Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Anderson Pereira Deluiz Santos

HIGHSPEED TCP:

Comparando o HSTCP a outras implementações TCP

Rio de Janeiro

2010

Anderson Pereira Deluiz Santos

HIGHSPEED TCP: Comparando o HSTCP a outras implementações TCP

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Orientador:

Moacyr Henrique Cruz de Azevedo, M.Sc., UFRJ, Brasil

Rio de Janeiro

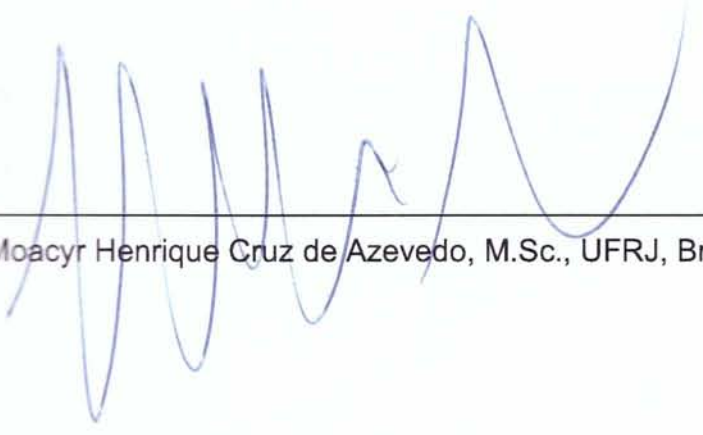
2010

Anderson Pereira Deluiz Santos

HIGHSPEED TCP: Comparando o HSTCP a outras implementações TCP

Monografia apresentada para obtenção do título de Especialista em Gerência de Redes de Computadores no Curso de Pós-Graduação Lato Sensu em Gerência de Redes de Computadores e Tecnologia Internet do Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – NCE/UFRJ.

Aprovada em abril de 2010.



Moacyr Henrique Cruz de Azevedo, M.Sc., UFRJ, Brasil

Este trabalho é dedicado àqueles que me apóiam em todo o tempo, minha família, meus reais amigos e a você, Fernanda. Não posso esquecer de todo o apoio dado pelos meus companheiros do curso MOT-CN turma 2006 que perdura até os dias de hoje.

AGRADECIMENTOS

Gostaria de agradecer a todos que de alguma forma viabilizaram a confecção deste material, seja pelo apoio moral e principalmente por ter entendido minha ausência física durante o período de confecção deste trabalho.

RESUMO

SANTOS, Anderson Pereira Deluiz. **HIGHSPEED TCP: Comparando o HSTCP a outras implementações TCP**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2010.

HighSpeed TCP (HSTCP) é uma proposição de atualização ao algoritmo de controle de congestionamento utilizado no protocolo TCP quando este último utiliza grandes janelas de congestionamento em redes de alta velocidade e latência. Definida na RFC 3649 do IETF, o HSTCP ainda detém o status de experimental, apesar de estável e utilizável. Neste trabalho são apresentadas as características do HSTCP conforme originalmente desenvolvido, comparando-o com outras implementações muito usadas do TCP.

ABSTRACT

SANTOS, Anderson Pereira Deluiz. **HIGH SPEED TCP: Comparando o HSTCP a outras implementações TCP**. Monografia (Especialização em Gerência de Redes e Tecnologia Internet). Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2010.

HighSpeed TCP (HSTCP) is a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows and high spped networks. Defined in IETF RFC 3649, HSTCP still holds the status of experimental, although it is stable and usable. This work presents the characteristics of HSTCP as originally developed by comparing it with other widely used TCP implementations.

LISTA DE FIGURAS

	Página
Figura 2-1 - Cabeçalho TCP	13
Figura 2-2 - Janela Deslizante	15
Figura 2-3 - Opção SACK-Permitted	18
Figura 2-4 - Opção SACK	18
Figura 3-1 - Fast Retransmit	24
Figura 3-2 - TCP Reno	26
Figura 3-3 - TCP NewReno	28
Figura 3-4 - TCP SACK	31
Figura 3-5 - Exemplo de retransmissão no TCP Vegas	34
Figura 4-1 - Topologia de rede de simulação	39
Figura 5-1 - Utilização de banda	40
Figura 5-2 - Evolução dos números de sequência	41
Figura 5-3 - Janela de congestionamento (Reno, NewReno e SACK)	42
Figura 5-4 - Janela de Congestionamento (Tahoe, Vegas e Westwood)	43
Figura 5-5 - Janela de Congestionamento Highspeed TCP	43
Figura 5-6 - Janela de Congestionamento Scalable TCP	44
Figura 5-7 - Duração da fase de slow start	45
Figura 5-8 - Taxa de perda de segmentos	46
Figura 5-9 - Tempo de recuperação em função do aumento de banda	47
Figura 5-10 - Janela de Congestionamento em função do uso de banda	48

LISTA DE ABREVIATURAS E SIGLAS

CWR	*Congestion Window Reduced*
ECE	*Explicit Congestion Notification Echo*
FIN	*Finalize*
GB	*Gigabyte*
HBDP	*High Bandwidth Delay Product*
HSTCP	*High Speed TCP*
IP	*Internet Protocol*
KB	*Kilobyte*
MB	*Megabyte*
Mbps	*Megabits per second*
MSS	*Maximum Segment Size*
MTU	*Maximum Transfer Unit*
NOP	*No Operation*
PSH	*Push*
RFC	*Request For Comments*
RST	*Reset*
RTO	*Retransmit Timeout*
RTT	*Round Trip Time*
SACK	*Selective Acknowledgement*
STCP	*Scalable TCP*
SYN	*Synchronize*
TCP	*Transmission Control Protocol*
URG	*Urgent*

SUMÁRIO

	Página
1 INTRODUÇÃO	11
2 O PROTOCOLO TCP	12
2.1 VISÃO GERAL	12
2.2 CABEÇALHO TCP	12
2.3 JANELA DESLIZANTE (*SLIDING WINDOW*)	15
2.4 MSS – *MAXIMUM SEGMENT SIZE*	16
2.5 *WINDOW SCALE*	17
2.6 *TIMESTAMP*	17
2.7 SACK – *SELECTIVE ACKNOWLEDGMENT*	18
2.8 *SLOW START*	19
2.9 *CONGESTION AVOIDANCE*	20
2.10 *FAST RETRANSMIT* E *FAST RECOVERY*	22
3 IMPLEMENTAÇÕES TCP	24
3.1 TAHOE	24
3.2 RENO	25
3.3 NEWRENO	27
3.4 SACK – *SELECTIVE ACK*	29
3.5 VEGAS	31
3.6 TCP WESTWOOD	34
3.7 SCALABLE TCP	35
3.8 HIGH SPEED TCP	36
4 METODOLOGIA	38
4.1 AMBIENTE COMPUTACIONAL	38
4.2 TOPOLOGIA DE REDE UTILIZADA	38
5 TESTES REALIZADOS	40
6 CONCLUSÕES	49
7 REFERÊNCIAS	51

1 INTRODUÇÃO

O protocolo Internet Protocol (IP) é, indiscutivelmente, o protocolo de camada de rede mais utilizado por sistemas computacionais para interconexão à redes de computadores heterogêneas. Entretanto, sua operação é baseada no modo de datagrama, não permitindo que a transmissão de dados ocorra de forma segura, confiável e sequenciada. Sempre que se faz necessário o uso de transmissões confiáveis, como na transferência de arquivos entre sistemas, é utilizado o Transmission Control Protocol (TCP) na camada de transporte, operando acima do protocolo IP.

Inicialmente definido na RFC 793 em 1981, o TCP foi aprimorado ao longo dos anos. Várias versões foram criadas e estas implementam várias características distintas. Por ser um protocolo aberto, cada fabricante pode escolher qual versão e características deseja implementar em seus equipamentos e sistemas. Estas características vão desde aspectos de configuração até a reação a determinados eventos, como a reação ao congestionamento de tráfego de dados. A detecção, controle e recuperação do congestionamento do tráfego de dados no TCP é essencial para que sejam alcançadas melhores taxas de transmissão.

Este estudo compara os algoritmos de controle de congestionamento das versões do TCP mais utilizadas, as versões conhecidas como Reno, NewReno, Tahoe, SACK e Vegas, com o algoritmo de controle e recuperação de congestionamento HighSpeed TCP (HSTCP), definido na RFC 3649. Ainda em status experimental, o algoritmo já é utilizado em equipamentos comerciais destinados a aceleração na transmissão de dados através de redes metropolitanas ou de maior abrangência.

2 O PROTOCOLO TCP

2.1 VISÃO GERAL

Conforme a RFC 2581, um aprimoramento da RFC 793, o TCP é um protocolo de comunicação, e não um software [1]. O protocolo especifica o formato dos dados, como devem ser feitas e desfeitas as conexões lógicas e a troca de informações de controle entre dois equipamentos para que seja possível a troca confiável de dados. Especifica também especifica como o TCP distingue dentre os vários destinos em cada máquina e como os equipamentos em comunicação se recuperam de erros.

Todavia, o protocolo não detalha as interfaces entre aplicativos e o TCP, e pouco assume a respeito do sistema de comunicação subjacente, levando em consideração apenas as operações que o TCP fornece. Esta ausência de detalhamento possibilita liberdade na implementação do protocolo a um programador, portabilidade entre os mais variados sistemas e o uso do protocolo através de uma grande quantidade de sistemas de entrega, como linhas telefônicas discadas, conexão sem fio com ruídos ou uma rede de fibra ótica de alta velocidade.

2.2 CABEÇALHO TCP

No TCP, a unidade de transferência entre dois equipamentos é chamada de segmento. Segmentos são trocados entre sistemas para transferência de informações de controle do TCP e de dados provenientes da camada de aplicação. Os segmentos são divididos em duas partes, um cabeçalho, que é um conjunto de informações necessárias que possibilitam uma transferência segura e controlada dos pacotes pela rede, seguido por um conjunto de dados.

A figura 2-1 mostra o formato do segmento TCP.

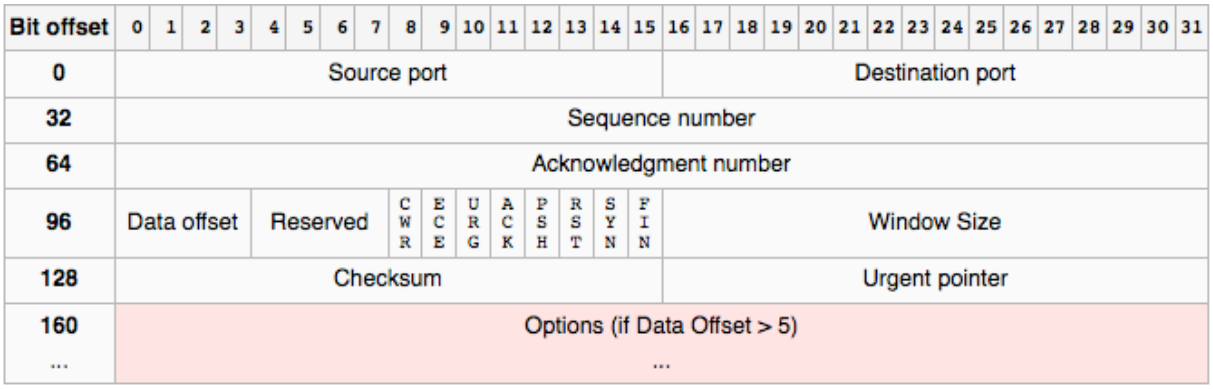


Figura 2-1 - Cabeçalho TCP

Source port e *Destination port* identificam a porta de envio e destino do segmento, e servem também para identificar internamente nos sistemas qual processo enviou o pacote e a qual processo se destina o pacote.

Sequence number é o valor que identifica o segmento para que haja um controle dos segmentos já recebidos, quais foram perdidos e em que ordem devem ser armazenados.

Acknowlegment number é o campo que o receptor informa ao emissor dos dados que bytes já foram recebidos com sucesso até o momento. Este campo tem o formato numérico e é cumulativo, de modo que mesmo que nenhuma confirmação anterior tenha sido recebida pelo emissor após o envio de um determinado número de segmentos dentro de um intervalo de tempo válido, o recebimento de uma confirmação referente ao último segmento enviado valida a recepção com sucesso de todos os outros anteriores a este.

Window size especifica a capacidade do receptor de receber novos segmentos.

Data Offset especifica o tamanho do cabeçalho em blocos de 32 bits.

Reserved é um campo sem uso ainda e deve ter seu valor sempre 0.

Checksum é um campo utilizado para checagem de ocorrência de erros na transmissão dos dados entre emissor e receptor do segmento.

CWR, ECE, UGR, ACK, PSH, RST, SYN e FIN são campos que especificam bits de controle de um fluxo TCP.

