Universidade Federal de Santa Catarina
Sistemas de Informação

Análise do Comportamento do Protocolo TCP em Redes de Alta Velocidade

Acadêmica: Renata Santos de Souza
Orientador: Carlos Becker Westphal
Co-Orientador: Edison Tadeu Lopes Melo
Banca examinadora: Rafael Righi

Trabalho apresentado ao Departamento de Sistemas de Informação da Universidade Federal de Santa Catarina, como parte dos requisitos para obtenção do título de graduado em Sistemas de Informação.

Florianópolis - Santa Catarina – Brasil
Novembro, 2006

Agradecimentos

Primeiramente, agradeço a Deus por minha vida e de minha família e por todas as oportunidades que ele tem me proporcionado.

Agradeço ao meu co-orientador, Edison Tadeu Lopes Melo, pela oportunidade de ter participado, na qualidade de bolsista, no desenvolvimento de projetos sob sua coordenação.

Agradeço aos amigos do NPD, especialmente a Augusto Castelan Carlson e a Fausto Vetter que estiveram sempre prontos a me ajudar e a esclarecer quaisquer dúvidas que surgissem, das mais simples às mais complexas.

Ao amigo Vilmar Longino Barborza Junior, pela paciência e apoio na fase final desde trabalho.

Além disso, eu gostaria de manifestar gratidão a minha família, em especial, aos meus pais, Antônio Luciano de Souza e Rosângela Santos de Souza, pois sem eles nada disso seria possível. E estendo esta gratidão as minhas irmãs, Roberta e Kamilla Santos de Souza, pela paciência e apoio.

Resumo

Com a chegada das redes de alta velocidade, a escolha de um protocolo de transporte é de vital importância para garantir o bom desempenho da rede. Atualmente, o TCP é o protocolo de transporte mais utilizado na internet, desta forma o bom desempenho da Internet está ligado à eficiência deste protocolo. Mas o mecanismo de controle de congestionamento, usado no protocolo TCP, tem dificuldade para utilizar, de maneira eficiente, os recursos disponíveis em redes de alta velocidade e longa distância. A presente pesquisa teve como propósito estudar o comportamento do protocolo TCP em redes de alta velocidade e longos atrasos, identificando os padrões e as deficiências do TCP. Uma visão geral do protocolo TCP é apresentada inicialmente. A seguir, são explicados os conceitos dos mecanismos de congestionamento, e as principais implementações do TCP, atualmente em uso. Simulações sobre as diferentes implementações são realizadas, a fim de identificar as peculiaridades de cada uma delas, quando submetidas a redes de alta velocidade e longas distâncias. Na conclusão deste trabalho, apresenta-se uma análise comparativa entre as implementações que foram utilizadas nesta pesquisa.

Palavras chaves: TCP, Controle de Congestionamento, Redes de Alta Velocidade, Desempenho.

Abstract

With the arrival of high speed networks, choose a transport protocol is of vital importance to guarantee a good network performance. Currently, TCP is the transport protocol most used in Internet, so the Internet performance is depends on the efficiency of that protocol. But the TCP congestion control mechanism can not use the high speed and long distance network resources efficiently. This research had as intention to study the behavior of TCP protocol in high speed and long distance networks with high delay, identifying the standards and the deficiencies of TCP. Initially, a global view of the TCP protocol is presented, followed by an explanation of the concepts of congestion mechanisms, and the main TCP implementations used today. Simulations using the different implementations are carried in order to identify the peculiarities of each one of them, when submitted in high speed and long distance networks. In the conclusion, a comparative analysis between the implementations used is presented.

Key words: TCP, Congestion Control, High speed Network, Performance.

Índice

Lista de Figuras

Abreviaturas

ACK Acknowledgment
AIMD Additive Increase Multiplicative Decrease
BDP Bandwidth Delay Product
BSD Berkeley Software Distribution
FIFO First In First Out
FTP File Transfer Protocol
CWND Congestion Window
DT DropTail
HIPPI High Performance Parallel Interface
HSTCP HighSpeed TCP
HTTP Hyper Text Transfer Protocol
MAX_SSTHRESH Maximum Slow-Start Threshold
MSS Maximum Segment Size
MTU Maximum Transfer Unit
NS-2 Network Simulator Version 2
RFC Request For Comments
RTO Retransmit Timer
RTT Round-Trip Time
SACK Selective Acknowledgement
SSTHRESH Slow-Start Threshold
SYN Synchronization Packet
TCP Transmission Control Protocol
XCP Explicit Control Protocol
WWW World Wide Web

Objetivo

A proposta geral deste trabalho foi estudar o comportamento das diferentes implementações do protocolo TCP em enlaces de alta velocidade e longa distância, como um mecanismo para transferência de grande volume de dados.

Para cumprir este objetivo geral, este trabalho realiza um estudo das diferentes implementações do TCP padrão. Este protocolo TCP é responsável pela detecção de erros, ordenação de segmentos, controle de fluxo e controle de congestionamento na rede de dados. Os diferentes mecanismos de controle de congestionamento do TCP são de grande importância para este estudo, pois estes mecanismos têm como objetivo otimizar a vazão, mantendo alta a eficiência de utilização do canal e ao mesmo tempo evitar que diversos fluxos concorrentes causem congestionamento na rede.

As implementações do TCP utilizadas no presente estudo são: Tahoe [17], Reno [18], New Reno [19], Vegas [24] e [25] e TCP SACK [20]. Para analisar o comportamento do protocolo TCP, em redes de alta velocidade, algumas destas implementações foram submetidas a simulações de rede, que tiveram como propósito identificar padrões e deficiências do protocolo TCP, quando opera em enlaces de alta velocidade e longas distâncias.

1 - Introdução

Uma das grandes questões na área de conectividade, atualmente, é a crescente demanda por largura de banda. Diversas tecnologias têm emergido e aumentado em muito a capacidade dos canais de comunicação. Atualmente, vários meios estão disponíveis para conectar dois pontos em alta velocidade, tanto para conexões locais, como para ligações de longa distância.

Com a chegada das redes de alta velocidade, diversas aplicações, tais como as aplicações multimídias distribuídas em tempo real e as aplicações de grade computacional de larga escala, estão sendo implantadas. Para a utilização dessas aplicações em redes de longa distância, como a Internet, a escolha de um protocolo de transporte adequado é de vital importância.

O TCP (Transmission Control Protocol) foi desenvolvido na década de 70 e, desde então, tem sido constantemente modificado para adaptar-se à novas aplicações [1], [2]. Além de acomodar-se às características particulares dos novos meios de transmissão, e melhorar sua capacidade de transferência de dados. Atualmente, o TCP é o protocolo de transporte mais utilizado na Internet. Aplicações populares como *Web*, FTP e correio eletrônico foram projetadas para operarem sobre este protocolo [3]. O desempenho fim-a-fim que os usuários esperam destes serviços, depende muito do desempenho do próprio TCP.

As questões referentes ao desempenho são muito importantes nas redes de computadores. Infelizmente, compreender o desempenho de uma rede é mais uma arte que uma ciência. Alguns problemas de desempenho, como o congestionamento, são causados pela sobrecarga temporária de recursos, ou seja, se um roteador receber um tráfego maior do que é capaz de manipular, haverá um congestionamento e uma queda de desempenho. Outro problema referente ao desempenho é quando há um desequilíbrio nos recursos estruturais. Este tipo de dificuldade ocorre quando um *host* não é capaz de tratar os segmentos com a mesma rapidez com que eles são recebidos da rede, causando perda de segmentos [2].

As redes de alta velocidade trazem com elas novos problemas de desempenho relacionados ao protocolo TCP, já que, este protocolo opera com confirmação de segmentos. Isto significa que um *host* transmissor ao enviar uma quantidade de segmentos, só enviará uma nova quantidade depois de receber a confirmação dos segmentos enviados. Portanto, para atingir a velocidade máxima de uma conexão, um

host transmissor precisa manter o canal de comunicação cheio até receber a primeira confirmação do *host* receptor e assim evitar desperdício de largura de banda.

Outro fato, é o controle de congestionamento do TCP que é constituído por diferentes mecanismos. Em estado estacionário, o principal algoritmo de controle de congestionamento utilizado é o AIMD (Additive-Increase, Multiplicative- Decrease). Portanto, a janela de congestionamento do TCP aumenta linearmente a cada reconhecimento positivo (ACK) recebido e decresce exponencialmente a cada segmento perdido. Através disto, o AIMD adapta a taxa de transmissão ao congestionamento da rede. No entanto, o AIMD apresenta alguns problemas para redes de alta velocidade, principalmente quando o produto da largura de banda pelo atraso (Bandwidth delay product, BDP) é grande, pois a lentidão do aumento da janela de congestionamento provoca uma baixa utilização da largura de banda disponível.

O BDP representa a capacidade do canal desde o transmissor até o receptor. Este cálculo nos leva a concluir que, para se obter um bom desempenho, a janela TCP do receptor deve ser, pelo menos, tão grande quanto o produto da largura de banda pelo retardo [2].

Compreendido o contexto do trabalho, a sua organização ocorre da seguinte maneira. O capítulo 2 apresenta os fundamentos do protocolo TCP. O capítulo 3, detalha como é feito o Controle de Congestionamento em relação ao desempenho do TCP. Neste mesmo capítulo, também são citados os problemas relacionados ao desempenho. O capítulo 4, são apresenta várias das implementações do protocolo TCP e suas principais características, além de apresentar a proposta e o cenário da simulação desenvolvidos neste trabalho.

O capítulo 5 apresenta os resultados dos experimentos deste estudo e suas interpretações.

O capítulo 6 é dedicado à conclusão e indicação de futuros trabalhos que podem seguir nesta linha de pesquisa.

2 - Fundamentos TCP

2.1-Descrição Geral

O TCP e o UDP são os dois protocolos da camada de transporte na Internet. O TCP fornece confiabilidade para aplicações sensíveis a perda, através de retransmissões de segmentos perdidos, enquanto o UDP fornece um transporte mais leve, sem retransmissões para aplicações sensíveis ao atraso.

Embora o desempenho da rede possa ser elevado com o alto desempenho de hardware, a função do serviço TCP, geralmente implementado em *software*, não pode ser ignorada. Este serviço tem total controle sobre o transporte de cada byte da aplicação que o utiliza. O TCP é um protocolo complexo que interage com muitos elementos externos em um caminho fim-a-fim. O hardware, sozinho, pouco pode fazer para ajudar no desempenho da rede, a menos que o serviço TCP seja otimizado.

Um dos problemas mais pesquisados do TCP é sobre o controle de congestionamento e o controle de fluxo. Os algoritmos e parâmetros que são adequados para um ambiente, nem sempre são adequados para outros. Logo, é necessário adaptar o TCP para diferentes ambientes.

As funcionalidades básicas do TCP estão definidas nas seguintes RFCs: RFC 793 [29], RFC 1122 [31] e RFC 2581 [32]. Algumas extensões também estão definidas nas RFC 2018 [30] e RFC 1323 [4].

2.2-Aspectos Relevantes TCP

O TCP, através do esquema de reconhecimento positivo e de retransmissão, provê a entrega confiável de segmentos de dados. Além de fornecer serviço confiável, podemos citar outras características importantes do protocolo TCP:

- **Orientado à conexão,** antes das duas aplicações iniciarem a transmissão elas devem estabelecer uma conexão, assim como em telefonia. Em tese, um aplicativo faz uma chamada que deve ser aceita pelo outro. Durante o estabelecimento da conexão, alguns parâmetros referentes à transmissão dos dados são determinados. A conexão é terminada quando uma das partes decide fazê-lo.
- **Orientado ao fluxo**, quando dois programas aplicativos transferem grande volume de dados, estes dados são considerados como um fluxo

de *bits*, dividido em octetos de *bits*, os quais são informalmente chamados de *bytes*. O serviço de transmissão de fluxos, da máquina de destino, passa para o receptor exatamente a mesma seqüência de octetos que o transmissor passa para ele na máquina de origem.

- **Transmissão bufferizada**, os dados da aplicação transmissora são enviados para o *software* TCP, em quantidade de *bytes*, escolhida da forma mais conveniente, variando de aplicação para aplicação. O *software* TCP, por sua vez, para otimizar a taxa de transferência, e para minimizar o tráfego de rede, ao receber os dados da aplicação transmissora, acumula-os em buffer de transmissão, enviando-os em blocos cujo tamanho geralmente difere daqueles que ele recebe da aplicação. O mesmo raciocínio vale para o nó destino, ou seja, os dados são armazenados em um buffer de recepção e entregues para a aplicação, não necessariamente em bloco, nem com o mesmo tamanho utilizado na transmissão através da rede.

- **Conexão *full duplex***, as conexões fornecidas pelo serviço de fluxo TCP/IP permitem transferência em ambas as direções. Essas conexões são denominadas *full duplex*. Do ponto de vista de um processo aplicativo, este tipo de conexão consiste em dois fluxos independentes, fluindo em direções opostas, sem qualquer operação aparente. A vantagem de uma conexão *full duplex* é que o *software* básico de protocolos pode enviar, de volta à origem, as informações de controle de um fluxo, em datagramas que estejam transportando dados na direção oposta. Essa carona (*piggyback*) reduz o tráfego na rede.

Outro fato importante a esclarecer sobre em que consiste o TCP, é que independente de suas diferentes implementações, este é um protocolo de comunicação e como tal consiste em uma série de especificações que devem ser seguidas. Porém, a forma como estas especificações serão implementadas pode variar. Sendo assim, não se deve confundi-lo com um trecho de seu código.

O protocolo TCP define o formato dos dados e reconhecimentos que as aplicações utilizam para atingir um serviço confiável, assim como os procedimentos efetuados para

assegurar o recebimento correto dos dados. Este protocolo também especifica como o *software* TCP distingue múltiplos destinos em um mesmo nó, como tratar erros, perda de *segmento*s, como iniciar e terminar uma conexão.

A unidade de dados padronizada para o TCP é denominada segmento, e é constituída de um cabeçalho TCP, seguido de um campo de dados, composto de dados vindos da camada de aplicação. O tamanho do campo de dados recebido da camada de aplicação é variável, porém não deve ser maior que o *Maximun Segment Siza (MSS)*, definido pelo receptor durante o estabelecimento da conexão. O segmento deve estar dentro do limite de tamanho imposto pelo *Maximum Transfer Unit (MTU)*, padronizado para cada tipo de rede. O cabeçalho do TCP tem formato fixo de 20 *bytes*, seguido de campos opcionais. O formato do cabeçalho TCP está na figura 2, e as definições das funções de cada campo estão logo a seguir [29].

Como cada segmento é transmitido como sendo um segmento separadamente, então o segmento não necessita de delimitadores de início e fim. Cada byte recebido através do fluxo de dados enviado pela camada de aplicação, corresponde a um determinado número de seqüência na conexão TCP. O cabeçalho de cada segmento leva o número da seqüência do primeiro byte presente no segmento. O valor padrão para o MSS é de 536 *bytes*, este valor está dentro do valor padrão definido para o MTU, que é de 576 *bytes* (MSS + TCP header + IP header) [1], [2] e [29].

2.3-Cabeçalho TCP

Um segmento TCP é encapsulado em um datagrama IP como mostrado na figura 2.1, e cada segmento tem duas partes, o cabeçalho padrão com 20 *bytes* e os dados da aplicação. O cabeçalho contém os campos com informações úteis, tais como, tamanho da janela adversativa, número ACK, entre outros. Para entender as operações TCP, é necessário examinar os princípios e propósitos destes campos que estão definidos em [4], [29] e [30].

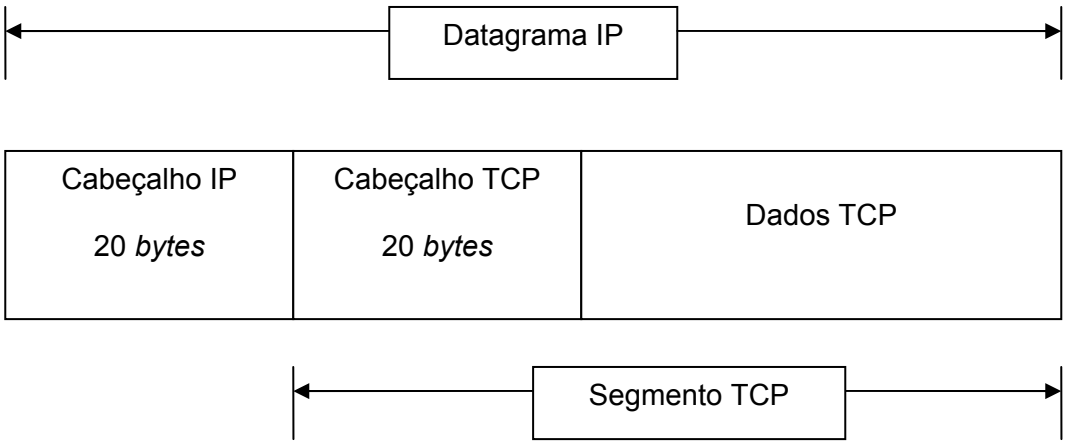

Figura.2.1 – Encapsulamento do segmento TCP

Número de Porta de Origem (16bits)								Número de Porta de Destino (16bits)
Número de Seqüência (32 bits)								
Número de Reconhecimento – ACK (32 bits)								
Tamanho do Cabeçalho (4 bits)	Reserved (6 bits)	U R G	A C K	P S H	R S T	S Y N	F I N	Janela (16bits)
Verificação (16bits)								Ponteiro Urgente (16bits)
Opções (Variável)								
Dados (Variável)								

Figura 2.2 – Cabeçalho TCP

Número da porta de origem (Source Port - 16*bits*): Contém o número da porta TCP de origem que identifica o programa aplicativo da conexão.

Número da porta de destino (Destination Port - 16*bits*): Contém o número da porta TCP de destino que identifica o programa aplicativo da conexão.

Número de seqüência (Sequence Number - 32*bits*): identifica a posição no fluxo de *bytes* do transmissor dos dados no segmento.

Número de reconhecimento (Acknowledge Numeber - 32*bits*): identifica o próximo octeto que a origem espera receber.

Tamanho do cabeçalho (TCP Header Length - 4*bits*): contém um número inteiro que especifica o comprimento do cabeçalho do segmento, medido em múltiplos de 32*bits*. Isto é necessário porque o campo **Opções** varia em comprimento, dependendo de quais opções foram incluídas. Assim, o tamanho do cabeçalho TCP varia de acordo com as opções selecionadas.

Reservado (6*bits*): é reservado para utilização futura ou experimental.

***Bits* de código (Flags - 6*bits*)**: alguns segmentos transportam apenas uma confirmação, outros transportam dados e outros transportam solicitações para estabelecer ou encerrar uma conexão. O *software* TCP utiliza o campo de seis *bits* para determinar a finalidade e o conteúdo do segmento. Os seis *bits* indicam como interpretar outros campos do cabeçalho, segue a definição para cada um dos seis *bits*.

Bit	Descrição se o valor for 1
URG	Campo de ponteiro urgente é válido
ACK	Campo de reconhecimento é válido
PSH	Este segmento requer push
RST	Restabelecer a conexão
SYN	Início de uma nova conexão
FIN	Final do segmento do transmissor

Janela (Window Size - 16*bits*): informa quantos dados o receptor está esperando receber, sempre que envia um segmento, especificando o espaço disponível de seu

buffer. O campo contém um número inteiro de 16 *bits*, que é utilizado pelo transmissor com o propósito de controle de fluxo.

Verificação (*Checksum* – 16*bits*): Número de verificação de *bits* que estão sendo transferidos para descobrir erros na transferência. A soma de todos os *bytes* mais o *checksum* deve ser igual a zero.

Ponteiro urgente (Urgent Pointer - 16*bits*): um segmento TCP pode transmitir dados que precisam de prioridade de tratamento. O programa receptor deve ser notificado de sua chegada o mais rápido possível, independente de sua posição no fluxo. O campo URG deve estar setado neste segmento e, neste campo, especifica a posição no segmento onde os dados urgentes terminam.

Opções (tamanho variável): possui diversas funções. Diversas opções podem ser utilizadas no cabeçalho TCP. Novas RFCs, principalmente [2] e [4], definem opções adicionais para o TCP, muitas destas são somente encontradas nas últimas implementações. As opções relevantes para o desempenho do TCP são as seguintes:

- MSS (Tamanho máximo de segmento – Maximum-receive-segment-size): esta opção é usada durante o estabelecimento da conexão para negociar o MSS que será usado na conexão. Ela deve ser usada na descoberta do MTU máximo da rede.

O tamanho de um segmento TCP pode variar ao longo de uma conexão. Porém, deve haver um acordo sobre o tamanho máximo que um segmento pode assumir durante a existência de uma conexão. A escolha do valor do MSS tem influência direta no desempenho do protocolo TCP, da mesma forma que valores muito pequenos acarretarão em uso ineficiente da largura de banda da rede e valores altos podem fazer com que o datagrama IP seja fragmentado ao longo do percurso. Esta fragmentação obrigará a espera de todos os fragmentos para que o *software* TCP possa enviar o reconhecimento, já que fragmentos não podem ser reconhecidos separados. Logo, deve-se encontrar o valor ótimo para o MSS.

No entanto, determinar o valor ótimo não é tão simples. Em redes locais torna-se mais simples, pois é possível utilizar o valor de um datagrama IP com o tamanho igual ao MTU (Maximum Transfer Unit) de rede. Caso o caminho envolva várias redes

heterogêneas, pode-se tentar descobrir o tamanho ideal ou escolher o valor de 536 *bytes*, que resultará em um datagrama IP de tamanho padrão.

-Window-scale option (Janela adaptável): Esta opção tem por objetivo aumentar o tamanho máximo da janela de transferência do TCP, principalmente em redes com um valor alto para o produto da largura de banda pelo atraso. Sem esta opção, o tamanho máximo da janela é de 65535 *bytes* (campo de 16 *bits*). Esta opção é negociada no início da conexão TCP, junto com um *segmento* de SYN.

O segmento SYN foi escolhido por ser um segmento confiável, no qual cada lado pode comunicar o fator de adaptação da janela, na opção inicial do TCP. No entanto, este método tem a desvantagem de o fator de adaptação poder ser estabelecido somente quando a conexão é iniciada, não podendo ser mais alterado.

Esta opção de *window scale* aumenta a definição da janela do TCP de 16 *bits* para 32 *bits*. Com isso, em vez de mudar o cabeçalho do TCP para acomodar uma janela maior, o cabeçalho mantém o valor de 16 *bits*, e esta opção aplica uma operação de modificação neste valor. O TCP mantém então o tamanho "real" da janela internamente como um valor de 32 *bits*.

Em [4] temos que o valor máximo da janela é 2^{30}, permitindo uma janela de 1Gbyte, e o valor máximo permitido para o fator da *window scale* é 14, se o fator recebido for maior que 14, o TCP irá informar um erro, mas usará o 14 ao invés do valor informado.

- Timestamp (8 *bytes*): esta opção é utilizada para calcular o valor RTT (round-trip time) com mais precisão.

Este valor é fundamental para manter o equilíbrio da rede, já que, para manter este equilíbrio uma boa estimativa para o valor de *timeout* de retransmissão deve ser alcançada.

- SACK-permitted option and SACK option: Esta opção altera o comportamento do TCP. SACK significa Selective Acknowledgement. Esta opção é oferecida ao nó remoto, no início da conexão (*segmento* SYN) e, quando habilitada, o transmissor obtém informações suficientes sobre os segmentos recebidos corretamente pelo receptor e, dessa forma, é capaz de retransmitir em um RTT múltiplos segmentos perdidos de uma janela de dados.

Devemos lembrar que o procedimento padrão do TCP é enviar ACK somente para o número de seqüência mais alto dos segmentos recebidos corretamente, podendo provocar no transmissor um reenvio de blocos que chegam bem ao destino, pois o transmissor não sabe que um segmento anterior foi perdido.

Qualquer implementação robusta do TCP deve assegurar-se que a sessão está utilizando o maior tamanho de segmento possível sem fragmentação, que o tamanho da janela está adequado ao produto de "largura de banda X atraso" da rede, e que o ACK seletivo (SACK) pode ser usado para recuperar rapidamente erros e pequenos congestionamentos.

2.4- O MTU – Maximum Transmission Unit

O tamanho dos blocos de dados utilizados numa conexão TCP é negociado no início da sessão. O transmissor tenta utilizar o maior tamanho possível de segmento não fragmentado para a transferência de dados, dentro dos limites da rede, do transmissor e do receptor.

O processo do usuário pode enviar mensagens de qualquer tamanho, e o TCP divide essa mensagem em blocos iguais ou menores que 64 K*bytes*, enviando cada peça separadamente. O tamanho máximo do segmento é de 65535 *bytes*, menos os cabeçalhos TCP e IP, como pôde ser visto, anteriormente, na figura 1. Se o segmento passar por uma rede com MTU menor, deve ser fragmentado, recebendo um novo cabeçalho TCP e IP (40 *bytes* a mais), prejudicando o desempenho do sistema, portanto, isso deve ser evitado.

Para o Ethernet, o MTU é 1500 *bytes*, para o PPP é 1492, e muitas conexões dial-up utilizam um MTU de 576 *bytes*. Cada segmento consiste de cabeçalho+dados. O tamanho dos dados é conhecido como MSS (Maximum Segment Size), e define o máximo de dados que podem ser transmitidos em um segmento TCP. Essencialmente, MTU = MSS + cabeçalhos TCP e IP (o mínimo dos cabeçalhos TCP e IP é de 20 *bytes* cada, totalizando 40 *bytes* de cabeçalhos) [29], [34] e [35].

Tamanho do pacote X Latência

Os nós intermediários normalmente trabalham numa filosofia de "*store-and-forward*", causando um atraso devido ao empacotamento do pacote. Esse atraso acontece em cada nó, por onde o pacote passa. Supondo um canal E1 (2.048.000 bps), tem-se que a latência por nó é dada por:

Latência = (MSS+cabeçalho) * 8 / 2.048.000.

Assim, usando-se dois MTUs distintos, tem-se:

- MTU=1500: (1460+40)*8 / 2.048.000 = 5,86ms por nó.
- MTU=576: (536+40)*8 / 2.048.000 = 2,25ms por nó.

Em uma transferência com 10 nós, através de linhas E1, o MTU de 1500, atrasaria 58,6ms, enquanto o MTU de 576 atrasaria 22,5ms. Esse dado pode ser particularmente útil em transmissões de tempo real, quando o baixo atraso é fundamental para a aplicação.

Vale lembrar que com MTU maior, consegue-se uma vazão de dados também maior, como é visto a seguir [31] e [35].

Tamanho do pacote X Vazão de dados

Usando a mesma fórmula anterior e supondo uma transferência de 2M*bytes*, tem-se:

- Com o MTU=1500: número de pacotes a serem enviados é:

  2M/1460 = 2.097.152/1460 = 1436,4 ou, de fato, 1437 cabeçalhos.

  O total de dados transmitidos será:

  2.097.152 (*bytes*) * 8 (*bits*) + 1437 (pacotes) * 40 (*bytes*) * 8 (*bits*).

  Isso equivale à:

  17.237.056 *bits* /2.048.000 = 8,41 segundos, mais os atrasos de empacotamento por nó.

- Com o MTU=576: número de pacotes a serem enviados é

  2M/536 = 2.097.152/536 = 3912,6 ou, de fato, 3913 cabeçalhos.

O total de dados transmitidos será:

2.097.152 (*bytes*) * 8 (*bits*) + 3913 (pacotes) * 40 (*bytes*) * 8 (*bits*).

Isto equivale à:

18.029.376 *bits* /2.048.000 = 8,8 segundos.

A diferença é pelo do fato de que para transmitir pacotes de tamanho maior, o overhead é menor, como pode ser observado em [33].

2.5- Fornecendo Confiabilidade

O TCP oferece um serviço de transmissão de fluxo de dados confiável, no entanto, não é óbvio perceber como um protocolo que utiliza serviços do protocolo IP possa fornecer um serviço confiável, já que este último é totalmente não confiável. Para fornecer tal confiabilidade, o protocolo TCP utiliza mecanismos próprios para a garantia de confiabilidade. Para atingir este objetivo, o protocolo TCP executa as seguintes tarefas:

1- Os dados passados para a aplicação são divididos em blocos de tamanhos convenientes, que são denominados Segmentos.

2- Para cada segmento recebido com sucesso do outro ponto-final da conexão, uma confirmação positiva é enviada. Estas confirmações podem ser enviadas imediatamente ou após um atraso devidamente escolhido. Este último procedimento é conhecido como mecanismo de reconhecimentos atrasados, *delayed acks*, que tem como objetivo aumentar a eficiência da conexão, já que utiliza apenas um segmento para reconhecer uma quantidade maior de dados. No entanto, é importante observar que este último reconhecimento recebido é um reconhecimento cumulativo, ou seja, ele reconhece a chegada de todos os segmentos até aquele que desencadeou a geração deste sinal de reconhecimento.

3- Para cada segmento enviado um temporizador associado ao segmento é disparado. O TCP então espera pela confirmação do recebimento deste segmento pelo nó destino. Caso o tempo se esgote sem que a confirmação tenha sido recebida, o segmento é retransmitido.

4- Mantém um mecanismo de proteção da integridade para o cabeçalho, através de um mecanismo de *checksum*, de forma a detectar qualquer alteração no conteúdo da

informação transmitida. Caso haja algum erro, o segmento é descartado no destino e a sua retransmissão é aguardada.

5- Por utilizarem os serviços de uma rede IP, os segmentos podem chegar ao destino fora de ordem. O TCP deve efetuar então a reordenação dos dados a fim de garantir a entrega destes em ordem para a aplicação. Outro fato que pode ocorrer é a duplicação dos segmentos [18], que também não é tratado pelo protocolo IP, logo, a responsabilidade de descartar dados duplicados é do protocolo TCP, fazendo-o de forma transparente.

6- Exerce um controle de fluxo fim-a-fim, fazendo com que o nó destino permita que o nó fonte envie no máximo a quantidade de *bytes* correspondente ao espaço livre no *buffer* de recepção.

Das tarefas citadas anteriormente, as mais importantes para a confiabilidade são a segunda e a terceira, pois a junção das duas forma o conceito de reconhecimento positivo e retransmissão. O termo positivo aqui se refere ao fato do receptor enviar as confirmações correspondentes aos segmentos recebidos, e os segmentos que não forem confirmados serão retransmitidos.

A retransmissão é o princípio básico do serviço de transferência de dados confiável do TCP. Se o segmento é perdido deve ser retransmitido. Para detectar a perda de um segmento, o transmissor mantém um registro de cada segmento que envia e espera uma confirmação antes de enviar o próximo segmento. Para cada registro, o TCP mantém um cronômetro e inicia um temporizador quando um segmento é enviado. O temporizador é configurado por um intervalo de tempo, chamado *timeout* de retransmissão (retranmission *timeout* - RTO). Se uma confirmação, também chamada de reconhecimento (ACK - aknowledgment), é recebida durante o RTO, o temporizador é zerado, caso contrário, o temporizador expira e, neste caso, deverá haver uma retransmissão do segmento.

A fig.2.3 ilustra o esquema que garante que todos os segmentos serão recebidos em algum momento, inclusive no caso em que ocorra o estouro do temporizador. Um fato que se deve observar, é que os segmentos e suas confirmações são numerados. A importância disto é para que o nó fonte associe cada confirmação a seu respectivo segmento. E para isto o segmento de confirmação contém um campo, indicando o segmento recebido com sucesso, que foi responsável por sua geração.

Na figura 2.3, exemplifica o caso de transmissão *stop-and-wait*, no qual o emissor aguarda o recebimento de um sinal de reconhecimento para só então enviar um segmento de dados.

Das tarefas citadas anteriormente, as mais importantes para a confiabilidade são a segunda e a terceira, pois a junção das duas forma o conceito de reconhecimento positivo e retransmissão. O termo positivo aqui se refere ao fato do receptor enviar as confirmações correspondentes aos segmentos recebidos, e os segmentos que não forem confirmados serão retransmitidos.

