Uma Proposta para TCP Reno no Controle de Congestionamento em Alta Velocidade

Franklin Magalhães Ribeiro Junior[1], Ricardo José de P. B. Salgueiro[1]

[1]Departamento de Computação
Universidade Federal de Sergipe (UFS) – São Cristóvão, SE – Brasil
{franklin.mr3,ricardo.salgueiro}@gmail.com

Abstract. *With the computer network's diffusion, the competition for traffic increased, on the other side the bandwidth also increased, however this demand is not enough to solve problems of packet loss on network congestion control. This paper proposes a new analytical approach to congestion control in high-speed networks and proposes a change to the TCP Reno variant. Theoretically and partially, experimentally, the new proposal showed better handle packet loss that variant Reno.*

Resumo. *Com a difusão das redes de computadores, a disputa pelo tráfego aumentou, por outro lado a largura de banda também aumentou, porém esta demanda não é suficiente para sanar problemas de perda de pacotes no controle de congestionamento de redes. Este artigo propõe uma nova abordagem analítica para o controle de congestionamento em redes de alta velocidade de tráfego e propõe alterações na variante TCP Reno, mais utilizada atualmente. Teoricamente e parcialmente, experimentalmente, a nova proposta mostrou lidar melhor com a perda de pacotes que a variante Reno.*

1. Introdução

Atualmente, apesar do aumento na largura de banda nas redes de computadores, ainda há um crescente esforço pela manutenção e integridade na transferência dos dados através das mesmas. Com isto são geradas preocupações acerca de congestionamento em redes, sobretudo na camada de transporte [El-Sayed et. al. 2011]. Onde o agente TCP atua no controle desse congestionamento.

O agente TCP possui implementações básicas, como algoritmos provem o controle no congestionamento de redes, tais como a "partida lenta", "prevenção de congestionamento" e "retransmissão rápida" [Tanenbaum 2003] [IETF - Allman et. al. 1999]. Também existem variantes do TCP, como por exemplo, o TCP Reno que implementa os algoritmos citados e também o "recuperação rápida" [Qureshi 2009].

Devido à busca por melhorias para determinadas situações de congestionamento de rede, investigações são realizadas acerca das variantes TCP existentes [Bhanumathi and Dhanasekaran 2010], como o TCP New Reno, Vegas, FAST, BIC, entre outros. Sendo uma das grandes preocupações das pesquisas, a busca pelo aprimoramento dos

algoritmos de variantes TCP para o controle do congestionamento em tráfegos de alta velocidade.

Esta pesquisa apresenta com apoio da ferramenta *network simulator* 2 (ns2) [NS-2 2007] uma investigação das características de uma das variantes TCP mais usadas, o TCP Reno [Kurose and Ross 2010] em um cenário de congestionamento de rede, bem como o comportamento do Reno em alta velocidade. Tal investigação serve como embasamento para a segunda parte do trabalho que apresenta minha nova proposta do TCP Reno num tráfego de alta velocidade.

O restante do artigo esta organizado da seguinte forma: a seção 2 explana controle de congestionamento TCP, a seção 3 apresenta o ambiente de simulação, a seção 4 mostra os trabalhos correlatos. A seção 5 apresenta a metodologia. A seção 6 contem a simulação, análises e nova proposta, cuja subseção 6.3 apresenta a nova proposta e o algoritmo híbrido, e finalmente, a seção 7 apresenta as conclusões e trabalhos futuros.

2. Controle de Congestionamento TCP

Uma rede está congestionada na existência de um roteador gargalo o qual limita a velocidade da conexão [Tanenbaum 2003]. O controle de congestionamento TCP usa o controle fim a fim, e o congestionamento é somente percebido por esgotamento de temporização ou 3 *acks* duplicados [Kurose and Ross 2010].

