreas [7].

9.2 Modificações no Slow Start

Como mostrado no capítulo 8, modificar o algoritmo Slow Start pode fornecer um incremento no *throughput* quando da transferência de arquivos em canais de satélite. Além disso, nossos testes mostram pequena desvantagem no uso desses mecanismos no ambiente de satélite para arquivos relativamente grandes. Contudo, esses mecanismos incrementam agressividade no TCP e nossos testes não incluíram disputa de tráfego. Por isso, estudos mais aprofundados tornam-se

necessários para garantir que essas modificações não terão um impacto negativo numa rede compartilhada, como a Internet.

9.3 Modificações no Congestion Avoidance

O MFTP também sugere que seja necessário favorecer o estudo sobre o Congestion Avoidance. Como explicado na seção 2.2.3.1, o Congestion Avoidance requer uma quantidade grande de tempo para incrementar o tamanho da janela sobre os canais de satélite devido ao longo RTT. O MFTP é capaz de multiplicar esse incremento pelo número de conexões sendo usadas (isto é, cada conexão de dados pode incrementar a janela efetiva de 1 segmento). Trabalho futuros nessa área podem tentar incluir alguma mudança no algoritmo Congestion Avoindance para levar em conta o RTT quando incrementar o *cwnd* ao invés de simplesmente usar o número de RTTs que passaram desde a última ocorrência de uma perda.

9.4 Estimação do Ponto de Parada do Slow Start

Finalmente, o MFTP sugere que também é necessário estudar a escolha quando o TCP termina o Slow Start. Perdas maciças acontecem quando o MFTP utiliza muitas conexões de dados porque a janela deslizante efetiva sobrecarrega o roteador intermediário. O TCP poderia evitar a maioria dessas perdas, enquanto estivesse mantendo uma boa utilização, se ele pudesse terminar o Slow Start justamente antes da perda ocorrer. A janela requerida pode ser computada usando a equação 6.3. O TCP já observa o RTT, mas necessitaria estimar a largura de banda disponível do *link* congestionado para computar a janela requerida. Uma vez feita uma estimação, a janela necessária poderia ser determinada à variável *ssthresh* (a qual determina quando será terminado o Slow Start).

9.5 ACKs Seletivos

A necessidade por SACKs foi mostrada na seção 2.4.2. Os SACKs permitem o receptor reportar precisamente quais segmentos chegaram. Essa informação explícita permite o transmissor retransmitir os segmentos que não foram recebidos. O transmissor é capaz de quase eliminar a retransmissão dos segmentos que foram entregues com sucesso pelo uso de uma estratégia de retransmissão deficiente (isto é, usando Slow Start para retransmitir todos os segmentos que não têm sido reconhecidos (*ACKed*) cumulativamente). Num estudo amigável, o uso de SACKs foi mostrado para ser efetivo no ambiente de satélites. Em adição, os SACKs têm sido mostrados para ser benéficos em quantidades variantes de perdas em redes terrestres simuladas [28].

9.6 Novos Mecanismos de Recuperação de Perdas

Melhoramentos no algoritmo Fast Recovery que fazem retransmissões de dados mais efetivamente também têm sido sugeridos atualmente. Hoe [21] introduziu um mecanismo no lado do transmissor chamado de “Fase de Rápida Recuperação” (*Fast Recovery Phase*). Esse algoritmo é usado na momento em que um segmento é retransmitido usando o algoritmo Fast Retransmit até que todas as perdas tenham sido recuperadas. Já na Fase de Rápida Recuperação, o TCP usa indícios de ACKs entrantes para determinar quais segmentos não foram entregues e, portanto, precisam ser retransmitidos. Esse mecanismo trabalha melhor do que o TCP padrão com Fast Recovery, mas não tão bem quanto o TCP com SACKs [28]. Uma vantagem desse mecanismo sobre os SACKs é que ele só precisa ser implementado no lado do transmissor da conexão TCP, ao invés dos SACKs que necessitam de implementação em ambos os lados da transmissão.