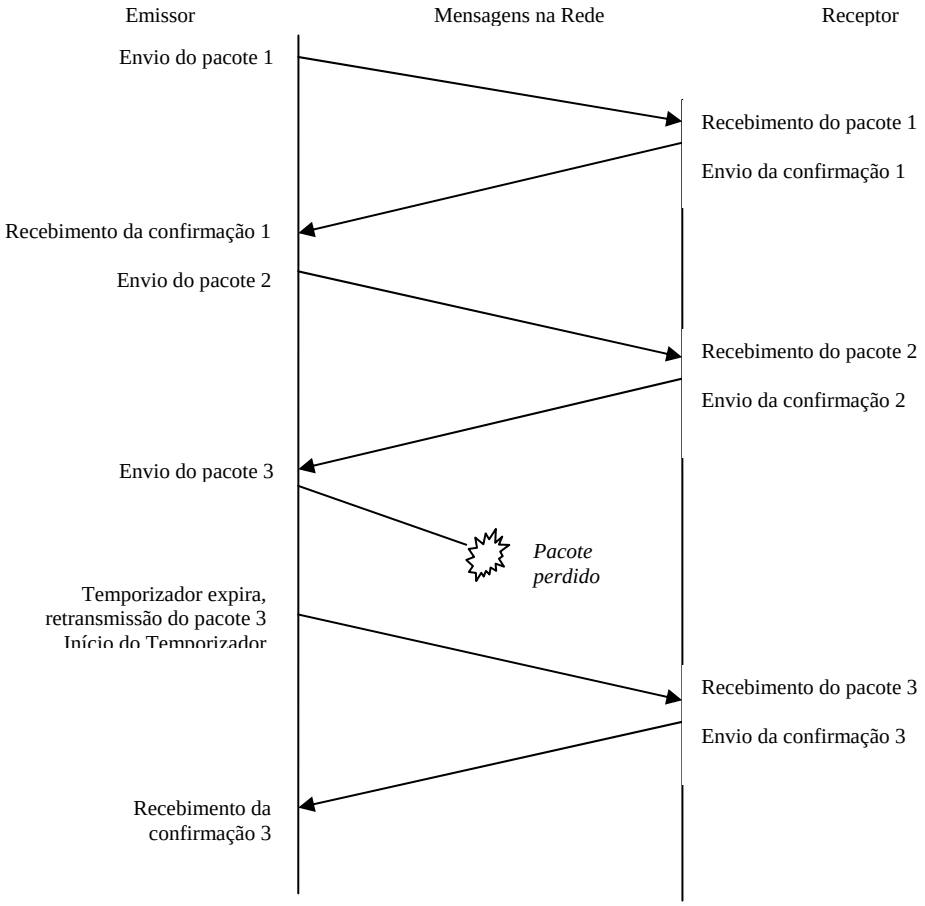

Figura 2.3- Reconhecimento positivo e retransmissão

2.6- Timeout de retransmissão

Em uma determinada conexão a estimativa do tempo total de transmissão de ida e volta é fundamental para o cálculo do RTO. Como esta estimativa muda com o tempo,

devido a mudanças de rotas e padrões de tráfego, o TCP deve monitorar estas mudanças e modificar o tempo de RTO apropriadamente [1].

Sabendo disto, o protocolo TCP aceita estas variações através de um algoritmo adaptável de retransmissão, monitorando o desempenho de cada conexão e calculando valores razoáveis para o RTO. Este algoritmo registra o instante que cada segmento é enviado e o instante em que chega a confirmação. Através da diferença destes valores, o TCP calcula uma nova amostra para o RTT. Sempre que obtém uma amostra de tempo de ida e volta, o TCP ajusta sua noção de média de tempo de ida e volta para a conexão. Normalmente, o *software* TCP armazena o RTT estimado como uma média ponderada, e utiliza novas amostras de tempo de ida e volta para alterar lentamente a média. A especificação original do TCP atualizava a estimativa do RTT, usando uma soma ponderada entre a nova amostra e o valor de RTT utilizado anteriormente, como mostra o quadro1.

$$RTT = (\alpha * RTT) + (1 - \alpha) * M$$

Quadro 1

onde o M representa a última estimativa e RTT o novo valor amostrado, α assume um valor entre 0 e 1, o valor recomendado é 0,9. Escolher um valor α próximo a um, torna a média ponderada imune a alterações de curta duração, mas, se escolhido um valor próximo a zero, a média ponderada reagirá mais rapidamente às alterações por intervalos.

Uma vez que o RTT é atualizado, o RTO, *timeout* de retransmissão, para o próximo segmento segue a fórmula no quadro 2.

$$RTO = (\beta * RTT)$$

Quadro 2

onde β é o fator de variação, e é maior que 1. Fazendo o valor β próximo de 1 aumenta a vazão na medida em que o valor do *timeout* se aproxima do RTT. Porém, um pequeno

atraso causará uma retransmissão desnecessária. A especificação original do protocolo TCP especifica um valor de 2 para o parâmetro β.

Em [5] é detalhado os problemas com esta abordagem. Primeiramente, ela não se adapta a flutuações muito altas do RTT, causando retransmissões desnecessárias e aumentando ainda mais a carga na rede, quando esta já está sobrecarregada.

O problema final de confiabilidade surge quando há ambigüidade dos reconhecimentos, isto ocorre porque o TCP usa um esquema de confirmação cumulativo no qual uma confirmação refere-se aos dados recebidos dentro de um fluxo e não ao segmento especificamente. Outra dificuldade para a medição do RTT advém do fato de que, no caso de retransmissões, um segmento com os mesmos dados é enviado e não há meios de saber se a confirmação que chega ao transmissor refere-se ao primeiro ou ao segundo segmento.

Estimar o valor do RTT quando houver retransmissão pode acarretar em problemas na rede. Isto é, associando a confirmação à transmissão original, fará com que a estimativa de ida e volta cresça sem limites, se na rede houver perdas de segmentos. E associar a confirmação à retransmissão mais recente também pode haver falhas, ou seja, sabendo-se que o protocolo TCP, ao enviar um segmento, utiliza a estimativa anterior do tempo de ida e volta para calcular o *timeout*, então, se um intervalo fim-a-fim aumentar repentinamente, o temporizador poderá terminar antes que a confirmação chegue.

Para solucionar o problema, metade da solução consiste em fazer com que o protocolo TCP ignore reconhecimentos referentes a segmentos retransmitidos, e com isso não atualizar a estimativa do tempo de ida e volta para segmentos retransmitidos. Esta técnica é conhecida como Algoritmo de Karn/Partridge, evita o problema de confirmações ambíguas, ou seja, o tempo de ida e volta é somente ajustado para as confirmações dos segmentos que não tiverem sido retransmitidos. No entanto, esta idéia também pode levar a falhas, pois se o TCP calcula o *timeout*, usando a estimativa existente do tempo de ida e volta, quando houver um aumento no intervalo, o *timeout* será muito pequeno, o que pode acarretar em uma retransmissão, como esta confirmação será ignorada, o cálculo da estimativa do RTT não poderá ser atualizado.

Para tratar este problema, o algoritmo de Karn/Partridge exige que o transmissor combine os *timeout*s da retransmissão com uma *estratégia de backoff do temporizador*. A técnica de *backoff* calcula um *timeout* inicial, aplicando uma fórmula semelhante a mostrada anteriormente. Entretanto, se o temporizador terminar e provocar uma retransmissão, o TCP aumentará o *timeout*.

As implementações usam diversas técnicas para calcular o *backoff*. A maioria escolhe um fator multiplicativo γ, e fixa o novo valor como segue o quadro 3.

novo_*timeout* = (γ * *timeout*)

Quadro 3

Geralmente, γ é igual a dois. Outras implementações usam uma tabela de fatores multiplicativos, permitindo um *backoff* arbitrário em cada etapa. E para evitar que o valor de *timeout* cresça indefinitivamente, no caso de retransmissões sucessivas, um limite igual ao maior atraso possível na Internet é mantido pela maioria das implementações.

No entanto, mesmo com todas estas modificações, ainda assim o resultado não é satisfatório, pois os procedimentos descritos até o momento não permitem que as estimativas para o *timeout* acompanhem grandes variações de atraso da rede. Esta limitação justifica-se pela teoria das filas, que diz que o atraso de ida e volta e a sua variação são proporcionais a $1/(1 - \rho)$, onde ρ é a utilização da rede, $0 \leq \rho \leq 1$. Se uma interligação de redes estiver sendo executada com 50% e 80% de sua capacidade, deve-se esperar que o retardo varie por um fator de 4 e 10, respectivamente [1]. Levando em consideração o valor dois para β, a estimativa para o valor do RTT poderá se adaptar bem a valores de utilização iguais ou inferiores a 30%[1].

Sendo assim, a especificação de 1989, do protocolo TCP, introduz estimativas tanto para a média quanto para a variância do RTT, e utiliza o cálculo desta última ao invés da constante β para o cálculo do *timeout*. As expressões que seguem no quadro abaixo descrevem os passos para o cálculo do *timeout*:

Dif = *Amostra_RTT – RTT_Estimado*
RTT_Estimado = *RTT_Estimado* + δ * *Dif*
Desvio = *Desvio* + ρ (| *Dif* | - *Desvio*)
Timeout = *RTT_Estimado Desvio* + ρ η

Quadro4

onde, “Desvio” é o menor desvio estimado, δ é uma fração entre zero e um, que controla a rapidez com que as novas amostras afetam a média ponderada, ρ é uma fração entre zero e um que controla quão rápido novas amostras afetam o menor desvio e η é um fator que controla o quanto o desvio médio afeta o *timeout*.

Pesquisas na área sugerem que δ =1/2^3, ρ = 1/2^2 e η = 3 irão funcionar bem. Este algoritmo foi desenvolvido pelo trabalho de Jacobson/Karels e está descrito em [3], o efeito deste algoritmo é uma melhora no desempenho TCP em relação à estimativa do valor do *timeout*.

As figuras 2.4 e 2.5 mostram a diferença do cálculo do *timeout* em situações com os algoritmos Karn/Partidge e Jacobson/Karels [5].

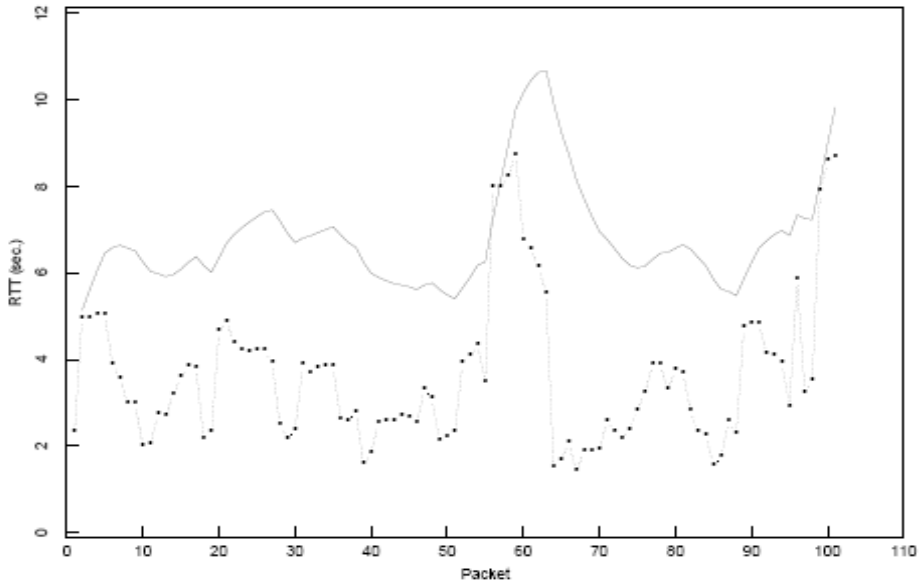

Fig.2.4 – Cálculo do *timeout* pelo algoritmo de Karn/Prtridge, que segue as regras da RFC793 [29].

Na figura 2.4 o eixo das abscissas – X é o número de segmentos, o eixo das ordenadas – Y é o tempo percorrido para que um segmento enviado receba o seu ACK. Durante este teste, nenhum segmento foi perdido ou retransmitido. A linha cheia mostra o comportamento do cálculo do *timeout* de acordo com as regras da RFC793 [29].

A figura 2.5 apresenta os mesmos dados que o gráfico anterior, mas a linha sólida mostra o *timeout* calculado de acordo com o algoritmo Jacobson/Karels. Mais detalhes sobre este experimento encontram-se em [3].

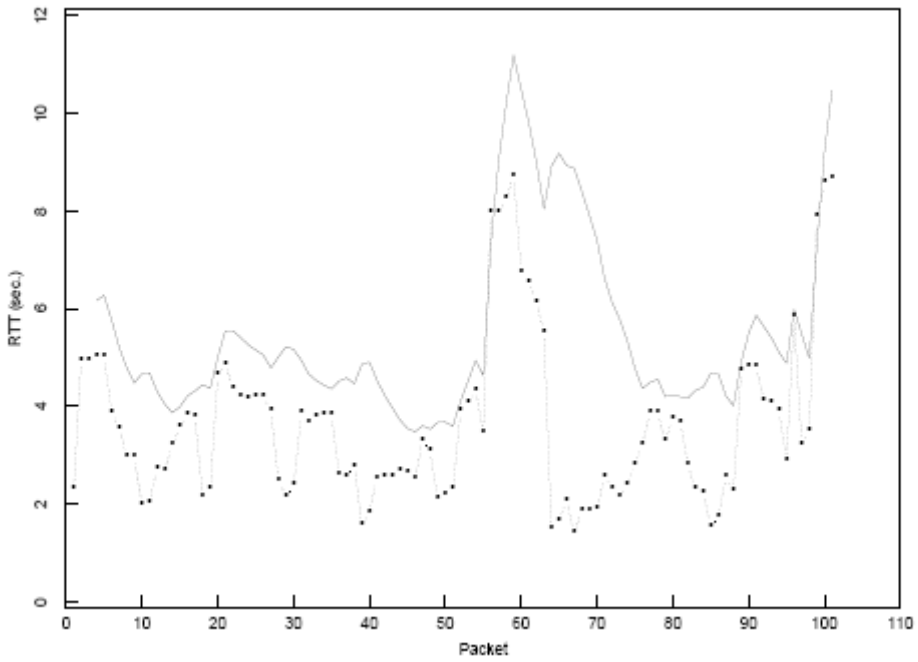

Figura 2.5 – Cálculo do *timeout* pelo algoritmo de Jacobson/Karels.

2.7- Janela Deslizante

O TCP considera o fluxo de dados uma seqüência de octetos ou *bytes* que ele divide em segmentos para transmissão. Normalmente, cada segmento trafega em uma rede em um único datagrama IP.

Para alcançar a confiabilidade, o protocolo TCP faz com que o transmissor transmita um segmento e depois aguarde uma confirmação antes de transmitir outro segmento, como ilustrado na figura 2.3, com isso, a rede pode ficar completamente inativa durante o tempo em que a máquina retarda as confirmações. Se for uma rede com grandes intervalos entre as transmissões, o problema torna-se claro, ou seja, um protocolo simples, de confirmação positiva, gasta uma quantidade substancial de largura de banda, pois precisa retardar a transmissão de um novo segmento, até que receba uma confirmação sobre o anterior.

Para tornar o mecanismo mais eficiente, introduz-se o mecanismo de janela deslizante para resolver dois problemas importantes: transmissão eficaz e controle de fluxo. Esta técnica é uma forma mais complexa de confirmação positiva e de retransmissão do que o método simples citado anteriormente. Os protocolos de janela deslizante utilizam melhor a largura de banda da rede, pois utilizam uma janela com um determinado tamanho e transmitem todos os dados dentro desta janela sem esperar por

reconhecimento algum. Sendo assim, o número de segmentos não reconhecidos fica limitado ao tamanho da janela e da velocidade com que a rede aceita os segmentos, a figura 2.6 ilustra o esquema de janela deslizante.

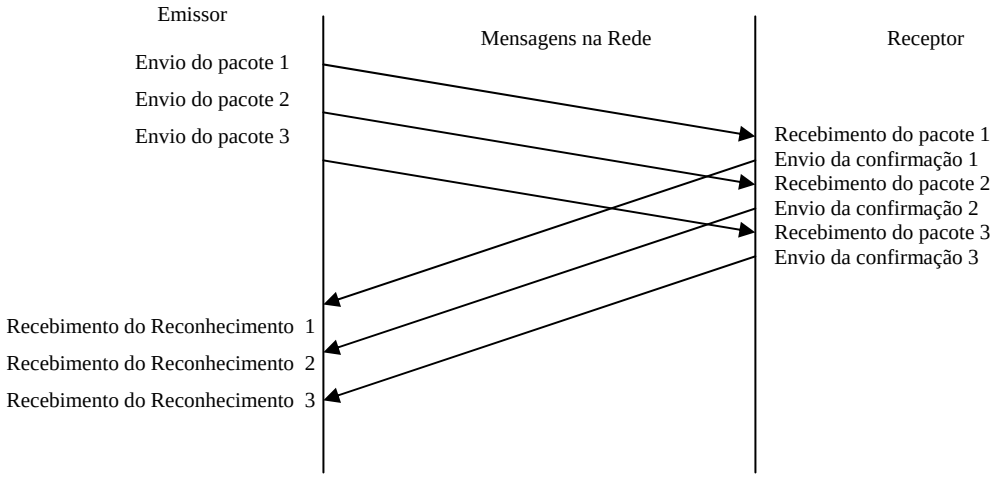

Figura 2.6- Reconhecimento positivo e retransmissão

Entretanto, a maneira como o mecanismo de janela deslizante é implementado no TCP difere um pouco do tradicional, isto é, a diferença ocorre porque o TCP permite que o tamanho da janela varie com o passar do tempo. Cada confirmação contém uma especificação de quantos octetos foram recebidos e um notificador de janela que define quantos octetos de dados adicionais o receptor está preparado para aceitar. O notificador de janela é considerado o especificador do tamanho atual do *buffer* no receptor.

A vantagem de utilizar uma janela de tamanho variável é que ela fornece controle de fluxo e uma transferência confiável. Se o *buffer* do receptor começar a ficar cheio, a janela não poderá suportar mais segmentos e, então, enviará um notificador de janela menor. Em casos extremos, o receptor indica um tamanho de janela zero para interromper todas as transmissões. Tão logo o espaço do *buffer* torne-se disponível, o receptor indicará um tamanho de janela diferente de zero para iniciar o fluxo de dados novamente.

Aumentando o tamanho da janela, é possível eliminar completamente o tempo de inatividade da rede. Isso significa que, com o aumento da janela e supondo um RTT (Round-Trip time) fixo, chega-se a uma situação na qual a rede se torna totalmente ocupada, o que acarreta um aumento na vazão, ou seja, em estado estacionário, o

transmissor pode transmitir segmentos de acordo com a velocidade que a rede pode transmiti-los. Um fator importante sobre isto é, quando um protocolo de janela deslizante apropriado mantém a rede completamente saturada de segmento, ele obtém um throughput substancialmente maior do que um protocolo simples de confirmação positiva.

No caso do protocolo TCP, este mecanismo opera em termos de *bytes*, e não em termos de segmentos ou pacotes. Logo, o tamanho da janela controla o número de *bytes* em trânsito. O seu tamanho é ajustado dinamicamente de acordo com espaço em *buffer* no receptor. O receptor TCP pode aumentar ou diminuir o tamanho da janela em um segmento ACK, utilizando o campo, tamanho de janela, no cabeçalho do segmento TCP. O transmissor TCP mantém uma variável, Janela de anúncio (*advertised window* - *awnd*), para guardar o tamanho da janela atual, ou seja, a capacidade do *buffer* do receptor, que tem como finalidade o controle de fluxo em uma conexão TCP.

Os controles do mecanismo da Janela deslizante possibilitam estimar a taxa de transmissão possível entre sistemas heterogêneos e otimizar esta taxa. Na origem do fluxo, esse mecanismo controla o envio de mais de um segmento de cada vez, antes de receber confirmação. Esse envio é chamado de rajada. Já o destino receberá toda rajada para depois confirmar os segmentos recebidos em uma só mensagem ACK. Se houver falha de algum segmento, a confirmação será apenas dos segmentos recebidos na seqüência correta, até onde houve a falha. Como pode ser observado na figura 2.7, retirada de [1], o emissor mantém 3 ponteiros para cada conexão, que delimitam quatro situações:

- o ponteiro 1, na figura 2.7, define que até o octeto 2 já houve confirmação;
- o ponteiro 2, na figura 2.7, define que os octetos 3 e 4 foram enviados e aguardam confirmação;
- o ponteiro 1, na figura 2.7, define os octetos 5 e 6não foram enviados, mas serão sem atraso;
- e na quarta situação os octetos maiores que 6 aguardam a janela se mover.

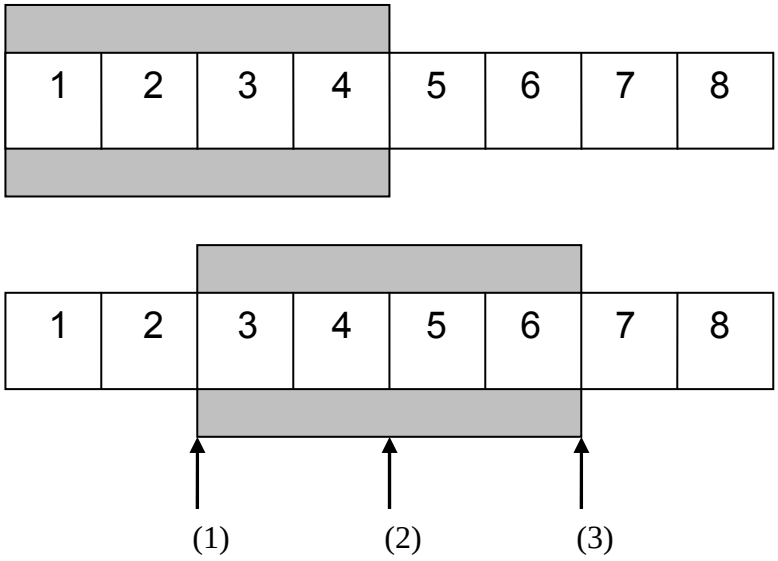

Figura 2.7- Janela Deslizante

O controle de fluxo tem como objetivo prevenir que um transmissor muito rápido sobrecarregue um receptor lento. Cada receptor aloca algum espaço em memória "*buffer*" para uma conexão TCP. Os dados recebidos são colocados neste *buffer* para que a aplicação correspondente possa lê-los e limpar o *buffer* o mais rápido possível. No entanto, em muitas situações, a aplicação no *host* receptor pode não ser capaz de tratar a taxa de dados que está chegando, levando ao excesso de dados no *buffer* do receptor. Nesta situação, a tarefa do controle de fluxo é ajustar a taxa de transmissão do transmissor TCP, ou seja, diminuir a quantidade de dados transmitidos para prevenir a falta de *buffer* no receptor TCP.

Possuir um mecanismo para controle de fluxo é essencial em um ambiente de interconexão de dados, onde máquinas de várias velocidades e tamanhos comunicam-se através de redes e de roteadores de várias velocidades e capacidades. Quando máquinas intermediárias tornam-se sobrecarregadas, a condição é denominada congestionamento.

Os mecanismos para resolver o problema são denominados controle de congestionamento. O TCP utiliza o mecanismo de janela deslizante para resolver o problema de controle de fluxo fim-a-fim e não possui um mecanismo explícito para controle de congestionamento. Entretanto, se uma implementação for cuidadosamente

programada pode detectar congestionamento e recuperar-se dele, enquanto uma implementação ineficaz pode acentuar este problema.

2.8 - Mecanismo de Reconhecimento

O TCP conta com o reconhecimento do receptor para confirmar a entrega correta dos dados. Algumas das importantes características do mecanismo ACK do TCP são as seguintes:

Reconhecimento acumulativo: o mecanismo de confirmação TCP é denominado cumulativo porque informa a área do fluxo que foi acumulada. Há vantagens e desvantagens em confirmações cumulativas. Uma vantagem é que as confirmações perdidas são fáceis de serem geradas. Uma outra vantagem é que as confirmações perdidas não forçam necessariamente uma retransmissão, já que estas somente informam qual o próximo octeto esperam receber. A principal desvantagem é que o transmissor não recebe informações sobre as transmissões bem-sucedidas, somente sobre uma única posição no fluxo que foi recebido.

Segmento ACK ou *Piggybacking*: um segmento ACK é indicado através do campo ACK no cabeçalho TCP. Então, para o reconhecimento dos *bytes* corretamente recebidos, um receptor pode criar somente um segmento ACK, ou pode enviar o ACK dentro de um segmento de dados. Quando um ACK atravessa dentro de um segmento de dados, o processo é chamado de *piggybacking*. *Piggybacking* reduz o tráfego ACK na direção reversa.

ACK atrasado: o receptor TCP tem a escolha de enviar um ACK assim que receber o segmento ou atrasá-lo por um período. Atrasando-o, o receptor pode ser capaz de confirmar dois segmentos ao mesmo tempo e reduzir o tráfego ACK, no entanto, um receptor TCP não deve atrasar um ACK por longos períodos, a fim de evitar o *timeout* de retransmissão no transmissor.

ACK duplicado: A finalidade do ACK duplicado é indicar ao emissor que um segmento foi recebido fora de ordem e qual o número de seqüência esperado. No entanto, se um ACK for duplicado, o transmissor pode interpretar como uma perda antes que o RTO expire.

2.9 - Estabelecimento e Encerramento de Conexão

O TCP fornece um serviço orientado à conexão através de dois procedimentos: estabelecimento de conexão e terminação de conexão. Uma conexão é estabelecida

antes da transferência dos dados. Quando a transferência dos dados é completada, a conexão é explicitamente terminada.

2.9.1- Estabelecendo uma conexão TCP

Para estabelecer uma conexão, o TCP utiliza o mecanismo de handshake de três vias. Este mecanismo é suficiente para a sincronização correta entre as duas extremidades da conexão.

O primeiro segmento do handshake pode ser identificado porque ele possui um conjunto de *bits* SYN no campo de código. A segunda mensagem possui ambos os conjuntos de *bits* SYN e ACK, indicando que confirma o primeiro segmento SYN. A mensagem final do handshake é apenas uma confirmação e é utilizada simplesmente para informar ao destino que ambos os lados concordam que uma conexão foi estabelecida.

O handshake é cuidadosamente projetado para funcionar mesmo que ambas as máquinas tentem iniciar uma conexão simultânea. Assim, uma conexão pode ser estabelecida de qualquer extremidade ou de ambas, simultaneamente. Quando a conexão tiver sido estabelecida, os dados podem fluir, igualmente bem, nas duas direções.

O handshake de três vias cumpre duas funções importantes. Garante que ambas as extremidades estejam prontas para transferir os dados, e permite que ambas concordem quanto aos números de seqüência iniciais. Os números de seqüências são enviados e confirmados durante o handshake. Cada máquina precisa escolher aleatoriamente um número que usará para identificar os *bytes* no fluxo de dados. Os números de seqüência escolhidos por cada uma das extremidades não precisam ser os mesmos, ou seja, cada extremidade é livre para escolher o número de seqüência que desejar, a seguir, na figura 2.8, é exemplificado uma conexão TCP sendo estabelecida.

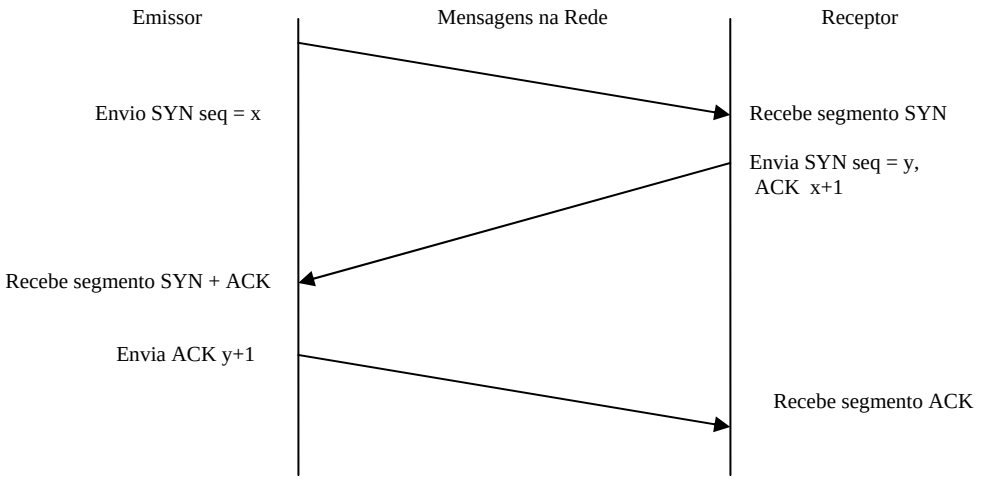

Figura 2.8– Estabelecimento de uma conexão

2.9.2- Encerramento de Conexão

Dois programas que utilizam o TCP para se comunicar podem facilmente encerrar a comunicação, usando a operação close. Internamente, o TCP usa um handshake de três vias, modificado para encerrar as conexões, pois as conexões TCP são *full duplex* e as transferências de fluxos são independentes. Quando um programa aplicativo diz ao TCP que não há mais dados a serem enviados, o TCP escolherá a conexão em uma direção. Para encerrar a metade de uma conexão, o transmissor completa a transmissão dos dados restantes, espera que o receptor confirme-os e, a seguir, envia um seguimento com o conjunto de *bits* FIN. O TCP receptor confirma o segmento FIN e informa ao seu programa aplicativo que não há mais dados disponíveis.

Quando uma conexão tiver sido concluída em determinada direção, o TCP se recusará a aceitar mais dados para esta direção. Mas, os dados podem continuar a fluir na direção oposta até que o transmissor encerre a conexão. É claro que as confirmações continuarão a fluir de volta para o transmissor, mesmo depois que uma conexão tiver sido encerrada. Quando ambas as direções tiverem sido concluídas, o *software* TCP utilizará um handshake de três vias modificado para encerrar a conexão.

A diferença entre os handshake de três vias, usados para estabelecer e encerrar conexões, ocorre depois que uma máquina recebe um segmento FIN inicial. Em vez de gerar um segundo segmento FIN, imediatamente, o TCP envia uma confirmação e, a seguir, informa ao aplicativo sobre a solicitação a ser encerrada. Informar ao programa aplicativo sobre a solicitação e obter uma resposta, pode levar um tempo considerável. A

confirmação evita a retransmissão do segmento FIN inicial durante a espera. Finalmente, quando o programa aplicativo instrui o TCP para encerrar a conexão, o TCP envia o segundo segmento FIN e o site original responde com uma terceira mensagem, uma ACK. A figura 2.9, a seguir, exemplifica o término de uma conexão.

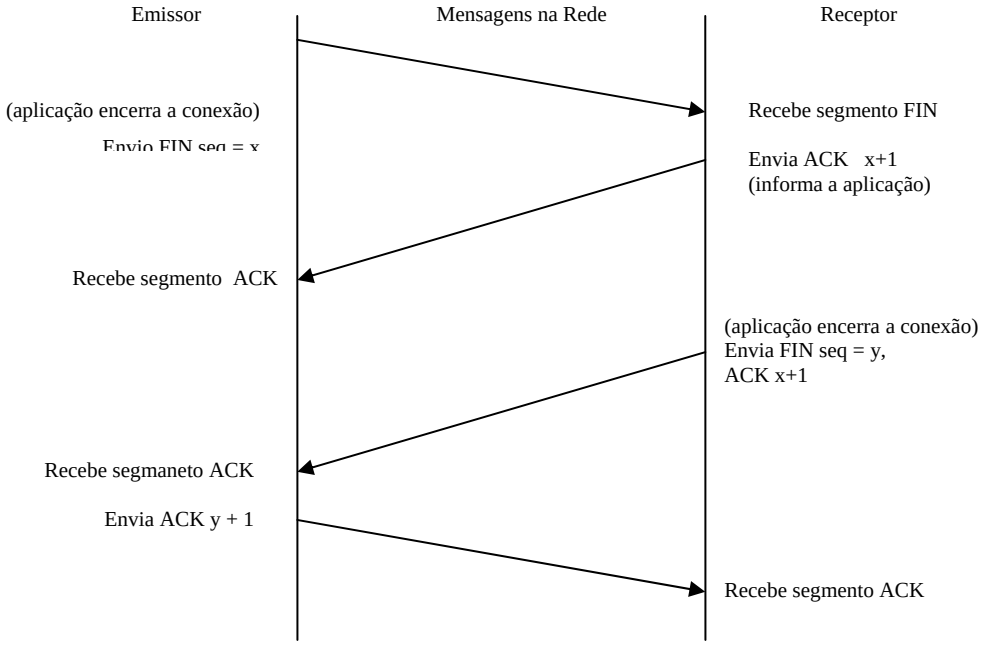

Figura 2.9 – Encerramento de uma conexão nos dois sentidos.

3 –Mecanismos de Controle de Congestionamento do Protocolo TCP

Neste capítulo serão abordados os Mecanismos do TCP usado para o controle de congestionamento. Os algoritmos utilizados pelo TCP para esta tarefa, serão apresentados neste capítulo, assim como as várias implementações TCP e suas principais características. As RFCs que definem estas funcionalidades são: RFC 896, RFC 2001, RFC 2582, RFC 2018 e RFC 2581.

3.1 - O Colapso do Congestionamento

A condição de congestionamento em uma rede de segmentos como a Internet ocorre quando o desempenho da rede é desgastado pelo excesso de pacotes. Vários fatores podem acarretar o congestionamento. Por exemplo, quando vários fluxos de pacotes chegam em três ou quatro entradas de um roteador, todos os fluxos deverão ser direcionados para a mesma porta de saída, fazendo com que a fila, nesta porta do roteador encha, e se a quantidade de memória para armazenar todos os pacotes for insuficiente os pacotes serão descartados. A adição de memória pode ajudar até certo ponto, mas, segundo Nagle [7] se os roteadores tivessem quantidade de memória infinita, o congestionamento seria pior, e não melhor, isso porque, quando os pacotes chegam à fila, eles já sofreram esgotamento de tempo (*timeout*) no transmissor, fazendo com que segmentos duplicados sejam reenviados e, assim, todos os pacotes serão encaminhados para o próximo roteador, aumentando a carga ao longo do caminho percorrido pelo pacote.

