Data Center TCP com Histerese em Switches P4

Vítor L. G. Silva[1], José A. M. Nacif[1], Marcos A. M. Vieira[2]

[1]Universidade Federal de Viçosa (UFV)
Rodovia LMG 818, km 06, s/n, Florestal - MG, 35690-000

[2]Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brasil

{vitor.luis,jnacif}@ufv.br, mmvieira@dcc.ufmg.br

Resumo. *O congestionamento de pacotes em uma rede de computadores causa perda de desempenho no sistema. Esta proposta busca melhorar o desempenho das redes de computadores em centro de dados, especialmente em cenários de congestionamento. O protocolo padrão de controle de congestionamento em centro de dados é o Data Center TCP (DCTCP). Este artigo propõe duas abordagens: i) o uso da técnica de histerese nas filas dos switches para marcar os pacotes, indicando o congestionamento; ii) o retorno de pacotes ao emissor para reduzir a taxa de envio quando houver congestionamento. As abordagens foram implementadas em linguagem P4 utilizando switches P4 programáveis e validado juntamente com o DCTCP. Os resultados mostram que ambas soluções propostas melhoram a vazão da rede.*

1. Introdução

Desde o surgimento da Internet, sua utilização se tornou cada vez mais importante para a sociedade, o que implica em um aumento significativo no tráfego de dados gerado. Como consequência, novos problemas surgiram nesse cenário e os antigos, que sempre existiram, se agravaram. Um dos maiores problemas é o congestionamento de pacotes, que causa grande impacto no desempenho da rede. Quando se analisa escalas maiores, como a de um centro de dados, a situação se torna preocupante.

O protocolo padrão de controle de congestionamento em centro de dados é o Data Center TCP (DCTCP). Devido ao seu bom desempenho, muitas empresas já o adicionaram em suas estruturas de redes, como Google e Microsoft [Alizadeh et al. 2010]. Assim, construir um algoritmo compatível com esse protocolo é de extrema importância para que o congestionamento seja tratado da forma mais rápida e eficiente possível, sem comprometer outras características da rede, como a vazão.

As principais contribuições desse trabalho são: i) desenvolvimento de uma nova política para marcar os pacotes congestionados nas filas dentro dos switches. Ela consiste no uso de dois limiares (histerese) para marcar se um pacote é de um fluxo congestionado; ii) desenvolvimento de um sistema para o switch notificar diretamente o emissor quando houver congestionamento, agilizando o tratamento dele. As soluções são práticas pois elas são capazes de serem sintetizadas e executadas em switches P4; iii) avaliação do benefício das contribuições anteriores executadas com o protocolo de transporte DCTCP.

Neste artigo foram desenvolvidas, na linguagem P4 [Bosshart et al. 2014], duas abordagens para tratar o congestionamento de pacotes em um switch P4 com arquitetura portátil (*Portable Switch Architecture* – PSA). O algoritmo de congestionamento do

protocolo TCP adotado foi o *Data Center* TCP (DCTCP). Essa escolha foi feita porque ele é mais reativo à pacotes com *Round Trip Time* (RTT) pequeno [Alizadeh et al. 2011], utiliza o campo Notificação Explícita de Congestionamento (*Explicit Congestion Notification* – ECN) para sinalizar a presença ou a falta de congestionamento e é o protocolo estado-da-arte para controle de congestionamento em centros de dados.

O uso de P4 para solucionar problemas de congestionamento em switches não é uma novidade em si. Como, por exemplo, em [Geng et al. 2019], os autores construíram um algoritmo que prevê congestionamento e gera pacotes de realimentação que informam a necessidade da redução na taxa de envio. Eles utilizam apenas um limiar para conferir se ocorreu congestionamento na rede e, ao invés de usarem o DCTCP para a redução, operam com um algoritmo próprio para cumprir essa tarefa e pressupõe que a camada de enlace seja compatível com RoCE (*Remote Direct Memory Access* – RDMA) sobre *Converged Ethernet*. Como esta abordagem utiliza hardware especializado, a abrangência deste trabalho é menor do que a proposta neste artigo que utiliza DCTCP.

