Universidade Federal de Santa Maria
Curso de Engenharia de Computação
Disciplina de Redes de Comunicação de Dados

Implementação TCP CUBIC

Professor: Carlos Henrique Barriquello
Alunos: Moises Goulart de Oliveira
Victor Dallagnol Bento

Santa Maria, RS - Junho/2019

1. INTRODUÇÃO

O TCP teve seu desempenho questionado com a evolução da internet para incluir uma quantidade grande de caminhos de rede de alta velocidade e longa distância. Essas redes são caracterizadas por alta largura de banda com alta latência (BDP - Bandwidth Delay Product, a quantidade de dados que podem estar em trânsito na rede), que representa o número total de pacotes transportados, mantendo a largura de banda totalmente utilizada, ou seja, o tamanho da janela de congestionamento. No TCP padrão, como o TCP-Reno, o TCP-NewReno e TCP-SACK, o TCP aumenta sua janela em um tempo de viagem por percurso (RTT - Tempo para remetente enviar uma mensagem para um destinatário e receber uma resposta de volta), fazendo com que a velocidade de transporte de dados do TCP usada em todos os principais sistemas operacionais, incluindo Windows e Linux, seja lenta, sobrecarregando as redes, especialmente se o comprimento dos fluxos for muito menor que o tempo em que o TCP cresce suas janelas para o tamanho total do BDP de um caminho. Por exemplo, se um fluxo terminar antes dessa hora, o caminho será sobrecarregado.

O CUBIC é uma implementação do TCP com um algoritmo de controle de congestionamento otimizado para redes de alta largura de banda com alta latência. O tamanho da janela é uma função cúbica do tempo desde o último evento de congestionamento, com o ponto de inflexão configurado para o tamanho da janela anterior ao evento. Por ser uma função cúbica, existem dois componentes para o crescimento da janela. A primeira é uma parte côncava onde o tamanho da janela aumenta rapidamente até o tamanho antes do último evento de congestionamento. Em seguida, está o crescimento convexo, onde CUBIC investiga mais largura de banda, lentamente no início e depois muito rapidamente. CUBIC passa muito tempo em um platô entre a região de crescimento côncava e convexa que permite que a rede se estabilize antes que CUBIC comece a procurar por mais largura de banda.

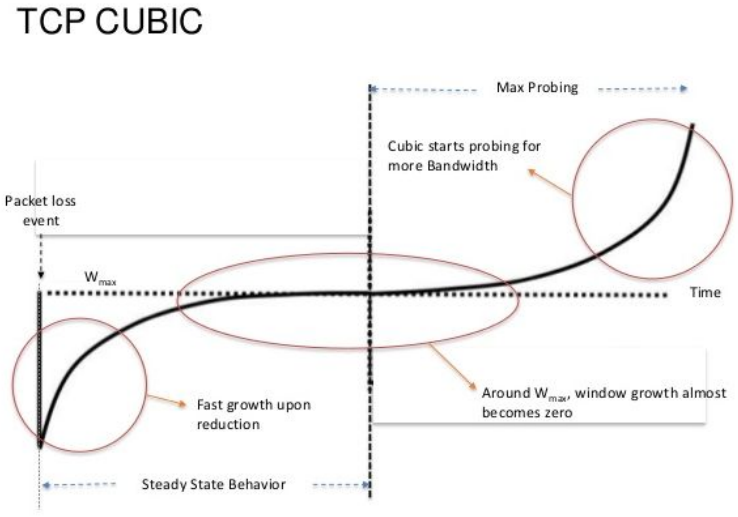

Figura 1: TCP CUBIC esquemático.

Outra grande diferença entre os sabores CUBIC e TCP padrão é que ele não depende da cadência dos RTTs para aumentar o tamanho da janela. O tamanho da janela do CUBIC depende apenas do último evento de congestionamento. Com o TCP padrão, os fluxos com

tempos de atraso de ida e volta muito curtos (RTTs) receberão os ACKs mais rapidamente e, portanto, suas janelas de congestionamento aumentaram mais rapidamente do que outros fluxos com RTTs mais longos. CUBIC permite mais justiça entre os fluxos, uma vez que o crescimento da janela é independente do RTT.