Processadores lentos também podem gerar congestionamento. Se os processadores dos roteadores demorarem a tratar os pacotes, a filas poderão acontecer, mesmo que exista largura de banda disponível no canal. Da mesma forma, canais com baixa largura de banda também podem causar congestionamento. A atualização das linhas sem alteração dos processadores ou vice-versa, sempre ajuda um pouco, mas sempre transfere o gargalo para outro lugar. Geralmente, o problema real é uma incompatibilidade entre partes do sistema e esse problema persistirá até que todos componentes estejam em equilíbrio.

Um fato importante a destacar, é a diferença entre controle de congestionamento e controle de fluxo, já que a diferença é bastante sutil. O controle de congestionamento se baseia na garantia de que a sub-rede é capaz de transportar o tráfego oferecido. É uma questão global, envolvendo o comportamento dos *host*s, de todos os roteadores, do

processamento de operações *store-and-forward* dentro dos roteadores e todos os outros fatores que tendem a reduzir a capacidade de transporte da sub-rede.

O controle de fluxo, no entanto, baseia-se no tráfego ponto-a-ponto entre um transmissor e um receptor. Sua tarefa é garantir que um transmissor rápido não possa transmitir dados continuamente com uma rapidez maior do que o receptor é capaz de tratar. O controle de fluxo quase sempre envolve algum feedback direto do receptor para o transmissor. Assim, o transmissor fica sabendo como tudo está sendo feito na outra extremidade.

A razão para a confusão entre o controle de fluxo e o controle de congestionamento é que alguns algoritmos de controle de congestionamento operam enviando mensagens de volta às diversas origens, informando-as de que devem diminuir a velocidade quando a rede enfrentar problemas. Dessa forma, um *host* pode receber a mensagem “reduzir velocidade”, seja porque o receptor não pode manipular a carga ou porque a rede não é capaz de tratá-la.

O controle de congestionamento do TCP foi introduzido na Internet depois de 1980, por Camionete Jacobson. Anteriormente a esta data, a Internet estava sofrendo do colapso de congestionamento. O mecanismo para o controle de congestionamento do TCP é a principal razão pela qual nós podemos usar, com sucesso, a Internet nos dias de hoje.

O TCP, no início de sua utilização, possuía apenas um mecanismo rudimentar de controle de congestionamento, que não era suficiente para prevenir o congestionamento em roteadores intermediários. Em outubro de 1986, a Internet experimentou o primeiro de uma série de fenômenos que ficaram conhecidos como "colapsos de congestionamento". Durante este período, a taxa de dados entre o Lawrence Berkeley Laboratory e a UC Berkeley (sites separados por 400 metros e dois hops) caiu de 32Kbps para 40bps. Van Jacobson e outros ficaram fascinados por essa queda brusca de um fator maior que 1.000, na taxa de transmissão Este fato motivou uma investigação do problema, como descrito em [3].

A condição conhecida como colapso de congestionamento ocorre porque os pontos terminais de uma conexão utilizam *timeout* e retransmissão. Para os *host*s finais, o congestionamento simplesmente significa um aumento do retardo. Com o aumento do retardo, os *host*s reagem com a retransmissão de datagramas. As retransmissões agravam o congestionamento em vez de amenizar. Se não for verificado, o aumento do

tráfego produzirá um aumento do retardo, que por sua vez ele verá o tráfego, tornando a rede inútil.

O fluxo em uma conexão TCP deveria obedecer ao princípio da conservação de pacotes que se fosse obedecido, colapsos devido ao congestionamento seriam uma exceção ao invés da regra. Contudo, controlar o congestionamento significa encontrar as causas da violação do princípio da conservação e consertá-las.

Para evitar o colapso do congestionamento, o TCP precisa reduzir as taxas de transmissão quando houver congestionamento. Para tratar este problema, o TCP implementa um grupo de mecanismos chamado de controle de congestionamento. O principio fundamental por trás do controle de congestionamento é o ajuste do tamanho da janela de transmissão, do emissor, de tal modo, que além de prevenir a sobrecarga no *buffer* do receptor, também previna a sobrecarga nos *buffer*s dos roteadores intermediários. Para alcançar este princípio, o parâmetro básico utilizado pelo TCP, no controle de congestionamento, é a janela de congestionamento (*congestion window*), *cwnd*, que trata da janela de congestionamento do emissor. O valor da *cwnd* é resultante de cálculos realizados, levando-se em conta, entre outros fatores, o tempo de ida e volta dos segmentos. Outro parâmetro é o *allowed window* (*allowed_wnd*), janela permitida de envio de dados cujo valor é utilizado para transmissão dos próximos segmentos. A expressão básica para definir a *aloowed_wnd* a cada instante é:

$$allowed\_wnd = min\ (\ awnd\ ,\ cwnd\ )$$

Então, se de alguma forma é possível saber o espaço de *buffer* disponível nos roteadores, envolvidos em uma conexão TCP, também é possível configurar a *cwnd* e selecionar a *allowed_wnd* como sendo o mínimo entre a *cwnd* e a janela anunciada pelo receptor (*adversative window*), *awnd* Isto prevenirá a sobrecarga no *buffer* do receptor e nos *buffer*s internos à rede.

O desafio é como saber o espaço disponível nos *buffer*s dos roteadores da rede se os roteadores não participam da camada de transporte, então, não podem usar os segmentos ACK para ajustar a janela. Para superar este problema, o TCP assume que há congestionamento na rede sempre que acontece uma retransmissão e reage ao congestionamento, ajustando a *cwnd*, usando três algoritmos, partida lenta (slow start), prevenção do congestionamento (congestion avoidance) e Incremento aditivo Decremento Multiplicativo (*Additive Increase Multiplicative Decrease* ou AIMD). Muitas modificações

destes algoritmos estão disponíveis atualmente. Nas próximas linhas serão descritos os algoritmos originais.

Proposto por Van Jacobson, os algoritmos de Partida Lenta e Prevenção de Congestionamento permitem que o TCP aumente sua taxa de transmissão sem sobrecarregar a rede. Eles utilizam a variável citada anteriormente, Janela de Congestionamento - *cwnd*. A *cwnd* é do tamanho da janela deslizante, usada pelo transmissor e não pode exceder à janela de anúncio - *awnd* do receptor. Portanto, a *cwnd* é o controle de fluxo imposto pelo emissor, enquanto a *awnd* é o controle de fluxo imposto pelo receptor [3].

3.2-Mecanismo de Controle de Congestionamento TCP

O controle de congestionamento do TCP tem como princípio básico o fato de que o fluxo de segmentos na rede deve ser conservativo. Isto significa que um segmento só pode ser inserido na rede depois que um segmento for retirado. O recebimento de um ACK indica para o transmissor que um segmento deixou a rede. Como o transmissor só pode enviar mais segmentos ao receber ACKs, o protocolo torna-se *self-clocking* [5], ou seja, o protocolo ajusta automaticamente sua taxa de transmissão ao enlace mais lento da rede. A figura 3.1, retirada de [5] mostra este esquema. Esta característica é importante para que o protocolo lide com a heterogeneidade da Internet.

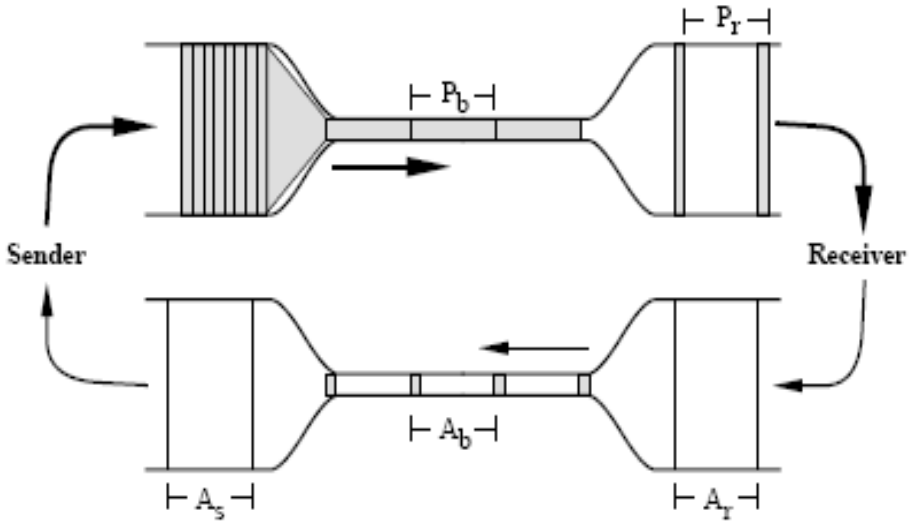

Figura 3.1: Mecanismo *self-clocking* utilizado pelo TCP

A figura 3.1 é uma representação do mecanismo *self-clocking*. Ela representa duas redes de alta velocidade, integradas por um enlace de gargalo, mais lento e com maior atraso. A dimensão horizontal é o tempo e a vertical é a banda passante. A área de cada

retângulo equivale, então, ao produto do atraso pela largura de banda, que é a quantidade de *bits* enviados. Assim, no enlace mais lento, esses *bits* demoram mais para serem transmitidos, levando um tempo Pb, como mostrado. Ao chegar ao enlace mais veloz do receptor, os pacotes ficam mais espaçados pelo intervalo entre as chegadas, que será o tempo Pb. Desta forma, os segmentos de reconhecimento serão gerados num espaçamento de tempo igual a Pb. Como os segmentos de dados são maiores que os respectivos reconhecimentos, esse espaçamento será mantido no enlace de gargalo, sendo Ab= Pb. Como novos segmentos só serão enviados com o recebimento dos reconhecimentos, e a taxa deste recebimento é igual à taxa do enlace mais lento, fica demonstrado que a taxa do emissor será igual à taxa do enlace mais lento, no caso, o de gargalo, mais detalhes ver [5].

A condição anteriormente citada, também é válida para uma conexão no estado estacionário. Logo, há a necessidade de um algoritmo que determine o comportamento transiente do TCP, desde o início da conexão até que se atinja o equilíbrio. Este algoritmo deve ser agressivo o suficiente para atingir o ponto de equilíbrio o mais rápido possível e, ao mesmo tempo, conservador o suficiente para manter a estabilidade da rede, evitando que se inunde as filas dos nós intermediários com rajadas de dados muito maiores que elas possam absorver.

3.3 - Partida Lenta (Slow Start)

Este algoritmo é usado para aumentar a quantidade de dados sem a confirmação de recebimento que o TCP injeta na rede, através do aumento gradual do tamanho da janela de transmissão. Ele inicia a transmissão com uma pequena janela e aumenta lentamente a cada confirmação. Para testar o espaço disponível no *buffer* na rede.

O termo, partida lenta, pode ser uma denominação equívoca porque, em condições ideais, o início não é tão lento. O TCP inicializa este algoritmo no início de uma conexão e, em certas circunstâncias, após a detecção de um congestionamento. A janela de transmissão é inicializada em um segmento, que é enviado e aguarda a confirmação. Quando a confirmação chega, o algoritmo aumenta a janela de transmissão para dois, envia novamente os dois segmentos e aguarda. Quando as duas confirmações chegam, cada uma delas aumenta a janela em um e, assim, o TCP pode enviar quatro segmentos. Este algoritmo habilita um crescimento exponencial no tamanho da janela. Mesmo para janelas extremamente grandes, ele leva apenas tempos de ida e volta iguais ao log2N, antes que o TCP possa enviar N segmentos. A Partida Lenta continua até que o tamanho

da janela alcance o valor do Limiar da Partida Lenta (*Slow-Start Threshold* ou *ssthresh*) ou quando a perda de um segmento de dados é detectada, quando então é encerrada.

O valor da janela de transmissão é inicializada no valor do MSS. O valor do MSS é baseado no MSS do receptor, obtido durante o estabelecimento da conexão TCP, na Unidade Máxima de Transferência (*Maximum Transfer Unit* ou MTU) do caminho da conexão, no MTU da interface do transmissor; ou na ausência de outra informação, 536 *bytes* que é o valor padrão.

O outro limite para o aumento da janela de transmissão, durante a Partida Lenta, é mantido pela variável *ssthresh*. Inicialmente o valor de *ssthresh* é colocado para o valor correspondente ao tamanho máximo da janela do receptor. Entretanto, quando um congestionamento é percebido, a variável *ssthresh* é reduzida para a metade da janela atual, fornecendo ao TCP uma memória do ponto onde ele poderá antecipar o congestionamento da rede no futuro.

O aumento da janela de transmissão, durante a fase de Partida Lenta, é interrompido quando a janela de transmissão exceder à *awnd* do receptor, ou seja, se o valor aumentar, passando do valor do congestionamento memorizado em *ssthresh* ou quando estiver além da capacidade da rede, o controle de fluxo do TCP é mudado do algoritmo de Partida Lenta para o algoritmo de Prevenção de Congestionamento.

Isto é necessário, pois quando a taxa de envio é maior que o nível que pode ser sustentado pela rede, segmentos de dados podem ser descartados por transbordamento de *buffer*. O TCP pode detectar a perda de segmentos de dois modos. Primeiro, se um único segmento é perdido dentro de uma seqüência de segmentos, a entrega bem sucedida dos segmentos posteriores ao segmento perdido fará que o receptor gere um ACK duplicado para cada entrega bem sucedida. A chegada destes ACKs duplicados é um sinal de que houve perda de um segmento. Segundo, se um segmento é perdido no final de uma seqüência de segmentos enviados, não há segmentos consecutivos para a geração de ACKs duplicados. Neste caso, não há a geração do correspondente ACK deste segmento. Em conseqüência, o Temporizador de Retransmissão (*Restransmit Timer* ou RTO) do transmissor irá expirar e o transmissor assumirá a perda do segmento.

Em um protocolo baseado em ACKs, o transmissor é responsável pela detecção de perdas de segmentos. Segmentos perdidos são revelados pelas falhas na ordem dos números de seqüência, com confirmação de recebimento, devido à ausência de ACKs.

As figuras 3.2 e 3.3 retiradas de [5] mostram o comportamento da rede sem e com o algoritmo partida lenta. A falta do algoritmo resulta no comportamento oscilatório da

rede, já que várias retransmissões são feitas, reduzindo o desempenho do protocolo. A introdução faz com que a transmissão atinja o equilíbrio.

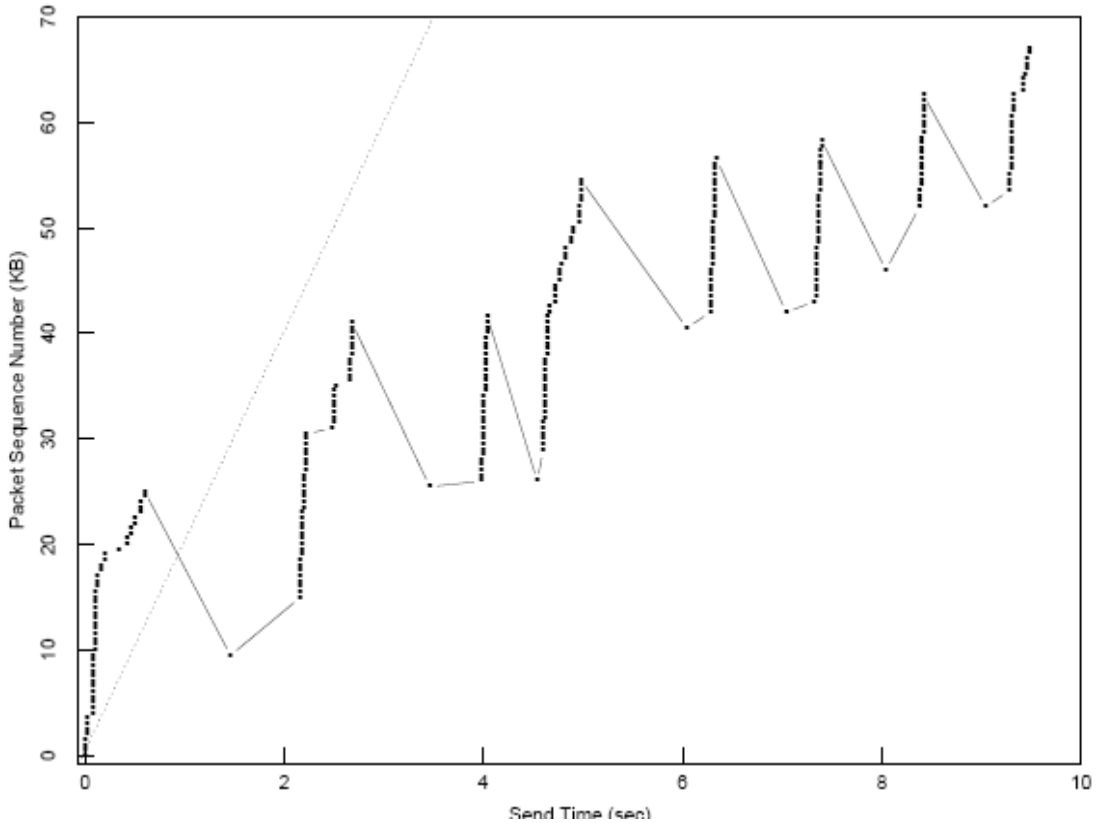

Fig.3.2 Comportamento inicial do TCP sem o algoritmo partida lenta

Cada ponto é um segmento de 512 *bytes*. O eixo x é o momento em que o segmento foi enviado, o eixo y é o número de seqüência do segmento. Logo, dois pontos com o mesmo valor de y e valores de x diferentes, indica uma retransmissão. A linha pontilhada representa a vazão máxima do canal, que corresponde a 20kBps. O comportamento desejável seria a linha pontilhada seguindo uma diagonal sobre os pontos.

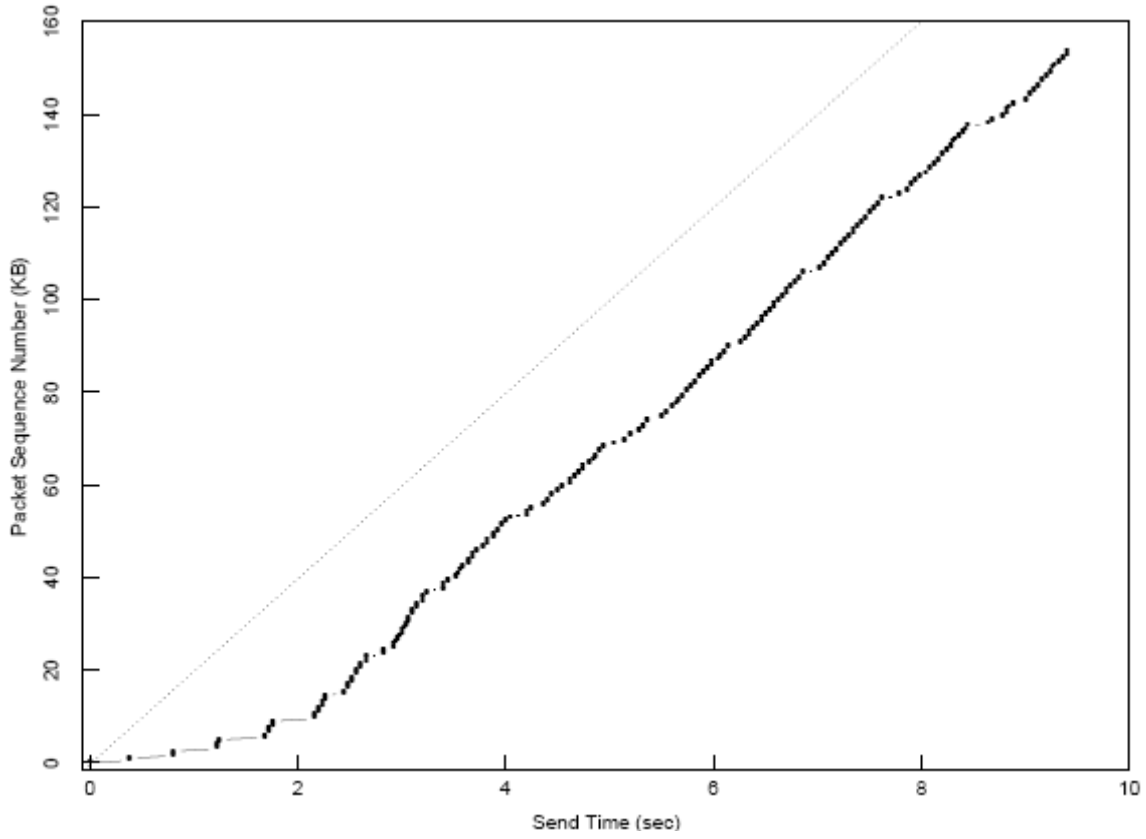

Fig.3.3 Comportamento inicial do TCP com o algoritmo partida lenta

Com as mesmas condições que o gráfico anterior, esta figura mostra que não houve nenhuma retransmissão, e assim, não houve desperdício de largura de banda. O comportamento deste gráfico é o desejável, já que a linha pontilhada segue uma diagonal em cima dos pontos, mais detalhes ver [5].

3.4 - Prevenção do Congestionamento (Congestion Avoidance)

Conforme dito anteriormente, o algoritmo Partida Lenta incrementa a janela de transmissão exponencialmente, como ilustrado na figura 3.4. Em um determinado ponto, a capacidade da rede será ultrapassada, provocando perda de segmentos. Isto indica que a janela de congestionamento ultrapassou o ponto de equilíbrio, a partir daí, o protocolo passa a buscar o seu ponto de equilíbrio de forma mais conservadora. O algoritmo que implementa esta etapa do controle de congestionamento chama-se Prevenção de Congestionamento.

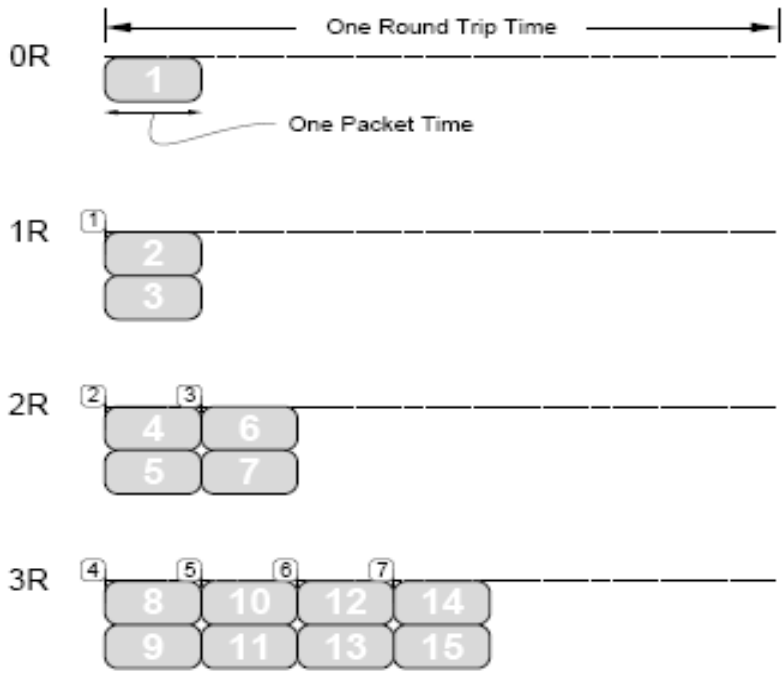

Fig.3.4 Algoritmo Partida Lenta

O algoritmo Prevenção do Congestionamento manipula a janela de congestionamento, seguindo a regra de incremento aditivo e decremento multiplicativo (*AIMD- additive increse - multiplicative decrease*), que será citado a seguir. Em situações de congestionamento (sinalizadas por perda de segmentos), a janela é reduzida de forma exponencial. Do contrário, a janela aumenta linearmente.

O AIMD é à base do Controle de Congestionamento do TCP. No estado estacionário em uma conexão não-congestionada, o TCP, a cada confirmação de recebimento, incrementa em um segmento a *cwnd* e corta pela metade a janela, a cada janela de dados, contendo um segmento perdido. Reduzir a *cwnd* reduz o tráfego que o TCP irá injetar na conexão. Para estimar o tamanho da *cwnd*, o TCP supõe que a maioria das perdas de datagramas advém de congestionamentos e por isso utiliza o Decremento Multiplicativo para prevenir um possível congestionamento, ou seja, reduz a *cwnd* pela metade, sempre que um segmento for perdido.

Isto justifica-se pelo seguinte: pode-se demonstrar que, quando a rede está congestionada, as filas nos roteadores aumentam exponencialmente. Portanto, a redução na taxa de envio também deve ser exponencial, de modo a permitir que as filas se estabilizem. Por outro lado, a rede não indica ao transmissor que ele está subutilizando a largura de banda disponível, logo, deve haver uma maneira de aumentar a taxa de

transmissão, de modo que se possa descobrir o limite atual da largura de banda. Um aumento exponencial, por mais lento que fosse, em um intervalo de tempo constante, faria com que a janela aumentasse muito rapidamente, causando instabilidade na rede.

A taxa adotada para decremento multiplicativo é de 0,5. Este número foi escolhido de acordo com o seguinte raciocínio: quando se verifica uma situação de congestionamento, o transmissor pode estar iniciando a conexão, ou seja, está na fase de Partida Lenta, ou estar no estado estacionário. No primeiro caso, em que o segmento é perdido na fase Partida Lenta, reduzir a janela à metade significa voltar à janela para o tamanho anterior, ou seja, para um tamanho que não provocou perdas. A partir deste ponto o aumento da janela passa a ser mais lento, buscando o equilíbrio. Na situação de uma conexão em estado estacionário, a perda de um segmento indica que a largura de banda disponível para a conexão diminuiu, provavelmente, porque outra conexão foi iniciada e está disputando recursos da rede. Supondo que a primeira conexão estivesse sozinha e que agora divida a banda com uma outra, é correto reduzir a janela pela metade, pois a banda passante foi reduzida nesta proporção. Este é o pior caso, no estado estacionário, já que, se houvessem duas conexões e uma terceira fosse iniciada, a redução da largura de banda seria de 1/3, e assim por diante. A redução pela metade é mais conservadora nestes casos, ajudando a garantir a estabilidade da rede.

O incremento aditivo é de 1/*cwnd*, para cada ACK recebido. Isto se traduz em um incremento de no máximo um segmento por RTT. Como o TCP é orientado a *bytes*, e não a segmentos, este incremento será de MSS*MSS/*cwnd*, que incrementará a janela de transmissão, no máximo, um MSS a cada RTT.

No entanto, é importante observar que esta é uma aproximação da realidade, já que, os ACKs não são recebidos no mesmo instante. Desta forma, a janela de transmissão, *cwnd*, aumenta gradualmente com o recebimento dos reconhecimentos, com uma taxa de aproximadamente um segmento por janela de ACKs recebidos.

O algoritmo de Prevenção de Congestionamento é realizado da seguinte forma:

1- quando ocorrer um estouro de temporização (*timeout*), *cwnd* recebe *cwnd*/2;
2- Para cada novo ACK recebido, desde que este ACK não seja duplicado, incrementa-se *cwnd* de 1/*cwnd*;
3- A janela de transmissão é a menor entre a janela anunciada (*awnd*) e *cwnd*.

Na prática, os algoritmos Partida Lenta e Prevenção do Congestionamento são implementados em conjunto. A fusão dos dois algoritmos é bastante simples. Além da

cwnd, que os dois usam, adiciona-se a variável *ssthresh*, ao limite da fase de Partida Lenta, o que determina qual dos dois algoritmos está ativo.

A implementação em conjunto fica:

1. Quando ocorrer um *timeout*, *ssthresh* recebe *cwnd*/2;
2. Se *cwnd* < *ssthresh*, a janela é manipulada pelo algoritmo Partida Lenta;
3. Se *cwnd* ≥ *ssthresh*, a jenela é manipulada pelo algoritmo Prevenção do Congestionamento.

De maneira simples, o controle de congestionamento do TCP pode ser expresso pelas equações descritas no quadro 5.

Partida lenta

ACK : CWND ← CWND + c

Prevenção Congestionamento

ACK : CWND ← CWND + a / CWND

DROP : CWND ← CWND – b x CWND

os termos *cwnd*, a e c são definidos em unidades de MSS. Os valores canônicos para a, b e c são: a = 1, b = 0.5 e c = 1.

Quadro 5

Com isto, podemos concluir que a característica geral da fusão dos algoritmos do TCP é de uma busca rápida inicial da capacidade da rede, ou seja, para estabelecer aproximadamente os seus limites de eficiência máxima, seguida de um comportamento adaptativo cíclico, que reage rapidamente a um congestionamento e incrementa lentamente a taxa de envio perto da área de máxima eficiência de transmissão. A perda de segmento, quando sinalizada pelo acionamento do RTO, faz com que o transmissor recomece a fase de Partida Lenta, após um intervalo de *timeout*.

Entretanto, desde o trabalho inicial de Van Jacobson, em 1988, foram incorporadas algumas melhorias importantes no controle de congestionamento TCP, que

afetam diretamente o seu comportamento em conexões de alta velocidade. Estas melhorias são as seguintes:

3.5 –Retransmissão Rápida (Fast Retransmit) e Recuperação Rápida (Fast Recovery)

Cada segmento que chega com sucesso ao seu destino desencadeia a geração de uma mensagem de reconhecimento, a qual é transmitida pelo canal reverso. O objetivo dessa mensagem é reportar ao transmissor o recebimento de um novo segmento e, garantir o auto-ajuste, explicado em [3]. No entanto, o teor dessa mensagem depende do esquema de reconhecimento adotado. No caso especifico do TCP, adotou-se, conforme já mencionado, o esquema de reconhecimento cumulativo.

No esquema de reconhecimento cumulativo, uma particular mensagem de reconhecimento informa o número do próximo segmento a ser recebido pelo receptor. A tabela 1 mostra o funcionamento deste esquema de reconhecimento. Supondo que em uma conexão estabelecida os 10 (dez) primeiros segmentos chegaram corretamente ao seu destino, em seqüência, o conteúdo das mensagens de reconhecimento informam o número do próximo segmento, por exemplo, com o recebimento do sétimo segmento será gerada uma mensagem de reconhecimento para o oitavo segmento. No entanto, supondo que o décimo primeiro segmento chegou com erro no receptor, o seu recebimento não gera uma mensagem de reconhecimento. Supondo que o décimo segundo segmento chega com sucesso ao receptor, será gerada uma mensagem de reconhecimento. Contudo, como este segmento chegou "fora de ordem", o conteúdo da mensagem de reconhecimento indica 11 ao invés de 13, tendo em vista que o segmento em ordem que está sendo aguardado pelo receptor é o décimo segundo. Este evento é comumente denominado reconhecimento duplicado.

A adoção do esquema de reconhecimento cumulativo, apesar de possuir o inconveniente apresentado acima, possui diversos atrativos, principalmente no que se refere à simplicidade de implementação e a robustez quanto às perdas das mensagens de reconhecimento.

O reconhecimento duplicado é utilizado pelo algoritmo Retransmissão Rápida como indicação de segmento perdido. Esse algoritmo tem como objetivo detectar a perda de segmento com maior rapidez do que a permitida com o procedimento convencional, além de evitar a redução drástica da taxa de transmissão.

Número de segmentos transmitidos	Fluxo de segmento na rede	Número de segmentos notificados na mensagem de reconhecimento
...	...	...
7	Sucesso	8
8	Sucesso	9
9	Sucesso	10
10	Sucesso	11
11	Perdido	Não é gerado mensagem
12	Sucesso	11
13	Sucesso	11
14	Sucesso	11
11	Sucesso	15
15	Sucesso	16

Tabela 1

Na tabela os valores em vermelho são os reconhecimentos duplicados;

E os valores em azul são os reconhecimentos dos blocos que já tinham sidos recebidos com sucesso, no entanto, não notificados por causa da perda do décimo primeiro segmento.

Devido ao comportamento aleatório do RTT, a duplicação de um reconhecimento não implica, necessariamente, na perda de um segmento. No entanto, a probabilidade do segmento ter sido perdido aumenta à medida que novos reconhecimentos repetidos chegam ao transmissor. Convencionou-se admitir que, com 3 reconhecimentos duplicados, o segmento foi perdido, sendo uma réplica sua reenviada mesmo que não seja transcorrido o tempo RTO.

Existe uma heurística bastante simples para a utilização do algoritmo Retransmissão Rápida e que vale a pena ser mencionada. Se o transmissor continua recebendo mensagens de reconhecimentos, o fluxo de segmentos na rede está sendo mantido com boa velocidade. Portanto, é bem provável que a conexão não esteja congestionada. Neste caso, existe uma grande probabilidade que tenha ocorrido um problema isolado com o segmento, com por exemplo, erros na informação nele contida. Sob este ponto de vista, deve-se transmitir o segmento recebido com problemas, sem, no entanto, reduzir a taxa de transmissão. O algoritmo Retransmissão Rápida procura implementar este procedimento, reenviando o segmento caso seja recebido três reconhecimentos duplicados em um intervalo de tempo inferior a RTO.