9.7 Conclusões

A solução em nível de aplicação estuda e apresentada por [14], bem como o nosso trabalho de comprovação da técnica de múltiplas conexões, mostram que extensões TCP provendo janelas largas e ACKs seletivos permitem o TCP conseguir boa performance sobre canais de satélite. Além disso, a solução em nível de aplicação sugeriu que mudanças nos algoritmos Slow Start e Congestion Avoidance poderiam também melhorar a performance. Os experimentos apresentados neste documento verificaram que modificações no Slow Start melhoraram a utilização da capacidade do canal. As extensões que nós testamos, bem como o trabalho citado de outros pesquisadores, permitiram que o protocolo TCP utilizasse totalmente os canais de satélite de longo atraso.

Dessa forma, pela observância dos testes e resultados apresentados, percebe-se que para conseguir bons melhoramentos da utilização da capacidade dos *links* de satélite são necessários estudos de pesquisa dedicados e aprofundados. Contudo, atualmente já se tem conseguido boas taxas de transmissão sobre canais de longo atraso, fazendo com que conexões paralelas com janelas largas e melhoramentos da capacidade de memória e processamento dos roteadores intermediários, permitindo que redes baseadas em TCP/IP como a Internet estejam cada vez mais próximas das localidades, cuja cobertura e acesso são estritamente por meio de satélites.

10. BIBLIOGRAFIA

[1] A. BAKRE AND B. R. BADRINATH. *I-TCP: Indirect TCP for Mobile Hosts. In Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS), May 1995.*

[2] AGËNCIA NACIONAL DE TELECOMUNICAÇÕES – ANATEL. URL: http://www.anatel.com.br acessada em 20 de agosto de 2002.

[3] B. KANTOR AND P. LAPSLEY. *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News, February 1986. RFC 977.*

[4] BLACK, UYLESS. *ATM Volume I: Foundation for Broadband Networks 2/e © 1999*

[5] BORLAND SOFTWARE CORPORATION ©. URL: http://www.borland.com acessada em 20 de agosto de 2002.

[6] COMMER, DOUGLAS E. *Internetworking with TCP/IP, Vol. I: Principles, Protocols, and Architectures Prentice Hall, 1995 (3rd Ed).*

[7] CURTIS VILLAMIZAR AND CHENG SONG. *High Performance TCP in ANSNET. Computer Communications Review, 24(5):45-60, October 1995.*

[8] DAVID CHERITON. *VMTP: Versatile Message Transaction Protocol, February 1988. RFC 1045.*

[9] DAVID CLARK, MARK LAMBERT, AND LIXIA ZHANG. *NETBLT: A High Throughput Transport Protocol. In ACM SIGCOMM, pages 353-359, August 1987.*

[10] DIAS, ROBERTO ALEXANDRE. TUTORIAL MPLS – UMA ARQUITETURA PARA SUPORTE A QoS EM REDES IP. Universidade Federal de Santa Catarina, setembro 2000.

[11] DOUGLAS E. COMER. *Internetworking with TCP/IP, Volume I, Principles, Protocols, and Architecture. Prentice Hall, 3rd edition, 1995.*

[12] EMPRESA BRASILEIRA DE TELECOMUNICAÇÕES – EMBRATEL S/A URL: http://www.embratel.com acessada em 18 de agosto de 2002.

[13] ENIO ROBOREDO SANCHES E MARCELO PANDOLFI BARCELLOS. Tecnologias de Videoconferência. Departamento de Eng. Elétrica, Universidade de Brasília, 1999.

[14] FRITZ J. AND DOLORES H. RUSS. *Improving TCP Performance Over Satellite Links. A thesis presented to The Faculty of the College of Engineering and Technology, Ohio University, June 1997.*

[15] GARY R. WRIGHT AND W. RICHARD STEVENS. *TCP/IP Illustrated Volume II: The Implementation. Addison-Wesley, 1995.*

[16] HANS KRUSE. *Performance Of Common Data Communications Protocols Over Long Delay Links: An Experimental Examination. In 3rd International Conference on Telecommunication Systems Modeling and Design, 1995.*

[17] HARI BALAKRISHNAN, SRINIVASAN SESHAN, ETAN AMIR, AND RANDY KATZ. *Improving TCP/IP Performance Over Wireless Networks. In ACM MobiCom, November 1995.*

[18] ERTZ, HEINRICH. *Istitute Berlim. URL: http://www.hhi.de/ acessada em 18 de agosto de 2002.*

[19] IETF - THE INTERNET ENGINEERING TASK FORCE. URL: http://www.ietf.org/

[20] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION - ISO – ISO 8248

[21] JANEY C. HOE. *Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In ACM SIGCOMM, August 1996.*

[22] JOHN NAGLE. *Congestion Control in IP/TCP Internetworks, January 1984. RFC 896.*

[23] JOHN NAGLE. *Congestion Control in IP/TCP Internetworks. Computer Communication Review, 14(4), October 1984.*

[24] JON POSTEL AND JOYCE REYNOLDS. *File Tranfer Protocol (FTP), October 1985. RFC 959.*

[25] JON POSTEL. *Simple Mail Transfer Protocol, August 1982. RFC 821*.