O TCP CUBIC é implementado e usado por padrão nos kernels 2.6.19 e posteriores do Linux, assim como na Atualização de Criadores de Queda do Windows 10.1709 e atualização do Windows Server 2016 1709.

2. PROTOCOLO TCP

O TCP é um protocolo da camada de transporte da arquitetura TCP/IP (**Figura 2**) com a função de transferir dados de forma confiável e econômica entre uma determinada origem e um determinado destino, independente das redes físicas em uso. É orientado a conexões, ou seja, passa por três fases: o estabelecimento, a transferência de dados e o encerramento, além de as conexões serem *full-duplex* (tanto o transmissor e receptor podem transmitir dados de forma simultânea, em ambos os sentidos).

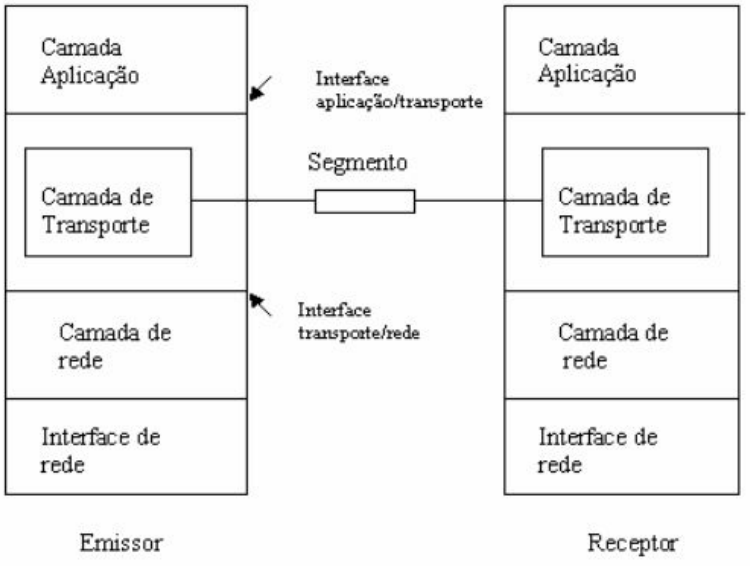

Figura 2 - Modelo de referência TCP/IP.

As entidades transmissoras e receptoras do TCP trocam dados na forma de segmentos, que consiste em um cabeçalho seguido de dados provenientes de aplicação. O cabeçalho de um segmento começa com um tamanho fixo de 20 bytes e é nele que se encontram os campos que auxiliam na realização das tarefas essenciais do protocolo. Para restringir o tamanho do segmento, tem-se os fatores de carga útil do IP e a unidade máxima de transferência (MTU, *Maximum Transmission Unit*) da rede.