- CWR (Congestion Window Reduced) – é indicado pelo sistema emissor para especificar que este recebeu um segmento com o sinal ECE ligado e está respondendo em modo de controle de congestionamento.
- ECE (ECN Echo) – se o sinal SYN estiver ligado, especifica que o sistema suporta ECN (Explicit Congestion Notification). Caso o sinal SYN não esteja ligado, indica que um datagrama IP com o sinal Congestion Experienced foi recebido durante a transmissão de dados.
- URG (Urgent) – indica que o campo *Urgent Pointer* é significativo.
- ACK (Acknowledgment) – indica que o campo *Acknowledgment number* é significativo.
- PSH (Push) – indica que os dados armazenados em buffer para transmissão devem ser transmitidos imediatamente.
- RST (Reset) - termina a conexão.
- SYN (Synchronize) – indica que os números de sequência (*sequence numbers*) devem ser sincronizados. Somente o primeiro segmento de cada parte envolvida na conexão pode ter este controle ligado.
- FIN (Finalize) – não existem mais dados a serem enviados.

Options é um campo opcional e seu tamanho é determinado pelo campo *Data Offset*, que determina o tamanho do cabeçalho TCP. As opções têm tamanhos variados de dados e usos diferenciados. Toda opção ocupa pelo menos um byte, que indica o tipo da opção. Com exceção das opões de tipo 0 e 1, todas as outras opções possuem mais um byte indicativo do tamanho da opção considerando todos os campos a ela associados.

- Tipo 0 – final da lista de opções.
- Tipo 1 – NOP (No Operation). Indica uma opção vazia servindo apenas como preenchedor (padding) de espaços para que o campo opções tenha tamanho sempre múltiplo de 32 bits.
- Tipo 2 – MSS (Maximum Segment Size). Indica o tamanho máximo de um segmento TCP. Ver seção 2.4.
- Tipo 3 – Window Scale. Ver seção 2.5.
- Tipo 4 – SACK (Selective Acknowledgment Permitted). Ver seção 2.7.
- Tipo 8 – Timestamps. Ver seção 2.6.

Existem ainda outras opções, mas são irrelevantes para o estudo aqui apresentado.

2.3 JANELA DESLIZANTE (*SLIDING WINDOW*)

Um mecanismo básico e presente em todas as versões do TCP é a janela deslizante.

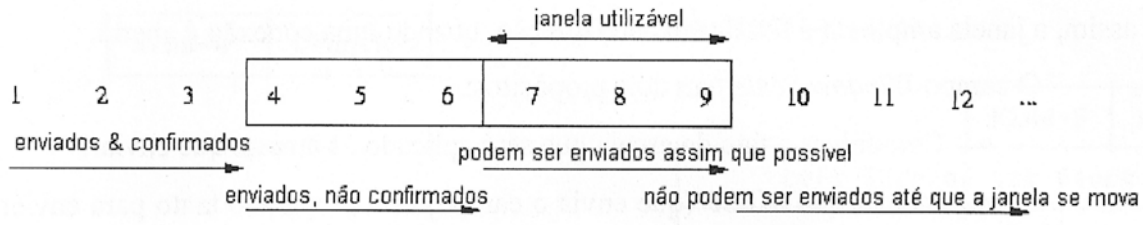


Figura 2-2 - Janela Deslizante

A figura 2-2 representa um modelo simplificado do conceito de janelas deslizantes utilizado no TCP para o remetente. Para efeito didático será considerado o uso de segmentos, mas na prática o TCP efetua o controle sobre bytes.

O retângulo representa a variável *sndwnd* (*sender window*) que é o numero máximo de segmentos que o TCP pode enviar sem receber confirmações (*acknowledgments*) do receptor.

No exemplo representado pela figura 2-2 os segmentos 1, 2 e 3 já foram enviados e confirmados e os segmentos 4, 5 e 6 já foram enviados mas não confirmados. A janela tem tamanho 6, portanto os segmentos 7, 8 e 9 ainda podem ser enviados assim que possível. Ao chegarem as confirmações para 4, 5 e 6, a janela é movida para a direita, daí o nome janela deslizante, passando a abranger os segmentos 10, 11 e 12, e então novos segmentos podem ser enviados.

A variável *sndwnd* é sempre o menor valor entre a janela anunciada pelo receptor (ver *window size* na seção 2.2) e a variável *cwnd* (*congestion window*) que será visto mais a frente. Como a janela deslizante tem tamanho variável, *sndwnd* também tem tamanho variável.

2.4 MSS – *MAXIMUM SEGMENT SIZE*

A opção de tamanho máximo de segmento presente no TCP permite que um receptor especifique o tamanho máximo de segmento que este receberá. A opção MSS é significativa pois permite que sistemas heterogêneos possam se comunicar mesmo tendo tamanhos diferentes de buffers para recepção.

A negociação do MSS também é importante para que todo o potencial de velocidade de transmissão da rede seja aproveitado pelo TCP, de modo que em redes rápidas o TCP negocia valores de MSS próximos ou do mesmo tamanho do MTU (*Maximum Transfer Unit*) da rede. Entretanto, em redes heterogêneas esta negociação torna-se mais difícil pois MSS muito pequenos ou muito grandes podem reduzir a performance do TCP em virtude de baixa utilização da rede ou fragmentação devido aos diferentes tamanhos de MTU encontrados ao longo do caminho.

2.5 *WINDOW SCALE*

Definida na RFC 1323, esta opção permite que o tamanho máximo de janela do TCP passe de 64KB para 1GB através da utilização de um fator de escala. Adicionalmente a definir o fator de escala a ser adotado pelo emissor, esta opção quando usada ainda denota que o emissor está preparado tanto para enviar quanto para receber janelas ampliadas.

Somente sendo possível o envio do *window scale* no início de cada conexão TCP (é necessário que o bit SYN esteja ligado), o fator de janela ampliada somente é negociado uma vez, em ambas as direções, durante uma conexão TCP.

O valor numérico definido na opção define a potência de 2 que será utilizada como fator de escala. Portanto, se o campo contiver o valor 4, significa que a janela quando recebida deverá ser multiplicada por 2^4. Contendo o valor 0 no campo, indica que o TCP do equipamento em questão está habilitado para trabalhar com *window scale*.

2.6 *TIMESTAMP*

Também definida na RFC 1323, esta opção foi desenvolvida com o intuito de ajudar o TCP a calcular os tempos de atraso na rede. Possuindo 4 campos (2 obrigatórios, de tipo e tamanho) a opção *timestamp* possui ainda mais dois campos chamado *timestamp value* e *timestamp echo reply*. O emissor coloca no campo *timestamp value* a hora de seu relógio atual ao enviar um segmento. O receptor copia o valor de *timestamp value* para *timestamp echo reply* ao enviar a confirmação para o segmento. Assim, quando a confirmação chega ao emissor é possível calcular o tempo gasto total de ida e volta da rede naquele instante.

2.7 SACK – *SELECTIVE ACKNOWLEDGMENT*

Definida na RFC 2018, a opção SACK – *Selective Acknowledgment* – também denomina um algoritmo de controle de congestionamento para o TCP. Na realidade a opção SACK divide-se em duas opções.

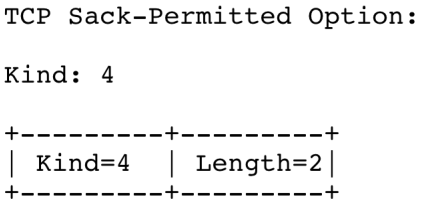


Figura 2-3 - Opção SACK-Permitted

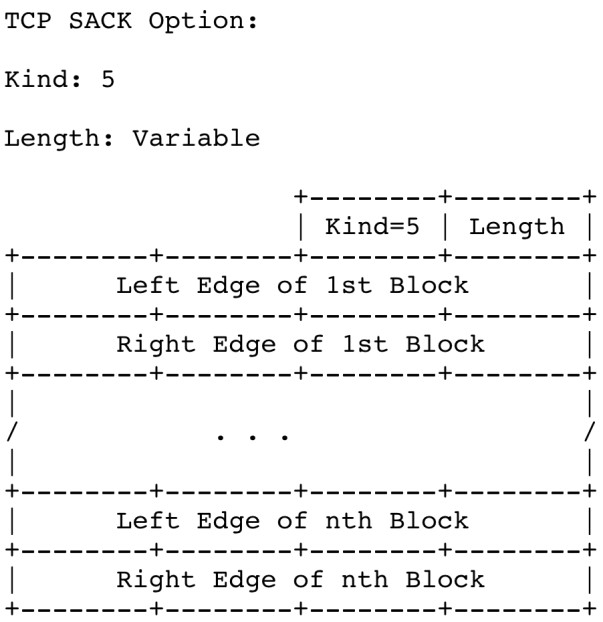


Figura 2-4 - Opção SACK

A figura 2-3 exibe a opção SACK de tipo 4, de nome *SACK-Permitted*, que somente é utilizada durante a negociação de conexão TCP (segmentos com bit SYN habilitado). A opção *SACK-Permitted* permite que um sistema informe ao outro que oferece suporte a opção SACK para a conexão. A figura 2-4 exibe a opção SACK de tipo 5, que é utilizada ao longo de uma conexão TCP para que o receptor de um fluxo de dados possa informar ao emissor que um ou mais blocos de dados fora de ordem foram recebidos com sucesso e enfileirados. O receptor então aguarda até que sejam recebidos os blocos faltantes para que os intervalos sejam preenchidos. Ao receber os blocos faltantes, o receptor os confirma normalmente e avança a janela de recebimento. Os campos *Left Edge* e *RIght Edge* informam os números de

sequência que formam os limites de cada bloco de *bytes* que foi recebido e ambos possuem tamanho de 4 *bytes*, mesmo tamanho do campo *Sequence Number* do cabeçalho TCP.

Para melhor entendimento, suponha que 5 segmentos são enviados, cada um contendo 1500 *bytes* de dados. Os números de sequência para os segmentos iniciam em 0 para o segmento 1, 1500 para o segmento 2, 3000 para o segmento 3, 4500 para o segmento 4 e 6000 para o segmento 5. Suponha ainda que o segmento 3 foi perdido durante o percurso.

Ao receber o segmento 1, o receptor envia ao emissor uma confirmação de chegada (ACK) com número de sequência 1500, informando que recebeu o segmento com sucesso e que aguarda o próximo segmento. É recebido o segmento 2, com número de sequência 1500, e é então é enviada uma nova confirmação (ACK) com número de sequência 3000. Como o segmentos 3 foi perdido, é então recebido o segmento 4 com número de sequência 4500. O receptor então armazena o segmento recebido e envia ao emissor uma confirmação (ACK) ainda com número de sequência 3000, mas com a opção SACK habilitada e contendo o valor 4500 no campo *Left Edge* e 6000 no campo *RIght Edge*. Ao receber o segmento 5, é enviado outra confirmação (ACK) como número 3000 e também com a opção SACK habilitada, mas desta vez com o valor de *Left Edge* em 4500 e *Right Edge* em 7500.

2.8 *SLOW START*

Proposto por Jacobson em [2], *slow start* é um algoritmo utilizado para controle de congestionamento de forma a ajustar a quantidade de dados enviados às condições da rede. Seu mecanismo é baseado na observação que a taxa que novos segmentos que devem ser injetados na rede deve ser a mesma em que as confirmações (ACK) são enviadas pelo receptor.

O *slow start* requer que seja mantida pelo TCP emissor uma janela, chamada de janela de congestionamento, que utiliza uma variável chamada *cwnd*. Quando uma nova conexão é estabelecida com outro equipamento na rede, a janela de congestionamento é inicializada com um segmento, ou seja, o tamanho do segmento anunciado pelo nó oposto. Toda vez que uma nova confirmação (ACK) é recebida, a janela de congestionamento é incrementada de um segmento. O valor de *cwnd* é mantido em bytes, mas o *slow start* sempre a incrementa em segmentos. O emissor pode transmitir até o mínimo entre a janela de congestionamento e a janela anunciada pelo receptor. A janela de congestionamento é o controle de fluxo imposto pelo emissor, enquanto a janela anunciada é controle de fluxo imposto pelo receptor.

O emissor começa transmitindo um segmento e esperando pelo ACK correspondente. Quando este ACK é recebido, a janela de congestionamento é incrementada de um para dois, e dois segmentos podem ser mandados. Quando cada um destes dois segmentos for confirmado, a janela de congestionamento é aumentada para três e logo após para quatro. Na prática, isto provê uma aumento exponencial.

2.9 *CONGESTION AVOIDANCE*

Estando a estimativa de tempo do TCP para retransmissão (RTO – *Retransmit Timeout*) bem ajustada, é possível afirmar que se o cronômetro de retransmissão expirar (timeout), um segmento foi perdido. Nas redes atuais, com baixa taxa de perda por erros (< 1%), o mais provável é que o pacote IP que transportava o segmento TCP tenha sido descartado em algum ponto devido ao congestionamento de rede enfrentado em algum enlace.