[26] JON POSTEL. *Transmission Control Protocol, September 1981. RFC 793*.

[27] JUNIPER NETWORKS. Supporting Differentiated Service Classes: TCP Congestion Control Mechanisms URL: http://www.juniper.net/techcenter/techpapers/200022.html acessada em 20 de agosto de 2002.

[28] KEVIN FALL AND SALLY FLOYD. *Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. Computer Communications Review, July 1996*.

[29] LINUX.COM. URL: http://www.linux.com acessada em 20 de agosto de 2002.

[30] LU, BHARGHAVAN. *Adaptive resource management for indoor mobile computing environments - 1996*

[31] MALEK, JAROSLAW ©. *Trace Graph. URL: http://www.geocities.com/tracegraph*

[32] MARK ALLMAN AND SHAWN OSTERMANN. *FTP Extensions for Variable Protocol Speci_cation, March 1997. Internet-Draft draft-allman-ftp-variable-04.txt (work in progress).*

[33] MARK ALLMAN AND SHAWN OSTERMANN. *Multiple Data Connection FTP Extensions. Technical Report TR-19971, Ohio University Computer Science, February 1997.*

[34] MARK ALLMAN, ADAM CALDWELL, AND SHAWN OSTERMANN. *ONE: The Ohio Network Emulator. Technical Report TR-19972, Ohio University Com puter Science, June 1997.*

[35] MARK ALLMAN, CHRIS HAYES, AND SHAWN OSTERMANN. *An Evaluation of TCP Slow Start Modi_cations, 1997. In preparation.*

[36] MARK ALLMAN, CHRIS HAYES, HANS KRUSE, AND SHAWN OSTERMANN. *TCP Performance Over Satellite Links. In Proceedings of the 5th Interna tional Conference on Telecommunication Systems, March 1997.*

[37] MATT MATHIS AND JAMSHID MAHDAVI. *TCP Rate-Halving with Bounding Parameters. Technical report, Pittsburgh Supercomputer Center, October 1996. URL:http://www.psc.edu/networking/papers/FACKnotes/current/.*

[38] MATTHEW MATHIS AND JAMSHID MAHDAVI. *Forward Acknowledgment: Re_ning TCP Congestion Control. In ACM SIGCOMM, August 1996.*

[39] MATTHEW MATHIS, JAMSHID MAHDAVI, SALLY FLOYD, AND ALLYN ROMANOW. *TCP Selective Acknowledgement Options, October 1996. RFC 2018.*

[40] MICROSOFT CORPORATION ©. URL: http://www.microsoft.com acessada em

20 de agosto de 2002.

[41] MIKAEL DEGERMARK, MATHIAS ENGAN, BJORN NORDGREN, AND STEPHEN PINK. *Low-Loss TCP/IP Header Compression forWireless Networks. In ACM MobiCom, November 1996.*

[42] NASA WTEC. *Global Satellite Communications Technology and Systems http://www.itri.loyola.edu/satcom2/04_05.htm acessado em 23 de julho de 2002.*

[43] NETWORK SIMULATOR. URL: http://www.isi.edu/nsnam acessada em 20 de agosto de 2002.

[44] NETWORK WORKING GROUP. RFC 1122 - URL: http://www.w3.org/Protocols/rfc1122/Overview.html acessada em 10 de maio de 2002.

[45] NETWORK WORKING GROUP. RFC 1633 - URL: http://www.w3.org/Protocols/rfc1633/Overview.html acessada em 14 de maio de 2002.

[46] NETWORK WORKING GROUP. RFC 2205 - URL: http://www.w3.org/Protocols/rfc2205/Overview.html acessada em 10 de maio de 2002.

[47] NETWORK WORKING GROUP. RFC 2460 - URL: http://www.w3.org/Protocols/rfc2460/Overview.html acessada em 23 de julho de 2002.