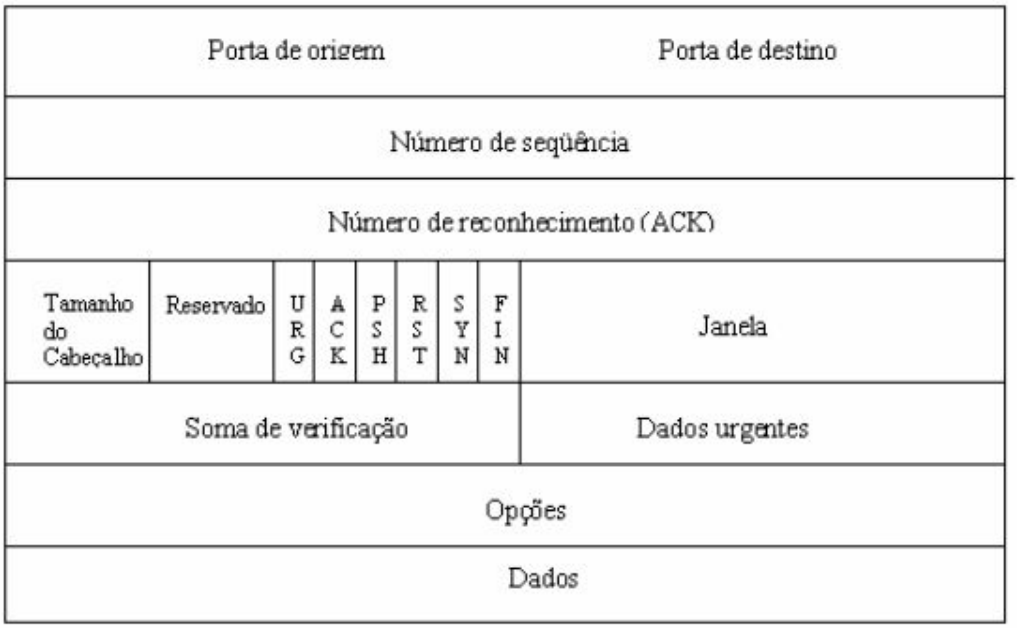

Figura 3 - Layout do cabeçalho de um segmento TCP.

Para o estabelecimento de uma conexão, o TCP utiliza a técnica do *3-way handshake*, ou seja, a máquina emissora envia um segmento de sincronização (SYN) para a receptora, caso a conexão seja aceita, o receptor envia para o emissor um segmento com o SYN e a confirmação (ACK), indicando que recebeu o segmento anterior sem problemas, e em seguida, como resposta, o emissor envia um ACK estabelecendo assim, a conexão.

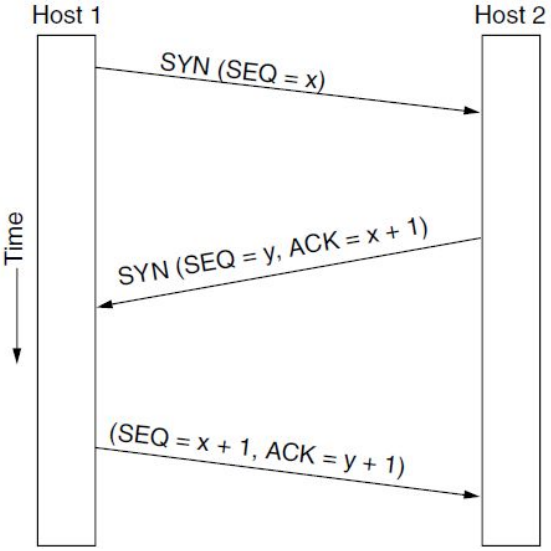

Figura 4 - Esquemático de estabelecimento de conexão entre 2 máquinas.

Durante a transmissão de um dado, é necessário que se tenha um controle para que não haja a sobrecarga de um receptor, já que o transmissor pode ser mais rápido e o receptor não ter capacidade de receber e processar os dados na mesma velocidade enviada. Para evitar esse problema, o TCP realiza um controle de fluxo entre os sistemas finais, que consiste em um campo do cabeçalho do segmento que informa a quantidade de *buffer* disponível no receptor, assim, é possível ter o controle da capacidade do destino final. Já o encerramento da conexão, qualquer uma das máquinas pode solicitar, apenas mandando o bit FIN ativado (bit que faz parte do cabeçalho do TCP - **Figura 3**).

É possível que ocorram problemas na rede de comunicação durante o processamento de dados e com isso, pacotes podem ser perdidos pelo trajeto e sistemas finais não ficam sabendo dessa perda, isso se não existisse a camada de transporte, que é fim-a-fim e exige uma confirmação de recebimento do pacote para o emissor (ACK). Para se obter o melhor desempenho na transferência confiável dos dados, o TCP utiliza o protocolo de janela deslizante, onde a janela é o número de segmentos que podem ser enviados, mas não reconhecidos (sem a confirmação).

O mecanismo de janela deslizante funciona da seguinte maneira: o emissor pode enviar vários pacotes dentro da janela sem receber um ACK, mas tem que iniciar um temporizador para cada um dos pacotes; O receptor tem que reconhecer cada pacote recebido, indicando o número de sequência do pacote que espera receber e por fim, o emissor desloca a janela para cada ACK recebido. O tamanho da janela é usado para a recuperação de erros, ordenação dos segmentos e também como mecanismo de controle de fluxo, além do uso da banda disponível de forma otimizada, pois aumenta o *throughput* (número de bytes transmitidos por segundo durante um intervalo de tempo).