A atual versão TCP (Reno) inicialmente aloca o tamanho de sua janela de congestionamento (*cwnd*) para 1 segmento a ser enviado. Essa janela aumentará seu valor exponencialmente de pelo algoritmo de "partida lenta" [Kurose and Ross 2010], até que o buffer do roteador gargalo não consiga mais armazenar pacotes e descarte próximos enviados a ele [El-Sayed et. al. 2011] [Kurose and Ross 2010]. Quando isso ocorrer o algoritmo ativa a "retransmissão rápida", para retransmitir os pacotes perdidos e logo em seguida a "recuperação rápida".

Na "recuperação rápida", ao invés de recomeçar o valor da janela com apenas 1 segmento (*cwnd=1*), esta iniciará do valor janela (*ssthresh=cwnd/2*) [El-Sayed et. al. 2011], feito isso, o próximo passo é a "prevenção de congestionamento", porém alocando o valor da janela para *ssthresh* segmentos [Kurose and Ross 2010].

Ainda no estado da prevenção de congestionamento, quando a janela alcançar o valor de *ssthresh,* ao invés de aumentar seu valor de segmentos exponencialmente, aumentará, com incremento de apenas 1 segmento e assim sucessivamente. Porém caso ocorra esgotamento de temporização, à janela é atribuído o valor 1.

3. Ambiente de Simulação

O simulador, Network Simulator, utilizado neste trabalho, é um software livre que simula dispositivos em cenários de redes de computadores [NS-2 2007]. O simulador possui 3 versões, ns1, 2 e 3. Sendo a segunda versão utilizada para embasamento experimental desta pesquisa. Embora não seja a mais recente, ainda é dada manutenção pelos desenvolvedores.

O simulador ns2 foi criado em 1997 e interpreta a linguagem de programação *Tool Comand Language* (Tcl) [Ousterhout 2012], criada por John Ousternhout para a

execução das simulações. Além disto, o NS2 contem em sua suite um ambiente visual para animação, o *Network Animator* (nam), dos dispositivos simulados, respectivos ao código Tcl e uma ferramenta gráfica de plotagem de dados dispostos em dois eixos x e y [NS-2 2007].

4. Trabalhos Correlatos

Em seu trabalho, Xiuchao et. al. (2009), apresentaram algoritmos clássicos usados no controle de congestionamento em alta velocidade e também uma nova abordagem do TCP para o controle de congestionamento, a qual baseia se no acréscimo ou decréscimo da janela de congestionamento (*cwnd*) por *delay* de forma independente e utiliza o sincronismo baseado em uma fila de valores *delay* na detecção do congestionamento, tal abordagem foi nomeada de Sync-TCP. Assim como em Leith et. al. (2005) propôs um framework chamado H-TCP, baseado em algoritmo adaptativo AIMD.

Os autores [Xiuchao et. al. 2009] e [Leith et. al. 2005] compararam o desempenho do Sync-TCP e H-TCP respectivamente com relação à banda das variantes CUBIC, FAST e Compound. Xiuchao et. al. (2009) observaram que o Sync-TCP mostrou-se melhor no que se refere à demanda do tráfego de rede e o mesmo foi descoberto por Leith et. al. Em seu framework H-TCP com relação às demais variantes.

Em sua investigação, Katto et. al. (2008), avaliaram o desempenho do controle de controle de congestionamento em variantes TCP híbridos, tais quais Compound TCP, ARENO, YeAH-TCP e TCP-Fusion com relação às variantes TCP FAST e TCP Reno.

Katto et. al. (2008) perceberam que o Fusion, YeAH e Compound proveram melhor largura de banda e justiça. Além de que as variantes híbridas apresentaram menor perda de pacotes conforme a largura de banda aumenta ao contrário do Reno.

Finalmente, Wang et. al. (2011), propôs em sua pesquisa um algoritmo para controle de congestionamento chamado TCP-FIT. Além da proposta, foi realizada uma comparação do TCP-FIT com as variantes TCP CUBIC, Compound, TCPW, FAST, Vegas e Reno em 5 cidades distintas em redes WI-FI, Ethernet, CDMA e ADSL.

Wang et. al. (2011) concluíram que o TCP-FIT apresentou melhor desempenho de tráfego e justiça que as variantes TCP CUBIC, TCP Compound, Veno, TCPW e Reno.