[48] NETWORK WORKING GROUP. RFC 2475 - URL: http://www.w3.org/Protocols/rfc2475/Overview.html acessada em 10 de junho de 2002.

[49] NETWORK WORKING GROUP. RFC 2488 - URL: http://www.w3.org/Protocols/rfc2488/Overview.html acessada em 30 de maio de 2002.

[50] NETWORK WORKING GROUP. RFC 791 - URL: http://www.w3.org/Protocols/rfc791/Overview.html acessada em 10 de maio de 2002.

[51] NETWORK WORKING GROUP. RFC 959 - URL: http://www.w3.org/Protocols/rfc959/Overview.html acessada em 05 de maio de 2002.

[52] PHIL KARN AND CRAIG PARTRIDGE. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In ACM SIGCOMM, pages 2-7, August 1987.

[53] R. BAUER AND T. VONDEAK. *Advanced Communications Technology Satellite (ACTS) and Experiments Program Descriptive Overview. Technical report, NASA Lewis Reseach Center, 1991.*

[54] R. FIELDING, JE_REY C. MOGUL JIM GETTYS, H. FRYSTYK, AND TIM BERNERS-LEE. *Hypertext Transfer Protocol - HTTP/1.1, January 1997. RFC 2068.*

[55] ROBERT BRADEN, DAVID CLARK, JON CROWCROFT, BRUCE DAVIE, STEVE DEERING, DEBORAH ESTRIN, SALLY FLOYD, VAN JACOBSON, GREG MINSHALL, CRAIG PARTRIDGE, LARRY PETERSON, K. RAMAKRISHNAN, S. SHENKER, J. WROCLAWSKI, AND LIXIA ZHANG. *Recommendations on Queue Management and Congestion Avoidance in the Internet, March 1997. InternetDraft draft-irtf-e2e-queue-mgt-00.txt (work in progress).*

[56] ROBERT BRADEN, VAN JACOBSON, AND LIXIA ZHANG. *TCP Extension for High-Speed Paths, October 1990. RFC 1185.*

[57] ROBERT BRADEN. *Requirements for Internet Hosts Communication Layers, October 1989. RFC 1122.*

[58] SALLY FLOYD AND KEVIN FALL. *Router Mechanisms to Support End-to-End Congestion Control. Technical report, LBL, February 1997. Submitted to SIGCOMM.*

[59] SALLY FLOYD AND VAN JACOBSON. *Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, 1(4):397-413, August 1993.*

[60] SALLY FLOYD, *February 1997. Note to end2end-interest mailing list.*

[61] SRINIVASAN KESHAV. *A Control Theoretic Approach to Flow Control. In ACM SIGCOMM, pages 3-15. SIGCOMM, ACM, September 1991*.

[62] THE MATHWORKS, INC. URL: http://www.mathworks.com acessada em 20 de agosto de 2002.

[63] TIM BERNERS-LEE, R. FIELDING, AND H. NIELSEN. *Hypertext Transfer Protocol - HTTP/1.0, May 1996. RFC 1945.*

[64] UNIVERSAL INTERNET EXCHANGE, UNIX. URL: http://www.unix.com acessada em 20 de agosto de 2002.

[65] VAN JACOBSON AND MICHAEL J. KARELS. *Congestion Avoidance and Control. In ACM SIGCOMM, 1988.*

[66] VAN JACOBSON AND ROBERT BRADEN. *TCP Extensions for Long-Delay Paths, October 1988. RFC 1072.*

[67] VAN JACOBSON, ROBERT BRADEN, AND DAVID BORMAN. *TCP Extensions for High Performance, May 1992. RFC 1323.*

[68] VAN JACOBSON. *Compressing TCP/IP Headers For Low-Speed Serial Links, February 1990. RFC 1144.*

[69] VAN JACOBSON. *Modi_ed TCP Congestion Avoidance Algorithm. Technical report, LBL, April 1990. Email to the end2end-interest mailing list. URL: http://ftp.ee.lbl.gov/email/vanj.90apr30.txt.*

[70] VERN PAXSON. *End-to-End Internet Packet Dynamics. In ACM SIG-COMM, September 1997. To Appear.*

[71] W. RICHARD STEVENS. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. RFC 2001. [VHS84] D. Velten, Robert Hinden, and J. Sax. Reliable Data Protocol, July 1984. RFC 908.*