Congestion avoidance é um algoritmo elaborado para evitar uma condição conhecida como colapso de congestionamento, em que a performance dos enlaces

de rede não atende aos requisitos de transmissão de dados fazendo com que as filas de transmissão dos roteadores transbordem e pacotes IP passem a ser descartados.

Congestion avoidance e *slow start* são algoritmos diferentes com objetivos diferentes. Enquanto *congestion avoidance* diminuiu a janela de transmissão para evitar um colapso de congestionamento, *slow start* é invocado para recomeçar o processo de aumento da janela e taxa de transmissão.

Ambos os algoritmos requerem que duas variáveis sejam monitoradas para cada conexão: *cwnd*, que representa a janela de congestionamento, e *sstresh*, que é o limite de janela para o algoritmo de *slow start*. O algoritmo combinado funciona da seguinte maneira:

1. Durante a inicialização de uma conexão, a *cwnd* é igual a um segmento e a *ssthresh*, a 65536 *bytes*;
2. A rotina de emissão do TCP sempre envia o mínimo entre a *cwnd* e a janela anunciada pelo receptor;
3. Quando ocorre congestionamento (indicada por *timeout* ou o recebimento de ACKs duplicados), metade do valor atual da janela de transmissão (o mínimo entre a *cwnd* e a janela anunciada pelo receptor) é armazenado em *ssthresh*. Além disso, se o congestionamento foi causada por *timeout*, a *cwnd* passa a valer um segmento (ou seja, *slow start*);
4. Quando novos dados forem confirmados pelo nó destino, a *cwnd* é aumentada, mas a maneira como isto é feito depende se está sendo feito o *slow start* ou o *congestion avoidance*.

Se a *cwnd* for menor ou igual a *ssthresh*, o TCP está em *slow start*; caso contrário, ele está realizando o *congestion avoidance*. O *slow start* prossegue até que a janela de transmissão do TCP esteja com metade do tamanho de quando

ocorreu o congestionamento (guarda-se metade do valor da janela que causou problema no passo 3), e, então, passa-se para a fase de *congestion avoidance*.

O *slow start* faz a *cwnd* começar valendo um segmento e ser incrementada de um segmento toda vez que um ACK é recebido. Como mencionado anteriormente, isto abre a janela exponencialmente: um segmento é enviado, então dois, quatro, e assim por diante. O *congestion avoidance*, por sua vez, faz com que a *cwnd* seja incrementada por *segsize*segsize*/*cwnd* toda vez que um ACK for recebido, onde *segsize* é o tamanho do segmento (*segsize* e *cwnd* são mantidos em bytes).

Isto faz com que a *cwnd* tenha um aumento linear, comparado com o aumento exponencial do *slow start*. O aumento da *cwnd* deve ser de, no máximo, um segmento a cada RTT (*Round Trip Time*), independente de quantos ACKs sejam recebidos neste RTT, enquanto que o *slow start* incrementa a *cwnd* baseado no número de ACKs recebidos em um RTT.

2.10 *FAST RETRANSMIT* E *FAST RECOVERY*

Criado por Jacobson em [3] como modificação de sua proposta inicial [2], *fast retransmit* é um algoritmo que assume que ocorreu perda de algum segmento após o recebimento de 3 ACKs duplicados pelo remetente, ou seja, 3 ACKs confirmando o recebimento de um mesmo segmento, ocasionando a retransmissão do segmento perdido sem esperar que o temporizador de timeout expire e o reinício do temporizador de monitoramento de *timeout* para este segmento. Esta técnica aumenta o *throughput* da rede nos casos de perda ocasional de um segmento.

Fast recovery é um algoritmo com o objetivo de não reduzir de maneira drástica o tamanho da janela quando ocorre uma retransmissão, sendo uma variação do algoritmo *slow start*. No algoritmo *fast recovery*, após a execução de *fast retransmit* para reenvio de um segmento, a *cwnd* é reduzida a metade e é aumentada de um

MSS (*Maximum Segment Size*) a cada ACK duplicado que chegou anteriormente ou que chegou depois que a retransmissão ocorre. Como resultado, enquanto ocorre o *fast retransmit*, o TCP mantém o envio de segmentos enquanto não recebe a confirmação do segmento retransmitido. Após a chegada do ACK referente ao segmento retransmitido, o algoritmo de *congestion avoidance* é executado.

É importante ressaltar que na ausência do algoritmo *fast recovery* na implementação do TCP após a execução do *fast retransmit*, é executado o mesmo processo de tratamento de congestionamento feito no *congestion avoidance* quando ocorre um *timeout*, resultando no *slow start*.

3 IMPLEMENTAÇÕES TCP

Ao longo do tempo o TCP sofreu modificações para melhorar sua performance, tratamento de anomalias de funcionamento e adequação a novas tecnologias, como a melhora significativa da qualidade e velocidade dos enlaces de dados.

Abaixo segue um breve descritivo das implementações mais usadas.

3.1 TAHOE

A primeira proposta de implementação do TCP baseada no algoritmo de *congestion avoidance* proposto por Jacobson em [2] é conhecida como Tahoe.

Em sua implementação inicial estão presentes os algoritmos *slow start* e *congestion avoidance*. Posteriormente foi incluído o algoritmo *fast retransmit* para melhora visando a melhora em sua performance.

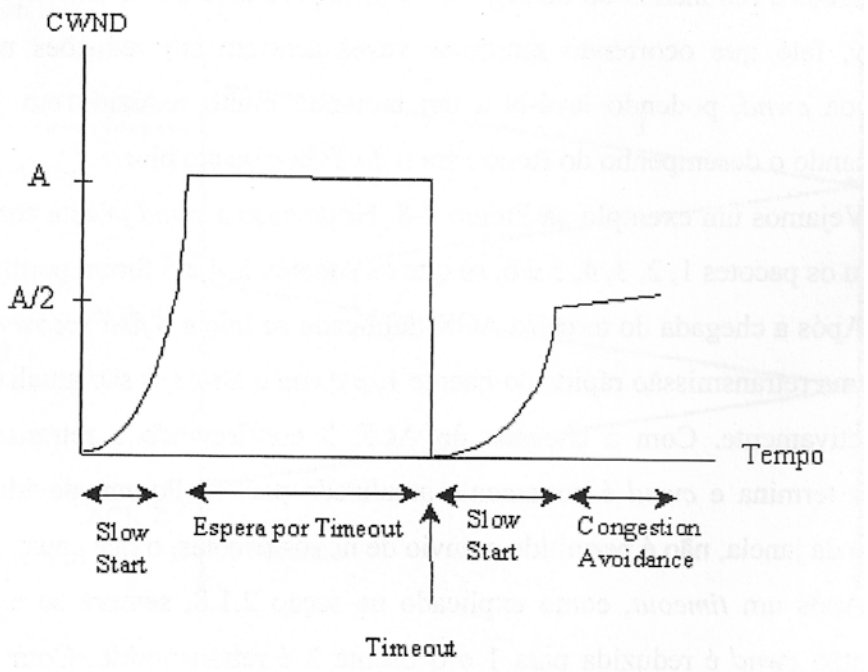


Figura 3-1 - Fast Retransmit

Esta implementação utiliza granularidade grossa para medir o RTT (*Round Trip Time*) e determinar o RTO (*Retransmit Timeout*), o que acaba impondo uma valor mínimo grande para o RTO, sendo este da ordem de segundos. Granularidade

grossa é um termo usado para indicar que a unidade de medida é considerada grande para o propósito. Por exemplo, uma medição efetuada com a unidade segundo quando a utilização da medida milisegundo, além de ser possível, possibilitaria maior precisão.

A figura 3-1 apresenta um gráfico de variação de *cwnd*, representado por A, conforme perdas ocorrem. O gráfico representa o comportamento da implementação TCP conhecida por TahoeNoFR (*No Fast Retransmit*), em que perdas só serão identificadas após a ocorrência de um *timeout*. Havendo a ocorrência de *timeout*, o TCP entra em *slow start*, fazendo com que *sndwnd* seja igual a 1 MSS, e vai aumentando até que a janela seja igual a *ssthresh*, que é metade do valor de *cwnd* antes da perda ocorrer, e então começa o algoritmo *congestion avoidance*, provocando um crescimento linear da janela.

3.2 RENO

Reno é a alteração proposta por Jacobson em [3] que implementa os algoritmos de *fast retransmit* e *fast recovery*. Aspectos básicos da implementação Tahoe são mantidos, como o uso do *slow start* e o uso de granularidade grossa para o temporizador de timeout.

O uso do algoritmo *fast retransmit* permite que um segmento seja retransmitido mesmo antes do RTO timer expirar, o que permite maior *throughput* da rede. Adicionalmente, com o uso do algoritmo *fast recovery*, a recuperação da condição de congestionamento passa a ser mais rápida, pois após a recepção de 3 ACKs duplicados, indicando que segmentos fora de ordem foram recebidos, o segmento faltante imediatamente posterior ao ACK recebido duplicadamente é retransmitido e *cwnd* é reduzido a metade e não a 1 MSS, como no *slow start*.

O principal problema desta implementação é que sua performance somente é boa para casos em que as perdas de segmentos são pequenas. Em casos em que há grande perda de segmentos a performance é similar a implementação Tahoe. A razão para isto é que somente é possível detectar a perda de um único segmento. Em casos com perdas consecutivas de segmentos, só será percebida perda de um segundo segmento após a recepção do ACK referente ao primeiro segmento. Com isso, o TCP sai da fase de *fast recovery*, efetua a retransmissão do segundo segmento e inicie uma nova fase de *fast recovery*. Ocorrendo esta situação por múltiplas vezes, o *cwnd* também é reduzido múltiplas vezes a metade, o que em um curto espaço de tempo leva o *cwnd* a um tamanho muito reduzido.

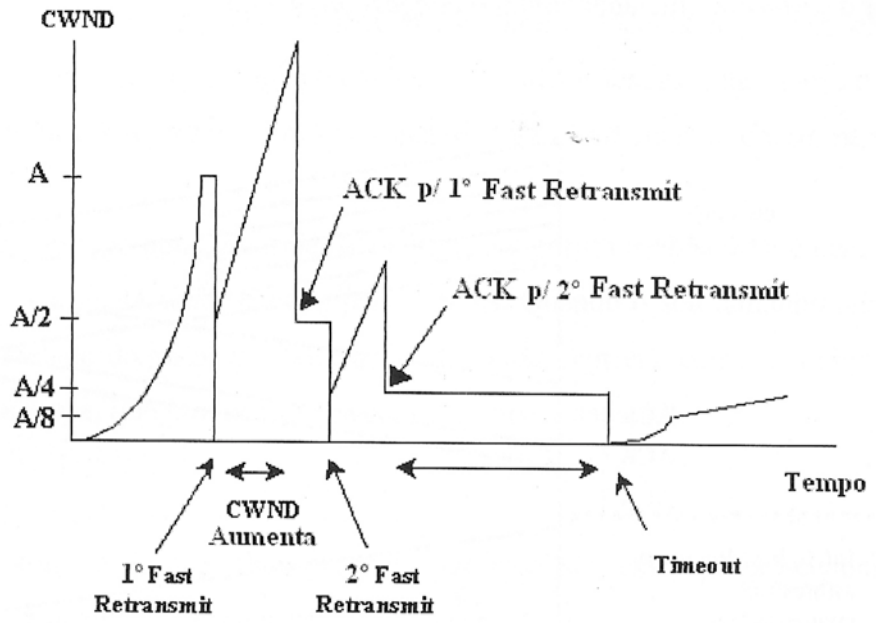


Figura 3-2 - TCP Reno

A figura 3-2 representa um caso de variação de *cwnd*. Após a primeira retransmissão, a janela é imediatamente reduzida a metade e então passa a aumentar linearmente para cada ACK duplicado que chega. Então o ACK da retransmissão é recebido e *cwnd* volta para o valor de quando o *fast recovery* havia

começado. É percebido então que um segundo segmento foi perdido e *cwnd* é reduzido a metade novamente. É então recebido o ACK para a segunda retransmissão. Ocorre então a perda de mais um segmento na rede, mas desta vez não são recebidos ACKs duplicados, impossibilitando ao TCP inferir que um terceiro segmento foi perdido. Como *cwnd* neste momento está muito reduzido após dois *fast recovery*, não é permitido o envio de novos segmentos. Neste caso, ocorre um timeout e então é efetuada a retransmissão do terceiro segmento e o TCP entra em *slow start* e *congestion avoidance*.

3.3 NEWRENO

NewReno é uma implementação TCP baseada na implementação Reno, mas com pequenas mudanças. A implementação NewReno é capaz de detectar múltiplas perdas de segmentos sendo muito mais eficiente que o TCP Reno nessas condições.