A fim de estimar a quantidade de dados em circulação na rede, algumas versões atuais do protocolo TCP incluem o algoritmo Recuperação Rápida em conjunto com o algoritmo Retransmissão Rápida, para ajustar o tamanho da *cwnd*, imediatamente após a ação do algoritmo Retransmissão Rápida. O algoritmo Recuperação Rápida é descrito a seguir.

Quando o terceiro reconhecimento duplicado chega ao transmissor, a variável *ssthresh* recebe o valor correspondente a metade do valor contido na variável *cwnd*, o segmento é reenviado e, logo em seguida, a variável *cwnd* é atualizada de acordo como mostra o quadro 6.

cwnd = *ssthresh* + 3

Quadro 6

onde o número três representa a quantidade de reconhecimentos duplicados. Cada vez que um novo reconhecimento duplicado chega, o que pode ocorrer em função de segmentos que já haviam sido transmitidos e que chegaram corretamente ao seu destino, a variável *cwnd* é incrementada por um segmento. Quando não chega mais reconhecimentos duplicados, a variável *cwnd* é igualada ao valor da variável *ssthresh*. O aumento da taxa de transmissão fica agora vinculado ao algoritmo Prevenção do Congestionamento.

3.6 - Retardo de Confirmações (Delay Acknowledment)

Outra modificação é o Retardo de Confirmações, que permite que o receptor envie um ACK cumulativo de volta para o transmissor, após ter recebido um número pré-definido de segmentos, ao invés de ter de reconhecer o recebimento de cada segmento.

Esta modificação passou a evitar a *Síndrome da Janela desnecessária* [1], já que as implementações anteriores do TCP apresentavam este problema que ocorre, porque o TCP armazena os dados de entrada em *buffer.* No entanto, se o receptor decidir ler um octeto por vez, um octeto de espaço estará disponível em seu *buffer*, e assim, irá gerar uma confirmação que utiliza o campo da janela para informar ao transmissor. Quando o transmissor receber a notificação da janela, responderá com um segmento que contém um octeto de dados.

Há vantagens e desvantagens em retardar as confirmações. A principal vantagem surge porque os retardos entre as confirmações podem reduzir o tráfego e, assim, aumentar o throughput. As desvantagens dos retardos entre as confirmações é que se o

receptor fizer retardos muito longos entre as confirmações, o TCP transmissor retransmitirá o segmento. As retransmissões desnecessárias reduzem o throughput porque consomem largura de banda da rede. Além disso, elas exigem um overhead computacional nas máquinas transmissoras e receptoras. O TCP também utiliza a chegada de confirmações para estimar os tempos de ida e volta, e retardos entre confirmações podem confundir a estimativa e tornar os períodos de retransmissão muito longos.

Para evitar problemas potenciais, os padrões TCP colocam um limite no tempo de retardo de transmissão, para uma confirmação. As implementações não podem retardar uma confirmação por mais de 500ms. Também, para assegurar que o TCP receba um número suficiente de estimativas do tempo de ida e volta, o padrão recomenda que um receptor confirme, pelo menos, um em dois segmentos de dados.

3.7- Opção de Reconhecimento Seletivo

O TCP tradicional realiza um esquema de reconhecimentos cumulativos, como citado anteriormente, em que ACK(n) indica que todos os segmentos até (n - 1) foram recebidos com sucesso e o transmissor pode transmitir o segmento (n). Isto impõe uma grave limitação de desempenho, em situações nas quais há mais de uma perda em uma mesma janela de transmissão, porque o transmissor tem que esperar um RTT para identificar cada segmento perdido. Isto ocorre porque, caso haja diversas perdas, os reconhecimentos são enviados, referenciando o primeiro segmento perdido, não dando maiores informações sobre as demais perdas ao emissor. Assim, o emissor reenvia apenas o segmento referenciado pelos ACKs duplicados, tendo que aguardar por um RTT a chegada do reconhecimento do segmento retransmitido, que estará indicando o próximo segmento perdido. Desta forma, apenas um segmento perdido pode ser notado a cada RTT. A alternativa para esta limitação seria retransmitir mais do que o necessário, como por exemplo, todos os segmentos após aquele primeiro que foi perdido. Medidas desse tipo, entretanto, além de ineficientes podem acabar agravando uma situação de congestionamento na rede.

Uma solução para este problema seria o uso de reconhecimentos seletivos (*selective acknowledgement,* ou *SACK*) por parte do TCP. Com esta modificação o emissor tem mais informações sobre quais segmentos foram recebidos corretamente e quais não foram, podendo, portanto, retransmitir somente os segmentos necessários, aumentando a eficiência na recuperação de perdas.

Esta modificação usa o campo OPTIONS (opções) do cabeçalho TCP para incluir informações de reconhecimento seletivo. Existem dois tipos de campos opcionais que são usados pelo TCP SACK: um para estabelecer se a opção de SACK será ou não usada, que é enviada ao iniciar uma conexão, e o SACK propriamente dito, que passa informações sobre os segmentos reconhecidos. A figura 3.5 ilustra a estrutura deste último campo.

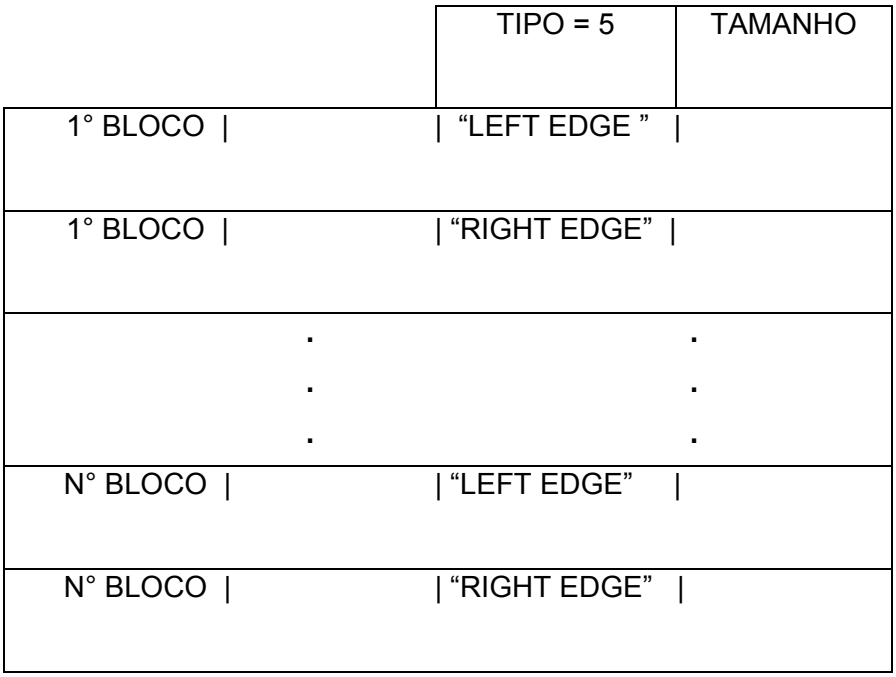

Figura 3.5: Campo de reconhecimento seletivos, o SACK

Como se vê na figura, o receptor envia informações sobre blocos de dados que foram recebidos e que se encontram isolados. O *LEFT EDGE* (*left edge of block*, borda esquerda do bloco) é o número de seqüência do primeiro byte deste bloco, enquanto *RICHT EDGE* (*right edge of block*; borda direita do bloco) é o número de seqüência seguinte ao último byte deste bloco. Isto mostra uma analogia com o reconhecimento normal, que tem o número seguinte ao último recebido. É importante frisar que o campo ACK do cabeçalho mantém o significado, ou seja, ele tem o número do primeiro segmento que não foi recebido em seqüência. Quando o receptor preenche as lacunas anteriormente expressas no campo de SACK, ele avança normalmente o número do ACK para denotar um conhecimento cumulativo.

O campo de opção do TCP pode conter, no máximo, 40 *bytes*. O campo SACK que especifique n blocos terá 8*n + 2 *bytes*, o que dá um limite teórico de quatro blocos

que podem ser reconhecidos. Como na prática outras opções são usadas, o número de blocos em geral, fica limitado a três.

3.7.1- Comportamento do Receptor

O objetivo da opção de reconhecimento seletivo é que o transmissor tenha mais informação sobre quais segmentos foram recebidos, de modo a definir uma estratégia de retransmissão mais eficiente. Portanto, existem regras bem definidas para geração dos SACKs, por parte do receptor, para que o transmissor tenha o retrato mais recente e fiel do estado do *buffer* do receptor.

Ao gerar um SACK, o receptor deve obedecer às seguintes regras:

1. O primeiro bloco deve conter o segmento que disparou este ACK (a menos que o último segmento tenha dado origem a um ACK normal), ou seja, o primeiro bloco mostra a mudança mais recente no estado do *buffer* do receptor;
2. O receptor deve incluir o maior número possível de blocos;
3. Os blocos de reconhecimento seletivo devem ser preenchidos do mais recente (primeiro bloco) para o mais antigo; desse modo, em um bloco de dados será incluído no mínimo, três ACKs, o que agrega redundância em caso de perda de ACKs.

Com relação à última regra, deve-se notar que mesmo que sejam perdidos todos os ACKs que reportem um bloco de SACK, a pior conseqüência que isso pode ter é a retransmissão desnecessária dos segmentos pertencentes ao bloco. Como sem o SACK essas retransmissões desnecessárias seriam feitas de qualquer modo, o comportamento do TCP com SACK, no pior dos casos, é igual ao dos normais.

O receptor mantém um *buffer* com os segmentos recebidos à espera de que sejam lidos pela aplicação. Quando isso acontece, os dados lidos são apagados do *buffer*. Em caso de falta de espaço em *buffer*, o receptor pode descartar segmentos que tenham sido reconhecidos através da opção de SACK. Neste caso, o próximo ACK gerado deve ter no primeiro bloco a mudança mais recente nos segmentos reconhecidos, conforme as regras já citadas.

3.7.2- Comportamento do Transmissor

No TCP tradicional, o transmissor mantém um *buffer* de retransmissão com os segmentos que foram enviados e ainda não foram reconhecidos. Ao receber um ACK, ele

descarta todos os segmentos cujos números de seqüência sejam menores que o número indicado no ACK.

O reconhecimento seletivo estabelece que para cada segmento em seu *buffer* o transmissor tenha um *flag* “SACKed”, que indica que o segmento foi reportado em um bloco de SACK. Em outras palavras, quando o transmissor recebe um ACK com reconhecimentos seletivos, ele deve colocar em um este *flag* para todos os segmentos que estejam dentro dos blocos especificados na opção de SACK. Deste modo, quando houver retransmissões, somente os segmentos que tiverem o *flag* SACKed em 0 são passíveis de retransmissões.

Quando houver um *timeout*, o transmissor deve zerar todos os *flags* “SACKed” da fila de retransmissão, pois isto pode indicar que o receptor descartou os segmentos anteriormente reconhecidos. Neste caso, o transmissor deve retransmitir o segmento indicado no último ACK, independente do estado dos *flags*.

Os segmentos que são reconhecidos seletivamente não podem ser apagados do *buffer* de retransmissão, porque o receptor pode descartá-los. Um segmento só é apagado quando for reconhecido por um ACK normal.

3.8 - Problemas de desempenho do protocolo TCP

Em redes de alta velocidade, o desempenho das aplicações TCP é sempre limitado pela capacidade dos sistemas finais de gerar, transmitir, receber e processar dados sobre a velocidade da rede [5].

Nos próximos parágrafos, serão apresentados os principais problemas enfrentados pelo TCP, para alcançar um alto desempenho em transmissões volumosas de dados em ligações de alta velocidade.

A introdução de tecnologias de redes de alta velocidade tem causado uma mudança dramática no desempenho das aplicações baseadas em TCP. Sabe-se que o desempenho do TCP depende da velocidade da rede, do tempo de percurso e da perda de segmentos. Quando a velocidade de uma ligação aumenta, a possibilidade de alcançar o produto da largura de banda pelo atraso diminui.

Os enlaces de longa distância e alta velocidade possuem algumas características peculiares e importantes para a determinação do comportamento de uma conexão TCP. Obter uma boa compreensão dos efeitos provocados por estas características é importante para estabelecer ou desenvolver algoritmos eficientes para este ambiente. Dentre estas características, destaca-se o enorme atraso de transmissão provocado pela grande distância que separa os interlocutores responsáveis pela conexão. O elevado

atraso de transmissão e a alta velocidade dos canais determina enormes tempos de realimentação, induzindo um uso ineficiente da largura de banda do canal, além de prejudicar aplicações em tempo real.

Antes de expor estes problemas, é necessário introduzir formalmente o conceito de produto da largura de banda pelo atraso. A capacidade de uma conexão é normalmente medida em termos de produto da largura de banda pelo atraso (*Bandwidth Delay Product* ou BDP). Isto é expresso no quadro 7.

Capacidade(*bits*) = banda(*bits*/seg) × atraso(seg)

Quadro 7

O atraso em uma rede é a latência de ida e volta necessária para a informação propagar-se do nó transmissor para o receptor. Banda é o número de *bits* que podem ser transmitidos em um certo período de tempo. BDP é o produto do atraso pela largura de banda, isto é, o número de *bits* (ou *bytes*) que uma rede pode suportar. É possível imaginar a rede como uma tubulação, cujo comprimento representa o tempo de latência de ida e volta da rede e a espessura representa a largura de banda da conexão. Desta forma, BDP representa o volume da tubulação.

Nas camadas de transporte e enlace, o BDP representa a quantidade máxima de dados sem confirmação de recebimento pendentes na rede, em qualquer momento, para manter a conexão cheia. O desempenho do TCP não depende da própria taxa de transferência, mas do produto da taxa de transferência pelo atraso de ida e volta. O BDP, portanto, mede a quantidade de dados que pode encher a *tubulação*. Quanto maior for o BDP, mais tempo levará para o TCP utilizar a capacidade disponível.

A quantidade de dados que uma conexão deve manter em tráfego, para utilizar eficientemente a capacidade de um canal, é dada pelo produto entre a largura de banda e o RTT. Para enlaces onde este produto é extremamente elevado em função do valor do RTT e da largura de banda, o TCP precisa manter uma grande quantidade de dados no enlace para aproveitar os recursos disponibilizados da melhor forma possível. O que é difícil de obter, em decorrência do limite imposto pelo próprio TCP, no tamanho da janela deslizante.

Os algoritmos Partida Lenta e Prevenção do Congestionamento, de um modo geral, apresentam problemas nas rotas que possuem as características previamente

citadas. O valor elevado do BDP compromete a eficiência dos enlaces. Como já foi discutido neste trabalho, no início de uma conexão TCP, ou durante, após a perda de um segmento, a taxa de transmissão é reduzida e o seu aumento é condicionado ao RTT. Para uma janela de transmissão aumentar em W segmentos, em uma rede com RTT igual a T e utilizando-se o algoritmo Partida Lenta, gasta-se um tempo, como é mostrado no quadro 8.

t= Tlog 2 W

Quadro 8

O algoritmo Prevenção do Congestionamento sofre dos mesmos problemas. Durante o tempo em que o tamanho da janela de congestionamento está aumentando, seja pela ação do algoritmo Partida Lenta, ou pelo algoritmo Prevenção do Congestionamento, recursos da rede estão sendo desperdiçados.

Além dos problemas enumerados, numa conexão em que todos os segmentos são reconhecidos em tempo hábil e a janela de congestionamento opera com a quantidade máxima de segmentos, ainda existe uma restrição na vazão provocada pelo limite, no tamanho da janela de transmissão e pelo RTT. Este limite pode ser expresso matematicamente como segue no quadro 9.

max_vazão = *advertised* / RTT

Quadro 9

Em suma, o limite do tamanho da janela de transmissão (65535 *bytes*) imposto pelo TCP, pelos elevados valores de RTT e a ação conservadora dos algoritmos Partida Lenta e Prevenção do Congestionamento, determinam uma ineficiente utilização do enlace com um alto valor do BDP. Diversas modificações nesses algoritmos têm sido sugeridas, bem como novos esquemas têm sido propostos com o intuito de amenizar o impacto do RTT elevado e contornar os problemas apresentados pelos algoritmos Partida Lenta e Prevenção do Congestionamento. Alguns destes mecanismos são discutidos mais adiante.

Atualmente, as implementações do TCP somente podem alcançar grandes janelas de congestionamento, quando houver uma taxa de perda de segmentos muito baixa. Perdas randômicas acarretam uma significativa deterioração da capacidade de

transmissão quando o produto da probabilidade de perda pelo quadrado do produto banda atraso for maior que um [22]. Por exemplo, para que uma transmissão com TCP padrão, de pacotes com tamanho de 1500 *bytes* e tempo de percurso de 100ms, possa atingir uma taxa de transmissão de 10Gbps seria necessária uma janela de congestionamento média de 83.333 segmentos e uma taxa de descarte de pacotes de, aproximadamente, um evento de congestionamento a cada 5.000.000.000 pacotes, ou equivalentemente a, no máximo, um evento de congestionamento a cada 100 minutos, conforme demonstrado em [21]. Isto está muito além do que é possível hoje em dia, com a atual tecnologia de fibras óticas e de roteadores.

O protocolo TCP em estado estacionário, ou seja, com uma baixa taxa de perda de pacotes, *p*, a janela de congestionamento média do TCP é aproximadamente 1.2/sqrt(*p*) segmentos [21]. Quanto maior o RTT, o tempo entre uma perda de segmento e a próxima, seria ainda maior. Este fato, em um ambiente real, coloca uma série de restrições na janela de congestionamento.

O protocolo TCP aumenta sua janela de congestionamento em um segmento a cada reconhecimento e diminui pela metade a cada janela de dados, contendo um descarte, seguindo o algoritmo clássico do AIMD. Na fase prevenção de congestionamento, seu comportamento é expresso pelas equações no quadro 6, já discutidas no capítulo 3.

O número de tempos de percurso entre os eventos de congestionamento, requeridos para um fluxo do TCP atingir uma taxa de transferência média alta, aumenta diretamente com a banda disponível.

O número de tempos de percurso entre eventos de congestionamento, para um fluxo TCP com tamanho de segmento D *bytes*, em função da taxa de transmissão média da conexão B em *bits*/seg, pode ser calculado através da janela de congestionamento média w de BR/(8D), sendo R o tempo de percurso em segundos. Em estado estacionário, a janela de congestionamento média do TCP é aproximadamente 1.2/sqrt(p). Isto equivale a um evento de perda a cada 1/p segmentos, ou a cada 1/(pw) = w/1.5. Substituindo w, isto representa um evento de congestionamento a cada (BR)/(12D) tempos de percurso.

Para pacotes de 1500 *bytes* e tempo de percurso (RTT) de 0.1 segundos, os números para a janela de congestionamento e taxa de perda de pacotes são apresentados na Tabela 3.1. Esta tabela também fornece a janela de congestionamento média w e a taxa de eventos de congestionamento em estado estacionário p.

Taxa de Transferência (Mbps)	RTTs entre perdas (segs)	Janela de Cong. (pcts)	Taxa de Perdas
1	5.5	8.3	0.02
10	55.5	83.3	0.0002
100	555.5	833.3	0.000002
1000	5555.5	8333.3	0.00000002
10000	55555.5	83333.3	0.0000000002

Tabela 3.1: RTTs entre Perdas de Pacotes para o TCP Padrão

Analisando a tabela 3.1, observou-se, que para o TCP atingir uma elevada taxa de transferência, é preciso um taxa de perda de pacote muito pequena e um grande intervalo entre as perdas de pacotes, o que se torna muito difícil nas redes atuais.

3.8.1- Tamanho do Quadro

Atualmente, o tamanho típico do MSS do TCP é de 1448 *bytes*, devido ao valor de 1500 *bytes* da MTU de uma interface *Ethernet*. Este tamanho é útil para ser usado em múltiplas velocidades, bem como em ambientes de *hubs* e *switche*s. Todavia, ele cria dificuldades para aplicações que necessitam enviar uma grande quantidade de dados, tais como transmissões volumosas de dados via *FTP*, porque estas aplicações trabalham melhor com quadros maiores. Um MSS maior melhora a velocidade de recuperação do TCP e reduz a taxa de interrupção da *CPU* por pacote. Outras mídias já suportam MTU maiores: *FDDI* tem um MTU de 4392 *bytes*, *Gigabit Ethernet Jumbo Frame* tem uma MTU de 9000 *bytes*, *IP-over-ATM* usa uma MTU de 9180 *bytes* e *HiPPI* usa uma MTU de 65535 *bytes*.

3.8.2- Buffers TCP

Os nós transmissores e receptores requerem um espaço de *buffer* para lidar com os segmentos de chegada e partida em uma conexão. Este espaço deve ser de, pelo menos, a quantidade de dados sem confirmação de recebimento que o TCP precisa lidar, de forma a manter o canal de comunicação cheio. Problemas de desempenho do TCP surgem quando o espaço para *buffer* não é adequado para acomodar o produto banda atraso. Se os *buffers* são muito pequenos, a janela de congestionamento do TCP nunca abrirá completamente a ponto de encher a conexão [11], [27].

A janela de anúncio, *awnd*, do receptor também precisa ser grande o suficiente. Ela limita o quanto o transmissor pode enviar, porque o transmissor não pode enviar mais

dados do que a janela de anúncio permite. Todavia, a janela de congestionamento máxima é proporcional à quantidade de espaço de *buffer* que o núcleo do sistema operacional aloca para cada *socket*. Por exemplo, se o RTT é de 50ms e a banda desta conexão é de 100 M*bits*/seg, o *buffer* TCP deverá ser de 625.000 *bytes*. Como a capacidade de transmissão da rede aumentou nos últimos anos, os sistemas operacionais têm gradativamente modificado o tamanho de *buffer* padrão, normalmente congelado em valores entre 8 K*Bytes* a 64 K*Bytes*. Entretanto, estes valores ainda são muito pequenos para as redes de alta velocidade atuais [11], [27], e impedem que o TCP faça uso de toda a banda disponível.

O protocolo TCP é responsável pela transmissão confiável e pelo controle de congestionamento na Internet. Cada segmento TCP contém um campo de 16 *bits*, chamado janela de anúncio, que indica o número de *bytes* que o receptor pode aceitar sem estourar o seu *buffer*. Assim, os 16 *bits* do campo chamado tamanho da janela de anuncio, como já dito anteriormente, seriam um empecilho para que o número de segmentos em trânsito fosse maior do que 64k*bytes*. Para tratar este problema, foi criado um mecanismo de multiplicação do valor da janela de recepção, por um fator negociado na fase de estabelecimento de conexão. Este mecanismo denominado *Window scale* está descrito na RFC 1323.

3.8.3- Buffers de Rede

O TCP tem, por natureza, característica de rajada. Esta característica de rajada do TCP pode resultar num fraco desempenho devido a uma limitada *bufferização* da rede. Grandes rajadas de dados colocadas na rede, em curtos intervalos, tendem a criar longas filas nos roteadores intermediários. Na maioria dos casos práticos, o tamanho máximo da janela, que reflete o maior tamanho possível de uma rajada de dados, é muito maior que a capacidade de enfileiramento de qualquer roteador intermediário. À medida que os transmissores TCP sobrecarreguem as filas dos roteadores, eles começarão a descartar segmentos. O TCP irá assumir estes descartes de segmentos, devido a este gargalo nas filas e ao congestionamento na rede. Isto pode resultar num fraco desempenho do TCP, com baixa taxa de transmissão e compartilhamento de banda desproporcional. Por outro lado, grandes filas no roteador podem introduzir atrasos adicionais nos fluxos TCP, aumentando seus RTT [25] e [27].

4 - Implementações TCP

O protocolo TCP concebido originalmente não determina como a camada de *software* TCP deve ser implementada em cada equipamento. Sendo assim, é possível implementá-lo de maneiras distintas desde que o protocolo seja respeitado. Existem muitos trabalhos realizados sugerindo novas implementações, mas as principais implementações do TCP são: TCP Tahoe, TCP Reno, TCP NewReno e TCP SACK.

4.1 - TCP Tahoe

As primeiras implementações do TCP não incluíam qualquer mecanismo de controle de congestionamento. Elas simplesmente usavam a janela do receptor para transmissão e um temporizador para retransmissão. Esta retransmissão era feita de acordo com a estratégia "*go-back-n*", ou seja, todos os segmentos após o segmento dado por perdido eram retransmitidos. Não havia nenhum tipo de busca de realimentação de informação que indicassem a situação da rede. Além disso, os temporizadores do TCP eram deficientes no cálculo da estimativa do RTT, o que provoca problemas nas retransmissões.

O TCP Tahoe foi a implementação que acrescentou alguns algoritmos e refinamentos em relação a implementações anteriores. Este tipo de TCP contém uma combinação de algoritmos que são, Partida Lenta, Prevenção de Congestionamento e Retransmissão Rápida. Além de modificações na estimação do RTT [5].

4.2 - TCP Reno

Esta implementação retém todos os melhoramentos introduzidos no TCP Tahoe, porém, com um novo algoritmo denominado recuperação rápida (*Fast Recovery*), que é incorporado ao de retransmissão rápida. Esta implementação é a versão do protocolo TCP que se encontra mais difundida na internet atualmente.

Este novo algoritmo, presente no TCP Reno [18], evita que o canal de comunicação esvazie depois do algoritmo Retransmissão Rápida, evitando que o algoritmo de Partida Lenta seja executado depois de uma única perda. Nesta situação, fica claro que qualquer situação de congestionamento que porventura tenha causado a suposta perda, já se esgotou, pois o fluxo de transmissão sobrevive à medida que reconhecimentos continuam chegando. Portanto, não há necessidade de se reduzir o fluxo abruptamente, fazendo com que o algoritmo de partida lenta seja executado.

4.3 - TCP New Reno

O TCP New Reno [19] inclui uma pequena alteração no algoritmo do TCP Reno que só tem efeito, na verdade, quando múltiplos segmentos são perdidos dentro de uma mesma janela de congestionamento.

O algoritmo do New-Reno utiliza o conceito de reconhecimento parcial. O TCP percebe que ocorreu a perda de um segmento através do recebimento de um determinado número de reconhecimentos duplicados, a próxima informação só será dada ao emissor com a chegada do reconhecimento para o segmento que foi retransmitido. No caso de uma única perda, esse reconhecimento irá confirmar o recebimento de todos os segmentos transmitidos antes do início da Retransmissão Rápida. Se ocorrer a perda de múltiplos segmentos, entretanto, o reconhecimento irá confirmar alguns segmentos, mais precisamente, todos os reconhecimentos até a próxima perda. Esse reconhecimento que não confirma todos os segmentos enviados antes do início da Retransmissão Rápida, é chamado reconhecimento parcial, visto em [8].

Pode-se concluir, então, que reconhecimentos parciais duplicados, da mesma forma que os reconhecimentos duplicados iniciais, indicam muito provavelmente que um outro segmento da mesma janela foi perdido ao longo do caminho.

O TCP Reno não diferencia os reconhecimentos recebidos durante a Recuperação Rápida, sejam eles parciais ou não, reconhecendo todos os dados pendentes no início do algoritmo de Recuperação Rápida. Assim, no TCP Reno, a cada reconhecimento parcial recebido o valor da janela cai pela metade, fazendo o protocolo entrar novamente na Recuperação Rápida, mesmo que as perdas ocorram na mesma janela de transmissão. Este fato acaba por prejudicar o desempenho do Reno, pois sua janela é reduzida repetidamente, quando uma redução já poderia ser suficiente para contornar a situação de congestionamento.

Já o TCP New Reno, responde aos reconhecimentos parciais. Assim, ao receber reconhecimentos parciais, o TCP se mantém no algoritmo de Retransmissão Rápida, evitando que ocorram múltiplas reduções no valor da janela de congestionamento.

O algoritmo Recuperação Rápida do New Reno utiliza, então, a variável *recover* que guarda o número de seqüência mais alto já transmitido. O algoritmo é descrito a seguir:

1. O recebimento do terceiro ACK duplicado faz com que *ssthresh* receba *cwnd*/2. Grava-se o maior número de seqüência na variável *recover*;
2. Retransmite-se o segmento perdido. A *cwnd* recebe *ssthresh* + 3*MSS – "inflando artificialmente" a janela de congestionamento pelo número de segmentos que deixaram a rede (três);
3. A cada novo ACK duplicado, incrementa-se *cwnd* de 1*MSS, se for possível, transmite-se um novo segmento;
4. Ao chegar um novo ACK:
    a. Se o ACK inclui o dado cujo número de seqüência ficou guardado em *recover* , faz *cwnd* receber *ssthresh* e sai da Recuperação Rápida;
    b. Se o ACK não reconhece todos os dados, ele é um ACK parcial, então, Retransmitir o primeiro segmento reconhecido, "desinflar" a janela de transmissão pelo número de reconhecimentos recebidos. Somar 1*MSS à janela de congestionamento e envia novos segmentos, se o tamanho de *cwnd* permitir. Não sair da Recuperação Rápida (ou seja, se algum novo ACK duplicado chegar, repetir a partir do passo 3 citado anteriormente).

A idéia principal do New Reno consiste em fazer com que, quando múltiplos segmentos são perdidos em uma janela de dados, o fluxo de transmissão se recupere sem que aconteça um esgotamento do temporizador, retransmitindo um segmento por RTT até que todos os dados pendentes tenham sido reconhecidos.

4.4- Vegas

Uma implementação proposta para o TCP que apresenta algumas modificações mais profundas nos mecanismos tradicionais deste protocolo, é o TCP Vegas, descrito em [24] e [25]. As alterações do Vegas resumem-se ao lado do transmissor, ou seja, sua política de emissão de reconhecimentos é a mesma do TCP Reno. Isto facilitaria sua adoção por parte da comunidade da Internet, já que os seus benefícios podem ser desfrutados mesmo que na outra ponta haja uma versão mais antiga (ao contrário, por exemplo, do SACK). O transmissor segue as linhas gerais propostas por [5]. O Vegas possui um algoritmo de início de conexão, um para o estado estacionário e um para controlar as retransmissões. Todavia, os três algoritmos são bastantes diferentes dos vistos no TCP Reno.

4.4.1 Prevenção do Congestionamento

O mecanismo de Prevenção do Congestionamento do Vegas é baseado no cálculo da estimativa da vazão da conexão. A partir desta estimativa, são arbitrados um limite mínimo e um máximo da vazão por parte da conexão TCP. O Vegas busca manter a vazão dentro destes limites através da manipulação da janela de congestionamento.

A estimativa das filas dos roteadores é feita a partir de outros valores: a vazão esperada e a vazão real da conexão. A vazão esperada é definida como Expected = WindowSize/BaseRTT, onde *WindowSize* é o tamanho atual em *bytes* da janela de congestionamento (supostamente igual ao número de *bytes* no meio) e *BaseRTT* é o menor valor de RTT, medido para uma dada conexão (que representa o RTT da conexão sem que haja congestionamento). para o cálculo da vazão real, o Vegas utiliza o RTT real. Para efetuar essa estimativa, ele registra o tempo em que um segmento de referência foi enviado e conta os *bytes* enviados até a chegada do ACK relativo àquele segmento, cujo tempo também é registrado. A vazão real é calculada como o número de *bytes* enviados sobre o RTT. Este cálculo é efetuado uma vez a cada RTT.

Então, o Vegas compara a vazão real com a esperada, gerando o parâmetro Diff = Espected – Actual. Por construção, Diff será sempre igual ou maior que zero, pois a vazão real maior que a esperada implica que o último RTT é menos que BaseRTT, e como este último é atualizado sempre, isto não pode ocorrer. São arbitrados também os parâmetros α e β, que serão usados como base para determinar se a janela de congestionamento deve aumentar, diminuir ou permanecer igual. Se Diff < α, o Vegas aumenta a janela de congestionamento de forma linear. Se Diff > β, a janela de congestionamento é reduzida também de forma linear, ao contrário do algoritmo Prevenção de Congestionamento tradicional, que segue a regra aumento aditivo e decremento multiplicativo, já comentado anteriormente. Se α < Diff < β, a janela de congestionamento fica inalterada.

Como são usados para comparação com taxas de vazão, os parâmetro α e β são expressos em KB/s, entretanto, é mais fácil entender o raciocínio por trás deste algoritmo se pensarmos em termos do espaço de *buffer* ocupado nos nós intermediários. Tomemos como exemplo uma conexão BaseRTT = 100ms e MSS = 1KB. Para uma vazão de 10KB/s, podemos dizer que um segmento se encontra em trânsito na rede, ocupando um *buffer* de um roteador. Se α = 30KB/s e β = 60KB/s, isto significa que a conexão deverá sempre ter entre três e seis segmentos, ocupando *buffers* nos roteadores por onde passa a conexão. Se estiver ocupando menos de três, a conexão não está usando

suficientemente a largura de banda disponível da rede. Por outro lado, se estiver ocupando mais de seis *buffers*, estará gerando congestionamento.