O restante deste artigo está estruturado da seguinte forma: na seção 2 é descrito o funcionamento do protocolo DCTCP. Em seguida, na seção 3 são mencionados os trabalhos que compõem o estado da arte e têm relação com o nosso. Na seção 4, são apresentadas a arquitetura proposta e o seu funcionamento. Depois, na seção 5 são descritos os experimentos e a análise dos resultados obtidos. Finalmente, na seção 6 discutem-se os trabalhos futuros e a conclusão.

2. O protocolo DCTCP e o mecanismo ECN

O DCTCP pressupõe que os roteadores detectem congestionamento e marquem os bits de indicação de congestionamento nos cabeçalhos dos pacotes. Neste caso, os roteadores marcam o campo Notificação Explícita de Congestionamento (*Explicit Congestion Notification* – ECN) para sinalizar se há congestionamento. A indicação de congestionamento é comunicada de volta para a outra ponta através dos campo ECN nos pacotes de confirmação. Um roteador compatível com ECN pode marcar no cabeçalho IP em vez de descartar um pacote para sinalizar um congestionamento iminente.

A RFC 3168 [Floyd et al. 2001] especifica a incorporação de ECN ao TCP e IP. O ECN usa os dois bits menos significativos (mais à direita) do campo Classe de Tráfego no cabeçalho IPv4 ou IPv6 para codificar quatro casos de código diferentes, que são mostrados na tabela 1. Quando o congestionamento é detectado, o campo ECN é marcado com o valor igual à três (11_2).

Bits	Significado
00	Transporte não compatível com ECN
10	Transporte capaz de ECN
01	Transporte capaz de ECN
11	Congestionamento encontrado (CE).

Tabela 1. O campo ECN no IP.

Para tratar congestionamento de pacotes, o DCTCP [Alizadeh et al. 2010] reage de acordo com a proporção do congestionamento e não com a presença dele, como funciona com o TCP padrão. O DCTCP utiliza o campo ECN para este propósito. Isso sig-

nifica que a taxa de envio não será reduzida necessariamente pela metade quando ocorrer o congestionamento, mas sim proporcionalmente à quantidade de pacotes que possuem o campo ECN marcado. A tabela 2 demonstra dois exemplos de cenários diferentes. No primeiro, de dez pacotes que circulam pela rede, oito (representa 80%) estão com o campo ECN marcado. Nessa situação, o TCP irá diminuir a janela em 50%, enquanto o DCTCP reduzirá em 80%*50%=40%. Já no segundo caso, apenas um pacote na rede está marcado, o que representa 10%. Mesmo com essa baixa presença de congestionamento, o TCP ainda reduzirá a janela em 50%, enquanto a redução do DCTCP será de apenas 10%*50%=5%.

Pacotes com ECN	TCP	DCTCP
0 1 1 1 1 1 1 0 1 1	Corta a janela em 50%	Corta a janela em 40%
1 0 0 0 0 0 0 0 0 0	Corta a janela em 50%	Corta a janela em 5%

Tabela 2. Exemplo de funcionamento do TCP e DCTCP.

Basicamente, o DCTCP funciona assim: um valor α é calculado para que a janela não seja sempre reduzida pela metade, como é feito no TCP. O cálculo, como mostra a equação 1, utiliza um parâmetro fixo *g* (com valor entre 0 e 1), a fração de pacotes que foram marcados *F*, e o valor de α calculado no tempo anterior. Os pacotes são marcados de acordo com o tamanho da fila do switch. Assim, caso ele seja maior do que um valor *K*, o pacote será marcado. Dito isso, a redução da janela é feita com base na equação 2. Essa janela de congestionamento (*J*) é calculada com base no valor anterior dela. A janela é utilizada para informar qual deve ser o tempo de envio dos pacotes. Assim, ela será reduzida de acordo com a intensidade do congestionamento. Caso ele seja muito intenso, a redução irá acontecer como no TCP, ou seja, pela metade (α igual a um).

$$\alpha_t \leftarrow (1-g)\alpha_{t-1} + gF \tag{1}$$

$$J \leftarrow (1-\frac{\alpha}{2})J_{anterior} \tag{2}$$

É importante ressaltar que o valor de *K* não é arbitrário, mas sim determinado. Ele é calculado como mostra a equação 3 [Alizadeh et al. 2011], onde *C* é a capacidade da rede (medida em pacotes/segundo) e *RTT* é o tempo de ida e volta (*Round Trip Time*).

$$K \leftarrow 0.17 * C * RTT \tag{3}$$

Como é possível observar, o DCTCP possui uma alta tolerância à estouro, pois a origem do pacote consegue reagir antes que ele seja descartado. Além disso, ele também tem baixa latência, já que o *delay* da fila é mais baixo e a ocupação do *buffer* é menor [Alizadeh et al. 2011].