Assim como o TCP Reno, o TCP NewReno entra na fase *fast retransmit* quando recebe ACKs duplicados passando a fase *fast recovery*, mas este último não sai da fase *fast recovery* até que todos os segmentos enviados tenham sido confirmados.

A fase *fast retransmit* é a mesma no TCP Reno e no TCP NewReno. A diferença está na fase *fast recovery*, que permite múltiplas retransmissões no TCP NewReno. No TCP NewReno, no momento que é iniciada a fase *fast recovery* é verificado qual o último segmento que está aguardando confirmação de recebimento. Então, o processamento se dá como no TCP Reno até que um novo ACK é recebido. Neste momento duas ações são possíveis:

- Se o ACK recebido confirma todos os segmentos enviados após o início do *fast recovery*, então a fase *fast recovery* é finalizada e atribui *ssthresh* a *cwnd* e entra em *congestion avoidance*, como no TCP Tahoe.

- Se o ACK recebido confirmar parte dos segmentos enviados após o início do *fast recovery*, o chamado ACK parcial, então o TCP NewReno infere que o próximo segmento não confirmado foi perdido, o retransmite e atribui zero ao valor de ACKs duplicados recebidos. Enquanto todos os segmentos não forem confirmados a fase *fast recovery* não é finalizada.[4]

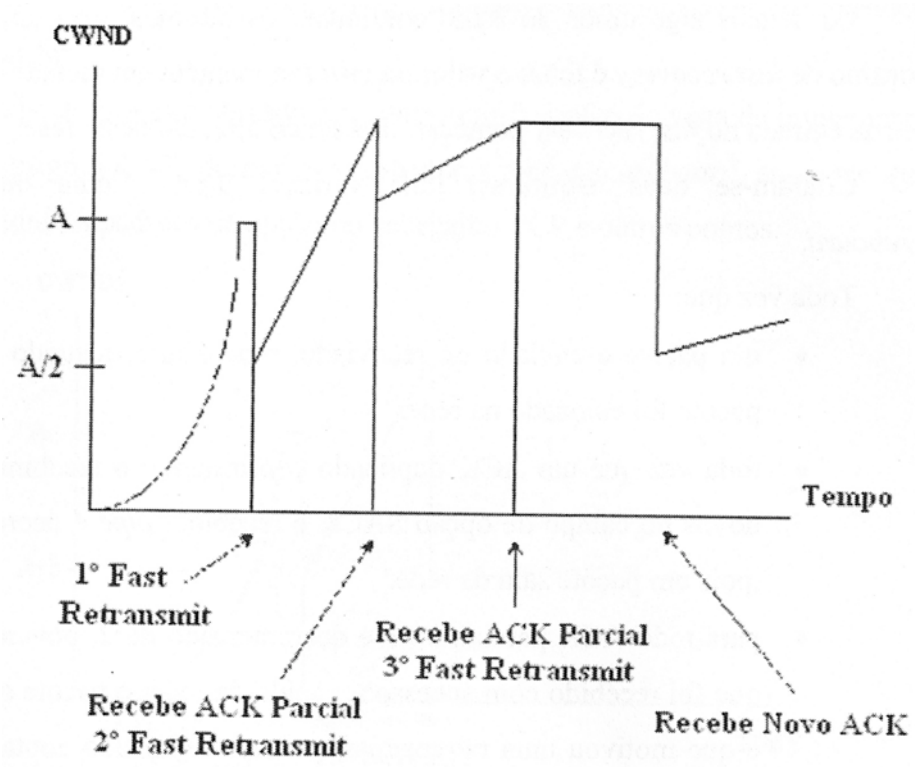


Figura 3-3 - TCP NewReno

A figura 3-3 mostra que a simples alteração de não sair do *fast recovery* quando um ACK parcial é recebido faz com que a janela não seja reduzida múltiplas vezes e que a perda de mais de um segmento seja percebida facilmente. É verificado também que a janela reduz de tamanho após o recebimento do primeiro ACK parcial e que ao receber o segundo ACK parcial, esta se mantém estável. Ao receber um novo ACK confirmando todos os segmentos enviados após o início do *fast recovery* é iniciado o *congestion avoidance*.

O TCP NewReno, contudo, sofre de um mesmo problema que o TCP Reno: somente é percebida a perda de qualquer segmento após um RTT. Somente é possível detectar o próximo segmento perdido após a chegada da confirmação da última retransmissão.

3.4 SACK – SELECTIVE ACK

Selective ACK, ou SACK, é uma alteração no TCP proposta na RFC 2018 com a finalidade de sanar o problema de somente perceber a perda de um segmento a cada RTT.

A implementação SACK mantém os algoritmos de *slow start* e *fast retransmit* presentes no Reno, mas o algoritmo de *fast recovery* possui uma pequena alteração: o valor de *cwnd* se mantém em metade do valor usado antes da entrada no *fast recovery* começar, não sendo alterado nesta fase. Também mantém a granularidade grossa para timeouts da implementação Tahoe para o caso de um segmento perdido não ser detectado pelo algoritmo modificado.

A diferença chave desta implementação frente às outras é que na implementação SACK são acrescentados ACKs que permitem que segmentos não contínuos de dados sejam confirmados. Toda vez que um receptor receber dados fora de ordem, este deve retornar um segmento contendo um ACK duplicado confirmando o último segmento enviado com sucesso em ordem, mas com o campo de opção SACK (ver seção 2.7) contendo a informação de qual segmento foi recebido com sucesso fora de ordem.

Para que esta confirmação seletiva seja possível, são implementadas uma variável chamada *pipe* e uma estrutura chamada *scoreboard*. A função da variável *pipe* é ser um contador de quantos segmentos foram enviados e ainda não confirmados, e sua alteração se dá quando umas das seguintes situações ocorre:

- Sempre que um segmento é enviado ou reenviado, *pipe* é incrementado de 1 pois um novo segmento foi colocado na rede.
- Todas as vezes que um ACK duplicado é recebido confirmando o recebimento de novos dados no campo SACK, *pipe* é decrementado de 1, pois o segmento foi retirado da rede.
- Para todo ACK parcial, *pipe* é decrementado de 2, pois além do segmento que foi recebido com sucesso ter saído da rede, o segmento que foi perdido e que motivou a retransmissão não havia sido contabilizado como enviado com sucesso.

Sempre que o valor de *pipe* é menor que o valor de *cwnd*, é verificado em *scoreboard* quais segmentos já foram confirmados pela opção SACK. Havendo segmentos a serem retransmitidos, estes são enviados, caso contrário novos dados são enviados.

A utilização do SACK recai sobre a necessidade de tanto o receptor como o emissor suportarem a implementação, caso contrário não é possível seu uso, diferentemente das outras implementações TCP vistas.

De acordo com a RFC 2018, somente o anúncio que o uso do SACK é permitido na conexão, isso não garante efetivamente seu uso pelos sistemas envolvidos.

A RFC 3517 padroniza o algoritmo SACK do ponto de vista de implementação, portanto é nela que são mencionas as variáveis *pipe* e *scoreboard*, enquanto a RFC 2018 padroniza a utilização da opção SACK no cabeçalho TCP.

A figura 3-4 representa como o uso do SACK permite ao emissor perceber rapidamente perdas de segmentos.

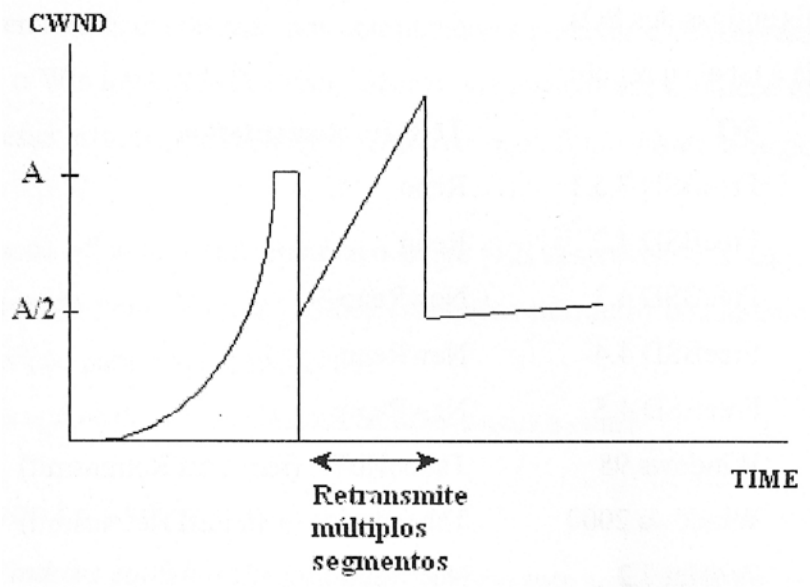


Figura 3-4 - TCP SACK

3.5 VEGAS

Inicialmente desenvolvido por Brakmo em 1994 [5], o TCP Vegas propõe um novo algoritmo de controle de congestionamento. Este novo algoritmo é um pouco diferente dos implementados no TCP Tahoe e Reno. As implementações Tahoe e Reno possuem algoritmos que detectam o congestionamento na rede através da ocorrência de perdas de segmentos e não possuem nenhum outro mecanismo para detectar a o congestionamento antes que a perda aconteça. Desta forma, ambas as implementações reagem ao congestionamento ao invés de tentar preveni-lo.

Na implementação Vegas o algoritmo de controle de congestionamento utiliza a diferença entre duas medidas, o *throughput* esperado e o *throughput* apurado da rede, como um modo de determinar se existe congestionamento. Com isso, não é mais necessário aguardar que haja perda de segmentos para que o congestionamento seja detectado. Contudo, o TCP Vegas mantém em sua implementação as técnicas de detecção de congestionamento por *timeout*, como no TCP Tahoe e no TCP Reno, e ainda propõe um algoritmo *slow start* modificado que

reduz a probabilidade de novo congestionamento e também um novo algoritmo para retransmissão de segmentos.

O algoritmo de controle de congestionamento funciona, basicamente, da seguinte forma:

1. Calcula-se o *throughput* real da rede da seguinte forma: para cada segmento enviado, é armazenada a hora de envio verificada no relógio do sistema e aguarda-se até que seja recebido o ACK correspondente. Ao chegar o ACK, calcula-se o RTT efetuando a diferença entre a hora de chegada e a hora de envio. Então é calculado o *throughput* da rede dividindo-se o tamanho da janela atual por RTT.
2. É calculado o *throughput* estimado dividindo-se o tamanho da janela atual pela variável *BaseRTT*, uma variável previamente definida que armazena o menor RTT apurado até o momento.
3. É então calculada a diferença entre o estimado e o real, por padrão não negativa, que será usada para ajustar o tamanho da janela.
4. O valor resultante é então comparado com os *thresholds* α e β, previamente definidos, de modo que $0 \leq \alpha < \beta$. Se a diferença for menor que α, a janela será incrementada linearmente durante o próximo RTT. Se a diferença for maior que α e menor que β, então a janela será decrementada linearmente durante o próximo RTT. Estando a diferença entre α e β, nada é feito.

O algoritmo, na realidade, tenta inferir as condições da rede em função da diferença entre o *throughput* esperado e o *throughput* real calculados. Se o *throughput* real for muito menor do que o esperado, então existe grande probabilidade da rede estar congestionada e, neste caso, a janela deve ser diminuída para minimizar o congestionamento. Por outro lado, se o *throughput* real apurado for

muito próximo do esperado, então há grande probabilidade de não estar sendo aproveitado todo o potencial da rede, logo a janela deve ser aumentada.

O algoritmo para retransmissão de segmentos é um aprimoramento do algoritmo utilizado na implementação Reno. Como dito anteriormente, para cada segmento enviado, é armazenada a hora de envio verificada no relógio do sistema para que quando é recebido o ACK correspondente ao segmento, seja possível calcular o RTT correspondente aquele segmento. Se um ACK duplicado é recebido verifica-se se o RTT é maior que o RTT esperado. Em caso positivo, o segmento é retransmitido sem que seja necessário esperar por três ACKs duplicados. Quando um ACK não duplicado é recebido, se for o primeiro ou segundo após a ocorrência de retransmissão de algum segmento, novamente verifica-se se o intervalo de tempo desde que o segmento foi enviado é maior do que o RTT esperado. Caso positivo, o segmento é retransmitido (ver figura 3-5). Desta forma é possível detectar a perda de múltiplos segmentos sem que seja necessário aguardar a recepção de três ACKs duplicados para cada segmento perdido, como no TCP Reno. Adicionalmente, caso sejam perdidos múltiplos segmentos e ocorra mais de uma retransmissão, a janela de congestionamento (*cwnd*) somente será reduzida para a primeira retransmissão.