5. Metodologia

A metodologia para obtenção dos resultados e análises está dividida em duas vertentes, a primeira experimental com auxílio da ferramenta Network Simulator 2 e a segunda, analítica e explana minha nova proposta sobre o TCP Reno no controle de congestionamento em alta velocidade de tráfego de redes.

5.1. Métodos do Experimento

Para a investigação experimental foi utilizado o simulador Network Simulator 2, na busca pela compreensão do funcionamento do controle de congestionamento em redes de computadores na variante TCP Reno (a mais usada) e a investigação experimental do Reno frente a problemas de tráfego de redes em alta velocidade.

Foram utilizados dois cenários (ver Tabela 1) em uma topologia (Figura 1) ilustrada pelo *Network Animator*, onde o nó "0" representa um cliente TCP em uma aplicação FTP, o nó "1", um roteador gargalo e o nó "2", um TCP *Sink* (Receptor).

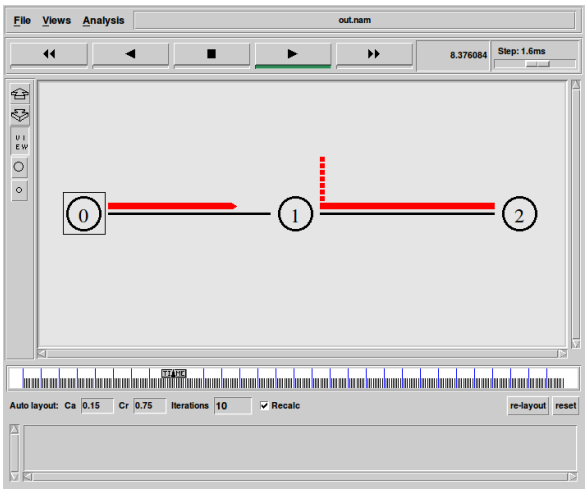

Figura 1. Topologia

Com relação aos cenários, os dados de entrada (atraso = 10ms para ambos) do primeiro correspondem à simulação do TCP Reno (em baixo tráfego) e o a entrada do segundo ao TCP Reno em velocidade aumentada, cujos dados de entrada estão descritos na tabela (ver Tabela 1).

Tabela 1. Dados de entrada dos cenários da simulação

	Cenário 1	Cenário 2 (Velocidade aumentada)
Tempo de Simulação	35 segundos	35 segundos
Tamanho da Janela (cwnd)	20	20
Largura de Banda do nó (0) a (1)	0.9 Mb	1000 Mb
Largura de Banda do nó (1) a (2)	45 Kbps	450 Kbps

6. Resultados, Análises e Proposta

Esta seção encontra-se dividida em três subseções, a 6.1 e 6.2, as quais apresentam a análise dos resultados obtidos de forma experimental com a variante TCP Reno e a 6.3, a qual mostra a minha proposta híbrida com o TCP Reno.

6.1. TCP Reno

Nesta subseção foi realizada a simulação da variante Reno no simulador Network Simulator 2, onde foram adotados os dados de entrada do cenário 1 (ver Tabela 1). Após a simulação com o Reno obteve-se os resultados nas figuras (Figura 2 e 3).

Com relação ao TCP Reno os dados obtidos estão dispostos nas Figuras 2 e 3, onde a Figura 2 corresponde ao gráfico dos pacotes (enviados em vermelho e momento da perda de pacotes em verde) e a Figura 3 ao cwnd.