Para garantir essas funcionalidades, o TCP utiliza retransmissões de pacotes que não foram entregues com sucesso, temporizadores para cada pacote enviado e campos no seu cabeçalho como número de sequência e número de reconhecimento. O TCP define o formato e a ordem das mensagens trocadas entre dois sistemas finais, bem como as ações realizadas na emissão e no recebimento de um pacote. Dentre os temporizadores, o mais importante é o de retransmissão, pois se o segmento é confirmado antes do tempo expirar, o temporizador é interrompido, caso contrário, o segmento é reenviado e o temporizador disparado novamente.

3. CONTROLE DE CONGESTIONAMENTO

Um dos problemas que uma rede como a Internet pode enfrentar é o de congestionamento, que ocorre quando um roteador recebe vários fluxos de redes diferentes e esse não possui uma quantidade de memória suficiente para armazenar os pacotes vindos dos vários fluxos de redes, levando a uma perda de pacotes por falta de memória disponível.

Para a identificação de um congestionamento, o TCP utiliza o estouro de temporizadores (*timeout*) e reconhecimentos duplicados. O estouro de temporizadores ocorre quando o reconhecimento de um pacote que foi enviado não chega ao receptor em um tempo pré determinado. E para a determinação do *timeout*, e assim, das retransmissões, é preciso fazer uma estimativa do tempo total de transmissão de ida e de volta (RTT, *round trip time*) em uma determinada conexão. O *timeout* não pode ser muito curto, pois ocorrerão retransmissões desnecessárias, sobrecarregando a rede com pacotes inúteis. Por outro lado, se o *timeout* for muito longo, o desempenho será afetado pelo simples fato de postergar a retransmissão do pacote perdido.

Como a estimativa do RTT muda com o tempo, devido às mudanças de rotas e padrões de tráfego, o TCP deve monitorar estas mudanças e modificar o tempo de *timeout* de forma apropriada. O cálculo para o RTT tem como base o tempo de envio de um segmento (outro temporizador) e o recebimento da confirmação do mesmo, se ocorrer a confirmação antes do tempo expirar, o TCP mede esse tempo (M) e atualiza a variável RTT, seguindo a fórmula:

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

sendo α um fator de suavização, que determina o peso do antigo valor. Geralmente $\alpha = \frac{7}{8}$.

Nas implementações iniciais, o valor usado era de $2 \times RTT$, porém usar uma constante não atendia os casos com variância maior. Com isso, foi proposto por Jacobson

(cientista da computação norte americano) calcular o *timeout* utilizando o desvio padrão dos tempos de chegada das confirmações, para aumentar a variância. Então chegou-se na fórmula:

$$D = ß \times D + (1 - ß) \times |RTT - M|$$

sendo D o desvio médio e $ß = \frac{3}{4}$. Atualmente, as implementações TCP utilizam:

$$Timeout = RTT + 4 \times D$$

onde o fator 4 é vantajoso pois, além de multiplicações por 4 serem realizadas em apenas um deslocamento, esse fator minimiza o número de timeouts e retransmissões desnecessárias, já que 1% de todos os pacotes chegam com atraso acima de quatro vezes o desvio padrão.

Os algoritmos do TCP que evitam o problema de congestionamento, forçam os sistemas finais (os quais detectam o tal problema) a diminuírem a velocidade com que enviam os pacotes para a rede durante os períodos em que a rede está congestionada. A detecção é feita quando os sistemas finais não recebem a confirmação dos pacotes enviados, com isso, há a retransmissão dos pacotes, dos quais não receberam a confirmação.

O controle feito pelo TCP para a taxa de transmissão de dados se deve ao uso de uma janela de congestionamento (CWND, *congestion window*), cujo o tamanho é o número de bytes máximo que o transmissor pode ter na rede em qualquer instante, além da janela de recepção (RWND, *receiver window*), que indica a quantidade de segmentos que o receptor pode receber sem confirmação. Ou seja, o CWND age como controlador de fluxo imposto pela rede e o RWND é o controlador de fluxo imposto pelo receptor.