3. Trabalhos Relacionados

Nesta seção, é discutido os trabalhos relacionados desse projeto ou que inspiraram a construção da abordagem desenvolvida pelo grupo.

Um trabalhos relacionado importante é o P4qcn [Geng et al. 2019]. Nele, foi criado um par de algoritmos capazes de tratar o congestionamento na rede. Segundo o artigo, cada algoritmo tem uma função específica que é: i) controle e ii) redução. Na fase de controle, os autores também utilizaram a linguagem P4 em conjunto com a biblioteca fornecida pela PSA para construir o algoritmo deles. Assim, eles comparam o tamanho da fila com um único valor (K) e, caso ele seja maior, o pacote será clonado e posteriormente marcado no estágio *Ingress* do *pipeline* da arquitetura. Após esse processo, o pacote que foi clonado e marcado é chamado de pacote de *feedback*. Ele é importante para o algoritmo de redução, que funciona da seguinte forma: existem três fases, que são i) redução da taxa de envio, ii) aumento rápido da taxa de envio e iii) aumento gradual da taxa de envio. Caso o *sender* receba um pacote de *feedback*, então a fase i) entra em ação. Quando não houver mais pacotes de *feedback* na rede, o estágio ii) ocorre em uma quantidade específica de ciclos para que aconteça uma rápida recuperação da taxa que foi reduzida. Se acontecer algum congestionamento nesse estágio, o algoritmo volta para a fase i). Se não, então a fase iii) entra em execução após o término de todos os ciclos. Nessa etapa, o aumento da taxa de envio acontece de acordo com a largura de banda disponível. Caso um pacote de *feedback* seja gerado, então a fase i) volta em execução. A figura 1 exemplifica melhor o funcionamento desse algoritmo.

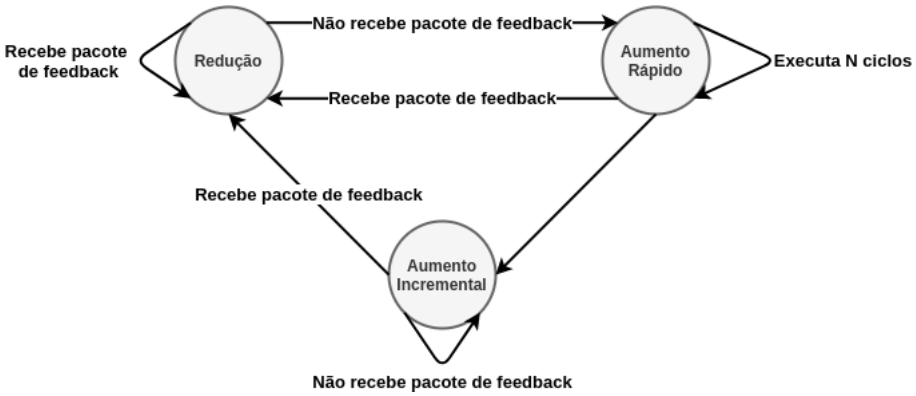

Figura 1. Funcionamento do algoritmo de redução do P4qcn.

Inicialmente, Ramakrishnan *et al.* [Ramakrishnan and Jain 1990] propuseram que os elementos de rede no meio do caminho poderiam marcar um dos bits dos pacotes caso eles pertencessem a um fluxo potencialmente congestionado. A indicação de congestionamento é comunicada de volta para o usuário através da confirmação. Depois desse inovador trabalho, a comunidade desenvolveu o padrão ECN. Os switches atuais tem suporte para ECN e permitem configurar o valor do limiar K da altura da fila para marcar os bits ECN. Porém, os switches legados não-programáveis não permitem configurar novas políticas para marcar os bits ECN, como a política de histerese proposta neste trabalho. O ECN é um dos exemplos de mecanismo de Gerenciamento de Fila Ativa (*Active Queue Management* – AQM). O outro mecanismo de AQM é descartar pacotes para indicar congestionamento. Como exemplo de AQM, temos o *Random Early Detection* (RED) [Floyd and Jacobson 1993], que descarta pacotes preventivamente antes que o buffer fique completamente cheio através da utilização de modelos probabilísticos.