Na prática, os valores de *α* e *β* são de fato definidos em termos de *buffers* ao invés de vazões. Durante a fase chamada *linear increase/decrease*, os valores *α*=1 e *β*=3, isto quer dizer que a conexão busca ocupar, ao menos, um e, no máximo, três *buffer* na rede. A importância de ter *α* maior que zero é que podemos ocupar ao menos um *buffer* na rede, a conexão pode reagir mais rapidamente ao aumento da banda disponível sem ter que esperar um RTT para que ocorra o aumento da janela de congestionamento. O valor de *β* é três para ocupar o mínimo de *buffers* sem que a janela fique oscilando permanentemente, o que aconteceria caso *β* fosse somente uma unidade maior que *α*.

4.4.2- Partida Lenta

A modificação feita no algoritmo Partida Lenta tem como meta evitar que o aumento exponencial, no início da conexão, gere congestionamento na rede, como pode ocorrer com o TCP Reno, por exemplo. Deste modo, no início de uma conexão do TCP Vegas ocorre o aumento exponencial, mas a cada dois RTTs, isto é feito aumentando a janela de um segmento para cada ACK recebido (como nos outros TCPs) em um RTT, e mantendo-a fixa durante o outro RTT. Durante o ciclo em que a janela fica inalterada, é realizada a comparação entre a vazão esperada e a real, descrita anteriormente. Quando a vazão real fica abaixo da esperada pelo equivalente a um *buffer* (um MSS), o Vegas muda para o modo *linear increase/decrease*.

4.4.3- Retransmissões

O Vegas tenta superar as limitações de temporização normalmente associadas ao TCP, devido ao uso de um relógio de baixa resolução. Em geral, como dito anteriormente, as implementações do TCP utilizam um relógio que dá um *tick* a cada 500ms. Esta resolução é muito baixa para determinar *timeout*s com RTTs, da ordem de 100ms, que é o que se encontra normalmente [24]. É dado um exemplo, em [24], de testes em que o temporizador deveria ter estourado com 300ms, mas, na prática, o Reno só percebia o *timeout*, em média, 1100ms após a transmissão dos segmentos. Em conseqüência disso, o Vegas tenta eliminar a dependência do TCP de temporizadores de baixa precisão.

A estimação do RTT é feita da seguinte maneira: a cada segmento enviado e a cada ACK recebido, o Vegas lê o relógio do sistema e registra o tempo lido. O cálculo do RTT é feito usando os mesmos algoritmos do Reno, mas as medidas de tempo usadas possuem

maior exatidão. Isto deve permitir que o TCP trabalhe com uma estimativa de RTT mais exata, o que possibilita as alterações propostas para algoritmos de retransmissão.
O mecanismo de retransmissão do TCP Vegas é disparado nas situações descritas a seguir:

- Quando é recebido um ACK duplicado, o Vegas confere se a diferença entre a hora atual e a hora registrada para o último segmento não reconhecido é maior que o valor de *timeout*, se for, o segmento é retransmitido sem que se tenha de esperar três ACKs duplicados. Este mecanismo é particularmente vantajoso em situações de congestionamento intenso em que haja perdas de ACKs.

-Quando um ACK normal é recebido, sendo este primeiro ou segundo após uma retransmissão, o Vegas procede da mesma forma, conferindo a hora de envio do último segmento contra a hora atual. Se a diferença for maior que o valor de *timeout*, o segmento é retransmitido.

O Vegas mantém o mecanismo de *timeout* do Reno como um último recurso, caso seus mecanismos não percebam a perda de um segmento. No entanto, o objetivo é justamente reduzir ao máximo o número de *timeouts* tradicionais, já que estes normalmente são mecanismos ineficientes para a determinação das retransmissões.

4.5- TCP SACK

O TCP com reconhecimento seletivo não incorpora qualquer mudança nos algoritmos de controle de congestionamento. Neste aspecto, ele é idêntico ao Reno, apenas apresentando a diferença na política de retransmissões, decorrentes da opção de reconhecimento seletivos, como citado anteriormente. A opção SACK [20] aumenta o desempenho do TCP em redes de alta velocidade e com alto atraso. Com a técnica SACK, o transmissor obtém informações suficientes sobre os segmentos recebidos corretamente pelo receptor e, dessa forma, é capaz de retransmitir em um RTT múltiplos segmentos perdidos de uma janela de dados.
Este tipo de TCP implementa a maioria das melhorias do TCP, disponíveis atualmente e permite que o receptor de uma conexão TCP informe ao transmissor sobre múltiplos segmentos descartados dentro de uma única janela de dados.

Nos próximos parágrafos, serão comentadas algumas das propostas para tratar o problema do baixo desempenho do protocolo TCP, em redes de alta velocidade. No entanto, estas propostas não farão parte das simulações realizadas no presente trabalho,

cujo o objetivo é o de estudar algumas das implementações padrão do protocolo TCP, em redes de alta velocidade, ou seja, acima de um gigabits.

4.6- Soluções propostas para a melhora do desempenho do protocolo TCP em redes de Alta Velocidade

Atualmente, existem duas linhas de pesquisa na atuação do protocolo TCP em redes de alta velocidade, o paralelismo com o TCP existente e mudanças no TCP existente, para permitir uma rápida aceleração nas taxas de um único fluxo.

4.6.1- Fluxos Paralelos TCP

O uso de fluxos paralelos tem como princípio aumentar o desempenho do TCP, esta técnica já existe há algum tempo. A especificação original do HTTP permitia o uso de sessões paralelas do TCP para executar *download*. Outra técnica para a transferência de arquivos é a que divide a carga da comunicação dentro de numerosos componentes discretos, e envia cada um destes componentes simultaneamente. Os componentes da transferência podem estar entre os mesmos dois pontos finais, como o GRID FTP, ou pode estender-se entre múltiplos pontos finais, tal como *BitTorrent*.

Usar o paralelismo como uma chave para altas velocidades é uma técnica computacional comum, é um truque por atrás de muitas arquiteturas do supercomputer. O mesmo pode aplicar-se à transferência de dados, na qual uma série de dados é dividida em inúmeros pequenos pedaços, e cada pedaço é transmitido, usando a sua sessão TCP. A idéia fundamental aqui é que ao usar-se algum número, N, de sessões paralelas TCP, um único evento de descarte de segmento, a causa mais provável é que rapidamente as N sessões tenham suas taxas divididas pela metade, porque as várias sessões terão a mais segmentos passando na rede, e conseqüentemente a sessão mais provável a ser impactada por um descarte de segmento. Estas sessões usarão então o algoritmo Prevenção do Congestionamento para aumentar a taxa, relembrando que a resposta a uma única perda de segmento é a redução da taxa de transmissão em aproximadamente 1/(2N). Por exemplo, usando cinco sessões TCP paralelas, a resposta para uma única perda de segmentos é a redução da taxa de transmissão para 1/(2 x 5), ou 1/10, quando comparado com a resposta de uma única sessão, onde uma única perda reduzirá a taxa de transmissão em 1/2. A característica essencial neste caso, é que em condições de perdas, o fluxo de dados com N sessões paralelas aumentará a taxa de transmissão N vezes mais rápido, que uma única sessão na fase de prevenção do congestionamento. No entanto, na ocorrência de uma perda reduzirá a taxa em 1/2N.

4.6.2- MultTCP

O MultTCP é uma implementação do protocolo TCP para operar em redes de alta velocidade. Esta implementação faz com que um único fluxo TCP comporte-se de maneira equivalente a N sessões TCP paralelas, onde as sessões virtuais são igualmente distribuídas a fim de alcançar um ótimo resultado em termos de taxa de transferência. As mudanças essenciais para o TCP são no algoritmo de Prevenção de Congestionamento e a reação a perda de segmentos, .no algoritmo de Prevenção de Congestionamento o MultTCP aumenta a janela de congestionamento, *cwnd*, em N segmentos por RTT, melhor que o padrão que aumenta somente em um segmento. Em relação a perda de segmentos, o MultTCP reduz a janela por W/(2N), melhor que o padrão que reduz W/2. O MultiTCP usa uma versão diferente do algoritmo Partida Lenta, ele aumenta a janela por 3 segmentos, por ACK recebidos, o padrão aumenta a janela por 2 segmentos por ACK recebido.

MultiTCP representa uma simples mudança no TCP que não muda radicalmente o algoritmo de controle de congestionamento do TCP. Para obter uma boa escolha no valor de N, um bom conhecimento das características da rede pode ajudar. Um fato que se deve levar em consideração é que se N for muito alto, a sessão do MultiTCP tem uma tendência a exigir uma quantia injusta da capacidade da rede, no entanto, um valor muito baixo para N, não trará grandes vantagens da capacidade disponível da rede. Experimentos com o MultiTCP podem ser encontrados em [14].

4.6.3- Scalable TCP

O Scalable TCP, desenvolvido por Tom Kelly da Universidade de Cambridge, tenta quebrar a relação entre o gerenciamento da janela TCP e o intervalo de tempo do RTT.

No TCP convencional, a resposta para cada ACK, no modo de Prevenção de Congestionamento, é aumentar o tamanho da janela de congestionamento (*cwnd*) por (1/*cwnd*) para cada intervalo do RTT. Similarmente, o tamanho da janela é dividido pela metade quando uma perda de segmento ocorre. O Scalable TCP substitui a função aditiva por um valor constante a. A diminuição multiplicativa é expressada como uma fração b, que é aplicada no tamanho da janela corrente.

No Sacalble TCP, para cada ACK a janela de congestionamento é aumentada por uma valor constante a, na ocorrência de uma perda de segmento a janela é reduzida por uma fração b.

A característica essencial do Scalable TCP é o uso de um aumento multiplicativo dentro da janela de congestionamento, melhor que o aumento linear do TCP padrão. Experimentos com esta implementação podem ser encontrados em [14].

4.6.4 - FAST

FAST é outra aproximação para a alta velocidade do TCP. Esta implementação tenta equilibrar o fluxo de segmentos em uma taxa que também estabiliza o atraso das filas, baseando-se no ajuste da janela, tal que o intervalo do RTT é estabilizado. Para isto, ele usa tanto o atraso de enfileiramento quanto a perda de pacotes como sinais de congestionamento.

O controle de congestionamento consiste em dois componentes, um algoritmo do transmissor, implementado no TCP, que adapta a taxa de envio à informação de congestionamento no seu caminho, e um algoritmo de enlace, implementado nos roteadores, que atualiza e realimenta a medida de congestionamento de volta para os transmissores que atravessam o enlace. No entanto, quando o valor do BDP é muito alto, isto é, quando o valor do atraso ou da largura de banda aumentam, tem sido comprovado que os algoritmos atuais podem se tornar instáveis. O FAST TCP propõe que, para manter a estabilidade, os transmissores devem diminuir suas respostas de acordo com a sua capacidade individual.

A função resposta da janela é baseada no ajuste do tamanho da janela, por uma quantidade proporcional à variação atual do RTT, em relação a sua média, ou seja, se sua variação é menor que a média, o tamanho da janela aumenta e, se for maior, a janela é decrementada. Mais detalhes, sobre experimentos feitos com esta implementação, podem ser encontrados em [14].

4.6.5 - XCP

Este protocolo faz uso do conceito ECN de maneira generalizada. Onde o *feedback* para relatar o congestionamento na rede é baseado em sinalização explícita, fornecida pelos roteadores habilitados para XCP, que informam aos transmissores sobre o grau de congestionamento no enlace gargalo.

O pacote XCP tem um cabeçalho de congestionamento que informa o estado do fluxo para os roteadores e realimenta as informações dos roteadores para os receptores. Um campo informa a janela de congestionamento atual do transmissor e outro comunica a estimativa presente de RTT do transmissor. Esta informação é preenchida pelo transmissor e não é modificada em trânsito. O terceiro campo é inicializado pelo

transmissor e recebe realimentações dos roteadores ao longo do caminho para controlar diretamente as janelas de congestionamento das fontes. Do mesmo modo que o TCP, o XCP é um protocolo de controle de congestionamento baseado em janelas, projetado para tráfego de melhor esforço.

O XCP faz com que elementos de troca na rede executem uma regra ativa de controle de fluxo fim-a-fim, ou seja, informem os sistemas nas extremidades a atual largura de banda disponível na rede. E, assim, permitindo que estes sistemas respondam rapidamente, aumentando a taxa de pacotes até o ponto que os roteadores suportem. Mais informações sobre esta implementação podem ser encontrada em [14] e [16].

4.6.6 - HighSpeed TCP

Para evitar a dependência de taxas de perdas irrealmente baixas, Sally Floyd propôs uma versão modificada do protocolo TCP para ser usado em conexões TCP com grandes janelas de congestionamento, chamada *HighSpeed* TCP (HSTCP). O HSTCP supera a dificuldade do TCP Padrão em atingir grandes janelas de congestionamento em ambientes com taxas de perda de pacotes muito baixas. Altera o crescimento e o decremento da janela de congestionamento para conexões com grandes janelas, na fase de prevenção de congestionamento. Por outro lado, o HSTCP não altera a fase de partida lenta.

A função de resposta do HSTCP é mantida linear, na escala *log-log*, como no TCP, para taxas de perda de pacote inferiores a 1%. A função resposta do HSTCP é especificada usando-se três parâmetros: *Low_Window*, *High_Window e High_P. Low_Window* é usado para estabelecer um ponto de transição e assegurar a compatibilidade. O HSTCP usa a mesma função resposta que o TCP padrão quando a janela de congestionamento atual é, no máximo, *Low_Window* e usa a função resposta do HSTCP quando a janela de congestionamento atual é maior do que *Low_Window*. *High_Window e High_P* são usados para especificar o ponto superior da função resposta do HSTCP. Ele é configurado a uma taxa de eventos de congestionamento específica *High_P*, necessária para a função resposta do HSTCP atingir a janela de congestionamento média de *High_Window*.

De acordo com [15], quanto maior a janela, maior sua taxa de crescimento e menor sua taxa de decrescimento. Com isso, uma janela média suficiente para transmissões de alta velocidade pode ser alcançada. Mais experimentos sobre este protocolo, podem ser encontrado em [14], [15], [27] e [28].

5- Simulações

Este capítulo visa descrever todo o trabalho feito, envolvendo as simulações destinadas à analise de desempenho do protocolo TCP descrito no capítulo 4. E apresenta uma série de simulações, cujo foco é o estudo do comportamento das diferentes implementações do protocolo TCP em redes de alta velocidade e grandes atrasos. O objetivo deste capítulo é apresentar um estudo quantitativo do protocolo TCP, em cenários que permitam avaliar o desempenho do comportamento das diferentes implementações do protocolo TCP.

Foram escolhidos para simulação as implementações Tahoe, Reno e SACK do TCP. O primeiro foi o pioneiro na utilização de mecanismos de controle de congestionamento. O Reno, além de incluir o algoritmo *Rápida Recuperação*, é a implementação mais comum, de uso corrente na internet [23]. O TCP SACK, por sua vez, utiliza um mecanismo extra, o reconhecimento seletivo, que representa uma nova tendência.

Muito embora o uso de simulação permita a investigação de um protocolo em uma rica variedade de situações, nem todas as condições de rede serviram ao interesse do presente estudo. O cenário de investigação foi limitado ao caso de interesse selecionado, ou seja, o foco principal foi o comportamento das implementações do protocolo TCP em situações em que ele estivesse em estado estacionário ou perto do estado estacionário. Quando a janela de congestionamento oscila em torno de seu ponto de equilíbrio, é caracterizado por uma forma de onda em dente de serra.

Este capítulo é apresentado da seguinte maneira, inicialmente é realizada uma breve apresentação sobre a ferramenta de simulação utilizada. Em seguida, descreve-se a topologia empregada nas simulações e a configuração dos fluxos e, logo após, realiza-se a análise das implementações utilizadas nas simulações.

5.1- Simulador de Redes

Para a realização da análise do desempenho das implementações do protocolo TCP, foram realizadas simulações utilizando o *software NS* (*Network Simulation*) [28]. Esta ferramenta de simulação de redes foi desenvolvida na Universidade da Califórnia Berkeley, em parceria com o projeto VINT (*Virtual InterNetwork Testbed*), incluindo como colaboradores o USC/ISI, a Xerox PARC e o LBNL. O simulador *ns* foi escolhido para as simulações devido sua ampla aceitação no meio acadêmico.

5.2- Topologia

Esta análise foi desenvolvida, usando um cenário com uma topologia simples para evitar interações mais complexas e reduzir o número de variáveis a coletar e estudar.

A topologia de rede escolhida para as simulações foi a conhecida barra de pesos, com um único gargalo e está ilustrada na figura 5.1 e 5.2 O cenário na topologia 5.1 consiste em uma Rede de Longa Distância de alta velocidade. Uma única fonte (S1) gera o tráfego para um destino (D1) através de uma conexão TCP. Dois roteadores (R1 e R2) direcionam o tráfego entre as fontes e destinos. Os enlaces entre o nó fonte e R1 e entre R2 e o destino possuem uma largura de banda de 100G*bits*/s e retardo de 1ms, enquanto que o enlace central entre os roteadores é de 1Gbit/s com o atraso de 50ms. O tráfego gerado pelas fontes é do tipo FTP onde sempre existe dados disponíveis para serem enviados, ou seja, enquanto o valor da janela permitir, dados estarão sendo transmitidos com a taxa de transmissão permitida pelo canal. O cenário da topologia 5.2 difere da topologia 5.1, somente no número de fontes geradoras de tráfego e de destinos, já que este cenário tem como propósito estudar o comportamento das implementações do TCP, quando competindo entre si, logo o número de fontes geradoras é igual ao número de implementações utilizadas na simulação, ou seja, três, S1,S2 e S3, e o número de destinos utilizados, também é igual ao número de implementações utilizadas, D1, D2 e D3.

O enlace gargalo foi o enlace principal. A banda do enlace principal foi 1G*bits*/seg, o seu atraso foi de 50ms e o tipo de gerenciamento de enfileiramento do roteador foi *Drop Tail*. (DT).

O gerenciamento de enfileiramento, no roteador, foi restrito à técnica de descarte (DT). DT é a técnica tradicional mais simples de realizar o gerenciamento do tamanho de fila do roteador. Ele controla o comprimento da fila, usando um esquema *FIFO* (*First In First Out*). Neste esquema, cada novo pacote que chega na porta de entrada da fila, é descartado, quando o espaço de buffer da fila está cheio.

O *Buffer* de roteadores é um elemento essencial para uma rede por comutação de pacotes. Ele absorve a chegada de pacotes em rajadas e reduz o potencial de perdas. Quanto maior o *buffer*, maior será a capacidade de absorver grandes rajadas. No entanto, este fato faz com que a carga e os atrasos nos enfileiramentos aumentem. Van Jacobson propôs, em 1988, que o espaço de *buffer* na fila deveria ser não menos que o produto da banda pelo atraso, BDP. Logo, o valor padrão do tamanho do buffer utilizado nas

simulações, foi 8.333 pacotes, porque este valor coincide com o BDP para um enlace de 1Gbps, com RTT de 100ms e tamanho de pacote de 1500 *bytes*.

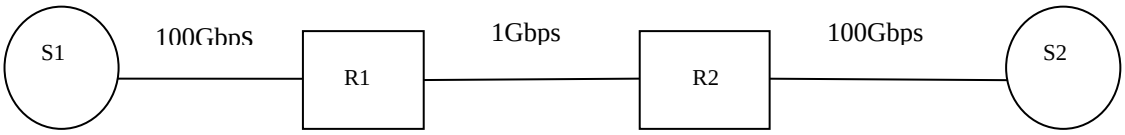

Figura 5.1- Topologia utilizada nas simulações dos testes dos fluxos individuais

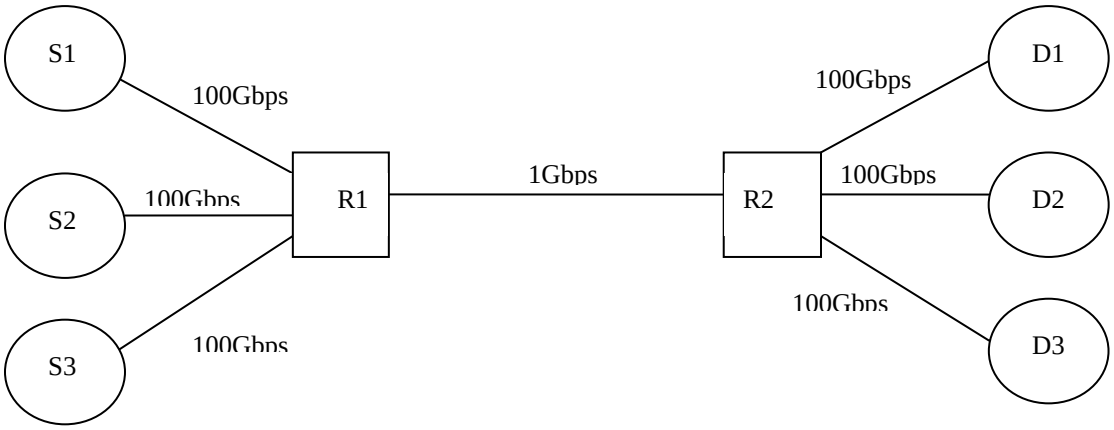

Figura 5.2- Topologia utilizada na simulação da competição dos fluxos

5.3- Configuração dos fluxos

Fluxos TCP de longa duração são os de maior interesse para enlaces de alta velocidade e longa distância, quando uma grande quantidade de dados precisa ser transmitida e então os fluxos usados neste trabalho tiveram longa duração, no qual o estado estacionário era bem maior que o estado transiente. Para isto, o tráfego gerado pelas fontes é do tipo FTP, onde sempre existem dados disponíveis para serem enviados, ou seja, enquanto o valor da janela permitir, dados estarão sendo transmitidos com a taxa de transmissão permitida pelo canal.

O conjunto de parâmetros comuns utilizados pelos fluxos TCP foram, o tamanho do pacote que foi de 1500 *bytes*, já que este é o tamanho de mais da metade do volume

de tráfego em conexões de longa distância [27]; o tamanho máximo da janela de janela foi 100000 segmentos, este valor foi configurado para ser grande o suficiente para não impor limites pelo tamanho máximo da janela de anúncio do receptor e o tamanho mínimo de cabeçalho TCP foi configurado, ou seja, 40 *bytes*, com nenhum cabeçalho opcional.

5.4- Simulações efetuadas

Esta seção, finalmente, apresentará os gráficos obtidos a partir das simulações realizadas, acompanhados de análises que explicam o comportamento das curvas, de acordo com a implementação do protocolo TCP utilizada.

Foram utilizados, neste estudo, quatro conjuntos de fluxos. O primeiro conjunto tem apenas um fluxo TCP Tahoe, o segundo apenas um fluxo TCP Reno, o terceiro somente um fluxo TCP SACK e o quarto conjunto possuía ambos os fluxos, Tahoe, Reno e SACK. O quarto conjunto permitiu observar a interação entre os fluxos Tahoe, Reno e SACK.

Estes quatro conjuntos de fluxos foram expostos a uma única condição de rede. No ambiente de rede criado, não havia outra fonte de tráfego e interferência além das geradas pelos conjuntos de fluxos testados. Este ambiente de rede é referenciado como Condição Ideal. Também é assumido que os pontos finais fornecem componentes de hardware suficientemente rápidos, ou seja, barramentos internos e memórias adequadas para tratar gigabits de dados por segundo entre memória e interface de rede. Esta situação é interessante porque é possível estudar o desempenho dos conjuntos de fluxo sem interferência.

A simulação deste experimento objetivou obter um padrão de comparação dos fluxos, bem como derivar informações do comportamento teórico de cada uma das implementações testadas, quando não houvesse interferência externa, de acordo com [26]. Cada simulação foi executada por 1500 segundos.

5.4.1- Tahoe

A primeira simulação efetuada utilizou a implementação Tahoe, e é apresentada nas figuras 5.3 e 5.4. Esta figura, assim como as demais analisadas neste capítulo, apresentam em seu eixo horizontal o tempo em segundos e em seu eixo vertical a quantidade de segmentos por janela. A partir do gráfico na figura 5.3, observa-se que o TCP Tahoe reinicia o modo Partida lenta nos tempos 2.7 e 5.4 segundos, neste instante ocorre uma perda, portanto, esta implementação ao receber um terceiro reconhecimento duplicado para o segmento perdido, faz com que o valor da variável *ssthresh* caia pela

metade e a janela de congestionamento reinicie com apenas um segmento, aumentando-a de forma exponencial. Porém, ao igualar o valor salvo em *ssthresh* o algoritmo Prevenção de Congestionamento é iniciado, fazendo com que a janela aumente de forma linear a uma taxa aproximada de um segmento por RTT, até o instante 1308 segundos, onde novamente ocorre uma perda e todo o processo citado anteriormente é iniciado.

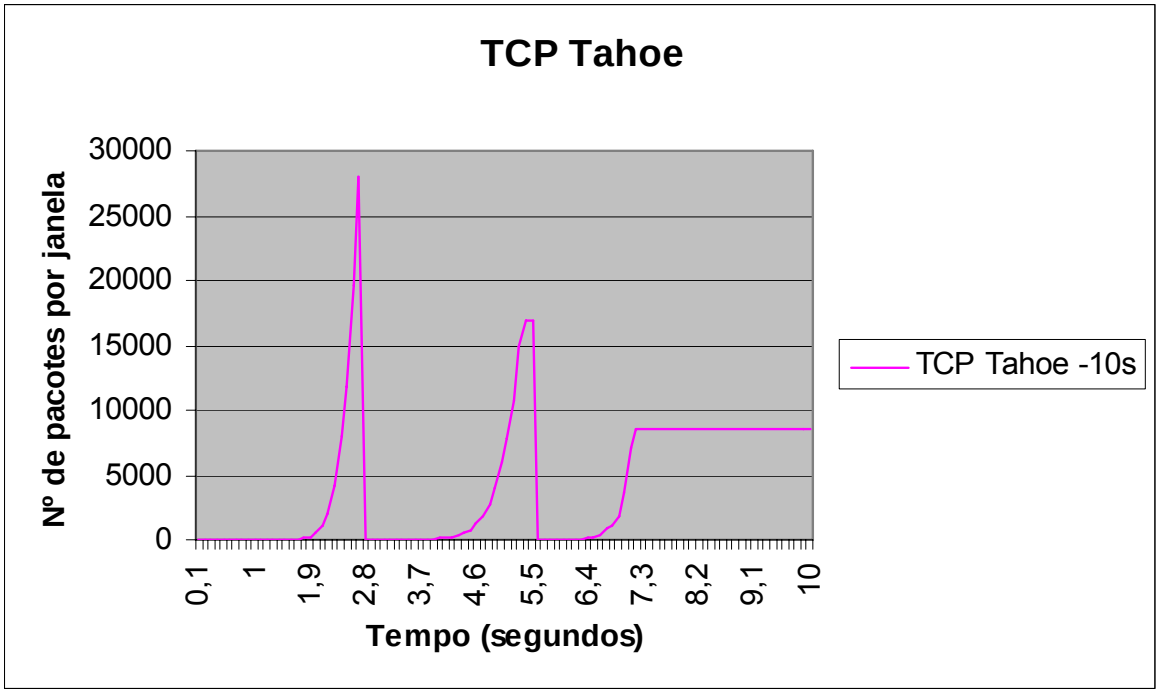

Figura 5.3- Partida Lenta Tahoe

Na figura 5.3 observa-se claramente o crescimento exponencial da janela de congestionamento, desde o início da transmissão até a primeira perda, 2.8 segundos com o valor da *cwnd* = 28036 segmentos, neste momento o modo Partida Lenta é reiniciado e novamente a *cwnd* =1 segmento, a segunda perda ocorre em 5.5 segundos, com a *cwnd* = 16999 segmentos, reiniciando novamente a *cwnd* = 1 segmento, no tempo 7.1 segundos, o Tahoe entra no modo de Prevenção de Congestionamento, como mostrado na figura 5.3, a partir deste ponto a janela deixa de ser incrementada de forma exponencial para ser incrementada de forma linear, esta implementação permanece neste modo até atingir a janela máxima, que ocorre no tempo 1308 segundos, ou seja, leva aproximadamente 22 minutos para atingir a janela máxima, este é o tempo que o Tahoe leva para completar um ciclo no estado estacionário. Ao encher a janela de congestionamento com o valor máximo, o Tahoe reinicia novamente o algoritmo Partida Lenta, fazendo com que a *cwnd* seja igual a um segmento e seu crescimento seja

exponencial até que ela atinja o *ssthresh* que tem com valor a metade da janela máxima, a figura 5.4 apresenta o comportamento do Tahoe no estado estacionário.

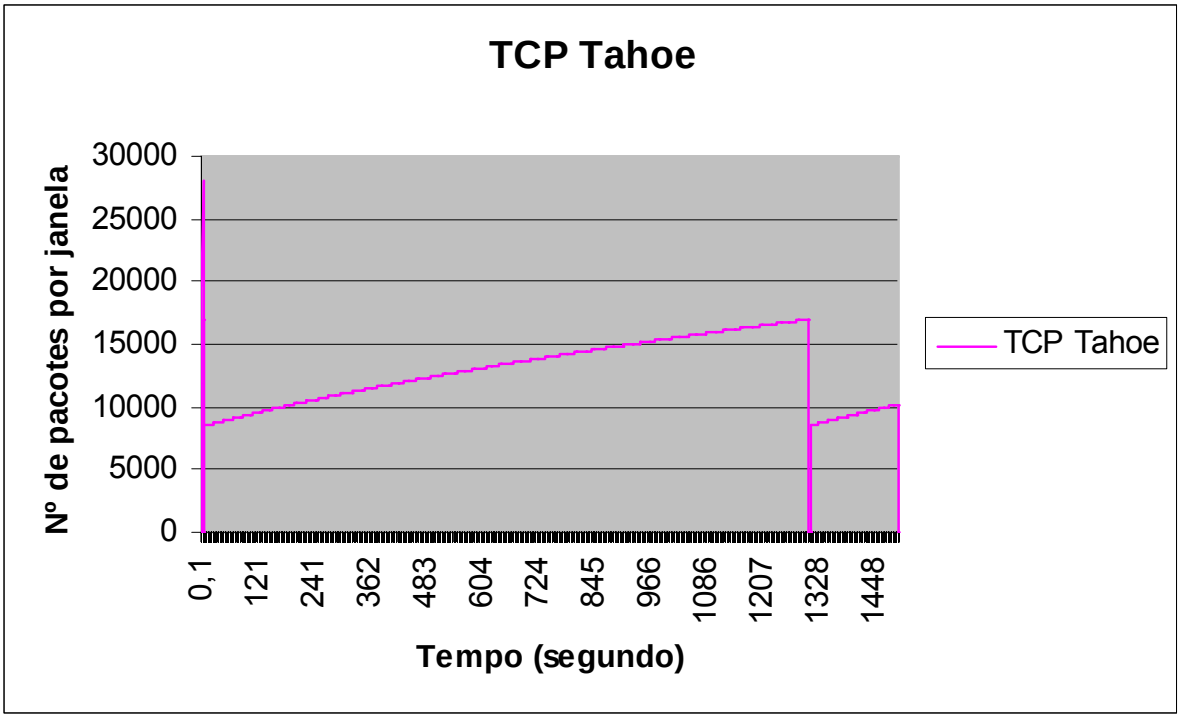

Fig. 5.4- TCP Tahoe em estado estacionário

5.4.2-Reno

A simulação para o TCP Reno é apresentada nas figuras 5.5 e 5.6, revelando que o protocolo se comporta, inicialmente, de forma semelhante ao TCP Tahoe, no modo Partida Lenta o Reno busca a ocupação do canal com o crescimento exponencial da janela de congestionamento. Da mesma maneira, como observado na figura 5.5, o transmissor percebe que ocorreu uma perda no tempo 2.7 quando o valor da *cwnd* = 28036 segmentos, neste momento, a janela de congestionamento é setada para *cwnd*/2, ou seja, *cwnd* = 16999 segmentos, no entanto, o TCP Reno ainda fica impedido de enviar mais dados pela enorme quantidade de dados pendentes. Como nenhum reconhecimento é enviado, O TCP Reno esgota o temporizador e os segmentos pendentes são transmitidos. Neste momento, o *ssthresh* é reduzido novamente pela metade da *cwnd*. Ao receber todos os dados pendentes, o TCP Reno entra novamente no modo de Partida Lenta até que a janela de congestionamento, *cwnd*, atinja o valor do *ssthresh*, onde entra no modo de prevenção de congestionamento e permanece até atingir o valor máximo da

janela, que é no instante 1306 segundos com a *cwnd* = 1700 segmentos, da mesma forma que o Tahoe, o Reno leva aproximadamente 22 minutos para atingir este valor. No entanto, quando este valor é atingido a janela de congestionamento passa a valer *cwnd*/2, e o algoritmo Prevenção de Congestionamento é reiniciado, ao contrário do Tahoe, o algoritmo de Partida Lenta não é reiniciado. O estado estacionário do Reno pode ser observado na figura 5.6.