4. TCP CUBIC

O algoritmo de controle de congestionamento Cubic utiliza uma função polinomial de terceira ordem para controlar a alteração da janela de congestionamento. O principal propósito do cubic é alcançar um crescimento escalável da janela de congestionamento mesmo em canais de alta velocidade. O cubic é uma evolução do TCP BIC sendo menos agressivo e possibilitando um melhor compartilhamento do canal de transmissão com os demais fluxos de dados.

A função cúbica utilizada tem como base o último evento de perda ocorrido, de modo a estipular qual será a nova taxa de transmissão, conforme definido na equação 1

$$W(t) = C\left(t - \sqrt{\frac{W_{máx}\beta}{C}}\right)^3 + W_{máx}$$

onde:

- *W(t)* é a nova taxa de transmissão
- *C* é uma constante escalar, utilizada para definir o peso da nova medição em relação ao maior valor de janela de transmissão mensurado;
- *t* é o tempo decorrido desde o último evento de perda;
- β é o fator de redução da largura utilizado após eventos de perda;
- W_{max} é o tamanho da janela antes do último evento de perda;

Empiricamente, os autores do Cubic recomendam a utilização de 0,4 e 0,2 para as constantes C e β, respectivamente, de modo a garantir que o Cubic apresente escalabilidade, convergência, equidade e possa coexistir com o TCP/Reno/New Reno.

a. Convergência Rápida

Quando ocorre uma perda de pacote, o Cubic reduz o tamanho de sua janela ao fator de β que é normalmente usado em 0,2. Um efeito indesejado em ajustá-lo a um valor menor que 0,5 é uma convergência lenta, porém ainda são estudadas várias formas de controlar essa taxa, possibilitando até que ele se adapte de acordo com o uso da rede.

b. CUBIC em Execução

Para melhorar a velocidade de convergência do Cubic, foi adicionada uma heurística no protocolo. Quando inicia um novo fluxo de pacotes na rede, os fluxos existentes terão que compartilhar a largura de banda com este, permitindo que os nós da rede compartilhem o meio de comunicação de forma justa. Seus testes demonstram essa convergência na **Figura 2**, onde são mostrados dois fluxos, onde um fluxo inicia enquanto outro já estava em execução, fazendo com que haja um compartilhamento mais justo do canal.

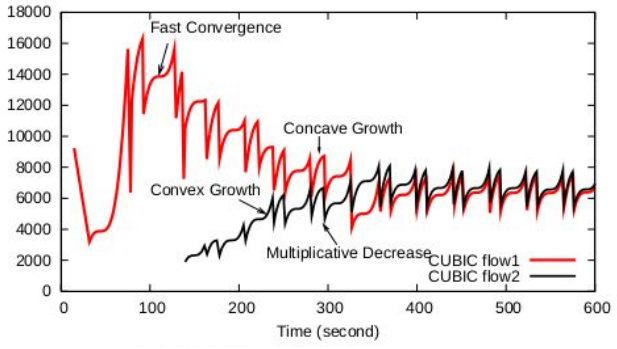

Figura 5: Gráfico da janela Cubic.

Através da **Figura 5** podemos observar que existem dois fluxos CUBIC e eles têm o mesmo RTT e o mesmo gargalo. Observe que as curvas têm platôs em torno de W_{max}, que é o tamanho da janela no momento do último evento de perda de pacotes. Observamos que dois fluxos usam todas as fases das funções CUBIC durante o tempo de execução e dois fluxos convergem para uma parcela justa dentro de 200 segundos.

5. PROCEDIMENTO EXPERIMENTAL

Os testes foram realizados utilizando a ferramenta de código aberto *NS* (Network Simulator) presente no linux e que é muito usado no meio acadêmico principalmente na parte de redes fornecendo suporte para simulações de TCP, roteamento e multicast em redes com e sem fio. Outra ferramenta foi usada para geração dos gráficos, a *NAM* (Network Animator).