O TCP PCC Vivace [Dong et al. 2018] e o TCP BBR [Cardwell et al. 2016] são duas das versões mais recentes do TCP. Em Vivace, foi proposto um novo protocolo de

controle de taxa baseado em otimização online (convexa) com aprendizado de máquina. Porém, ele não foi projetado para o cenário de centro de dados, que requer latência muito pequena e sua função de utilidade não consegue se adaptar nessa velocidade. Além disso, o TCP Vivace e TCP BBR não utilizam ECN, ou seja, não são compatíveis com o DCTCP. Para maiores detalhes sobre o histórico de modificações dos algoritmos de controle de congestionamento do TCP, é indicado o *survey* completo de Afanasyev *et al.* [Afanasyev et al. 2010].

4. Metodologia

Como mencionado anteriormente, duas abordagens foram desenvolvidas nesse artigo. A primeira delas foi o uso de dois limiares na marcação dos pacotes, ao invés de apenas um, em conjunto com a utilização do DCTCP. Já a segunda utiliza do retorno de pacotes com cabeçalho TCP para tratar o congestionamento na rede. Assim como a primeira, ela também utiliza o DCTCP. Vale pontuar que escolhemos a linguagem P4 pelo fato dela ser alvo de estudos recentes e aplicável na programação direta do plano de dados de switches [Kfoury et al. 2021].

4.1. Histerese

O objetivo desta abordagem é analisar o efeito do uso de dois limiares (histerese) na etapa que verifica se um pacote deve ser marcado ou não. Em muitos artigos, como em [Alizadeh et al. 2011] e [Geng et al. 2019], o tamanho da fila do switch é comparado apenas com um limiar. Assim, inserimos a histerese para avaliar os benefícios ou malefícios dela no cenário de congestionamento de pacotes. A ideia de usar dois limiares é melhorar o controle de marcação de pacotes, aumentando o rendimento e diminuindo o atraso, conforme mencionado por [Malhotra et al. 2009].

Histerese é um conceito amplamente utilizado no campo de eletromagnetismo e ele dita sobre a capacidade de um sistema conseguir manter suas propriedades na ocorrência de algum estímulo externo [Bertotti 1998]. No nosso caso, o sistema é todo o processo de envio de pacote, a propriedade que deve ser mantida é a marcação ou não dos pacotes e o estimulo externo é a ocorrência do congestionamento. A ideia é construir um intervalo de segurança para marcar (ou não) os pacotes, como mostra a figura 2. Para definir o valor do primeiro limiar, chamado de limite superior (L_S), optamos por utilizar o mesmo cálculo feito pelo DCTCP para definir K, como mostra a equação 3. No caso do segundo limiar, nomeado de limite inferior (L_I), realizamos alguns testes e, a partir deles, definimos que seu valor é igual ao piso de L_S menos 15% de L_S. Então, por exemplo, se L_S for igual a 10, temos que L_I é igual a 8.

O algoritmo funciona da seguinte maneira: quando o tamanho instantâneo da fila for maior do que o limite inferior (L_I), é verificado se o pacote anterior foi marcado ou não. Em caso afirmativo, o pacote que se encontra em verificação também deve ser marcado. Caso contrário, ele não é marcado. Enquanto o tamanho da fila for maior do que o limite superior (L_S), todos os pacotes são marcados. No momento em que ele for menor do que L_S, então analisa-se a marcação do pacote anterior (enquanto o tamanho da fila for maior que L_I). Por exemplo, na figura 2, quando o tamanho da fila é maior do que L_I no cenário (1), nenhum pacote é marcado. No momento (2), os pacotes passam a ser marcados, pois o tamanho da fila é maior do que L_S. Em (3), os pacotes continuam

sendo marcados, pois os anteriores também foram e o tamanho da fila ainda é maior do que *L_I*. Já em (4), o tamanho da fila é menor do que o limite inferior, então os pacotes não são mais marcados. Assim, quando o tamanho for maior do que *L_I* em (5), eles também não serão marcados. Ainda em (6), os pacotes continuam não sendo marcados, pois o tamanho da fila não ultrapassou o limite superior. O algoritmo 1 apresenta o pseudo código para o algoritmo de histerese explicado nesse parágrafo.