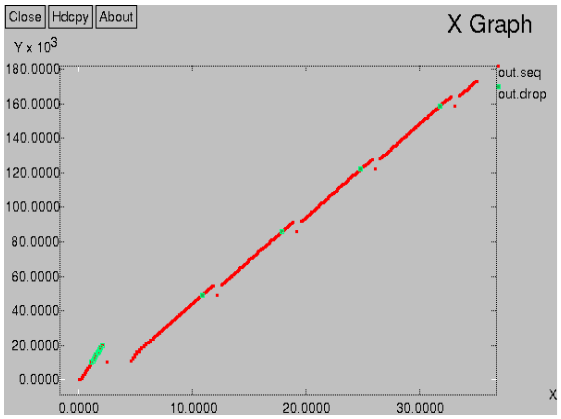

Figura 2. Reno (pacotes)

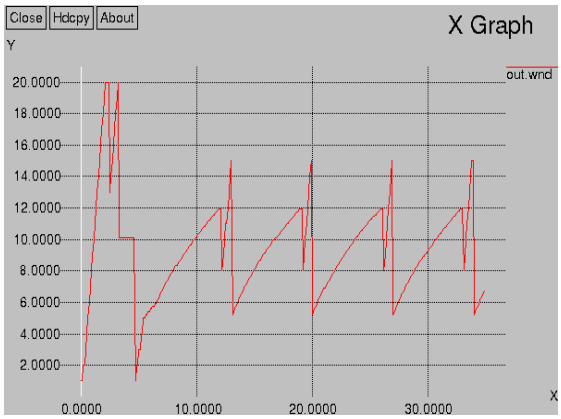

Figura 3. Reno (cwnd)

Percebeu-se que para a Figura 3 no primeiro momento ocorreu a execução da partida lenta, porém após esse momento, o *ssthresh* foi igualado a metade do *cwnd* (*ssthresh≈6*) e quando houve perda de pacote mesmo retornou ao mesmo valor.

Isso ocorre porque a variante TCP Reno usa o algoritmo de recuperação rápida, o qual define tal comportamento [Tanenbaum 2010], onde *ssthresh=cwnd/2*, onde o tamanho da janela inicia no valor de *ssthresh*, e é incrementado em +1 segmento ao invés de exponencialmente (como na partida lenta).

6.2. TCP Reno em Velocidade Aumentada

Nesta etapa do experimento foi realizada a simulação com os dados de entrada do cenário 2 (ver Tabela 1), a fim de simular o comportamento do Reno em um tráfego de velocidade aumentada. Obtiveram-se os resultados contidos nas figuras abaixo (Figura 4 e 5).

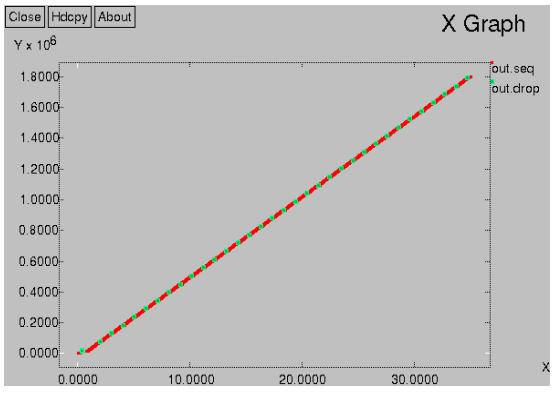

Figura 4. Reno em tráfego 1Gbps (pacotes)

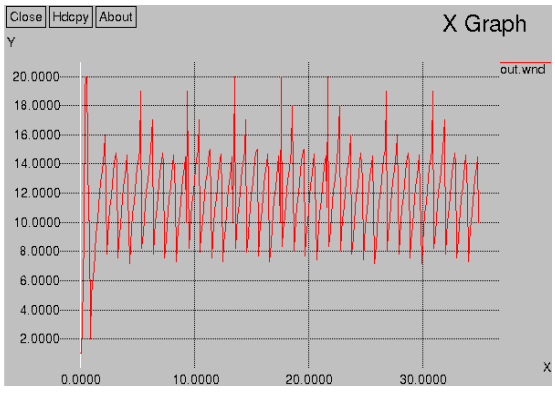

Figura 5. Reno em tráfego 1Gbps (cwnd)

Ao analisar as figuras 4 e 5 percebeu-se que houve num mesmo intervalo de tempo, embora em uma transmissão mais veloz que a das figuras 2 e 3, uma grande quantidade de perda de pacotes (indicada pelos pontos verdes na Figura 4 e diversas quedas no valor de *cwnd* na Figura 5).