Dessa forma, foi modelada a topologia e configuração da simulação que primeiramente constitui em 4 nós da rede, ligados de acordo com a topologia da **Figura 6** onde a largura de banda e latência dos nós é de:

- 2Gbps e 10ms entre 0 e 1

- 1,2Gbps e 100ms entre 1 e 2
- 2Gbps e 10ms entre 2 e 3

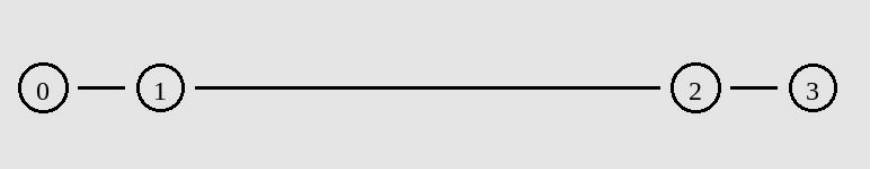

Figura 6: Topologia do experimento 1.

Como pode-se perceber, os nós das pontas possuem uma maior capacidade de transferência com seus nós mais próximos que entre os nós 1 e 2, estes que são usados para simular uma situação de *Bottleneck Link*, ou seja, se comportam como roteadores os quais controlam o fluxo de dados, caso a rede esteja fornecendo uma taxa de dados maior que o suportado pela capacidade da rede, ocorre um gargalo de rede.

Na simulação, foi gerado um tráfego entre os nós 0 e 3, e o tamanho da janela de congestionamento pode ser observada de acordo com a **Figura 7**, mostrando que a largura de banda utilizada irá crescer de acordo com a fatia disponível do canal de comunicação.

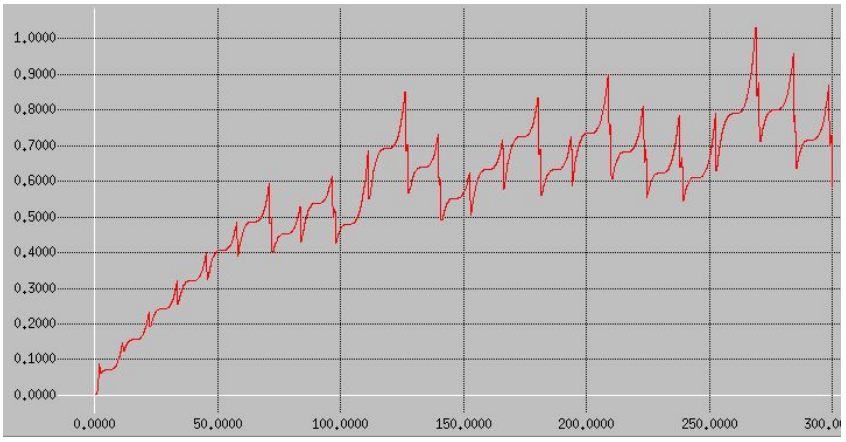

Figura 7: Convergência no experimento 1.

A convergência do uso da rede por cada canal também ocorre de forma dinâmica, ao passo que os nós da rede não são estáticos e podem se conectar a qualquer momento, como forma de exemplificar o comportamento do protocolo TCP Cubic, nessas situações, foram realizados testes onde foi possível observar o comportamento de outros nós da rede com a entrada de novos nós. A topologia utilizada no teste é a mostrada na **Figura 8** simulando a comunicação de uma rede que possui 6 nós ligados de acordo com as seguintes larguras de banda e latência:

- 2Gbps e 10ms entre 0 e 2
- 2Gbps e 10ms entre 1 e 2
- 1.5Gbps e 100ms entre 2 e 3
- 2Gbps e 10ms entre 3 e 4
- 2Gbps e 10ms entre 3 e 5

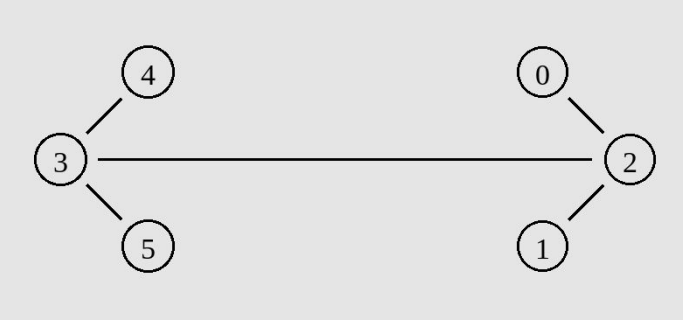

Figura 8: Topologia do experimento 2.