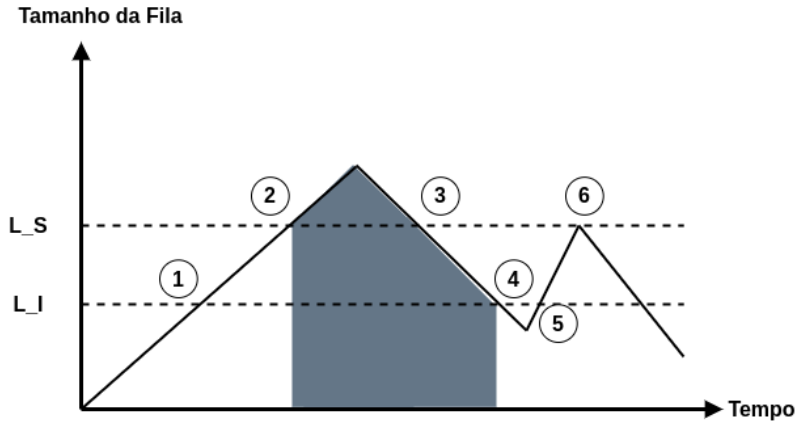

Figura 2. Ilustração simples do funcionamento da histerese. A linha representa o tamanho instantâneo da fila. A área rachurada indica quando os bits ECN do pacotes são marcados: 1) quando acima do limite superior (*L_S*) (cenário 2-3); 2) quando entre limite superior *L_S* e limite inferior *L_I* e o pacote anterior tinha sido marcado (cenário 3-4).

Algoritmo 1 Pseudo código para o algoritmo de Histerese

```
1:  função HISTERESE(tamanho da fila)
2:      se tamanho da fila ≥ L_S então
3:          estado ← CONGESTIONADO;                ▷ estado é variável global
4:          devolve MARCAR;
5:      senão se (tamanho da fila ≥ L_I) E (estado == CONGESTIONADO) então
6:          devolve MARCAR;
7:      senão
8:          estado ← NÃO_CONGESTIONADO;            ▷ estado é variável global
9:          devolve NÃO_MARCAR;
10:     fim se
11: fim função
```

Para ser mais específico, usamos um registrador de 1 bit para informar se a rede está congestionada ou não. Em outras palavras, a variável do pseudocódigo "status", que informa se o pacote anterior foi marcado ou não, é esse registrador. Esses registradores podem ser usados na linguagem P4 através da palavra reservada *register* e definidos com um tamanho de *n* bits.

É interessante ressaltar que o tamanho instantâneo da fila só pode ser obtido em um estágio específico do *pipeline*, que é o *Egress*. Isso acontece devido ao funcionamento padrão da biblioteca fornecida pela arquitetura utilizada, a PSA. Assim, é nele que os pacotes são marcados. Em sequência, o *checksum* é atualizado através de uma função

fornecida pela própria biblioteca. Isso é feito para que a integridade do pacote seja mantida. Por fim, ele é enviado para o destino original para que a redução seja feita pelo DCTCP caso necessária.

4.2. Sistema de retorno de pacotes

Esta abordagem possui duas etapas: i) controle e ii) redução. Na primeira parte, encontra-se todo o código em P4 e foi para ela que o foco do grupo foi direcionado. A segunda parte utiliza o DCTCP [Alizadeh et al. 2010].

Como mencionado em 4.1, existe um *pipeline* fornecido pela PSA. Ele é composto por seis estágios que podem ser programados em P4 e dois que têm função fixa. Na figura 3, aqueles que têm a função fixa estão coloridos. Os estágios programáveis mais importantes para essa abordagem são: *Ingress Parser*, *Ingress* e *Egress*.