Essa maior perda de pacotes ocorre como efeito colateral da variação (oscilação) do tamanho da janela de congestionamento, ou seja isso ocorre devido ao próprio controle de congestionamento da variante TCP Reno.

Pode-se perceber na Figura 5, que o valor da janela retorna ao valor de *sthresh* após a perda de pacotes com 3 acks duplicados e incrementa 1 em *cwnd* até que ocorra novamente a perda. "*Podendo suportar numa transmissão de 10Gbps apenas um evento de perda a cada 5 bilhões de segmentos*" [Kurose 2010].

6.3. Proposta e Algoritmo Híbrido

Nesta seção é estruturada a nova proposta para lidar com o problema da grande quantidade de perda de pacotes pelo TCP Reno em velocidade aumentada de tráfego, mencionado na seção 6.2.

Inicialmente, partindo dos experimentos anteriores das seções 6.1 e 6.2, partiu-se do princípio que o Reno sempre tentou aumentar o tamanho da janela, e que isso sempre alcançou um pico no *cwnd*, o qual acarretou em uma perda de pacotes.

No âmbito de solucionar esse problema foi simulado, com uso do cenário 2 (ver Tabela 1), uma visão inicial do comportamento do Reno em alta velocidade caso o valor da janela fosse constante e sempre igual ao *(ssthresh+ssthresh *0.5)*, já que a *cwnd* do Reno variou entre *cwnd* e *cwnd*0.5*, em outras palavras *cwnd*0.75*.

O resultado desta simulação está ilustrado em (Figura 6 e 7).

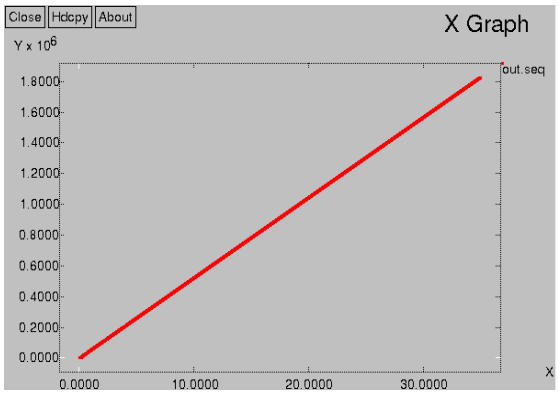

Figura 6. Pacotes Reno em alta velocidade cwnd(constante)

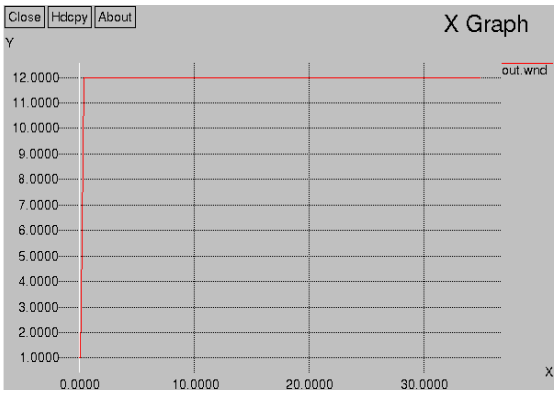

Figura 7. Janela Reno em alta velocidade cwnd(constante)

Percebeu-se que a Figura 6 não apresentou eventos de perda de pacotes (não houve pontos verdes) quando o tamanho da janela ficou constante e igual ao *(ssthresh+ssthresh *0.5)* correspondente ao da Figura 5.