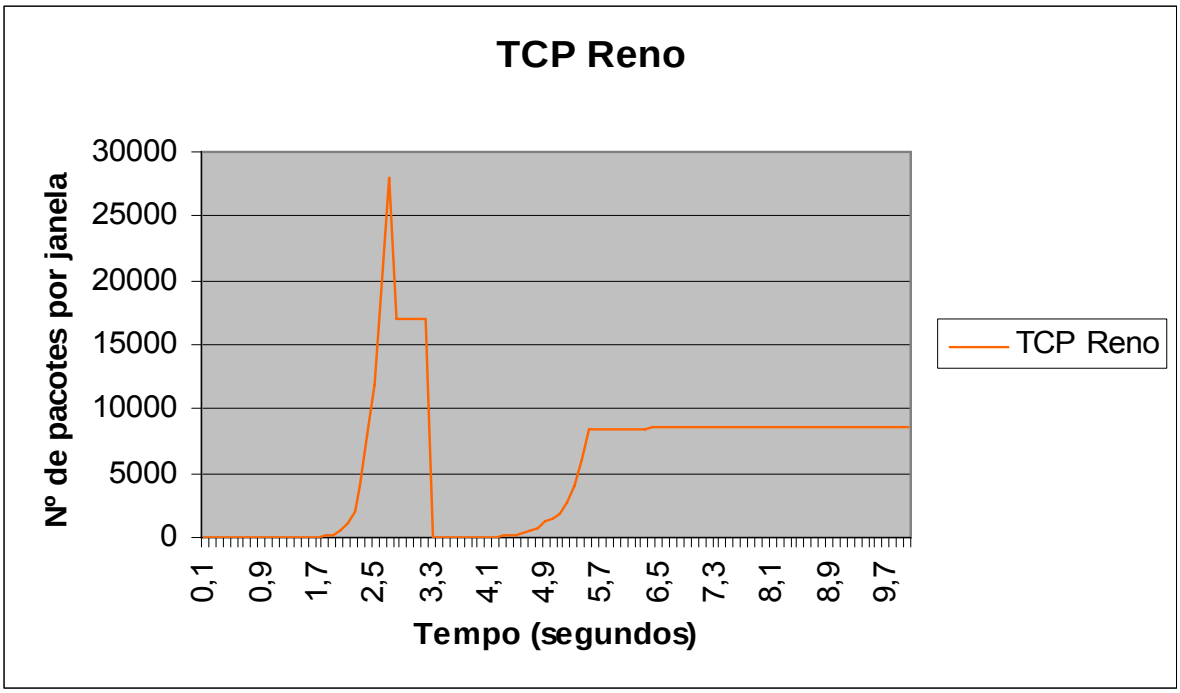

Fig.5.5- Partida Lenta Reno

Pode ser observado na f1gura 5.6 em relação à figura 5.4, um melhor desempenho na implementação do Reno, em relação ao Tahoe. Isto ocorre por que o Tahoe é uma implementação cautelosa em termos de contenção de congestionamento. O Tahoe age de forma a garantir, com larga margem de confiança, a estabilidade da rede mesmo que para isso tenha que prejudicar, em alguns casos, seu desempenho.

Este fato fica claro quando se observa que o Tahoe para qualquer perda de pacote o algoritmo Partida Lenta é iniciado, reduzindo a janela de congestionamento ao tamanho unitário, independentemente do nível de congestionamento da rede. Por outro lado, o TCP Reno busca manter a ocupação do canal através do mecanismo de Recuperação Rápida, permitindo uma recuperação mais rápida. No entanto, em algumas situações, como será mostrado mais adiante, o Tahoe pode ter um melhor desempenho que o Reno.

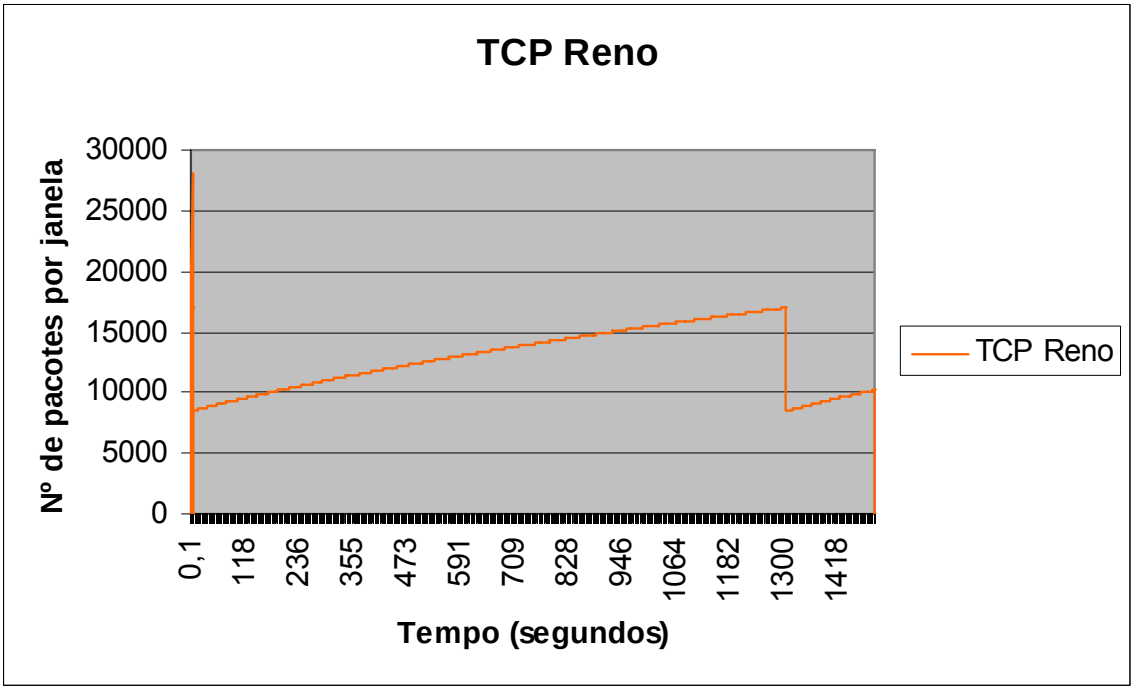

Fig. 5.6-TCP Reno no estado estacionário

5.4.3- SACK

As figuras 5.7 e 5.8 apresentam os gráficos nos quais o comportamento na simulação desta implementação é ilustrada.

Este protocolo como os já citados, também inicia com o algoritmo Partida Lenta, de maneira semelhante às outras implementações, tenta ocupar o mais rápido possível o meio de transmissão. É interessante observar que, neste caso, a opção por reconhecimentos seletivos é utilizada, ou seja, no instante 2.7 segundos, quando o SACK atinge o valor máximo da janela de congestionamento, *cwnd* = 28036 segmentos, ocorrem várias perdas na mesma janela e o SACK, ao contrário do Reno, não permite que o temporizador esgote, evitando assim que a *cwnd* passe a valer um segmento e o algoritmo Partida Lenta seja reiniciado, como pode ser observado na figura 5.7.

Quando todos os reconhecimentos dos segmentos pendentes são recebidos, o SACK entra no modo Prevenção do Congestionamento com variáveis *ssthresh* e *cwnd*, que já haviam sido reduzidas, duas vezes, pela metade do valor da janela de congestionamento, no momento das perdas. A partir deste ponto o SACK tem o comportamento igual ao do Reno. O estado estacionário pode ser observado na figura 5.8.

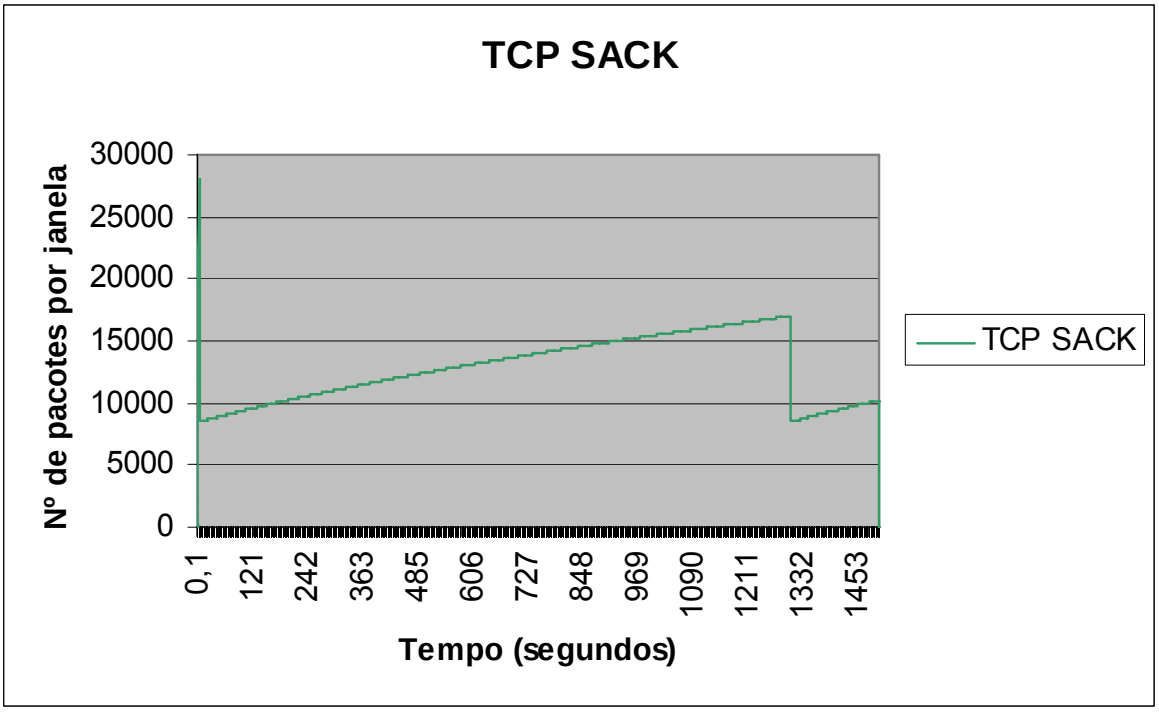

Figura 5.7- Partida Lenta SACK

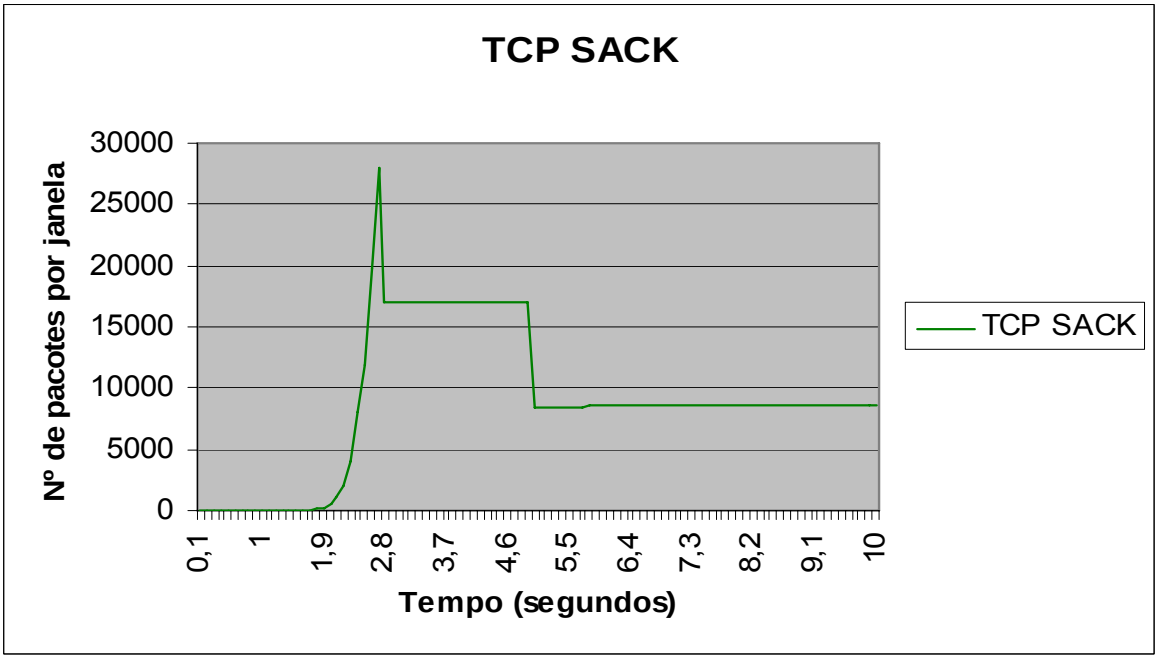

Figura 5.8- TCP SACK no estado estacionário

5.4.4- Simulação com a competição dos fluxos

A competição dos fluxos mostra como estas três implementações competem entre si. Na figura 5.9 observa-se o comportamento do Tahoe, Reno e SACK, e fica claro que o Sack obtém o melhor desempenho dentre as implementações. Este ganho de desempenho frente ao Tahoe é explicado pelo fato do SACK não utilizar o modo de Partida Lenta, reduzindo a janela para o valor de um segmento. Assim, o SACK entra imediatamente no modo Prevenção de Congestionamento, após o recebimento do reconhecimento total, conseguindo estar com um valor de janela sempre maior do que no caso do Tahoe.

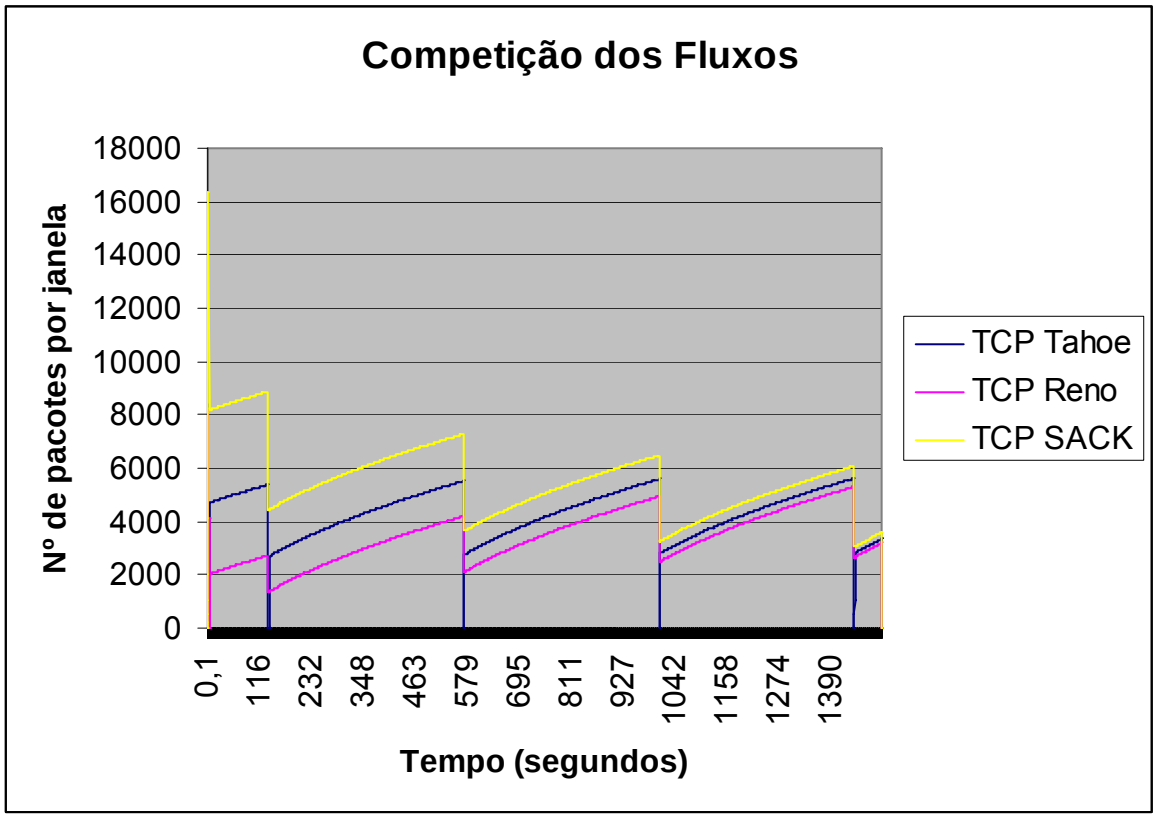

Fig. 5.9- Tahoe, Reno e SACK disputando o canal de comunicação

Neste caso, o Tahoe, em relação ao Reno, tem um melhor desempenho, isto acontece porque o Tahoe, em caso de perda, reinicia o modo Partida Lenta no mesmo instante, ajustando o valor do *ssthresh* pela metade do valor da *cwnd* somente uma vez, e reduz a *cwnd* a um segmento, aumentando-a exponencialmente até atingir o valor da

variável ssthresh, enquanto o Reno, na percepção das perdas, entra na Recuperação Rápida reduzindo a variável *ssthresh* pela metade, no entanto, acaba por esgotar o temporizador, levando-o ao modo de Partida Lenta, reduzindo novamente a *ssthresh*, como pode ser observado na figura 5.10. Isto ocorre porque à medida que mais segmentos são perdidos dentro de uma janela de dados, a janela de congestionamento cai pela metade a cada reconhecimento parcial. Isto pode fazer com que esta chegue a um valor baixo o suficiente a ponto de não permitir novos envios. A conseqüência de poucos envios será a ausência de reconhecimentos em quantidade para disparar a retransmissão de um dos segmentos que foram descartados dentro da janela e, assim, causando o esgotamento do temporizador. De forma muito clara este fato é apresentado em [10].

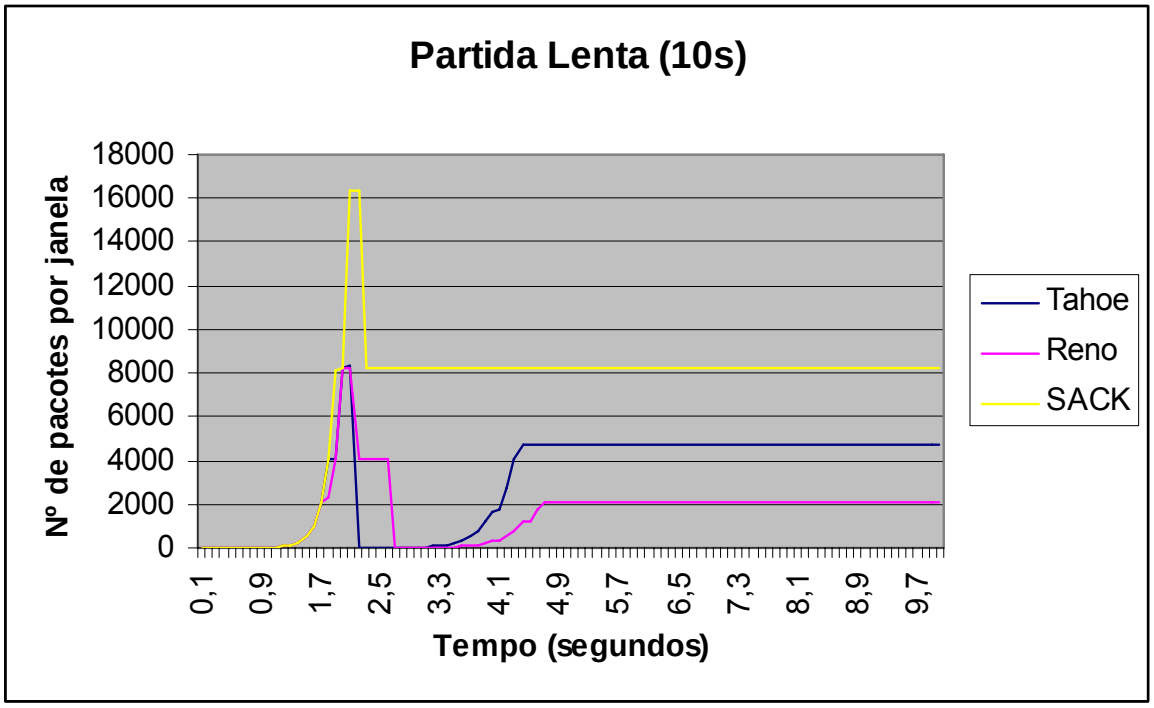

Figura 5.10- Tahoe, Reno e SACK no algoritmo Partida Lenta

5.5- Discussão dos Resultados

O melhor desempenho dentre os protocolos analisados em todas as simulações, foi o SACK, no entanto, nas simulações realizadas com um único fluxo, todas as três implementações levam aproximadamente vinte dois minutos, no estado estacionário, para atingir a janela máxima, como pode ser notado nas figuras, 5.3, 5.5 e 5.7, isto mostra a

dificuldade que as implementações padrões do TCP levam para conseguir encher o canal de comunicação. A dificuldade do protocolo TCP padrão atingir a janela máxima é justificada pelo algoritmo de prevenção de congestionamento, no qual é utilizado o algoritmo AIMD, que controla o tamanho da janela de congestionamento de forma dinâmica, ou seja, faz com que a janela, no estado estacionário, aumente a *cwnd* em um segmento por RTT, e o diminua pela metade quando uma perda é sinalizada pela duplicação de ACKs. Quando a perda é detectada por temporização, o *ssthresh* é ajustado para a metade do valor atual da *cwnd* e a *cwnd* é reiniciada em um segmento. Então, o algoritmo Partida Lenta é utilizado novamente até que a janela atinja o *ssthresh*, portanto, este mecanismo provoca um baixo desempenho em redes de alta velocidade.

È importante observar que o melhor desempenho do SACK em relação ao Reno, ocorre devido ao mecanismo de reconhecimentos, pois permite o reenvio de segmentos, no caso de múltiplas perdas, de forma mais eficiente que no Reno, Um fato que deve ser destacado, é em relação ao algoritmo de controle de congestionamento, que é o mesmo nas implementações do Reno e do Sack. Desta forma, a ocupação do canal no estado estacionário da conexão, será realizada da mesma forma no caso destas implementações. Portanto, podemos afirmar que a ocupação do canal no estado estacionário da conexão é independente da versão do protocolo, quando comparados o Reno e o SACK. Fica claro que a maior eficiência do SACK em relação ao Reno ocorre de fato na fase transitória de recuperação de perdas, que é muito mais robusta e rápida na implementação do protocolo TCP SACK.

No entanto, um fator que deve ser observado nos resultados destas simulações, é que outras restrições que impedem que o TCP alcance uma alta taxa de transferências, tais como: reduzida capacidade de buffer de rede, limitação no buffer do TCP, casos de congestionamento na rede e grande perdas de segmentos na Partida Lenta, não estão sendo tratadas neste caso.

O protocolo Tahoe é sem duvida uma implementação mais conservativa em condições de congestionamento à medida que, a cada nova perda de segmento que é sinalizada pelo estouro do temporizador ou por reconhecimentos duplicados, a implementação retorna ao modo de Partida Lenta. Isto faz com que um novo ponto de equilíbrio seja procurado com o custo de ter que reduzir a janela de congestionamento até o valor unitário.

Qualquer versão do TCP necessita de, pelo menos, três ACKs duplicados para executar o algoritmo de Recuperação Rápida. No caso do Tahoe, o algoritmo de

Retransmissão Rápida é seguido do algoritmo Partida Lenta, portanto, ele trata este instante como o início de uma conexão e não depende da chegada de mais reconhecimentos. A adoção desta estratégia pelo Tahoe implicará no esvaziamento do canal mesmo em situação de perda de um único segmento, em que não há necessidade de uma reação tão radical. Assim, a eficiência do Tahoe fica comprometida. Quanto maior o RTT da conexão, maior será a degradação da vazão, pois o aumento da janela demora mais a ocorrer. Quando a janela chega a *ssth*, o Tahoe entra no modo Prevenção do Congestionamento.

O protocolo Reno é um pouco menos conservativo do que o protocolo Tahoe, quando evita reduzir a janela de congestionamento a um segmento. Este protocolo tem como opção a retransmissão de um segmento perdido em cada RTT, que é o tempo necessário para a chegada de reconhecimentos após cada retransmissão, trazendo novas informações ao emissor. O Reno inclui o mecanismo de Recuperação Rápida, criado para evitar o esvaziamento do canal em situações onde não ocorrem múltiplas perdas de segmentos e, assim, melhorar esse comportamento em relação ao Tahoe. De fato, o Reno solucionou algumas limitações do Tahoe, mas por outro lado introduziu novos problemas. Como foi visto, em situações em que ocorrem múltiplas perdas em uma janela de dados, faz com que o Reno reduza a janela de congestionamento pela metade repetidamente, tantas vezes quanto forem os segmentos descartados em uma janela. Isso é causado pelo fato do Reno executar a Retransmissão Rápida e entrar na Recuperação Rápida a cada perda verificada. Logo, o TCP Reno acaba por reduzir a janela muito mais do que seria necessário para contornar uma situação de congestionamento. Este fato atrapalha em muito o desempenho do Reno, uma vez que, ao sair da Recuperação Rápida, ele passa para o modo de Prevenção do Congestionamento, no qual o crescimento da janela será linear, tornando muito lento o retorno da janela ao seu tamanho ideal.

A tendência é que o protocolo Reno se comporte melhor do que o Tahoe, quando houver poucas perdas. Quando o número de perdas aumenta, a ambição do protocolo Reno pode levar ao esgotamento do temporizador, o que acarretará a execução do algoritmo de Partida lenta. E, portanto, terá um desempenho inferior ao do Tahoe, pois este já terá se antecipado na execução do algoritmo Partida Lenta e atingirá o modo de Prevenção do Congestionamento antes do Reno.

Contudo, pode-se dizer que, nos momentos quando a perda de segmentos é baixa, o protocolo TCP pode tirar proveito das modificações introduzidas pelo TCP Reno.

Porém, quando o congestionamento aumenta, o TCP Tahoe parece ser a solução mais razoável de uma forma muito robusta e eficiente, pois nestes momentos a ambição do protocolo Reno se volta contra ele, fazendo com que seu desempenho seja menor que o Tahoe.

Em resumo, quando há múltiplas perdas de segmentos por janela, o TCP não pode tirar proveito das alterações introduzidas pelo TCP Reno, pois quando o congestionamento aumenta o TCP Tahoe se mostra como a opção mais razoável em relação ao Reno.

No entanto, o TCP SACK consegue alcançar o melhor desempenho em relação às implementações do TCP padrão, que utilizam o mecanismo de Recuperação Rápida, pois consegue se recuperar com mais rapidez quando ocorrem múltiplas perdas. Um fato importante a considerar é que as implementações do protocolo Reno e SACK, só diferem pelo fato do SACK ter introduzido o mecanismo de reconhecimentos seletivos, logo, a justificativa para o melhor desempenho do SACK, em relação ao Reno, é justamente a ausência do reconhecimento seletivo no Reno.

A fundamental conseqüência da ausência do mecanismo de reconhecimentos seletivos, é que o transmissor deverá escolher entre as seguintes estratégias, ou retransmitir no máximo um segmento descartado por RTT, ou retransmitir segmentos que já podem ter sido recebidos com sucesso. O Reno utiliza a primeira estratégia, enquanto o Tahoe utiliza a segunda.

6 - Conclusão e Trabalhos Futuros

6.1 - Conclusão

A necessidade de maior largura de banda tem produzido inovações tecnológicas que vem aumentando em inúmeras vezes a capacidade de banda disponível. Novas tecnologias, tais como enlaces óticos t~em possibilitado transferir diversos *terabytes* de informação em um pequeno espaço de tempo. No entanto, a utilização plena destes enlaces óticos é atingida com dificuldade pelo TCP, principalmente quando estes enlaces fazem parte de conexões de longa distância. Conseqüentemente, várias aplicações de redes tornam-se incapazes de utilizar estas novas redes de alta velocidade de maneira eficiente.

O objetivo deste trabalho foi estudar as implementações do protocolo TCP em redes de alta velocidade e longa distância. Nos próximos parágrafos são citadas as conlusões sobre os experimentos.

A partir das simulações, podemos verificar deficiências nas implementações do protocolo TCP padrão. A principal deficiência, observado no presente estudo, acontece no estado estacionário, no qual as três implementações levam um tempo aproximado de vinte dois minutos para atingir a janela máxima da rede. A razão para este fraco desempenho está relacionada com o produto de banda pelo atraso do enlace gargalo e com o algoritmo clássico AIMD, por sua maneira conservadora de aumentar a janela de congestionamento. Dado que, para cada segmento reconhecido durante a fase de Prevenção de Congestionamento, a *cwnd* é aumentada em 1/*cwnd* e o intervalo entre cada aumento é de 1 RTT, logo, a evolução da janela de congestionamento é lenta, mesmo que o TCP alcance a janela máxima, este crescimento lento faz com que a utilização do enlace seja subutilizada durante um período significativo de tempo. Esta situação é apresentada nas figuras 5.3, 5.5 e 5.7, onde é mostrado cada uma das implementações no estado estacionário.

6.2 – Trabalhos futuros

Em [15] e [27] são apresentados estudos sobre o TCP de Alta Velocidade (*HighSpeed* TCP – HSTCP). Nestes trabalhos, foram realizadas simulações e experimentos, nos quais implementações do TCP padrão e o TCP de Alta Velocidade são comparados.

Em [27] é argumentado que o TCP de Alta Velocidade de fato tem um desempenho melhor que o TCP SACK, para enlaces de alta velocidade e longa distância.

O TCP de alta Velocidade aumenta sua taxa de transferência mais rapidamente e sua recuperação de um evento de congestionamento leva menos tempo. Estas características aumentam a sua utilização de enlace. Foi também mostrado que a porção de banda usada pelos fluxos do TCP de Alta Velocidade foi maior que a usada por fluxos do TCP SACK, quando ambos os fluxos competiam pelo mesmo enlace. Outro fato que foi apresentado, é que o TCP de Alta Velocidade apresenta uma melhor adaptabilidade a ambientes com taxa de eventos de congestionamento variável.

No estudo feito por [15], os resultados obtidos sobre o desempenho do protocolo TCP em redes de alta velocidade foram os seguintes: o problema de eficiência do TCP está relacionado ao mecanismo de controle de congestionamento empregado que, por sua natureza conservadora, não é capaz de manter enlaces de alta velocidade totalmente ocupados; o algoritmo AIMD, que controla o tamanho da janela de congestionamento do TCP, depende de taxas de perda de segmentos irrealmente baixas para manter alto o valor da janela. Uma modificação deste mecanismo, chamada HSTCP, permite resolver esta deficiência através do uso de um novo algoritmo para o controle de congestionamento. Em altas taxas de perda, esse algoritmo tem um funcionamento similar ao TCP padrão, mas quando em baixas taxas de perda, é mais agressivo no crescimento da janela e, por outro lado, mais conservador na sua redução, quando ocorrem perdas de segmentos.

Para trabalhos futuros sugiro um estudo sobre o comportamento das implementações do protocolo TCP em redes de alta velocidade reais e também o estudo de implementações que apresenta, modificações para operarem redes de alta velocidade, como por exemplo o TCP de Alta Velocidade

7- Referência Bibliográfica

[1] Comer, Douglas E., “Interligação em Rede com TCP/IP Volume 1: Princípios, Protocolos e Arquitetura”, Editora Campus.

[2] A. S. Tanenbaum., “Redes de Computador”, Quarta Edição, Editora Campus.

[3] M. Hassan, Jain R., “High Performance TCP/IP Networking: Concepts, Issues, and Solutions”, Pearson Prentice Hal.

[4] V. Jacobson, R. Braden, and D. Borman. “TCP extensions for high performance”,Internet Engineering Task Force, Maio 1992. RFC1323.

[5] V. Jacobson, M. J. Karels, “Congestion avoidance and control”. In Proceedings of SIGCOMM 'Symposium on Communication Architectures and Protocols, pp 314-329 1988.

[6] S. Floyd, HighSpeed TCP for Large Congestion Windows. Internet draf, Fed. 2003.

[7] J. Nagle, 1984. *“*Congestion Control in IP/TCP Internetworks,” RFC 896, 9 pages(Jan.). Description of the Nagle algorithm.

[8] S. Floyd. Limited slow-start for TCP with large congestion windows, Maio 2002. Internet draft draft-floyd-tcp-slowstart-00b.txt, work in progress

[9] R. Carlson. Tackling the end-to-end performance problem. Large Scale Networking Workshop, Março 2001. URL http://www.ngisupernet.org/lsn2000/Argonne_Natl_Lab-Carlson.pdf.

[10] S. Floyd, K. Fall, “Simulation-based Comparisons of Tahoe, Reno and Sack TCP”, *Computer Communication Review*, Vol. 26 (3), Julho,1996.

[11] T. Duningan, M. Mathis, and B. Tierney. A TCP tuning daemon. In Procedings of IEEE Super Computing 2002, 2002.

[12] T. Lakshman and U. Madhow. Performance analysis of window-based flow control using TCP/IP: Effect of high bandwidth-delay products and random loss. In Fifth International Conference on High Performance Networking, pages 135–149, Junho 1994.