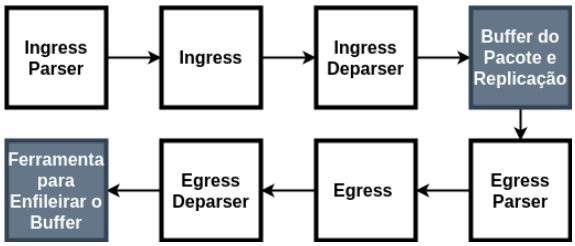

Figura 3. Todos estágios do *pipeline* da arquitetura.

No bloco do *Ingress Parser*, chega o pacote que foi enviado por um dos elementos da rede. Assim, é nessa etapa que determina-se a estrutura do pacote, ou seja, qual protocolo de transporte (TCP ou UDP, por exemplo) e o de internet (IPv4 ou IPv6) foram utilizados para definí-lo. Vale ressaltar que o *Ingress Parser* pode ser customizável para aceitar pacotes que possuem estruturas que também foram personalizadas. Após a decodificação realizada, existem duas situações possíveis: i) houve congestionamento na rede e o algoritmo de controle precisa entrar em ação ou ii) nada precisa ser feito.

Assim, o funcionamento da primeira etapa acontece da seguinte forma: o pacote entra no *Ingress Parser* para ser decodificado. É nessa fase que determina-se a estrutura do pacote. Após a decodificação feita pelo *parser*, existem três cenários possíveis: i) o pacote não foi clonado e nem marcado, ii) o pacote foi apenas clonado e iii) o pacote foi clonado e marcado.

Na primeira situação, o pacote ainda não passou pela verificação da ocorrência de congestionamento na rede. Desse modo, quando o pacote entrar na fila, essa verificação será feita. Caso o tamanho da fila seja maior do que o limiar que indica se o pacote deve ou não ser marcado (L_S), ele será marcado. Para realizar a marcação, primeiro o pacote deve ser clonado. Após realizar essa operação, uma cópia do pacote é feita e enviada para o *Egress*, enquanto o original segue o fluxo normalmente.

Caso o tamanho da fila seja menor que esse limiar, porém maior do que o limiar utilizado para estabilizar o algoritmo (L_I), então precisa-se conferir se o pacote anterior a esse foi clonado ou não. Se ele tiver sido clonado, então o pacote que encontra-se em verificação também será. Se não, o pacote em análise não será clonado. Por fim, se o tamanho da fila for menor do que os dois limiares, nenhuma clonagem do pacote será

feita. Vale ressaltar que um pacote que já foi clonado ou marcado, não será clonado novamente. Isso é feito para não sobrecarregar a rede com dados desnecessários.

No caso das outras duas situações, o tamanho da fila não importa, pois o congestionamento já aconteceu. Assim, as comparações são feitas na fase de *Egress*. Caso o pacote seja clonado, as informações de origem e destino dele devem ser trocadas, como o endereço e a porta. Isso é feito para que o pacote possa voltar para a entidade que o enviou, informando-a da ocorrência do congestionamento. Além dessas informações, o campo *Time To Live* (TTL) deve ser restaurado para seu valor padrão, pois só assim que o pacote clonado irá conseguir chegar em seu novo destino. Após a troca, o pacote é recirculado para o começo do *pipeline*, ou seja, ele é enviado para o *Ingress Parser*. Na situação em que o pacote já se encontra marcado, então ele já passou pelo processo de recirculação, de modo que o fluxo pode ser seguido normalmente.

A marcação do pacote acontece na fase de *Ingress*. Para todo pacote que passa nessa etapa, verifica-se se ele é um pacote retornado. Em caso afirmativo, um congestionamento aconteceu na rede, fazendo com que seja preciso alterar o campo *Explicit Congestion Notification* (ECN) para o valor três (11_2). Assim, através desse pacote marcado, é possível informar à entidade sobre a ocorrência do congestionamento e que ela deve reduzir a taxa de envio dos pacotes. A figura 4 ilustra o funcionamento geral do *pipeline*.