Com base na descoberta das Figuras 6 e 7 desta investigação, uma proposta analítica foi elabora de maneira algorítmica e de complexidade *n(log n),* descrita nos quadros abaixo (Quadro 1 e 2):

```
//além do pseudo-código abaixo, deve ser escrito na linha
//seguinte onde consta a atribuição "ssthresh=cwnd/2;" no código do TCP Reno um contador++
//para contar o número de oscilaçõese contador deve ser estático

void funcMenores_ssthresh(int quantidade)
{
    heapsort(Vetor_ssthresh[]);
    for (int j=1, j<controle, j++)
    {
        if (quantidade==3)
        {
            if ((Vetor_ssthresh[j]==Vetor_ssthresh[j-1]) AND ((Vetor_ssthresh[j]==Vetor_ssthresh[j+1]))
            {
                CWND_MAX = Vetor_ssthresh[j] + Vetor_ssthresh[j]*0.5;
                instavel = false;
                return CWND_MAX;
                break;
            }
        }
        else if(quantidade=2)
        {
            int aux = (Vetor_ssthresh[j] + Vetor_ssthresh[j-1])/2 ;
            CWND_MAX = aux + aux*0.5;
            instavel = false;
            return CWND_MAX;
            break;
        }
    }
}
```

Quadro 1. Função funcMenores_ssthresh (retorna valor máximo da janela cwnd_max)

```
Algoritmo_Para_Tornar_CWND_CONSTANTE:

  static int i=0;

  if (3_acks_duplicados==true OR timeout == true)
      {
        instavel=true;
      }

  while ((instavel == true) AND (i>=3)) //ativado após pelo menos 3 oscilações
      {
        Vetor_ssthresh[i]=ssthresh;
        if (i>N) // "se estiver muito instavel, muitas oscilações, N é o numero de oscilações
                 //ou seja, vários dispositivos de rede disputando e não disputando pelo roteador gargalo"
        {
           CWND = funcMenores_ssthresh(2);
        }
        else
        {
            CWND = funcMenores_ssthresh(3);
        }
          i=i+1;
      }

//FIM DO ALGORITMO
```

Quadro 2. Corpo principal do Algoritmo (chama a funcMenores_ssthresh)

O algoritmo é híbrido e funciona da seguinte forma:

Inicia-se a execução do TCP Reno normalmente, iniciando com a partida lenta, subindo o valor da janela *cwnd* exponencialmente. Quando houver perda de pacote e o algoritmo receber 3 *acks* duplicados, o tamanho da janela irá oscilar entre *cwnd/2* e *cwnd* (até aqui é assim que funciona o TCP Reno).

Logo após, **suponhamos**, 3 oscilações, a linha de execução entra no comando *while* (Quadro 2), onde é armazenado o valor de *Vetor_ssthresh[i]=ssthresh*, depois disto encontramos o *if (i>n)*, onde N é um número limite de oscilações permitidas. (*N*

oscilações permitidas, pois mesmo após o *cwnd* ficar constante, outros dispositivos podem eventualmente se conectar a rede e disputar por largura de banda do roteador gargalo, o que fará o número de oscilações aumentar).

Dentro deste *IF*, se as *i* rodadas são maiores do que *N*, chame a função *funcMenores* (Quadro 1) com parâmetro quantidade=2, onde o vetor *Vetor_thresh* tem seus valores ordenados com *heapsort*, e será atribuído a variável aux soma dos dois primeiros valores dividido por 2 e ao tamanho da janela(*CWND_MAX*) é lhe atribuído a soma de *aux + aux*0.5*.

Caso as i rodadas não alcancem as N oscilações, isso significa que ainda não há muitas oscilações e o algoritmo entra no *else* (Quadro 2). Quando isso ocorre, há uma chamada da função (*funcMenores*) com parâmetro quantidade=3. Nesta função (Quadro 1), após ordenado o vetor *Vetor_ssthresh[]* pelo *heapsort*, há uma verificação: se os três primeiros elementos do vetor são iguais. Caso afirmativo, o tamanho máximo da janela de congestionamento é igual ao elemento de índice 1 do vetor + o mesmo elemento*0.5.

No cálculo de *CWND_MAX*, quando *quantidade = 2* na função (*funcMenores*) do Quadro 1 é CWND_MAX = aux + aux*0.5, onde *aux=(Vetor_ssthresh[j]+ Vetor_ssthresh[j-1])/2* para que, mesmo que os valores dos elementos j do vetor sejam muito distantes, apenas a média dos menores seria calculada, o que garante que o tamanho da janela irá permanecer constante.