[13] J. Lee, D. Gunter, B. Tierney, W. Allock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area network. In Computing in High Energy and Nuclear Physics, Beijing, China, Abril 2001. LBNL-46269.

[14] Huston Geoff, Gigabit TCP, The Internet Protocol Journal, Volume 9, Number 2, june 2006.

[15] J F. Rezende, L. H. Costa, M. G. Rubistein, Avaliação Experimental e Simulação do Protocolo TCP em Redes de Alta Velocidade, XII Simpósio Brasileiro de Telecomunicações, setembro 2005.

[16] A. Falk, T. Faber, J. Bannister, A. Chien, R. Grossman, J. Leigh, Transport Protocols for High Performance, Communications of the ACM, November 2003, vol 46, n° 11, pg 43-49.

[17] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms”, Internet RFC 2001, Janeiro 1997.

[18] V. Jacobson, “Modified TCP Congestion Avoidance Algorithm”, tech. rep., E-mail to the end2end-interest mailing list, 30th. Apr. 1990. URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[19] S. Floyd e T. Henderson, “The New Reno Modification to TCP’s Fast Recovery Algorithm”, Internet RFC 2582, abril 1999.

[20] M. Mathis, J. Mahdavi, S. Floyd e A. Romanow, “TCP Selective Acknowledgment Options”, Ineternet RFC 2018, outubro 1996.

[21] S. Floyd, S. Ratnasamy, e S. Shenker, “Modifying TCP’s Congestion Control for High Speeds”, maio 2002, Disponível em *www.icir.org/floyd/papers/hstcp.pdf,* download setembro de 2006.

[22] T. V. Lakshman, U. Madhow, “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Randon Loss”, IEEE/ACM Transactions on Networking, Vol 5, nº 3, julho 1997.

[23] Tirumala, Cottrell, Dunigan, “Measuring end-to-end bandwidth with Iperf using Web, Pan 2003. Donwload setembro de 2006 em http://www.csm.ornl.gov/~dunigan/pam.pdf.

[24] L. S. Brakmo e L. L. Peterson, “TCP Vegas: End to End Congestion Avoidance on global internet”, *IEEE Journal on Selected Areas in Communications*, vol. 13, nº 8, pp. 1465-1480, outubro 1995.

[25] J. S. Ahn, P. Danzing, Z. Liu, e L. Yan, “Evaluation of Vegas: Emulation and Experiment”, em Proceedings of ACM SIGCOMM, *Symposium on Communication Architectures and Protocols*, 1995.

[26] M. Allman and A. Falk, "On the Effective Evaluation of TCP," ACM Computer Communication. Review, vol. 29, no. 5, Oct. 1999.

[27] E. de Souza, “Estudo por Simulação do Protocolo TCP de Alta Velocidade”, Dissertação de Mestrado, Agosto de 2003. 146 f., Dissertação (Mestre em Engenharia Elétrica)- Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2003.

[28] Manual NS, Disponível em: http://www.isi..edu/nsnam/ns/ns-documentation.html.

[29] Postel, J. B., ed. 1981 "Transmission Control Protocol", RFC 793, 85 páginas.

[30] Mathis, M., Mahdavi, j., Floyd, S, e Romanow, A., “TCP Selective Acknowledgement Options – RFC 2018”, Outubro 1996.

[31] Braden, R., “Requeriments for Internet *Host*s – Communication Layers RFC 1122” October 1989.

[32] Allman, M., Paxson, V., Stevens, W., “TCP Congestion Control – RFC 2581”, April 1999.

[33] Roesler, V., “Desempenho em Redes TCP/IP” maio 2001, Unisinos, UFRGS.

[34] Postel J., “TCP Maximum Segment Size and Related Topics – RFC 879”, Novembro 1983.

[35] Mogul J., Deering S., “Path MTU Discovery – RFC 1191” Novembro 1990.

8 – Anexo

8.1- Artigo

1 - Introdução

Uma das grandes questões na área de conectividade, atualmente, é a crescente demanda por uma largura de banda. Diversas tecnologias têm emergido e aumentado, em muito, a capacidade dos canais de comunicação. Atualmente, vários meios de conexões estão disponíveis para conectar dois pontos em alta velocidade, tanto para conexões locais, como para ligações de longa distância.

O protocolo TCP (Transmission Control Protocol) foi desenvolvido na década de 70 e, desde então, tem sido constantemente modificado para adaptar-se a novas aplicações [1], [2]. Além de acomodar-se às características particulares dos novos meios de transmissão, e melhorar sua capacidade de transferência de dados. Atualmente, o TCP é o protocolo de transporte mais utilizado na Internet. Aplicações populares como Web, FTP e correio eletrônico foram projetadas para operarem sobre este protocolo [3]. O desempenho fim-a-fim que os usuários esperam destes serviços depende muito do desempenho do próprio TCP.

O controle de congestionamento do TCP é constituído por diferentes mecanismos. Em estado estacionário, o principal algoritmo de controle de congestionamento utilizado é o AIMD (Additive-Increase, Multiplicative- Decrease). Portanto, a janela de congestionamento do TCP aumenta, linearmente, a cada reconhecimento positivo (ACK) recebido e, decresce exponencialmente, a cada segmento perdido. Através disto, o AIMD adapta a taxa de transmissão ao congestionamento da rede. No entanto, o AIMD apresenta alguns problemas para redes de alta velocidade, principalmente quando o produto da largura de banda pelo atraso é grande, pois a lentidão do aumento da janela de congestionamento provoca uma baixa utilização da largura de banda disponível.

O presente trabalho teve como propósito analisar o comportamento do protocolo TCP em redes de alta velocidade. Para isto, fez uso de simulações de redes com algumas das implemetações do TCP, que são ela: Tahoe, Reno e SACK. Cada uma destas implementações são diferenciadas, pelo conjunto de algoritmos que utilizam no controle de congestionamento. O Tahoe utiliza como mecanismo de controle de congestionamento os algoritmos, Partida Lenta, Prevenção de Congestionamento e Retransmissão Rápida, o Reno utiliza os mesmos algoritmos do Tahoe com a diferença de adicionar o de Recuperação Rápida ao de Retransmissão Rápida. O SACK, por sua vez, utiliza o mesmo

controle de congestionamento do Reno com a adição do Reconhecimento Seletivo e Retransmissão seletiva.

Este trabalho está organizado da seguinte maneira. A seção 2 descreve as características principais do protocolo TCP. A seção 3 descreve as implementações TCP utilizadas nas simulações. A seção 4 descreve os parâmetros utilizados nos cenários de experimentação das simulações, enquanto a seção 5 faz a análise comparativa dos resultados. A seção 6 conclui este trabalho.

2-TCP

2.1- Fundamentos

O protocolo TCP viabiliza a transmissão de dados de forma confiável através de um meio físico susceptível a falhas e, também, é responsável pelo controle do congestionamento. A confiabilidade baseia-se no procedimento de reconhecimento (ACK) positivo cumulativo, o qual mantém o transmissor informado dos pacotes que chegaram com sucesso a seu destino e do próximo que está sendo aguardado. De acordo com o procedimento adotado pelo TCP, se o transmissor não detectar o reconhecimento de um segmento, num intervalo de tempo especificado, ele será enviado novamente. A falta de um reconhecimento sinaliza um provável congestionamento. O tempo de espera que o TCP usa para sinalizar a perda de um segmento é denominado “Retransmit Timeout” (RTO) e é determinado a partir de estimativas do “Round Trip Time” (RTT), representando um parâmetro que afeta decisivamente no desempenho do protocolo.

O protocolo TCP, para evitar que o transmissor de dados sobrecarregue os recursos disponibilizados pelo receptor de dados, possui um mecanismo de controle de fluxo, permitindo que vários pacotes sejam transmitidos enquanto os seus reconhecimentos são aguardados. Os pacotes que podem ser enviados sem que sejam reconhecidos são agrupados em uma janela de dados denominada janela deslizante. À medida que os pacotes vão sendo reconhecidos, a janela é atualizada com novos pacotes e os já reconhecidos são excluídos.

Cada segmento TCP contém um campo de 16 bits, chamado tamanho de janela de recepção, que indica o número de bytes que o receptor pode aceitar sem estourar seu buffer de recepção. Com isso, o número máximo de segmentos que o TCP pode enviar, sem receber um ACK, é limitado pelo tamanho dessa janela de recepção, logo, esses 16 bits tornam-se uma limitação para que o número de segmentos em trânsito seja maior que

64kbytes. Para contornar este problema, foi criado um mecanismo de multiplicação do valor da janela de recepção por um fator negociado na fase de estabelecimento de conexão. Esse mecanismo denominado Window Scale está descrito na RFC 1323 [4].

As primeiras versões do TCP dispunham apenas do mecanismo de reconhecimento cumulativo e do comando *advertised* para controlar o funcionamento e o tamanho da janela deslizante, e consequentemente a taxa de transmissão da conexão. Versões atuais do TCP incorporam outros procedimentos para ajustar a taxa de transmissão de uma conexão, com a finalidade de prevenir o congestionamento, procurando manter a maior vazão possível. Estes procedimentos são implementados por quatro algoritmos denominados Partida Lenta, Prevenção do Congestionamento, Retransmissão Rápida e Recuperação Rápida.

2.2-Partida Lenta

O algoritmo Partida Lenta é usado no início de uma conexão TCP. Nesta fase, o TCP objetiva fazer com que a taxa de transmissão convirja lentamente para a capacidade disponível da rede. O tamanho da janela de transmissão, cwnd, começa em um segmento e aumenta um segmento para cada ACK recebido. O crescimento da cwnd é exponencial e acaba quando alcança o limiar, que é chamado ssthresh (SlowSstart Threshold).

Depois da fase Partida Lenta, o TCP passa à fase de Prevenção de Congestionamento, na qual é utilizado o algoritmo AIMD para controlar o tamanho da janela de congestionamento de forma dinâmica. O AIMD funciona da seguinte forma: primeiramente, a janela cwnd cresce a 1/cwnd a cada RTT. Quando uma perda é sinalizada, a cwnd é diminuída pela metade e a fase Prevenção do Congestionamento recomeça desse novo valor. Quando a perda é detectada por temporização, o ssthresh é ajustado para a metade do valor atual da cwnd e a cwnd é colocada em um segmento. Então, o algoritmo Partida Lenta é reiniciado até que a cwndI atinja o novo valor do ssthresh.

Resumidamente, para uma conexão TCP que transmite cwnd segmentos de tamanho MSS bytes a cada RTT segundos, a vazão é dada por:

$$(\text{cwnd} \times \text{MSS}) / \text{RTT}, \qquad (1)$$

na fase de Partida lenta, a cada ACK recebido a janela cresce de acordo com a seguinte equação:

$$cwnd = cwnd + c, \qquad (2)$$

onde c = 1. Na fase Prevenção de Congestionamento para cada ACK recebido:

$$cwnd = cwnd + a / cwnd, \qquad (3)$$

onde a = 1, e para cada segmento perdido:

$$cwnd = cwnd – bx\ cwnd, \qquad (4)$$

onde b = 0,5.

Este mecanismo de Prevenção de Congestionamento do TCP limita o tamanho máximo da janela qua pode ser alcançado nas atuais redes gigabits. Por exemplo, no caso de uma conexão TCP de segmentos de 1500 bytes e de RTT a 100ms, para que seja atingida uma vazão em estado estacionário de 10Gbps, obtém-se a partir da equação 1 que o tamanho médio de cwnd deve ser 83333 segmentos [27].

2.3- Retransmissão Rápida e Recuperação Rápida

Modificações no algoritmo de Prevenção do Congestionamento foram propostas em [17] e [18]. Antes de descrever a mudança, note que o TCP é obrigado a gerar uma confirmação imediata (um ACK duplicado) quando um segmento fora de ordem é recebido. A finalidade deste ACK duplicado é indicar ao emissor que um segmento foi recebido fora de ordem e qual o número de seqüência esperado.

Partindo do fato que não se sabe se um ACK duplicado foi causado por um segmento perdido ou é somente uma reordenação de segmentos, espera-se que um pequeno número de ACKs duplicados sejam recebidos antes que qualquer atitude seja tomada. É assumido que, se for somente uma reordenação de segmentos, só serão recebidos um ou dois ACKS duplicados antes do segmento fora de ordem alcançar o destino e ser processado, o que implicará em um novo ACK.

Se três ou mais ACKs duplicados forem recebidos em seguida, é um forte indício de que um segmento foi perdido. O TCP realiza, então, a retransmissão imediata do que aparenta ser o segmento perdido, sem esperar que o temporizador expire (timeout). Este é o algoritmo de Retransmissão Rápida. Em seguida, o algoritmo Prevenção de

Congestionamento, e não o algoritmo de Partida Lenta, é feito. Este é o algoritmo de Recuperação Rápida.

A razão pela qual não se faz slow start, nesse caso, é que o recebimento de ACKs duplicados diz mais do que simplesmente um segmento foi perdido. Sabe-se que o destino só pode gerar ACKs duplicados quando outro segmento for recebido, isto é, o segmento deixou a camada física e está no buffer do destino. Logo, ainda temos dados trafegando entre os dois nós. Então, não é aconselhável reduzir o fluxo abruptamente, usando o Partida Lenta.

2.4- Opção de Reconhecimento Seletivos

O TCP tradicional realiza um esquema de reconhecimentos cumulativos, como citado anteriormente, em que ACK(n) indica que todos os segmentos até (n - 1) foram recebidos com sucesso e o transmissor pode transmitir o segmento (n). Isto impõe uma grave limitação de desempenho, em situações nas quais há mais de uma perda em uma mesma janela de transmissão, porque o transmissor tem que esperar um RTT para identificar cada segmento perdido. Isto ocorre porque, caso haja diversas perdas, os reconhecimentos são enviados, referenciando o primeiro pacote perdido, não dando maiores informações sobre as demais perdas ao emissor. Assim, o emissor reenvia apenas o pacote referenciado pelos ACKs duplicados, tendo que aguardar por um RTT a chegada do reconhecimento do pacote retransmitido, que estará indicando o próximo pacote perdido. Desta forma, apenas um segmento perdido pode ser notado a cada RTT. A alternativa para esta limitação seria retransmitir mais do que o necessário, como por exemplo, todos os pacotes após aquele primeiro que foi perdido. Medidas desse tipo, entretanto, além de ineficientes podem acabar agravando uma situação de congestionamento na rede.

Uma solução para este problema seria o uso de reconhecimentos seletivos (selective acknowledgement, ou SACK) por parte do TCP. Com esta modificação o emissor tem mais informações sobre quais segmentos foram recebidos corretamente e quais não foram, podendo, portanto, retransmitir somente os pacotes necessários, aumentando a eficiência na recuperação de perdas.

Esta modificação usa o campo OPTIONS (opções) do cabeçalho TCP para incluir informações de reconhecimento seletivo. Existem dois tipos de campos opcionais que são usados pelo TCP SACK: um para estabelecer se a opção de SACK será ou não usada, que é enviada ao iniciar uma conexão, e o SACK propriamente dito, que passa informações sobre os segmentos reconhecidos.

3- Implementações

3.1- TCP Tahoe

O Tahoe foi o primeiro a adicionar novos algoritmos e refinamentos na sua implementação. Os novos algoritmos incluidos foram, Paritda lenta, Prevenção do Congestionamento e Retransmissão Rápida [5]. O refinamento inclui uma modificação na estimação do Round-trip time, usado para ajustar o valor do timeout de retransmissão, RTO. Todas estas modificações estão descritas em [5].

O algoritmo de Retransmissão Rápida é de interesse especial, já que este está modificado nas versões posteriores a esta. No Tahoe, o algoritmo de Retransmissão Rápida vem seguido do algoritmo Partida Lenta. Quando ACKs duplicados são recebidos, o transmissor ajusta a variável *ssthresh* pela metade do valor da *cwnd* e a *cwnd* para um segmento. Então, o algoritmo de Partida Lenta é reinicializando até que a *cwnd* atinja o valor da *ssthresh*, entrando no algoritmo Prevenção de Congestionamento.

3.2- TCP Reno

O Reno tem todos os melhoramento incorporados no Tahoe, modificando o algoritmo de Retransmissão Rápida com a inclusão do algoritmo de Recuperação Rápida [18]. Este novo algoritmo evita que o canal de comunicação esvazie depois do algoritmo Retransmissão Rápida, evitando que o algoritmo de Partida Lenta seja reiniciado depois que um único segmento seja perdido. Nesta situação, fica claro que qualquer situação de congestionamento que porventura tenha causado a suposta perda, já se esgotou, pois o fluxo de transmissão sobrevive à medida que reconhecimentos continuam chegando. Portanto, não há necessidade de se reduzir o fluxo abruptamente, fazendo com que o algoritmo de partida lenta seja executado.

3.3- TCP SACK

O TCP com reconhecimento seletivo não incorpora qualquer mudança nos algoritmos de controle de congestionamento. Neste aspecto, ele é idêntico ao Reno, apenas apresentando a diferença na política de retransmissões, decorrentes da opção de reconhecimento seletivos, como citado anteriormente. A opção SACK [20] aumenta o desempenho do TCP em redes de alta velocidade e com alto atraso. Com a técnica SACK, o transmissor obtém informações suficientes sobre os segmentos recebidos corretamente pelo receptor e, dessa forma, é capaz de retransmitir em um RTT múltiplos segmentos perdidos de uma janela de dados.

Este tipo de TCP implementa a maioria das melhorias do TCP, disponíveis atualmente e permite que o receptor de uma conexão TCP informe ao transmissor sobre múltiplos pacotes descartados dentro de uma única janela de dados.

4- Implementações nas Simulações

Muito embora o uso de simulação permita a investigação de um protocolo em uma rica variedade de situações, nem todas as condições de rede serviram ao interesse do presente estudo. O cenário de investigação foi limitado ao caso de interesse selecionado, ou seja, o foco principal foi o comportamento das implementações do protocolo TCP, em situações em que ele estivesse em estado estacionário ou perto do estado estacionário. Quando a janela de congestionamento oscila em torno de seu ponto de equilíbrio, é caracterizado por uma forma de onda em dente de serra.

4.1-Simulador

Para a realização da análise do desempenho das implementações do protocolo TCP, foram realizadas simulações, utilizando o *software NS* (*Network Simulation*) [28]. Esta ferramenta de simulação de redes foi desenvolvida na Universidade da Califórnia Berkeley, em parceria com o projeto VINT (*Virtual InterNetwork Testbed*), incluindo como colaboradores o USC/ISI, a Xerox PARC e o LBNL. O simulador *ns* foi escolhido para as simulações devido sua ampla aceitação no meio acadêmico.

Esta análise foi desenvolvida, usando um cenário com uma topologia simples para evitar interações mais complexas e reduzir o número de variáveis a coletar e estudar.

4.2-Topologia

A topologia de rede escolhida para as simulações foi a conhecida barra de pesos, com um único gargalo e está ilustrada na figura 1 e 2. O cenário na topologia 1 consiste em uma Rede de Longa Distância de alta velocidade. Uma única fonte (S1) gera o tráfego para um destino (D1) através de uma conexão TCP. Dois roteadores (R1 e R2) direcionam o tráfego entre as fontes e destinos. Os enlaces entre o nó fonte e R1 e, entre R2 e o destino, possuem uma largura de banda de 10Gbits/s e retardo de 1ms, enquanto que o enlace central entre os roteadores é de 1Gbit/s com o atraso de 50ms. O tráfego gerado pelas fontes é do tipo FTP, onde sempre existe dados disponíveis para serem enviados, ou seja, enquanto o valor da janela permitir, dados estarão sendo transmitidos com a taxa de transmissão permitida pelo canal. O cenário da topologia 2 difere da topologia 1, somente no número de fontes geradoras de tráfego e de destinos, já que este

cenário tem como propósito estudar o comportamento das implementações do TCP, quando competindo entre si, logo, o número de fontes geradoras é igual ao número de implementações utilizadas na simulação, ou seja, três, S1,S2 e S3, e o número de destinos utilizados, também é igual ao número de implementações utilizadas, D1, D2 e D3.

O enlace gargalo foi o enlace principal. A banda do enlace principal foi 1Gbits/seg, o seu atraso foi de 50ms e o tipo de gerenciamento de enfileiramento do roteador foi DT.

O gerenciamento de enfileiramento, no roteador, foi restrito à técnica de descarte DropTail (DT). DT é a técnica tradicional mais simples de realizar o gerenciamento do tamanho de fila do roteador. Ele controla o comprimento da fila, usando um esquema FIFO (First In First Out). Neste esquema, cada novo pacote que chega na porta de entrada da fila, é descartado, quando o espaço de buffer da fila está cheio.

O *Buffer* de roteadores é um elemento essencial para uma rede por comutação de pacotes. Ele absorve a chegada de pacotes em rajadas e reduz o potencial de perdas. Quanto maior o *buffer*, maior será a capacidade de absorver grandes rajadas. No entanto, este fato faz com que a carga e os atrasos nos enfileiramentos aumentem. Van Jacobson propôs, em 1988, que o espaço de *buffer,* na fila, deveria ser não menos que o produto da banda pelo atraso, BDP. Logo, o valor padrão do tamanho do buffer utilizado nas simulações, foi 8.333 pacotes, porque este valor coincide com o BDP para um enlace de 1Gbps, com RTT de 100ms e tamanho de pacote de 1500 bytes.

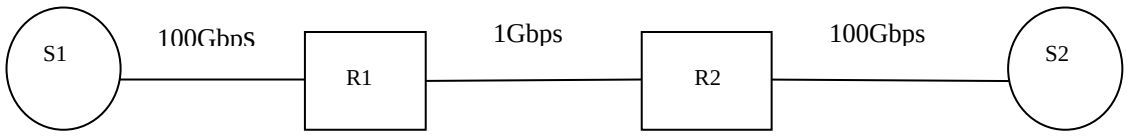

Figura 1- Topologia utilizada nas simulações dos testes dos fluxos individuais

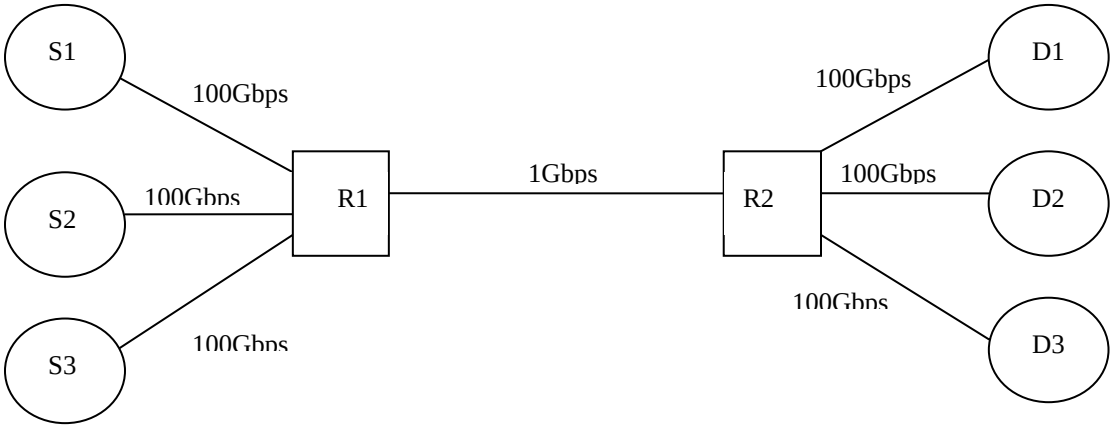

Figura 2- Topologia utilizada na simulação da competição dos fluxos

4.3- Configuração dos fluxos

Fluxos TCP de longa duração são os de maior interesse para enlaces de alta velocidade e longa distância, quando uma grande quantidade de dados precisa ser transmitida e então os fluxos usados neste trabalho tiveram longa duração, no qual o estado estacionário era bem maior que o estado transiente. Para isto, o tráfego gerado pelas fontes é do tipo FTP, onde sempre existem dados disponíveis para serem enviados, ou seja, enquanto o valor da janela permitir, dados estarão sendo transmitidos com a taxa de transmissão permitida pelo canal.

O conjunto de parâmetros comuns utilizados pelos fluxos TCP foram, o tamanho do pacote que foi de 1500 bytes, já que este é o tamanho de mais da metade do volume de tráfego em conexões de longa distância [27]; o tamanho máximo da janela de janela foi 100000 segmento, este valor foi configurado para ser grande o suficiente para não impor limites pelo tamanho máximo da janela de anúncio do receptor e o tamanho mínimo de cabeçalho TCP foi configurado, ou seja, 40 bytes, com nenhum cabeçalho opcional.

5 - Análise Comparativa

Esta seção, finalmente, apresentará os gráficos obtidos a partir das simulações realizadas, acompanhados de análises que explicam o comportamento das curvas, de acordo com a implementação do protocolo TCP utilizada.

Foram utilizados, neste estudo, quatro conjuntos de fluxos. O primeiro conjunto tem apenas um fluxo de dados da implementação Tahoe, o segundo conjunto tem um fluxo de dados da implementação Reno, o terceiro conjunto tem um fluxo de dados da implementação SACK e o quarto conjunto possui ambos os fluxos, Tahoe, Reno e SACK. O quarto conjunto permitiu observar a interação entre os fluxos Tahoe, Reno e SACK.

Estes quatro conjuntos de fluxos foram expostos a uma única condição de rede. No ambiente de rede criado, não havia outra fonte de tráfego e interferência além das geradas pelos conjuntos de fluxos testados. Este ambiente de rede é referenciado como Condição Ideal. Também é assumido que os pontos finais fornecem componentes de hardware suficientemente rápidos, ou seja, barramentos internos e memórias adequadas para tratar gigabits de dados por segundo entre memória e interface de rede. Esta situação é interessante porque é possível estudar o desempenho dos conjuntos de fluxo sem interferência.

A simulação deste experimento objetivou obter um padrão de comparação dos fluxos , bem como derivar informações do comportamento teórico de cada uma das implementações testadas, quando não houvesse interferência externa, de acordo com [26.] . Cada simulação foi executada por 1500 segundos.

5.1- Tahoe

A primeira simulação efetuada utilizou a implementação Tahoe, e é apresentada nas figuras 3 e 4 . Esta figura, assim como as demais analisadas neste capítulo, apresentam em seu eixo horizontal o tempo em segundos e em seu eixo vertical a quantidade de pacotes por janela. A partir do gráfico na figura 3, observa-se que o TCP Tahoe reinicia o modo Partida lenta nos tempos 2.7 e 5.4 segundos, neste instante ocorre uma perda, portanto, esta implementação ao receber um terceiro reconhecimento duplicado para o pacote perdido, faz com que o valor da variável *ssthresh* caia pela metade e a janela de congestionamento reinicie com apenas um segmento, aumentando-a de forma exponencial. Porém, ao igualar o valor salvo em *ssthresh,* o algoritmo Prevenção de Congestionamento é iniciado, fazendo com que a janela aumente de forma linear a uma taxa aproximada de um segmento por RTT, até o instante 1308 segundos, quado novamente ocorre uma perda e todo o processo citado anteriormente é iniciado.

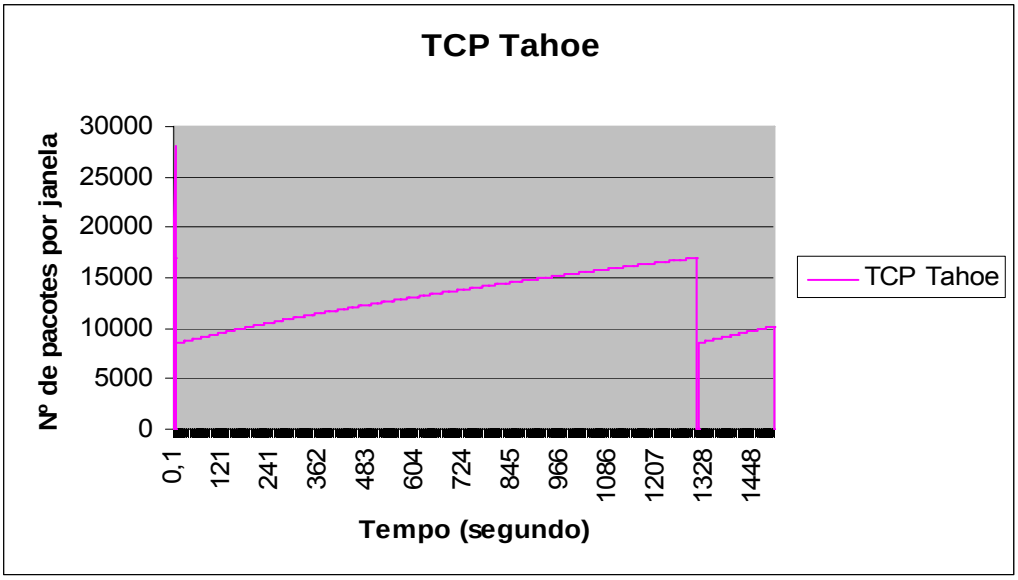

Figura 3- Implementação Tahoe

Na figura 3, observa-se claramente o crescimento exponencial da janela de congestionamento, desde o início da transmissão até a primeira perda, 2.8 segundos com o valor da *cwnd* = 28036 segmentos, neste momento o modo Partida Lenta é reiniciado e a *cwnd* =1 segmento, a segunda perda ocorre em 5.5 segundos, com a *cwnd* = 16999 pacotes, reiniciando a *cwnd* = 1 segmento, no tempo 7.1 segundos, o Tahoe entra no modo de Prevenção de Congestionamento, como mostrado na figura 4, a partir deste ponto a janela deixa de ser incrementada de forma exponencial para ser incrementada de forma linear, esta implementação permanece neste modo até atingir a janela máxima, que ocorre no tempo 1308 segundos, ou seja, leva aproximadamente 22 minutos para atingir a janela máxima, este é o tempo que o Tahoe leva para completar um ciclo no estado estacionário. Ao encher a janela de congestionamento com o valor máximo, o Tahoe reinicia novamente o algoritmo Partida Lenta, fazendo com que a *cwnd* seja igual a um segmento e seu crescimento seja exponencial até que ela atinja o *ssthresh* que tem como valor a metade da janela máxima, a figura 3 apresenta o comportamento do Tahoe no estado estacionário.

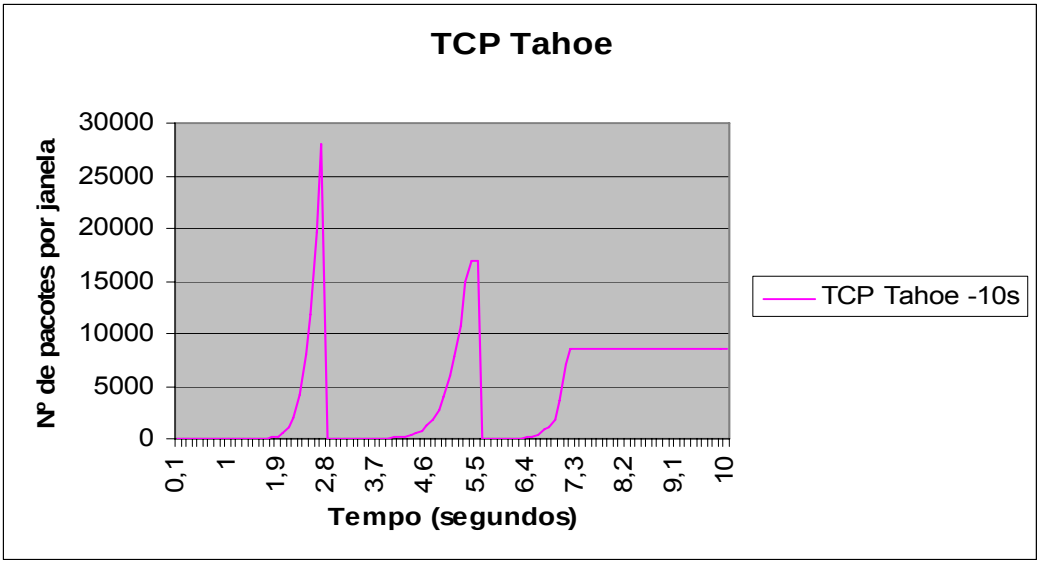

Fig. 4- TCP Tahoe em modo Partida Lenta (os primeiros dez segundos da simulação)

5.2-Reno

A simulação para o TCP Reno é apresentada nas figuras 5 e 6, revelando que o protocolo se comporta, inicialmente, de forma semelhante ao TCP Tahoe, buscando a ocupação do canal com o crescimento exponencial da janela de congestionamento. Da mesma maneira, como observado na figura 5, o transmissor percebe que ocorreu uma perda no tempo 2.7, quando o valor da *cwnd* = 28036 segmentos, neste momento, a janela de congestionamento é setada para *cwnd*/2, ou seja, *cwnd* = 16999 segmentos, no entanto, o TCP Reno ainda fica impedido de enviar mais dados pela enorme quantidade de dados pendentes. Como nenhum reconhecimento é enviado, O TCP Reno esgota o temporizador e os pacotes pendentes são transmitidos. Neste momento, o *ssthresh* é reduzido novamente pela metade da *cwnd*. Ao receber todos os dados pendentes, o TCP Reno entra novamente no modo de Partida Lenta até que a janela de congestionamento, *cwnd*, atinja o valor do *ssthresh*, quando entra no modo de prevenção de congestionamento, e permanece até atingir o valor máximo da janela, que é no instante 1306 segundos, com a *cwnd* = 1700 segmentos. Da mesma forma que o Tahoe, o Reno leva aproximadamente 22 minutos para atingir este valor. No entanto, quando este valor é atingido, a janela de congestionamento passa a valer *cwnd*/2, e o algoritimo Prevenção de Congestionamento é reiniciado, ao contrário do Tahoe, o algoritmo de Partida Lenta não é reiniciado.