O fluxo de dados entre os nós inicia no momento 0.1 segundos entre os nós 0 e 5, terminando em 375 segundos, enquanto a transferência entre os nós 1 e 4 iniciam a transferência em 100 segundos e terminando ao final da simulação, em 500 segundos. O resultado pode ser visto na **Figura 9**, onde mostra os dois fluxos se adaptando às novas condições de tráfego na rede, possibilitando um melhor compartilhamento do meio de comunicação.

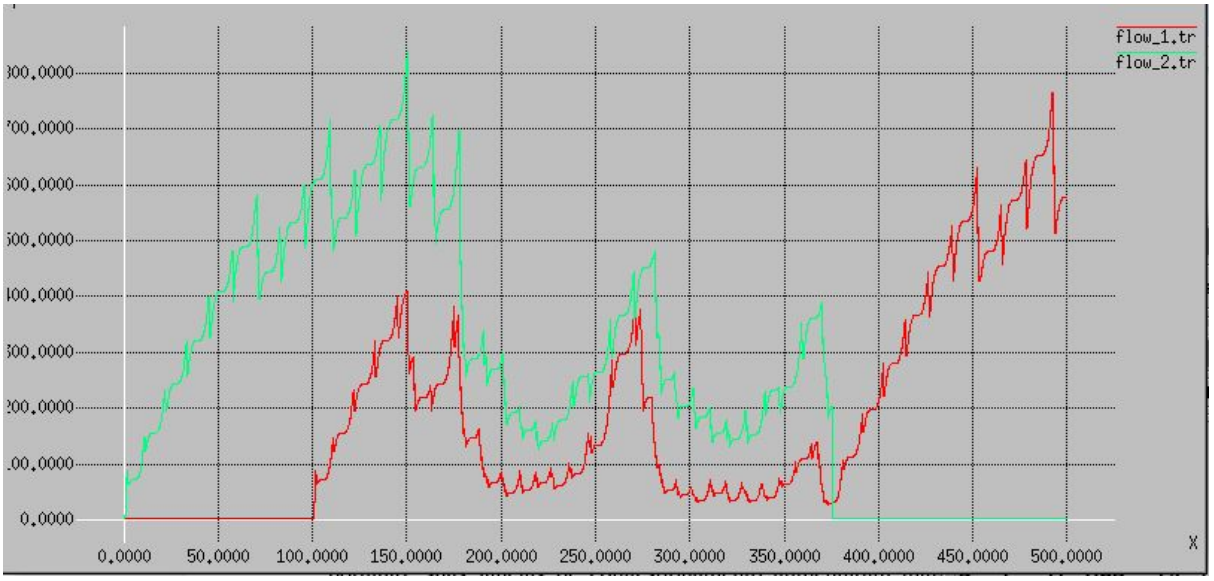

Figura 9: Janela de transferência do experimento 2.

6. ALGORITMO

Ao receber um ACK na prevenção de congestionamento, primeiro é necessário verificar se o protocolo está na região TCP ou não. Para isso pode-se analisar o tamanho da janela do TCP em termos do tempo decorrido t, encontrando assim o tamanho médio da janela de aumento aditivo e decréscimo multiplicativo (AIMD) com um fator aditivo α e um fator multiplicativo β para ser a seguinte função:

$$\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{2-\beta}{\beta}\frac{1}{p}}$$

Pela mesma análise, o tamanho médio da janela do TCP com $\alpha = 1$ e $\beta = 0.5$ é $\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}}$. Assim, para a Equação acima ser igual a do TCP, ela precisa ser igual à $\frac{3\beta}{2-\beta}$.

Se o TCP aumentar sua janela por fatores α e RTT, podemos obter a janela do tamanho do TCP em termos do tempo decorrido t da seguinte forma:

$$W_{tcp}(t) = W_{máx}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$$

Se $cwnd$ for menor que $W_{tcp}(t)$, então o protocolo está no modo TCP e $cwnd$ está definido para $W_{tcp}(t)$ em cada recepção de ACK.