O TCP Vegas também introduz modificações no *slow start*, para evitar gerar congestionamento na rede aumentando demais a janela de congestionamento. A janela de congestionamento só aumenta exponencialmente a cada RTT. Entre RTTs, o TCP Vegas calcula a taxa real e compara com a esperada. Quando a diferença for maior que um certo limite inicia-se a fase de prevenção de congestionamento [6].

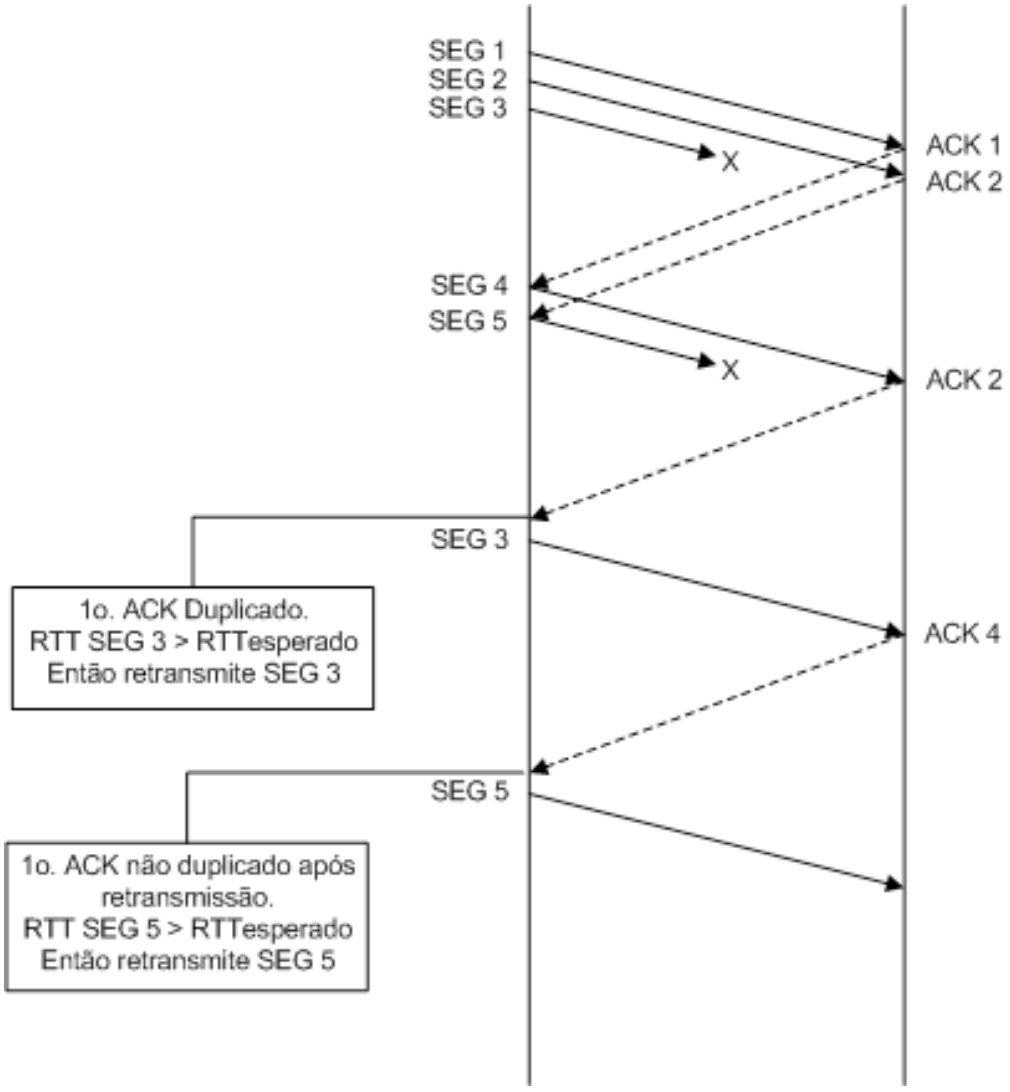


Figura 3-5 - Exemplo de retransmissão no TCP Vegas

3.6 TCP WESTWOOD

Proposto em [7] a implementação Westwood define um algoritmo para que o remetente numa conexão TCP possa estimar a largura de banda disponível e usá-la para se recuperar mais rapidamente de um congestionamento, atingindo assim um maior rendimento. É baseado em dois conceitos: a estimativa fim-a-fim da largura de banda disponível e na forma como esta estimativa é usada para definir o *slow start threshold* e a janela de congestionamento. Além disso, é importante observar que esta estimativa é apenas fim-a-fim e não depende de nenhum dos nós intermediários ao nível da rede. O TCP Westwood (TCPW) mede continuamente as

estimativas de largura de banda disponíveis utilizando o RTT médio dos ACKs recebidos. Esta estimativa é usada para calcular a janela de congestionamento e o *slow-start threshold* a ser utilizado após ser detectado um congestionamento – ou seja, após três ACKs duplicados ou um timeout. Ao contrário do TCP Reno, que somente reduz à metade a janela de congestionamento após três ACKs duplicados, o TCPW tenta tomar uma decisão mais inteligente. Ele seleciona um *slow start threshold* e uma janela de congestionamento que são consistentes com a largura de banda efetiva no momento do congestionamento. A característica principal sobre o TCPW é que ele verifica a largura de banda atual durante uma transferência de dados e não a largura de banda disponível antes da conexão ser iniciada.

Um importante elemento para o cálculo da janela de congestionamento após a detecção de congestionamento é o RTT. O RTT utilizado é o RTT mínimo medido durante a conexão. No TCPW a janela de congestionamento aumenta durante o *slow start* e se mantém durante a fase de *congestion avoidance*, como no TCP Reno.

3.7 SCALABLE TCP

Scalable TCP é uma modificação ao controle de congestionamento adotado no TCP padrão definida em [8] com a finalidade de aumentar a performance do TCP em redes de alta velocidade.

A técnica consiste em alterar a janela de congestionamento. Para cada ACK recebido normalmente, ou seja, em períodos que não é detectado congestionamento, a janela de congestionamento é somada de 0,01. Após a primeira detecção de congestionamento a janela de congestionamento é diminuída de 12,5% de seu próprio valor.

3.8 HIGH SPEED TCP

O High Speed TCP, ou HSTCP, foi introduzido por Sally Floyd [9] como uma modificação no mecanismo do controle de congestionamento do TCP para ser usado em conexões TCP com grandes janelas de congestionamento. Ele supera a dificuldade do TCP Tahoe, conhecido também como TCP Padrão, em atingir grandes janelas de congestionamento em ambientes com taxas de perda de pacotes muito baixas. O HSTCP propõe uma pequena modificação nos parâmetros de incremento e decremento do TCP padrão.

O HSTCP não modifica o comportamento do TCP em ambiente com taxa de perda de segmentos entre 1% e 5% e, portanto, não traz novas ameaças de colapso de congestionamento. Ele foi projetado para ter uma resposta diferente em ambientes com taxa de perda de pacote muito baixa e ter a mesma resposta do TCP padrão em ambientes com taxa de perda de pacotes de no mínimo 10^{-3}. Em ambientes com taxa de perda muito baixa, o HSTCP apresenta uma função de resposta muito mais agressiva que o TCP padrão, que utiliza o *slow start* para qualquer cenário. A função de resposta do HSTCP utiliza três parâmetros: *low_window, high_window* e *high_p*;

- *low_window* é utilizado para estabelecer um ponto de transição que define quando utilizar o *slow start* ou quando utilizar o algoritmo de alta velocidade. Quando a janela de congestionamento for igual ou menor a *low_window*, é utilizado o algoritmo *slow start*. Quando a janela de congestionamento ultrapassa *low_window*, é utilizado o algoritmo de alta velocidade HSTCP.
- *high_window* e *high_p* são utilizados para estabelecer o limite superior de janela do HSTCP.

Pesquisas relacionadas ao HSTCP incluem os trabalhos do brasileiro Evandro de Souza e de Deb Agarwal [10], e da inventora do HSTCP, Sally Floyd [11]. Em [10], o HSTCP é testado quanto a problemas de implementação. De acordo com Evandro de Souza, o HSTCP é adequado para um grande volume de transferência de dados por ser capaz de manter alta performance em diferentes condições e por ser fácil de implementar, quando comparado com outras soluções em uso. Em [11], um estudo conduzido por Sally Floyd, com colaboração de Evandro de Souza e Deb Agarwal, que faz parte da proposta do HSTCP, envolve o mecanismo limitado do *slow start* para o TCP com grandes janelas.

4 METODOLOGIA

Este capítulo explica os cenários de simulação utilizados na avaliação do desempenho destas oito implementações do TCP, a configuração das simulações e os parâmetros utilizados para avaliar o desempenho das implementações TCP ao longo de um único link. O tamanho do segmento em todas as simulações é tido como 1000 bytes. Além disso, o aplicativo, homônimo ao protocolo, utilizado para enviar dados é o FTP em todos os cenários. Todas as simulações usam o simulador NS-2 [12] versão 2.33-0ubuntu1.

4.1 AMBIENTE COMPUTACIONAL

Para as simulações necessárias foi utilizado um microcomputador portátil Apple Macbook White com sistema operacional Mac OS X 10.6 Snow Leopard e software de virtualização Parallels Desktop for Mac versão 5 build 5.0.9344.

O software de simulação de redes NS-2 versão 2.33-0ubuntu1 foi executado dentro de uma máquina virtual com sistema operacional Ubuntu Linux 9.10 64-bits, disponibilizada através do software Parallels Desktop. A versão utilizada é disponibilizada através dos repositórios oficiais de software mantidos pela Canonical, fabricante do sistema operacional Ubuntu Linux 9.10 64-bits.

O aplicativo utilizado para transferência dos dados utilizados para aferição foi o aplicativo *ftp*, homônimo ao protocolo, na versão 0.17-19. O aplicativo é nativo da instalação do sistema operacional Ubuntu Linux 9.10 64-bits.

4.2 TOPOLOGIA DE REDE UTILIZADA

A topologia de rede e os parâmetros utilizados na simulação para medir a performance das implementações TCP listadas no capítulo anterior é composta de um nó emissor, um nó receptor e dois nós roteadores interligados através de um enlace com alto produto banda x atraso (HBDP – *High Bandwith Delay Product*),

como mostrado na figura 4-1. Os valores máximos para as janelas de congestionamento das implementações TCP foram configurados para que as conexões pudessem utilizar toda a banda disponível. O atraso provocado pelo enlace para um RTT foi fixado com valor de 25ms.

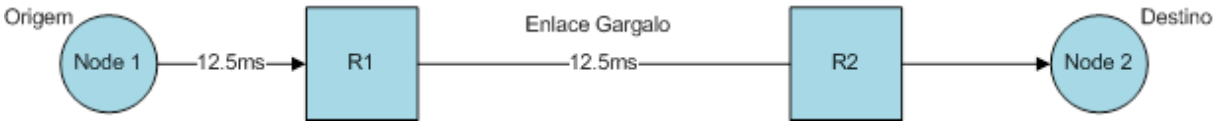


Figura 4-1 - Topologia de rede de simulação

A velocidade dos enlaces varia de 1.5Mbps a 1000Mbps, passando pelos valores de 10, 100, 250, 550 e 800 Mbps. Desta forma, o produto banda x atraso aumenta e com isso é possível mostrar a performance de cada implementação conforme aumenta-se a velocidade do enlace. O tamanho da fila para o enlace em R1 é fixado em 200 pacotes para que seja possível assimilar parte do congestionamento repentino.

Os experimentos conduzidos utilizando esta topologia serão feitos para verificar a utilização de banda, verificar a perda de pacotes durante a fase de *slow start* e o tempo de recuperação após a ocorrência de um congestionamento.

5 TESTES REALIZADOS

A figura 5-1 mostra a utilização de banda atingida por cada implementação TCP avaliada. Como a própria figura mostra, existem diferenças consideráveis entre as implementações. Há também uma tendência de diminuição de performance por algumas implementações quando a largura de banda é aumentada revelando claros problemas com escalabilidade. Este é um comportamento esperado e confirma o que pesquisadores já aferiram [9]. Somente as implementações Vegas, STCP e HSTCP apresentam melhor performance e escalabilidade. O pior desempenho foi registrado para o TCP Tahoe, seguido por Reno, NewReno, SACK, Westwood, STCP, HSTCP e Vegas. Esta sequência reflete o comportamento destas implementações de acordo com a reação à perda de segmentos, múltiplas ou não, dentro da mesma janela.