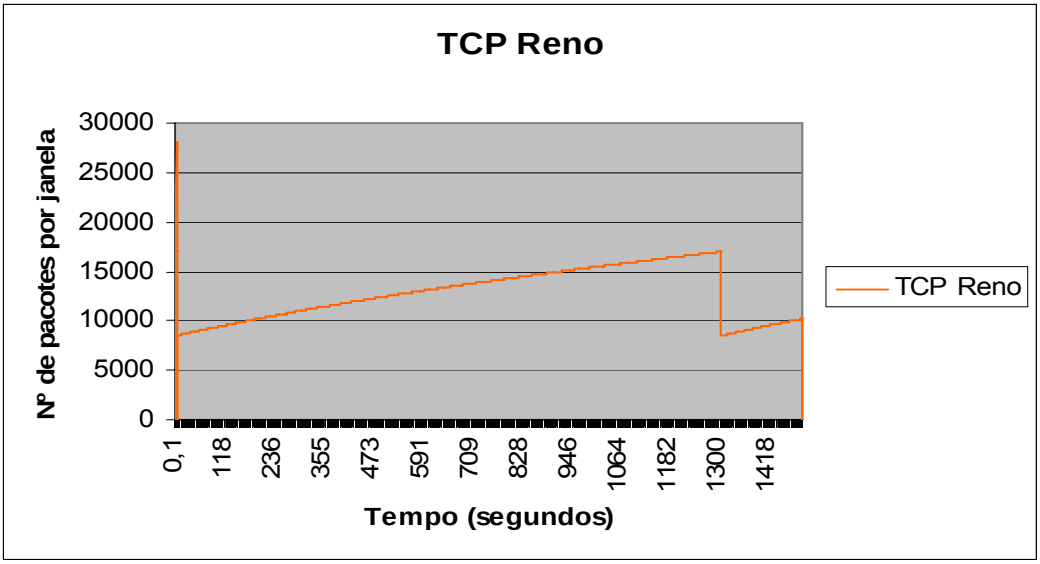

Fig. 5- Implementação Reno

Nas figuras 6 e 4, pode-se comparar um melhor desempenho na implementação do Reno, em relação ao Tahoe. Isto ocorre por que o Tahoe é uma implementação cautelosa em termos de contenção de congestionamento. O Tahoe age de forma a garantir, com larga margem de confiança, a estabilidade da rede mesmo que para isso tenha que prejudicar, em alguns casos, seu desempenho.

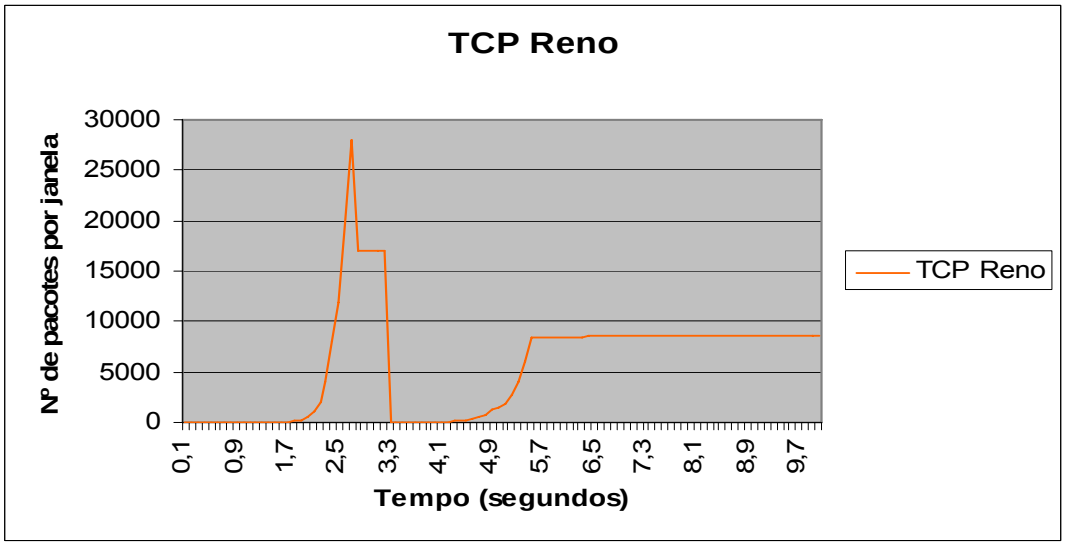

Fig. 6-TCP Reno no modo Partida Lenta (os primeiros dez segundos da simulação)

Este fato fica claro quando se observa que, para qualquer perda, o algoritmo Partida Lenta é iniciado, reduzindo a janela de congestionamento ao tamanho unitário,

independentemente do nível de congestionamento da rede. Por outro lado, o TCP Reno procura manter a ocupação do canal através do mecanismo Rápida Recuperação, permitindo uma recuperação mais rápida. No entanto, em algumas situações, como será mostrado mais adiante, o Tahoe pode ter um melhor desempenho que o Reno.

5.3- SACK

As figuras 7 e 8 apresentam os gráficos nos quais o comportamento na simulação desta implementação, é ilustrada.

Este protocolo, como os outros já citados, também inicia com o algoritmo Partida Lenta, de maneira semelhante às outras implementações, tentando ocupar o mais rapidamente possível o meio de transmissão. É interessante observar que, neste caso, a opção por reconhecimentos seletivos é utilizada de fato, ou seja, no instante 2.7 segundos, quando o SACK atinge o valor máximo da janela de congestionamento, *cwnd* = 28036 segmentos, ocorrem várias perdas na mesma janela e o SACK, ao contrário do Reno, não permite que o temporizador esgote, evitando assim que a *cwnd* passe a valer um segmento e o algoritmo Partida Lenta seja reiniciado, como pode ser observado na figura 8.

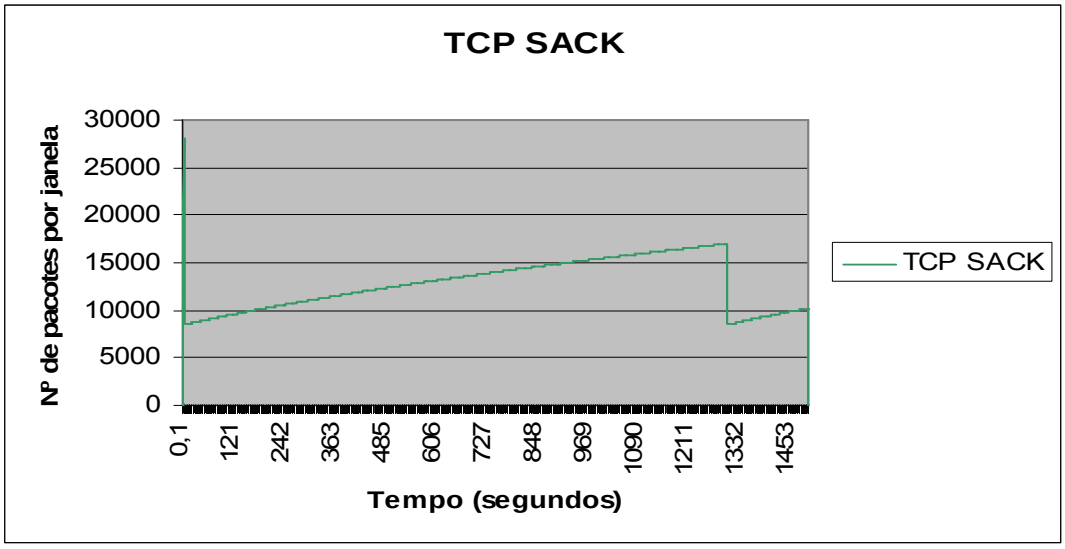

Fig 7- Implementação SACK

Quando todos os reconhecimentos dos pacotes pendentes são recebidos, o SACK entra no modo Prevenção do Congestionamento com variáveis *ssthresh* e *cwnd*, que já haviam sido reduzidas duas vezes seguidas pela metade do valor da janela de

congestionamento, no momento das perdas. A partir deste ponto, o SACK tem o comportamento igual ao do Reno.

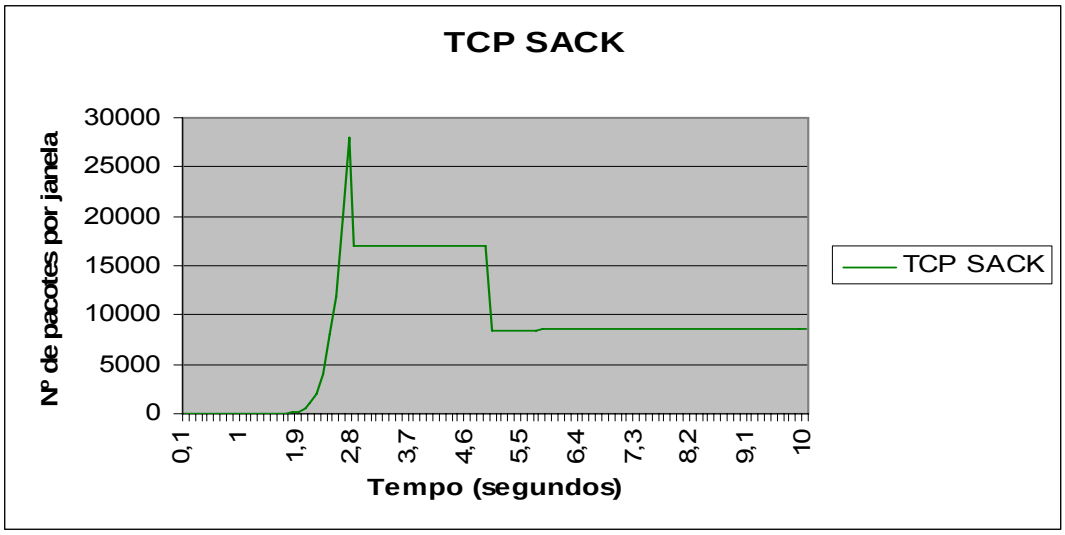

Fig 8- TCP SACK Partida Lenta (os primeiros 10 segundos da simulação)

5.4- Simulação com a competição dos fluxos

A competição dos fluxos mostra como estas três implementações competem entre si. Na figura 9, observa-se o comportamento do Tahoe, Reno e SACK, e fica claro que o Sack obtém o melhor desempenho dentre as implementações. Este ganho de desempenho frente ao Tahoe é explicado pelo fato do SACK não utilizar o modo de Partida Lenta, reduzindo a janela para o valor de um segmento. Assim, o SACK entra imediatamente no modo Prevenção de Congestionamento, após o recebimento do reconhecimento total, conseguindo estar com um valor de janela sempre maior do que no caso do Tahoe.

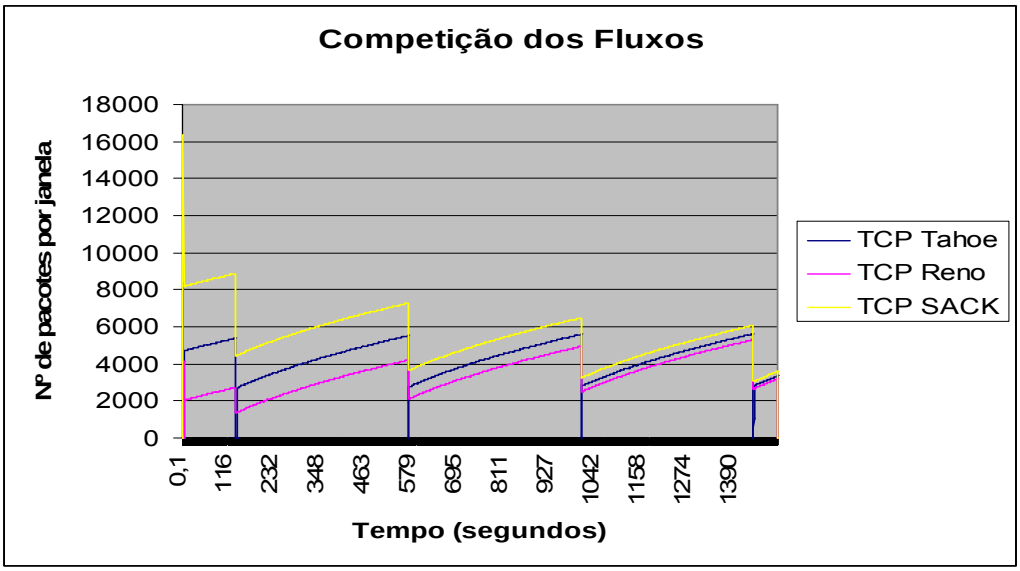

Fig.9- Tahoe, Reno e SACK disputando o canal de comunicação

Neste caso, o Tahoe, em relação ao Reno, tem um melhor desempenho, isto acontece porque o Tahoe, em caso de perda, reinicia o modo Partida Lenta no mesmo instante, ajustando o valor do sshthresh pela metade do valor da *cwnd* somente uma vez, e reduz a *cwnd* a um segmento, aumentando-a exponencialmente até atingir o valor da variável sshthresh, enquanto o Reno, na percepção das perdas, entra na Recuperação Rápida, reduzindo a variável *ssthresh* pela metade, no entanto, acaba por esgotar o temporizador, levando-o ao modo de Partida Lenta, reduzindo novamente a *ssthresh*, como pode ser observado na figura 10. Isto ocorre porque, à medida que mais pacotes são perdidos dentro de uma janela de dados, a janela de congestionamento cai pela metade a cada reconhecimento parcial. Isto pode fazer com que esta chegue a um valor baixo o suficiente a ponto de não permitir novos envios. A conseqüência de poucos envios será a ausência de reconhecimentos em quantidade suficiente para disparar a retransmissão de um dos pacotes que foram descartados dentro da janela e, assim, causando o esgotamento do temporizador. De forma muito clara, este fato é apresentado em [10].

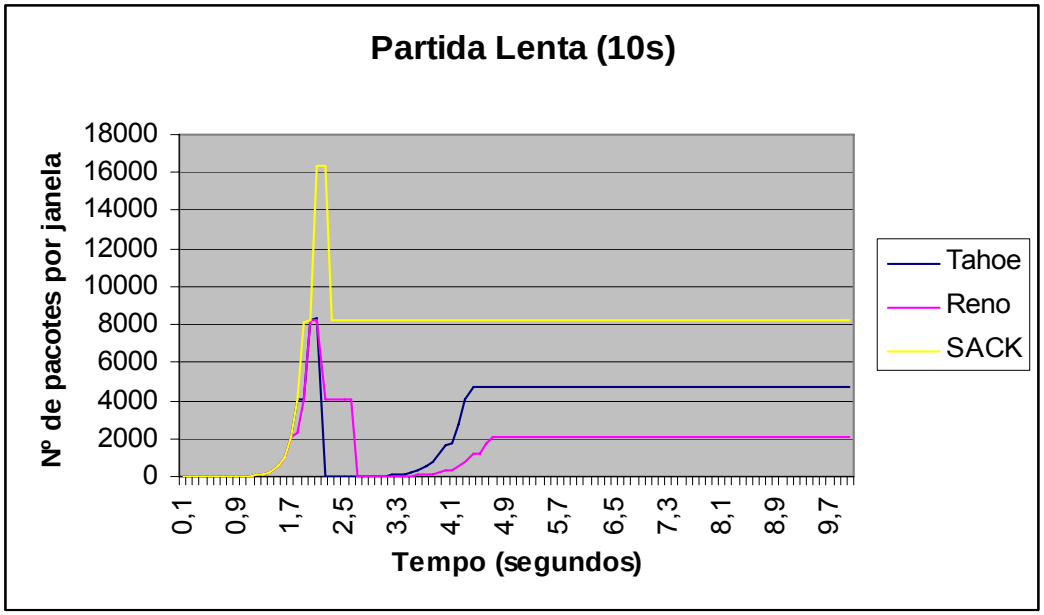

Fig 10- Tahoe, Reno e SACK no algoritmo Partida Lenta (os primeiros dez segundos da simulação)

O melhor desempenho dentre os protocolos analisados em todas as simulações, foi o SACK, no entanto, nas simulações realizadas com um único fluxo, todas as três implementações levam aproximadamente vinte dois minutos, no estado estacionário, para atingir a janela máxima, como pode ser notado nas figuras 3, 5 e 7, isto mostra a dificuldade que as implementações padrões do TCP levam para conseguir encher o canal de comunicação. A dificuldade do protocolo TCP padrão em atingir a janela máxima é justificada pelo algoritmo de prevenção de congestionamento, no qual é utilizado o algoritmo AIMD, que controla o tamanho da janela de congestionamento de forma dinâmica, ou seja, faz com que a janela, no estado estacionário, aumente a *cwnd* em um segmento por RTT, e o diminua pela metade quando uma perda é sinalizada pela duplicação de ACKs. Quando a perda é detectada por temporização, o *ssthresh* é ajustado para a metade do valor atual da *cwnd* e a *cwnd* é reiniciada em um segmento. Então, o algoritmo Partida Lenta é utilizado novamente até que a janela atinja o *ssthresh*, portanto, este mecanismo provoca um baixo desempenho em redes de alta velocidade.

É importante observar que o melhor desempenho do SACK em relação ao Reno, ocorre devido ao mecanismo de reconhecimentos, pois permite o reenvio de pacotes, no caso de múltiplas perdas, de forma mais eficiente que no Reno, Um fato que deve ser destacado, é em relação ao algoritmo de controle de congestionamento, que é o mesmo nas implementações do Reno e do Sack. Desta forma, a ocupação do canal no estado

estacionário da conexão, será realizada da mesma forma no caso destas implementações. Portanto, podemos afirmar que a ocupação do canal no estado estacionário da conexão é independente da versão do protocolo, quando comparados o Reno ao SACK. Fica claro que a maior eficiência do SACK em relação ao Reno ocorre de fato, na fase transitória de recuperação de perdas, que é muito mais robusta e rápida na implementação do protocolo TCP SACK.

6 - Conclusão e Trabalhos Futuros

6.1 - Conclusão

A necessidade de maior largura de banda tem produzido inovações tecnológicas que vem aumentando em inúmeras vezes a capacidade de banda disponível. Novas tecnologias, tais como enlaces óticos t~em possibilitado transferir diversos *terabytes* de informação em um pequeno espaço de tempo. No entanto, a utilização plena destes enlaces óticos é atingida com dificuldade pelo TCP, principalmente quando estes enlaces fazem parte de conexões de longa distância. Conseqüentemente, várias aplicações de redes tornam-se incapazes de utilizar estas novas redes de alta velocidade de maneira eficiente.

O objetivo deste trabalho foi estudar as implementações do protocolo TCP em redes de alta velocidade e longa distância. Nos próximos parágrafos são citadas as conlusões sobre os experimentos.

A partir das simulações, podemos verificar deficiências nas implementações do protocolo TCP padrão. A principal deficiência, observado no presente estudo, acontece no estado estacionário, no qual as três implementações levam um tempo aproximado de vinte dois minutos para atingir a janela máxima da rede. A razão para este fraco desempenho está relacionada com o produto de banda pelo atraso do enlace gargalo e com o algoritmo clássico AIMD, por sua maneira conservadora de aumentar a janela de congestionamento. Dado que, para cada segmento reconhecido durante a fase de Prevenção de Congestionamento, a *cwnd* é aumentada em 1/*cwnd* e o intervalo entre cada aumento é de 1 RTT, logo, a evolução da janela de congestionamento é lenta, mesmo que o TCP alcance a janela máxima, este crescimento lento faz com que a utilização do enlace seja subutilizada durante um período significativo de tempo. Esta situação é apresentada nas figuras 5.3, 5.5 e 5.7, onde é mostrado cada uma das implementações no estado estacionário.

6.2 – Trabalhos futuros

Em [15] e [27] são apresentados estudos sobre o TCP de Alta Velocidade (*HighSpeed* TCP – HSTCP). Nestes trabalhos, foram realizadas simulações e experimentos, nos quais implementações do TCP padrão e o TCP de Alta Velocidade são comparados.

Em [27] é argumentado que o TCP de Alta Velocidade de fato tem um desempenho melhor que o TCP SACK, para enlaces de alta velocidade e longa distância. O TCP de alta Velocidade aumenta sua taxa de transferência mais rapidamente e sua recuperação de um evento de congestionamento leva menos tempo. Estas características aumentam a sua utilização de enlace. Foi também mostrado que a porção de banda usada pelos fluxos do TCP de Alta Velocidade foi maior que a usada por fluxos do TCP SACK, quando ambos os fluxos competiam pelo mesmo enlace. Outro fato que foi apresentado, é que o TCP de Alta Velocidade apresenta uma melhor adaptabilidade a ambientes com taxa de eventos de congestionamento variável.

No estudo feito por [15], os resultados obtidos sobre o desempenho do protocolo TCP em redes de alta velocidade foram os seguintes: o problema de eficiência do TCP está relacionado ao mecanismo de controle de congestionamento empregado que, por sua natureza conservadora, não é capaz de manter enlaces de alta velocidade totalmente ocupados; o algoritmo AIMD, que controla o tamanho da janela de congestionamento do TCP, depende de taxas de perda de segmentos irrealmente baixas para manter alto o valor da janela. Uma modificação deste mecanismo, chamada HSTCP, permite resolver esta deficiência através do uso de um novo algoritmo para o controle de congestionamento. Em altas taxas de perda, esse algoritmo tem um funcionamento similar ao TCP padrão, mas quando em baixas taxas de perda, é mais agressivo no crescimento da janela e, por outro lado, mais conservador na sua redução, quando ocorrem perdas de segmentos.

Para trabalhos futuros sugiro um estudo sobre o comportamento das implementações do protocolo TCP em redes de alta velocidade reais e também o estudo de implementações que apresenta, modificações para operarem redes de alta velocidade, como por exemplo o TCP de Alta Velocidade

7 - Referências Bibliográficas

[1] Comer, Douglas E., “Interligação em Rede com TCP/IP Volume 1: Princípios, Protocolos e Arquitetura”, Editora Campus.

[2] A. S. Tanenbaum., “Redes de Computador”, Quarta Edição, Editora Campus.

[3] M. Hassan, Jain R., “High Performance TCP/IP Networking: Concepts, Issues, and Solutions”, Pearson Prentice Hal.

[4] V. Jacobson, R. Braden, and D. Borman. “TCP extensions for high performance”,Internet Engineering Task Force, Maio 1992. RFC1323.

[5] V. Jacobson, M. J. Karels, “Congestion avoidance and control”. In Proceedings of SIGCOMM 'Symposium on Communication Architectures and Protocols, pp 314-329 1988.

[6] S. Floyd, HighSpeed TCP for Large Congestion Windows. Internet draf, Fed. 2003.

[7] J. Nagle, 1984. *“*Congestion Control in IP/TCP Internetworks,” RFC 896, 9 pages(Jan.). Description of the Nagle algorithm.

[8] S. Floyd. Limited slow-start for TCP with large congestion windows, Maio 2002. Internet draft draft-floyd-tcp-slowstart-00b.txt, work in progress

[9] R. Carlson. Tackling the end-to-end performance problem. Large Scale Networking Workshop, Março 2001. URL http://www.ngisupernet.org/lsn2000/Argonne_Natl_Lab-Carlson.pdf.

[10] S. Floyd, K. Fall, “Simulation-based Comparisons of Tahoe, Reno and Sack TCP”, Computer Communication Review, Vol. 26 (3), Julho,1996.

[11] T. Duningan, M. Mathis, and B. Tierney. A TCP tuning daemon. In Procedings of IEEE Super Computing 2002, 2002.

[12] T. Lakshman and U. Madhow. Performance analysis of window-based flow control using TCP/IP: Effect of high bandwidth-delay products and random loss. In Fifth International Conference on High Performance Networking, pages 135–149, Junho 1994.

[13] J. Lee, D. Gunter, B. Tierney, W. Allock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area network. In Computing in High Energy and Nuclear Physics, Beijing, China, Abril 2001. LBNL- 46269.

[14] Huston Geoff, Gigabit TCP, The Internet Protocol Journal, Volume 9, Number 2, june 2006.

[15] J F. Rezende, L. H. Costa, M. G. Rubistein, Avaliação Experimental e Simulação do Protocolo TCP em Redes de Alta Velocidade, XII Simpósio Brasileiro de Telecomunicações, setembro 2005.

[16] A. Falk, T. Faber, J. Bannister, A. Chien, R. Grossman, J. Leigh, Transport Protocols for High Performance, Communications of the ACM, November 2003, vol 46, n° 11, pg 43-49.

[17] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms”, Internet RFC 2001, Janeiro 1997.

[18] V. Jacobson, “Modified TCP Congestion Avoidance Algorithm”, tech. rep., E-mail to the end2end-interest mailing list, 30th. Apr. 1990. URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[19] S. Floyd e T. Henderson, “The New Reno Modification to TCP’s Fast Recovery Algorithm”, Internet RFC 2582, abril 1999.

[20] M. Mathis, J. Mahdavi, S. Floyd e A. Romanow, “TCP Selective Acknowledgment Options”, Ineternet RFC 2018, outubro 1996.

[21] S. Floyd, S. Ratnasamy, e S. Shenker, “Modifying TCP’s Congestion Control for High Speeds”, maio 2002, Disponível em www.icir.org/floyd/papers/hstcp.pdf, download setembro de 2006.

[22] T. V. Lakshman, U. Madhow, “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Randon Loss”, IEEE/ACM Transactions on Networking, Vol 5, nº 3, julho 1997.

[23] Tirumala, Cottrell, Dunigan, “Measuring end-to-end bandwidth with Iperf using Web, Pan 2003. Donwload setembro de 2006 em http://www.csm.ornl.gov/~dunigan/pam.pdf.

[24] L. S. Brakmo e L. L. Peterson, “TCP Vegas: End to End Congestion Avoidance on global internet”, IEEE Journal on Selected Areas in Communications, vol. 13, nº 8, pp. 1465-1480, outubro 1995.

[25] J. S. Ahn, P. Danzing, Z. Liu, e L. Yan, “Evaluation of Vegas: Emulation and Experiment”, em Proceedings of ACM SIGCOMM, Symposium on Communication Architectures and Protocols, 1995.

[26] M. Allman and A. Falk, "On the Effective Evaluation of TCP," ACM Computer Communication. Review, vol. 29, no. 5, Oct. 1999.

[27] E. de Souza, “Estudo por Simulação do Protocolo TCP de Alta Velocidade”, Dissertação de Mestrado, Agosto de 2003. 146 f., Dissertação (Mestre em

Engenharia Elétrica)- Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2003.

[28] Manual NS, http://www.isi..edu/nsnam/ns/ns-documentation.html

[29] Postel, J. B., ed. 1981 "Transmission Control Protocol", RFC 793, 85 páginas.

[30] Mathis, M., Mahdavi, j., Floyd, S, e Romanow, A., “TCP Selective Acknowledgement Options – RFC 2018”, Outubro 1996.

[31] Braden, R., “Requeriments for Internet Hosts – Communication Layers RFC 1122” October 1989.

[32] Allman, M., Paxson, V., Stevens, W., “TCP Congestion Control – RFC 2581”, April 1999.

[33] Roesler, V., “Desempenho em Redes TCP/IP” maio 2001, Unisinos, UFRGS.

[34] Postel J., “TCP Maximum Segment Size and Related Topics – RFC 879”, Novembro 1983.

[35] Mogul J., Deering S., “Path MTU Discovery – RFC 1191” Novembro 1990.

8.2 - Código Fonte dos Testes

8.2.1 – Código do Teste Individual do TCP Tahoe

```
set ns [new Simulator]

set file1 [open out.tr w]
set winfile [open WinFile1000 w]

proc finish {} {
        global ns file1
        $ns flush-trace
        close $file1
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 100000Mb 1ms DropTail
$ns duplex-link $n1 $n2 1000Mb 50ms DropTail
$ns duplex-link $n2 $n3 100000Mb 1ms DropTail

$ns queue-limit $n1 $n2 8333
$ns trace-queue $n1 $n2 $file1

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 100000
$tcp set packetSize_ 1460

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

$ns at 1.0 "$ftp start"
$ns at 1499.0 "$ftp stop"

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 1500.0 "finish"
$ns run
```

8.2.2 – Código do Teste Individual do TCP Reno

```
set ns [new Simulator]

set file1 [open out.tr w]
set winfile [open WinReno w]

proc finish {} {
        global ns file1
        $ns flush-trace
        close $file1
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 100000Mb 1ms DropTail
$ns duplex-link $n1 $n2 1000Mb 50ms DropTail
$ns duplex-link $n2 $n3 100000Mb 1ms DropTail

$ns queue-limit $n1 $n2 8333
$ns trace-queue $n1 $n2 $file1

set tcp [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 100000
$tcp set packetSize_ 1460

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

$ns at 1.0 "$ftp start"
$ns at 1499.0 "$ftp stop"

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"

$ns at 1500.0 "finish"
$ns run
```

8.2.3 - Código do Teste dos fluxos competido

```
set ns [new Simulator]

set file1 [open out.tr w]
set winfile [open WinSack w]

proc finish {} {
        global ns file1
        $ns flush-trace
        close $file1
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 100000Mb 1ms DropTail
$ns duplex-link $n1 $n2 1000Mb 50ms DropTail
$ns duplex-link $n2 $n3 100000Mb 1ms DropTail

$ns queue-limit $n1 $n2 4166
$ns trace-queue $n1 $n2 $file1

set tcp [new Agent/TCP/Sack1]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/Sack1]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 100000
$tcp set packetSize_ 1460

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

$ns at 1.0 "$ftp start"
$ns at 1499.0 "$ftp stop"

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"

$ns at 1500.0 "finish"
$ns run
```

8.2.3 - Código do Teste dos fluxos competido

```
set ns [new Simulator]

set file1 [open out.tr w]
set winfile [open WinS w]
set winfile1 [open WinT w]
set winfile2 [open WinR w]

proc finish {} {
        global ns file1
        $ns flush-trace
        close $file1
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

$ns duplex-link $n0 $n3 100000Mb 1ms DropTail
$ns duplex-link $n1 $n3 100000Mb 1ms DropTail
$ns duplex-link $n2 $n3 100000Mb 1ms DropTail
$ns duplex-link $n3 $n4 1000Mb 50ms DropTail
$ns duplex-link $n4 $n5 100000Mb 1ms DropTail
$ns duplex-link $n4 $n6 100000Mb 1ms DropTail
$ns duplex-link $n4 $n7 100000Mb 1ms DropTail

$ns queue-limit $n3 $n4 8333
$ns trace-queue $n3 $n4 $file1

set tcp1 [new Agent/TCP/Sack1]
$ns attach-agent $n0 $tcp1
set sink1 [new Agent/TCPSink/Sack1]
$ns attach-agent $n5 $sink1
$ns connect $tcp1 $sink1
$tcp1 set fid_ 1
$tcp1 set window_ 100000
$tcp1 set packetSize_ 1460

set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n6 $sink2
$ns connect $tcp2 $sink2
$tcp2 set fid_ 2
$tcp2 set window_ 100000
$tcp2 set packetSize_ 1460

$ns attach-agent $n2 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n7 $sink3
```

```
$ns connect $tcp3 $sink3
$tcp3 set fid_ 3
$tcp3 set window_ 100000
$tcp3 set packetSize_ 1460

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP

$ns at 0.5 "$ftp1 start"
$ns at 1499.5 "$ftp1 stop"
$ns at 0.5 "$ftp2 start"
$ns at 1499.5 "$ftp2 stop"
$ns at 0.5 "$ftp3 start"
$ns at 1499.5 "$ftp3 stop"

proc plotWindow {tcpSack tcpTahoe tcpReno file1 file2 file3} {
global ns
set time 0.1
set now [$ns now]
set cwnd1 [$tcpSack set cwnd_]
set cwnd2 [$tcpTahoe set cwnd_]
set cwnd3 [$tcpReno set cwnd_]
puts $file1 "$now $cwnd1"
puts $file2 "$now $cwnd2"
puts $file3 "$now $cwnd3"
$ns at [expr $now+$time] "plotWindow $tcpSack $tcpTahoe $tcpReno $file1
$file2 $file3" }
$ns at 0.1 "plotWindow $tcp1 $tcp2 $tcp3 $winfile $winfile1 $winfile2"

$ns at 1500.0 "finish"
$ns run
```