Após o tamanho da janela de congestionamento (*cwnd*) permanecer constante, outros dispositivos podem disputar por largura de banda do roteador gargalo e com isso, a variante TCP Reno volta à execução a partir do início da linha de comando do algoritmo de prevenção de congestionamento.

O cálculo realizado para obtenção de *CWND_MAX* pela função quando o parâmetro quantidade=3, cujo *CWND_MAX=Vetor_ssthresh[j]+ Vetor_ssthresh[j]*0.5,* dá-se pelo fato da suposição que, pelo menos os três menores e iguais valores de *ssthresh* representariam uma média do valor (ver Figura 3) dessa mesma variável.

6.3.1. Esboço da Proposta

Com base nos cálculos do *cwnd* da proposta, foi abstraído o esboço do gráfico representado pelo algoritmo da proposta, *cwnd* vs. tempo (Figura 8).

Com o esboço abstraído do gráfico, percebeu-se que há uma limitação na proposta. Após o momento que vários dispositivos disputam pela rede o tamanho de *cwnd* fica constante e obviamente menor. Porém o algoritmo não utiliza nenhum mecanismo de acréscimo ao tamanho do *cwnd* máximo (*CWND_MAX*), e que deveria ser feito, já que os dispositivos podem deixar de competir por largura de banda no roteador gargalo.

Contudo, tal limitação pode ser mais bem adaptada ao indicar, por exemplo, um incrementador ao valor de cwnd depois de certo período de tempo na execução.

Uma das possíveis soluções consta em aumentar o valor da janela, utilizando o cálculo do *Compound* TCP [Tan 2006], onde *cwnd = cwnd + 0.125*cwnd^0.75*. Entretanto, isso somente deveria ser aplicado, após certo período de tempo na execução do algoritmo, para que não houvesse perda na estabilidade do valor *cwnd*.

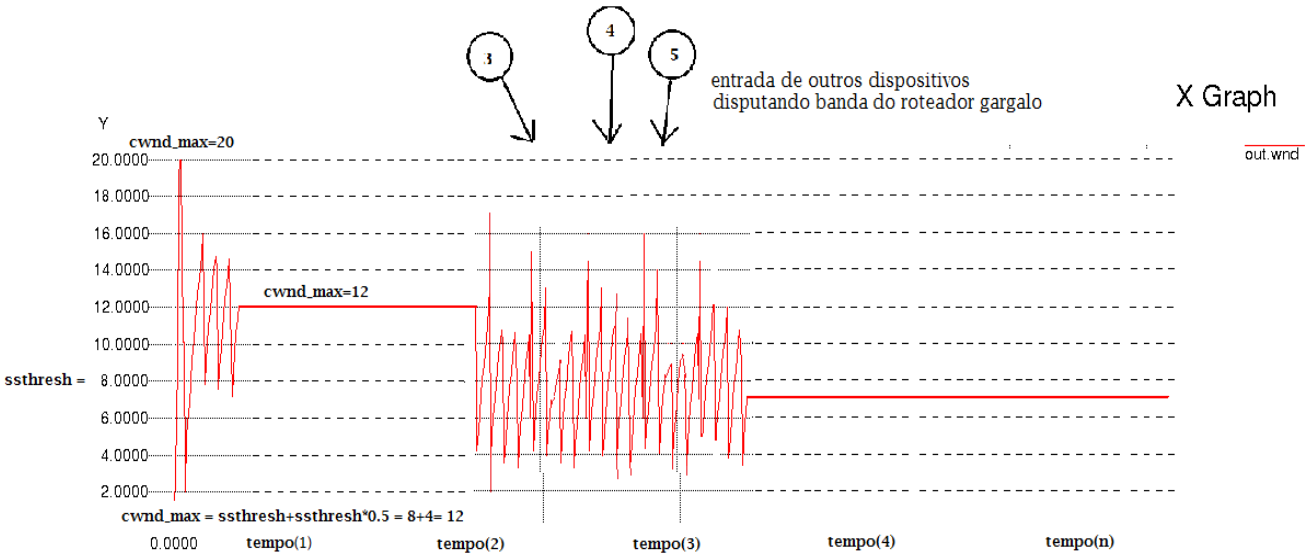

Figura 8. Esboço de gráfico aproximado pelo algoritmo a uma simulação real no ns2