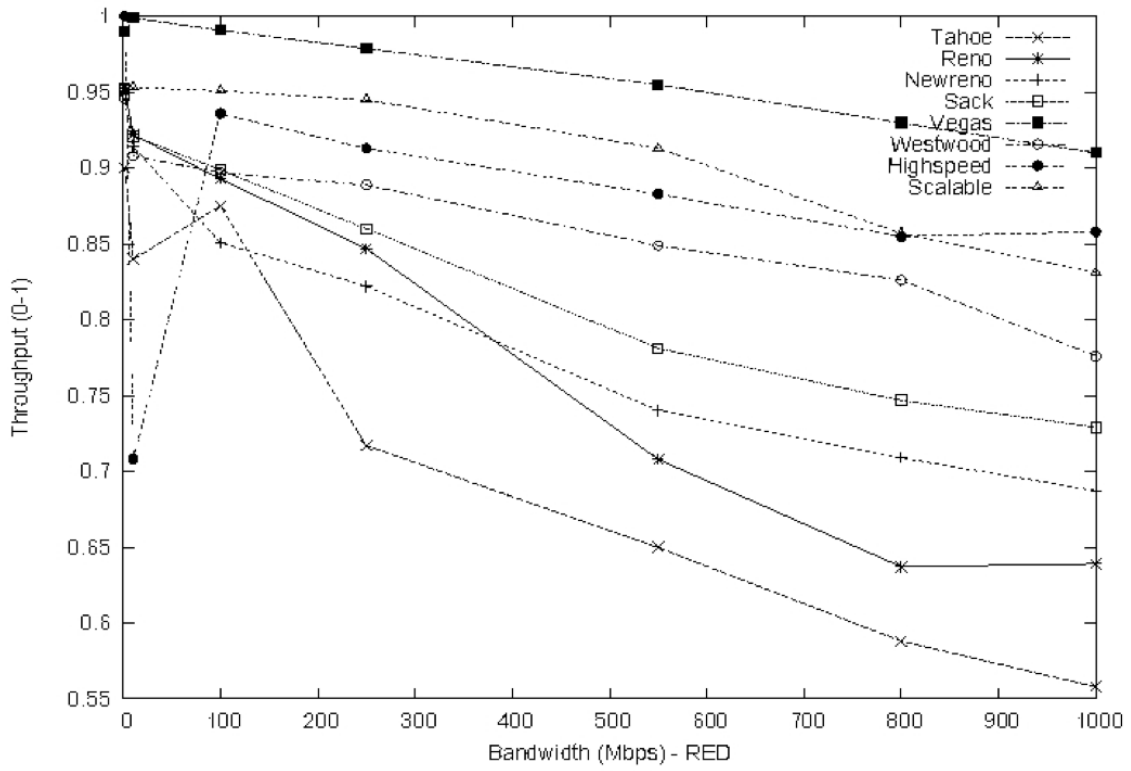


Figura 5-1 - Utilização de banda

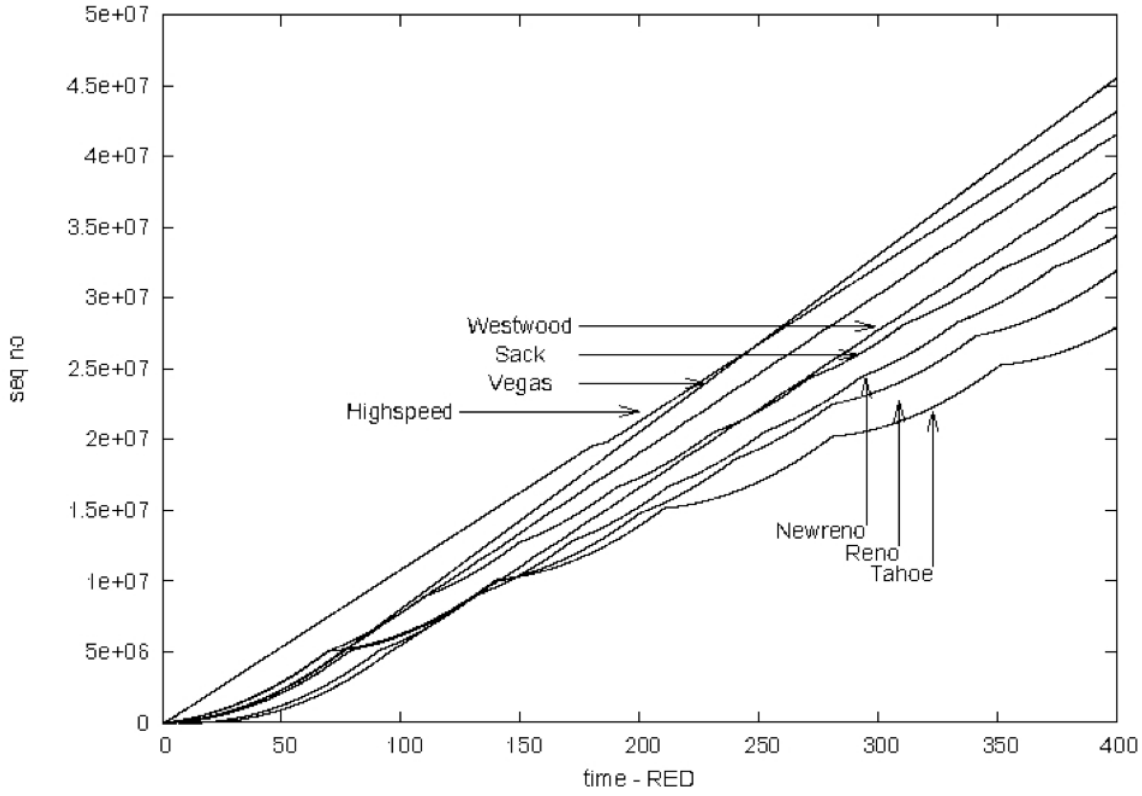


Figura 5-2 - Evolução dos números de sequência

A figura 5-2 revela a evolução dos números de sequência utilizados por cada implementação utilizando-se um enlace de 1000Mbps. A performance verificada pode ser explicada pelo comportamento da janela de congestionamento.

As figuras 5-3, 5-4, 5-5 e 5-6 mostram as janelas de congestionamento das implementações ao longo do tempo com o enlace que liga os roteadores R1 e R2 simulando a taxa de 1000Mbps. É possível verificar que a implementação Tahoe é a única que reduz a janela de congestionamento ao valor 1, indicando a execução do algoritmo de *slow start*, enquanto as outras implementações reduzem a janela de congestionamento à metade, ou até menos do que a metade, do valor corrente. A implementação Reno mostra reações mais longas, enquanto NewReno e SACK apresentam comportamento parecido. As implementações Vegas, Westwood e HSTCP apresentam comportamento diferenciado. A implementação Westwood consegue atingir uma melhor utilização pois sua janela de congestionamento não cai

a níveis muito baixos como as implementações mais tradicionais Tahoe, Reno e NewReno, devido a estimar o largura de banda disponível. Será visto mais adiante que a implementação Westwood terá uma longa fase de *congestion avoidance*. O mesmo comportamento de janela de congestionamento da implementação Westwood pode ser verificado para a implementação STCP, mas o STCP consegue atingir melhor performance pois não é necessário estimar a largura de banda em sua fase inicial como na implementação Westwood. A janela de congestionamento da implementação HSTCP também atinge valores parecidos ao da implementação Westwood, entretanto consegue atingir melhor performance ao transmitir mais segmentos devido a aumentar a janela de congestionamento mais rapidamente, principalmente no início da conexão. Finalmente, a implementação Vegas apresenta o melhor comportamento ao manter a janela de congestionamento após a fase de *slow start*.

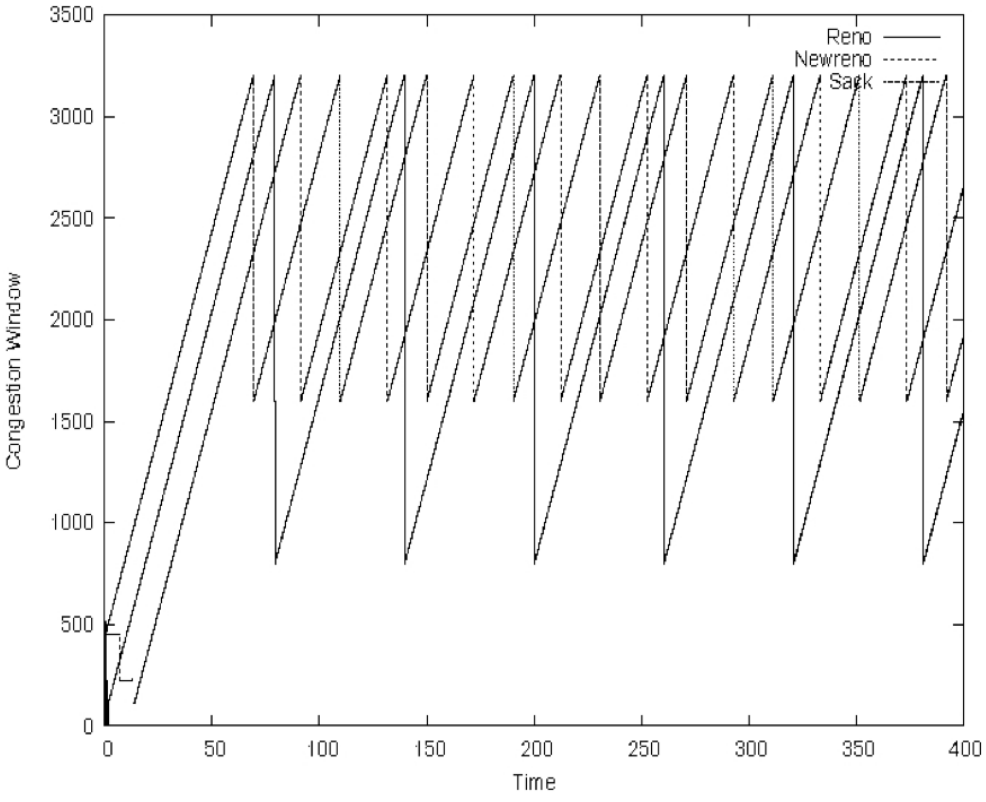


Figura 5-3 - Janela de congestionamento (Reno, NewReno e SACK)

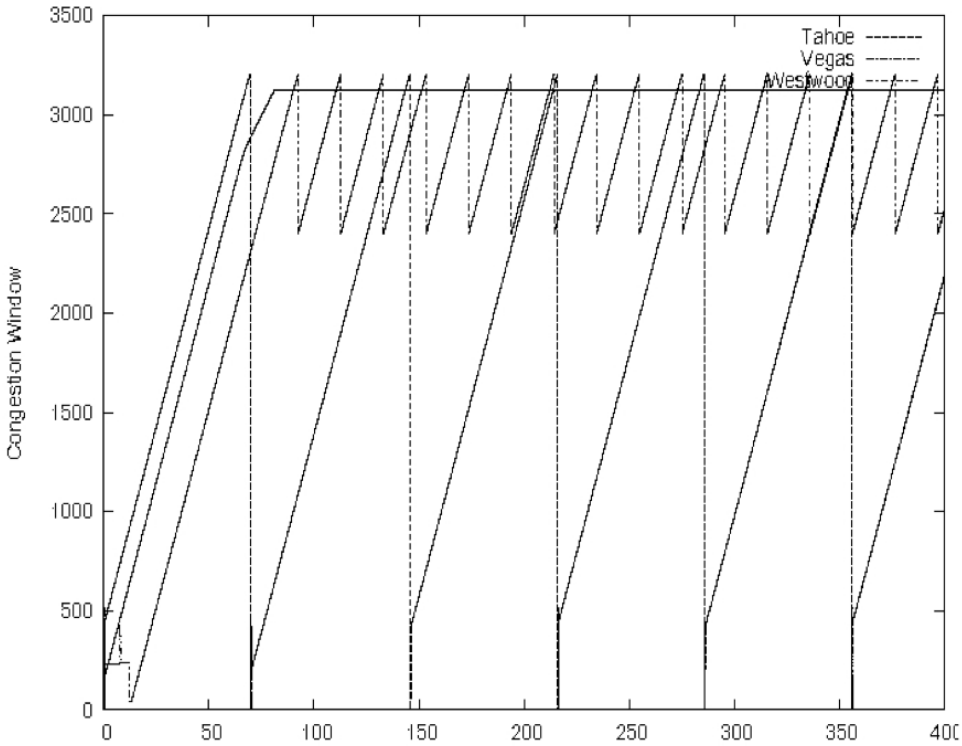


Figura 5-4 - Janela de Congestionamento (Tahoe, Vegas e Westwood)