Algorithm 1: Linux CUBIC algorithm (v2.2)

Initialization:
  $tcp\_friendliness \longleftarrow 1,\ \beta \longleftarrow 0.2$
  $fast\_convergence \longleftarrow 1,\ C \longleftarrow 0.4$
  $cubic\_reset()$
On each ACK:
begin
  if $dMin$ **then** $dMin \longleftarrow min(dMin, RTT)$
  else $dMin \longleftarrow RTT$
  if $cwnd \leq ssthresh$ **then** $cwnd \longleftarrow cwnd + 1$
  else
    $cnt \longleftarrow cubic\_update()$
    if $cwnd\_cnt > cnt$ **then**
      $cwnd \longleftarrow cwnd + 1, cwnd\_cnt \longleftarrow 0$
    else $cwnd\_cnt \longleftarrow cwnd\_cnt + 1$
end
Packet loss:
begin
  $epoch\_start \longleftarrow 0$
  if $cwnd < W_{last\_max}$ **and** $fast\_convergence$ **then**
    $W_{last\_max} \longleftarrow cwnd * \frac{(2-\beta)}{2}$ ..... **(3.7)**
  else $W_{last\_max} \longleftarrow cwnd$
  $ssthresh \longleftarrow cwnd \longleftarrow cwnd * (1-\beta)$ ..... **(3.6)**
end
Timeout:
begin
  $cubic\_reset()$
end
$cubic\_update()$: ..... **(3.2)**
begin
  $ack\_cnt \longleftarrow ack\_cnt + 1$
  if $epoch\_start \leq 0$ **then**
    $epoch\_start \longleftarrow tcp\_time\_stamp$
    if $cwnd < W_{last\_max}$ **then**
      $K \longleftarrow \sqrt[3]{\frac{W_{last\_max} - cwnd}{C}}$
      $origin\_point \longleftarrow W_{last\_max}$
    else
      $K \longleftarrow 0$
      $origin\_point \longleftarrow cwnd$
    $ack\_cnt \longleftarrow 1$
    $W_{tcp} \longleftarrow cwnd$
  $t \longleftarrow tcp\_time\_stamp + dMin - epoch\_start$
  $target \longleftarrow origin\_point + C(t-K)^3$
  if $target > cwnd$ **then** $cnt \longleftarrow \frac{cwnd}{target - cwnd}$ .. **(3.4,3.5)**
  else $cnt \longleftarrow 100 * cwnd$
  if $tcp\_friendliness$ **then** $cubic\_tcp\_friendliness()$
end
$cubic\_tcp\_friendliness()$: ..... **(3.3)**
begin
  $W_{tcp} \longleftarrow W_{tcp} + \frac{3\beta}{2-\beta} * \frac{ack\_cnt}{cwnd}$
  $ack\_cnt \longleftarrow 0$
  if $W_{tcp} > cwnd$ **then**
    $max\_cnt \longleftarrow \frac{cwnd}{W_{tcp} - cwnd}$
    if $cnt > max\_cnt$ **then** $cnt \longleftarrow max\_cnt$
end
$cubic\_reset()$:
begin
  $W_{last\_max} \longleftarrow 0,\ epoch\_start \longleftarrow 0,\ origin\_point \longleftarrow 0$
  $dMin \longleftarrow 0,\ W_{tcp} \longleftarrow 0,\ K \longleftarrow 0,\ ack\_cnt \longleftarrow 0$
end

Figura 10: Algoritmo CUBIC do linux.

7. CONCLUSÃO

O algoritmo de controle de colisão TCP Cubic se mostra uma excelente ferramenta no controle de colisões na rede, sua construção possibilita que seja escalável e adaptável em diversas configurações de rede, mesmo em redes onde outros protocolos trafegam. Suas principais vantagens são a rápida convergência e capacidade de compartilhar o canal de forma justa mesmo em redes com alta latência e grande largura de banda, pois seu mecanismo que controla o tráfego utilizado é baseado no tempo desde a última perda de pacotes, e não relacionado a latência, fazendo com que a largura de banda de um canal de longa distância seja melhor aproveitada.

REFERÊNCIAS

[1] **CUBIC TCP**, https://en.wikipedia.org/wiki/CUBIC_TCP

[2] **CUBIC: A New TCP-Friendly High-Speed TCP Variant**, https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf

[3] **CUBIC for Fast Long-Distance Networks,** https://tools.ietf.org/html/rfc8312