7. Conclusão

Neste trabalho, além de analisar os dados obtidos com simulações realizadas através da ferramenta Network Simulator 2, o comportamento do TCP Reno, também foi analisado o problema relacionado a perda de pacotes pelo Reno em alta velocidade de tráfego e uma nova abordagem para reter essa excessiva perda de pacotes, como base para fundamentar a minha proposta de TCP Reno de comportamento mesclado com a nova abordagem.

Este artigo apresentou uma nova proposta para o TCP Reno em altas velocidades de tráfego, baseado utilizando mecanismos inovadores, visto que até o presente momento não se havia pesquisado por variações relacionadas ao Reno adotando este modelo algorítmico.

O trabalho mostrou que há uma redução significativa na perda de pacotes em uma transmissão de alta velocidade ao utilizar a nova proposta híbrida do TCP Reno com a variante TCP Reno original. Para isso, a investigação analisou mecanismos do cálculo na delimitação do tamanho máximo da janela (*cwnd*), a fim de mantê-la constante com valor atribuído aos menores valores iguais (em uma conexão sem novos dispositivos adentrando) ou aos dois menores valores obtidos de *ssthresh*, onde *cwnd=ssthresh+ssthreash/2*.

Finalmente, como trabalhos futuros sugere-se a implementação de fato do algoritmo da proposta, bem como a comparação e análise do mesmo com outras variantes de alta velocidade, como o TCP BIC, CUBIC e FAST.

Referências

Allman, M., Paxson, V. and Stevens, W. R. (1999) “TCP Congestion Control”, in IETF RFC 2581.

Bhanumathi, V., Dhanasekaran, R., (2010) "TCP variants - A comparative analysis for high bandwidth - delay product in mobile adhoc network," *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, pp.600,604, 26-28

El-Sayed, A., El-Feshawy, N., Hagag, S. (2011) “A Survey of Mechanisms for TCP Congestion Control”, International Journal of Research and Reviews in Computer Science, IJRRCS, pp.676, 682, 2-3.

Jingyuan Wang; Jiangtao Wen; Jun Zhang; Yuxing Han, (2011) "TCP-FIT: An improved TCP congestion control algorithm and its performance," INFOCOM, 2011 Proceedings IEEE , pp.2894,2902, 10-15.

Leith, D., Shorten, R., and Lee, Y. (2005) “H-tcp: A framework for congestion control in high-speed and long-distance networks,” in PFLDnet Workshop.

Katto, J.; Ogura, K.; Akae, Y.; Fujikawa, T.; Kaneko, K.; Zhou, S. (2008) "Simple Model Analysis and Performance Tuning of Hybrid TCP Congestion Control," *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pp.1,6.

Kurose, J.F. and Ross K. W. (2010), Redes de computadores e a Internet: uma abordagem top-down, Addison Wesley, 5ª edição.

NS–2 (2007), Network Simulator, Disponível em: http://www.isi.edu/nsnam/ns, Acesso em: maio de 2013.

Ousterhout, J. (2012). “Tcl Language”. Disponível em: http://www.tcl.tk/, Acesso em: maio de 2013.

Qureshi, B.; Othman, M.; Hamid, N. A W, (2009) "Progress in various TCP variants (February 2009)," Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on, pp.1,6, 17-18.

Xiuchao Wu; Mun Choon Chan; Ananda, A.L.; Ganjihal, C. (2009), "Sync-TCP: A new approach to high speed congestion control," *In Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on,* pp.181,192, 13-16.

Tan, K., Song, J., Zhang, Q., and Sridharan, M. (2006) “Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks”, PFLDnet.

Tanenbaum A. S. (2003), Redes de computadores, Campus, 4ª edição.