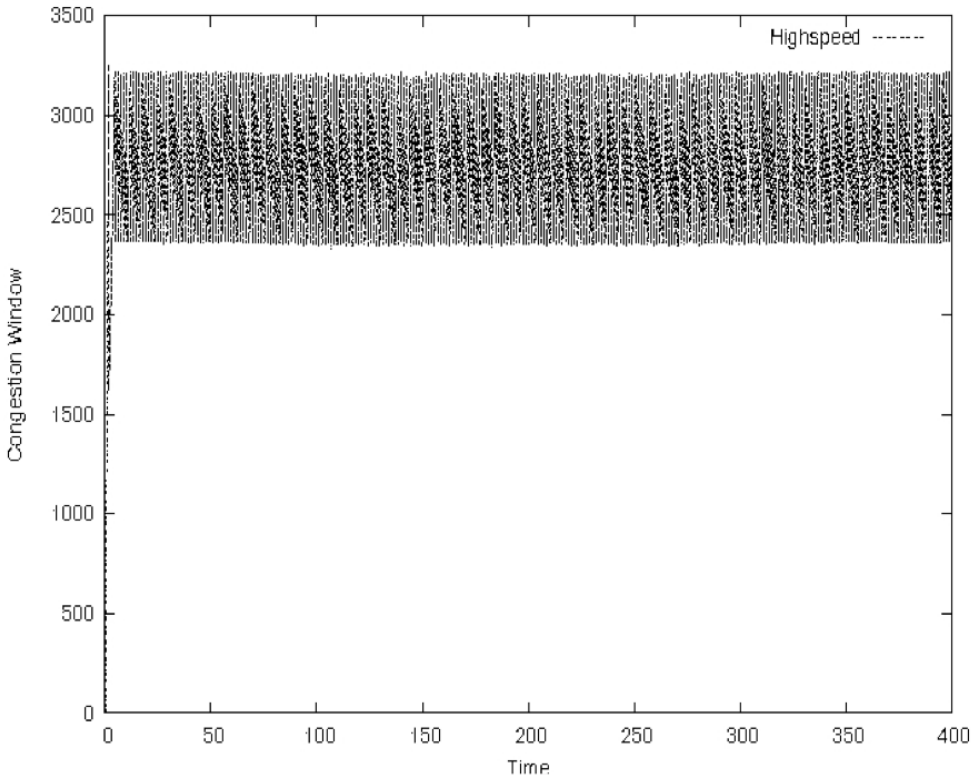


Figura 5-5 - Janela de Congestionamento Highspeed TCP

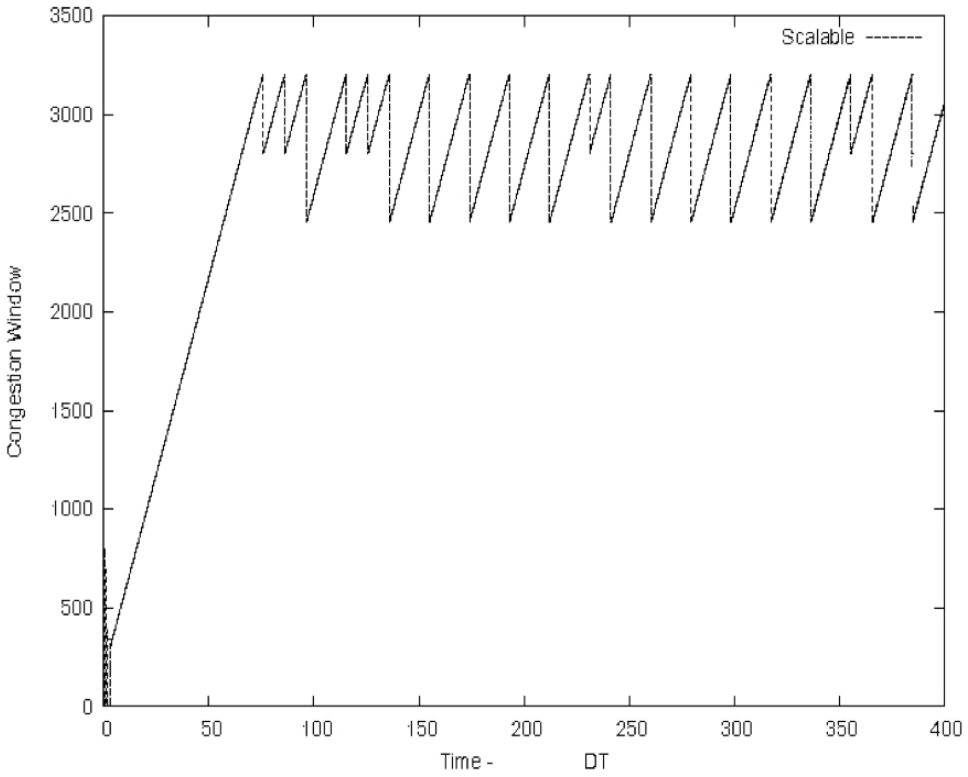


Figura 5-6 - Janela de Congestionamento Scalable TCP

Com os dados apresentados até o momento é possível verificar que não é possível atingir o limite máximo de uso de banda disponível com as técnicas de uso de janelas nas implementações TCP estudadas. O comportamento das janelas de congestionamento mostra que é necessário muito tempo até que seja atingido o tamanho máximo da janela em comparação com o tempo de redução da janela quando é detectada uma perda.

A seguir, foi testada a performance das implementações durante a fase de *slow start*. Para este teste o que se deseja aferir é, primeiramente, o tempo de duração da fase *slow start* e a taxa de perda de segmentos. O tempo de duração da fase de *slow start* é importante pois quanto mais tempo se passa nessa fase menor a capacidade do enlace é aproveitada. A taxa de perda de segmentos é medida através da divisão do número de segmentos perdidos pelo número total de segmentos enviados durante a fase de *slow start*. Pela figura 5-7 é possível verificar

que todas as implementações tiveram um comportamento similar e de pequena duração durante a fase de *slow start*. Como todas as implementações apresentam o mesmo mecanismo, este comportamento já seria esperado.

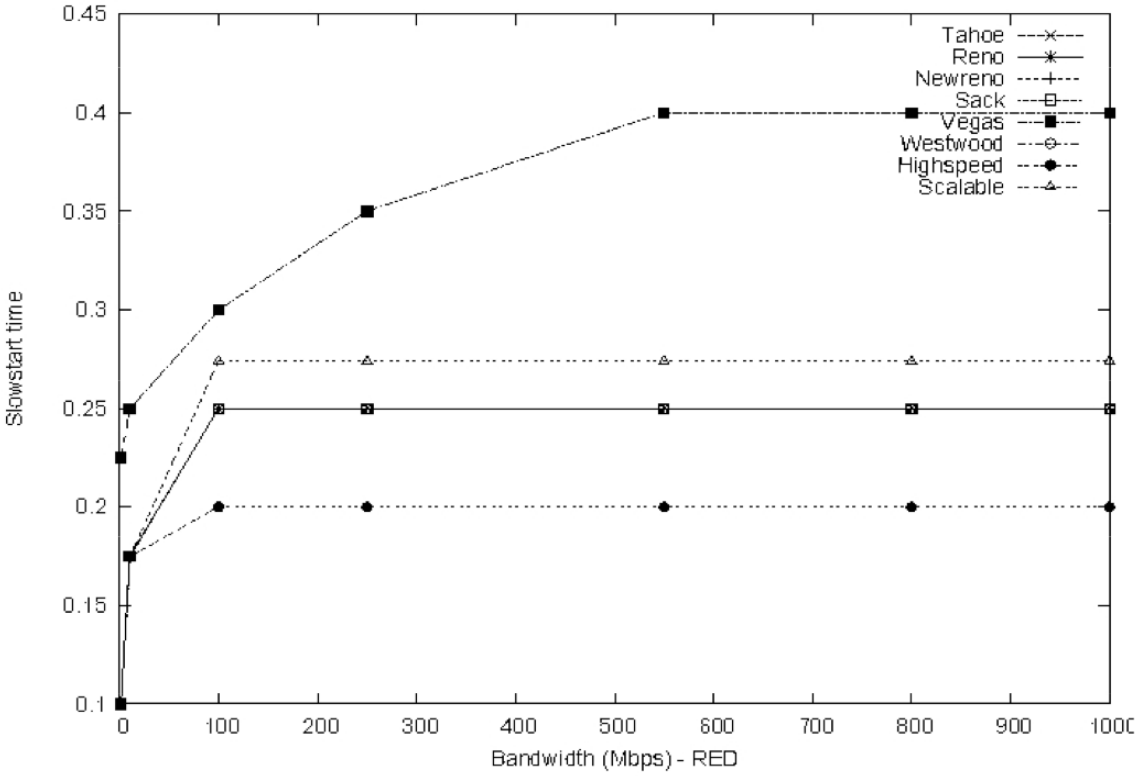


Figura 5-7 - Duração da fase de slow start

A figura 5-8 mostra a taxa de perda de segmentos atingida para as diferentes implementações durante a fase de *slow start*. Como pode ser visto, as implementações apresentam uma taxa de perda de segmentos similar e estável a medida que a largura de banda aumenta. Um ponto interessante analisado é o fato de somente as implementações Vegas e HSTCP terem taxa de perda de segmentos igual a zero. Apesar da fase de *slow start* na implementação Vegas ser um pouco maior que as demais, este comportamento é mais efetivo ao evitar perda de segmentos também durante a fase de *slow start* e demonstra alinhamento com as diretivas de implementação Vegas descritas em [6].

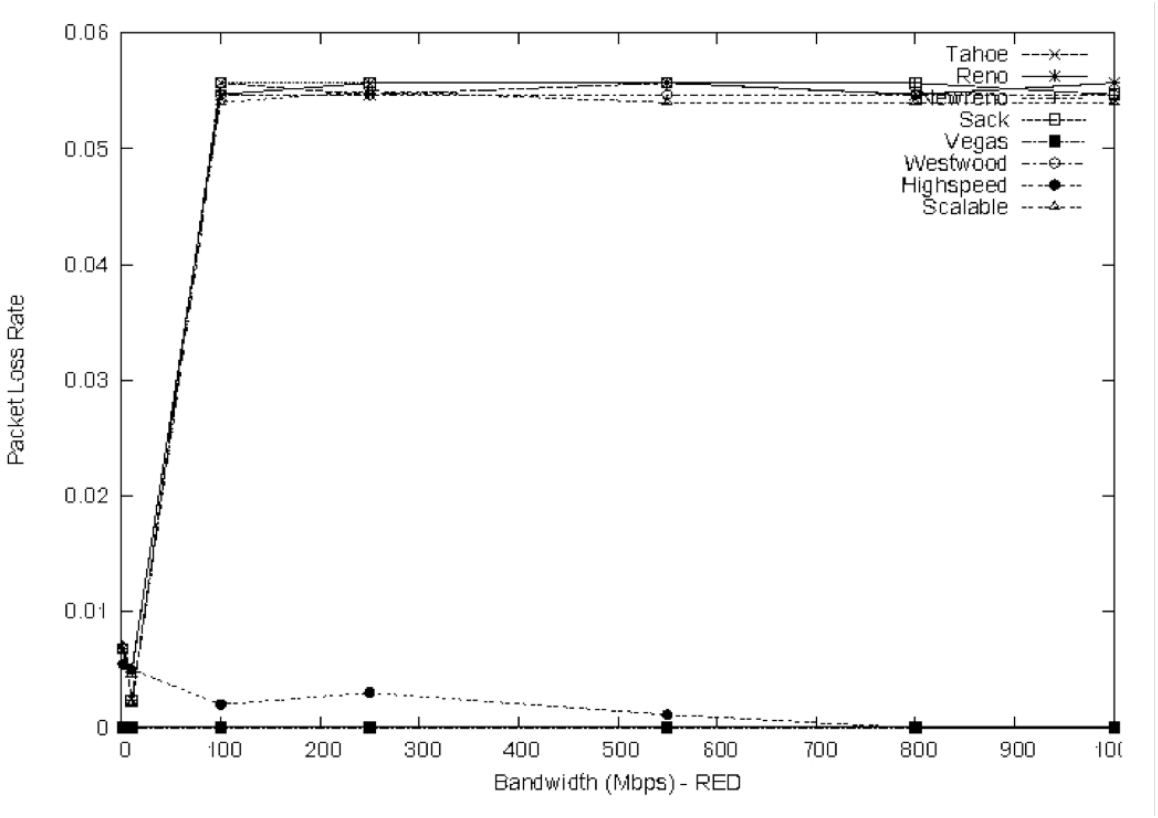


Figura 5-8 - Taxa de perda de segmentos

A performance das implementações durante a fase de *congestion avoidance* também é alvo deste comparativo, sendo o interesse no tempo de recuperação de cada implementação para atingir o tamanho máximo da janela de congestionamento novamente após a perda de um segmento. Para este teste somente foi considerada a primeira fase de *congestion avoidance* ocorrida a cada aumento de largura de banda. A figura 5-9 apresenta o tempo em segundos para que ocorra a recuperação em cada implementação enquanto a largura de banda é aumentada. Como esperado, o tempo de recuperação aumenta conforme a largura de banda aumenta. Pelo gráfico mostrado é possível verificar que o tempo de recuperação das implementações Tahoe, Reno, NewReno, SACK e STCP é por volta de 70 segundos, ou 2800 RTTs, enquanto que para as implementações Westwood e Vegas são necessários 10 segundos a mais. Entretanto, para a implementação HSTCP o tempo de recuperação é muito pequeno, da ordem de 2 segundos. Este

tempo reduzido é esperado visto o comportamento oscilatório apresentado pela implementação HSTCP conforme mostrado na figura 5-5. Tal fato deve-se ao rápido aumento da janela de congestionamento na fase de *slow start*. Isto faz com que a implementação HSTCP mostre-se bastante agressiva na utilização de banda disponível.