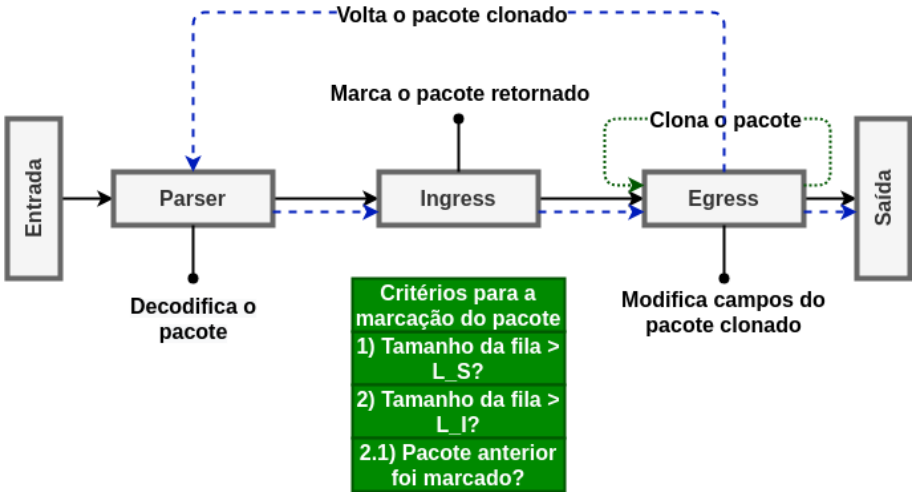

Figura 4. *Pipeline* da primeira etapa do algoritmo com histerese.

A parte de controle descrita diz respeito ao funcionamento do retorno de pacotes com histerese. No caso de apenas um limiar, não existe o L_I e nem a análise de se o pacote anterior foi marcado ou não. Nessa situação, a única regra para a marcação dos pacotes é se o tamanho da fila for maior do que L_S.

O funcionamento da etapa da redução acontece da mesma forma para os dois casos, de modo que o DCTCP é encarregado de reduzir a taxa de envio. Como mencionado, ele reage de acordo com a proporção do congestionamento. Isso significa que a taxa de transmissão é diminuída de acordo com o tamanho do congestionamento. Por conta dessa abordagem, existe alta vazão de pacotes e baixa variância ao se comparar com o TCP [Alizadeh et al. 2011].

5. Experimentos

Nesta seção, descrevemos os testes realizados na primeira (Histerese) e segunda (Sistema com retorno de pacotes) abordagens desenvolvidas pelo grupo, juntamente com os resultados obtidos. Ambas abordagens foram implementadas em um switch P4 com modelo comportamental (*Behavioral Model v2* – bmv2) [Bas 2020].

5.1. Resultados para Histerese

Para a realização dos experimentos, utilizamos uma máquina real com as seguintes configurações: sistema operacional Ubuntu 18.04.5 LTS (64-bit), 8 GB de memória RAM e processador Intel® Core™ i5-5200U CPU @ 2,20 GHz × 4.

Foi utilizado o emulador de redes Mininet [Kaur et al. 2014] para simular o ambiente de rede com os hospedeiros e os switches. Além disso, para gerarmos o tráfego na rede simulada, utilizamos a ferramente Iperf2 [Tirumala 1999]. Por fim, na criação das topologias presentes nos experimentos, nos inspiramos em uma topologia haltere (*dumbbell*), que é uma versão da topologia utilizada para avaliar o TCP CUBIC [Ha et al. 2008]. Nela, dois switches estão localizados no gargalo entre dois pontos finais, onde cada ponto final consiste em dois servidores Linux juntamente com dois geradores de tráfego para congestionar a rede. Entretanto, devido a limitações da máquina utilizada, não foi possível usar dois switches, de modo que alteramos outras características da rede para compensar essa falta.

Nos testes da primeira abordagem, criamos uma topologia com um total de seis hospedeiros e um switch. A figura 5 ilustra a topologia criada. Para executar os testes na primeira topologia, dividimos em dois cenários: i) hospedeiros com algoritmo DCTCP e um limiar (limite superior), que é a abordagem padrão do DCTCP, e ii) hospedeiros com algoritmo DCTCP e dois limiares (limite superior e inferior). Para facilitar, chamamos hospedeiros que possuem conexões TCP e UDP de hospedeiro híbrido. Também usamos nomenclaturas para definir melhor a intensidade do congestionamento, como mostra a tabela 3.