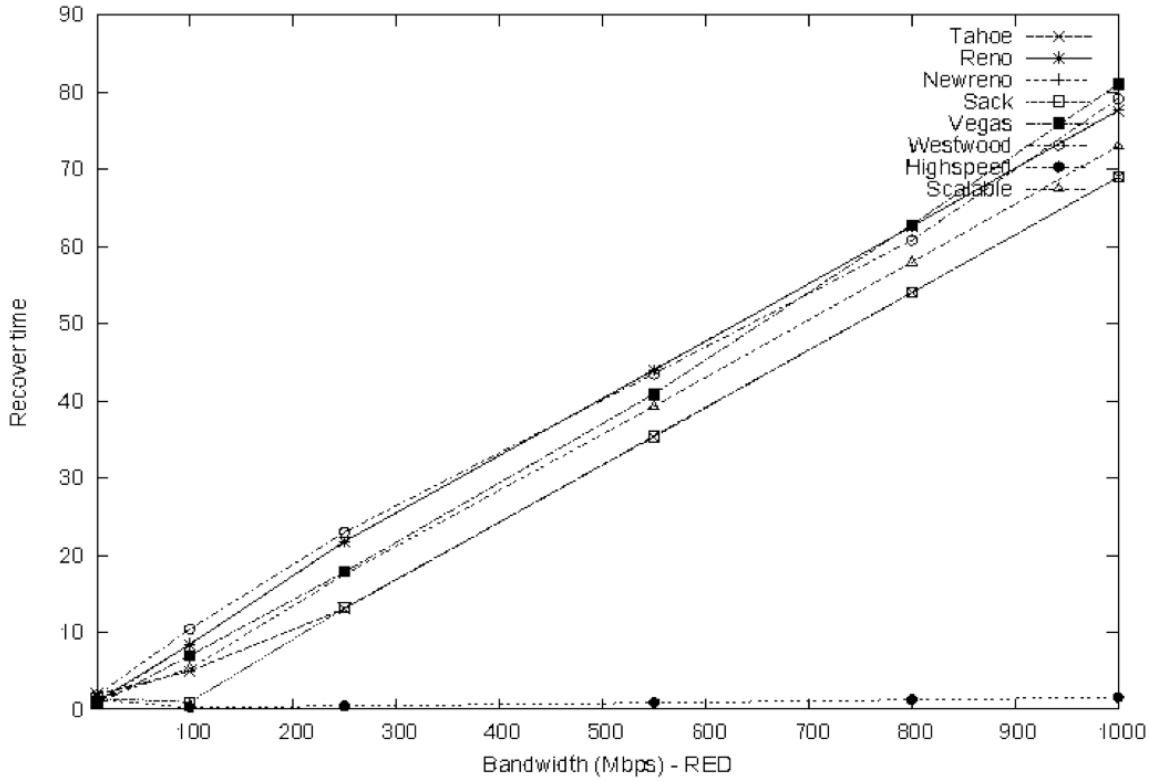


Figura 5-9 - Tempo de recuperação em função do aumento de banda

Outro ponto interessante de observação diz respeito a implementação Westwood. Verificou-se que o cálculo da estimativa de largura de banda durante o período inicial não é muito preciso, e com isso após as primeiras perdas de segmentos a janela de congestionamento e o *slow start treshold* recebem valores muito pequenos. A figura 5-10 exibe os valores da janela de congestionamento (*cwnd*) e de *slow start threshold* (*ssthresh*) ao longo do tempo quando a largura de banda do enlace é de 1000Mbps. Os valores de *cwnd* e *ssthresh* se mantêm pequenos durante o início da conexão, indicando o problema descrito acima. Como

resultado, a implementação Westwood desperdiça boa parte da banda disponível até que o cálculo da estimativa seja mais preciso.

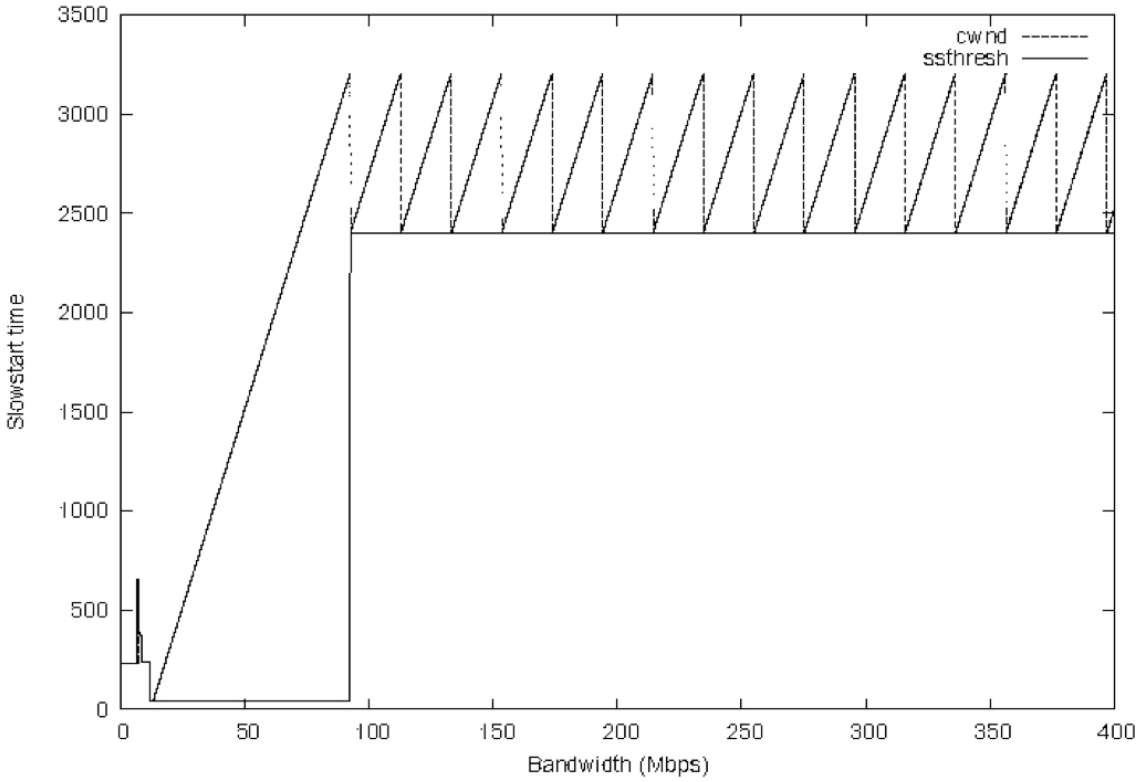


Figura 5-10 - Janela de Congestionamento em função do uso de banda

6 CONCLUSÕES

O TCP, apesar de concebido há mais de 20 anos, é o protocolo de camada de transporte mais utilizado atualmente em redes de computadores. Entretanto, a evolução do TCP não acompanhou o avanço de tecnologia das rede físicas, ocasionando a baixa utilização de banda que as novas redes provêem. Com isso, aprimoramentos do TCP original, também conhecido como TCP Tahoe, foram propostos ao longo do tempo para que fosse possível melhorar sua resposta a eventos de congestionamento e também melhorar a utilização de banda oferecida pelas novas redes físicas.

Neste estudo foram apresentadas as principais implementações do TCP encontradas nos sistemas computacionais: Tahoe, Reno, NewReno, SACK, Vegas, Scalable TCP e Westwood. Estas foram comparadas com a implementação HSTCP, uma proposição de padrão datada de 2003 [9] que ainda se encontra em caráter experimental e foi desenvolvida para otimização de uso em redes com grande largura de banda.

Pelos testes conduzidos, as implementações Vegas, Scalable TCP (STCP), High Speed TCP (HSTCP) e Westwood obtiveram performance superior às implementações Tahoe, Reno, NewReno e SACK conforme a velocidade do enlace aumentava. Os testes fornecem dados comparativos entre as implementações utilizando um único fluxo de dados por vez o que permite olhar para cada implementação individualmente.

Pelo comportamento averiguado da implementação HSTCP individualmente, esta apresenta a melhor taxa de utilização de banda disponível em relação às outras implementações. Entretanto, este comportamento agressivo no uso de banda sugere que esta implementação não seja justa no uso da rede num ambiente real onde

vários fluxos concorrem ao mesmo tempo. Embora esta tendência pareça prejudicial quando se leva em consideração o uso de rede no todo pois impede o uso justo, há casos em que o favorecimento é benéfico. Por exemplo, o uso em VLANs destinadas a backup de dados, onde são feitas grandes transferências de dados.

Para que seja comprovado este comportamento, fazem-se necessários testes com fluxos concorrentes. Como sugestão, futuros testes podem ser feitos utilizando-se uma topologia do tipo haltere (*dumbbell*), com vários fluxos, cada um representando as implementações aqui avaliadas. Testes com outras implementações em uso, como o BIC TCP e CUBIC TCP, sendo este último a implementação padrão dos sistemas Linux a partir do kernel 2.6.13, também podem ser efetuados para verificação.

REFERÊNCIAS

[1] COMER, Douglas. **Interligação de redes com TCP/IP, vol. 1 princípios, protocolos e arquitetura.** 5.ed. Rio de Janeiro: Campus, 2006.

[2] JACOBSON, V. **Congestion Avoidance and Control**. In Proceedings of SIGCOMM ’88 Symposium on Communication Architectures and Protocols, Agosto1988.

[3] JACOBSON, V. **Modified Congestion Avoidance Algorithm**, end2end-interest mailing list, April 30, 1990. Disponível em: ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail. Acessado em: 24/11/2009.

[4] FLOYD., S.; HENDERSON, T. **The New- Reno Modification to TCP’s Fast Recovery Algorithm** RFC 2582, Abril 1999. Disponível em: http://www.ietf.org/rfc/rfc2582.txt. Acessado em 23/11/2009.

[5] BRAKMO, L.; O’MALLEY, S.; PETERSON, L. **TCP Vegas: New Techniques for Congestion Detection and Avoidance**. Proceedings of the SIGCOMM '94 Symposium on Communication Architectures and Protocols, Agosto 1994, pg. 24-35.

[6] BRAKMO, L.; PETERSON, L. **TCP Vegas: End to end congestion avoidance on a global Internet.** IEEE Journal on Selected Areas in Communication, Vol 13, No. 8, Outubro 1995, pg. 1465-1480.

[7] GERLA, M.; SANADIDI, M.; WANG, R.; ZANELLA, A.; CASETTI, C.; MASCOLO, S.. **TCP Westwood: Congestion Window Control Using Bandwidth Estimation**. In Proceedings of IEEE GLOBECOM, Vol. 3, Novembro 2001, pg 1698-1702.

[8] KELLY, T.. **Scalable TCP: Improving Performance in Highspeed Wide Area Networks.** In Proceedings of First International Workshop on Protocols for Fast Long-Distance Networks, Fevereiro 2003.

[9] FLOYD, S.. **HighSpeed TCP for Large Congestion Windows**, RFC 3649, Dezembro 2003. Disponível em: http://www.ietf.org/rfc/rfc3649.txt. Acessado em 23/11/2009.

[10] SOUZA, E., AGARWAL, D.. **A HighSpeed TCP Study: Characteristics and Deployment Issues**. LBNL Technical Report Number LBNL-53215. Disponível em: http://www-itg.lbl.gov/evandro/hstcp/. Acessado em 02/02/2010.

[11] FLOYD, S.. **Limited Slow-Start for TCP with Large Congestion Windows**, RFC 3742, Março 2004. Disponível em: http://www.ietf.org/rfc/rfc3742.txt. Acessado em 23/11/2009.

[12] NS-2, software para simulação de redes de computadores. Disponível em http://www.isi.edu/nsnam/ns/. Acessado em 23/02/2010.

[13] PRETE, Ligia; SHINODA, Ailton. **Análise do Comportamento das Variações do Protocolo TCP.** Artigo publicado em: Anais do CNMAC v.2 - ISSN 1984-820X.

[14] TANENBAUM, Andrew. **Redes de Computadores**. 4a. Ed., Rio de Janeiro: Campus, 2003.