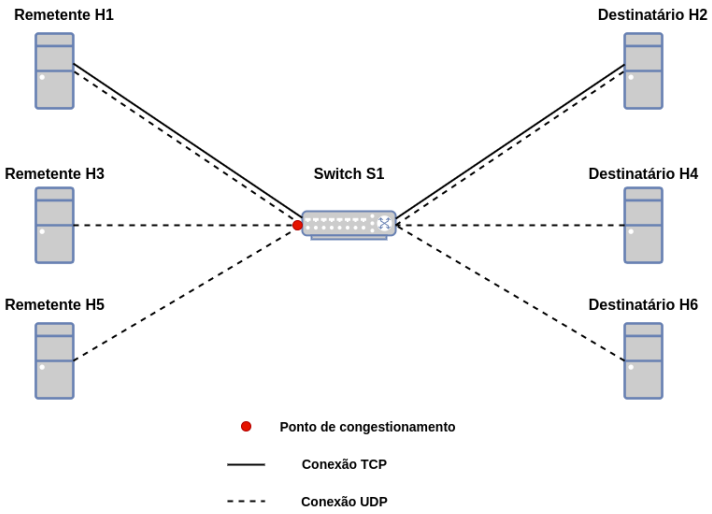

Figura 5. Topologia utilizada no primeiro experimento.

Nomenclatura	Significado
Leve	Fluxo UDP não interferiu no envio de pacotes do fluxo TCP do mesmo hospedeiro
Moderado	Fluxo UDP interferiu pouco no envio de pacotes TCP do mesmo hospedeiro, mas ainda permitiu envio simultâneo de pacotes
Intenso	Fluxo UDP interferiu muito no envio de pacotes TCP do mesmo hospedeiro, não permitindo envio de pacotes simultaneamente
Muito intenso	Fluxo UDP interferiu muito no envio de pacotes TCP do mesmo hospedeiro e de pacotes UDP de outros hospedeiros

Tabela 3. Nomenclatura para intensidade do congestionamento.

Assim, a configuração da rede foi montada da seguinte forma: cada uma das conexões entre os hospedeiros e o switch tem uma vazão máxima de 1000 Mbps e o atraso de 3000 µs com cada pacote de tamanho 1500 bytes, ou seja, o valor do limite superior é 42 e do limite inferior é 36, como mostra a explicação do cálculo na seção 4.1. No caso de teste (A), criamos duas conexões entre H1 e H2 (uma TCP e outra UDP). A conexão TCP foi mantida por 50 segundos, enquanto a UDP executou por 25 segundos e teve a largura de banda limitada em 2 Mbps. Também criamos outras duas conexões, uma UDP entre H3 com H4 e outra entre H5 com H6, ambas executaram por 25 segundos com a largura de banda limitada em 2 Mbps. Nessa situação, o congestionamento criado foi leve e impactou pouco nos resultados obtidos. No caso (B), as conexões entre H1 e H2 executaram por 50 segundos e a conexão UDP foi limitada em 10 Mbps. As outras duas conexões (H3 com H4 e H5 com H6) executaram por 25 segundos e com o limite máximo de 5 Mbps cada. A situação nesse caso de teste foi a seguinte: durante os primeiros 25 segundos, houve congestionamento intenso, onde o hospedeiro H1 nem conseguia enviar pacotes no fluxo TCP. Após esse período de tempo, o congestionamento foi moderado, de modo que o H1 conseguiu enviar pacotes tanto TCP quanto UDP. No caso (C), a conexão TCP executou por 50 segundos e a UDP do H1 apenas por 25 segundos e teve a largura de banda limitada em 10 Mbps, enquanto as conexões de H3 com H4 e H5 com H6 executaram por 50 segundos e foram limitadas em 5 Mbps cada. Nesse caso de teste, o congestionamento foi muito intenso durante os 25 segundos. Ao final desse intervalo de tempo, o congestionamento alterou para intenso. A tabela 4 mostra o resumo dos cenários criados.

Caso de Teste	Conexão TCP	Conexão UDP H1	Outros UDP	Congestionamento
A	50 s	25 s e 2 Mbps	25 s e 2 Mbps	Leve todo o período
B	50 s	50 s e 10 Mbps	25 s e 5 Mbps	Intenso, depois moderado
C	50 s	25 s e 10 Mbps	50 s e 5 Mbps	Muito intenso, depois intenso

Tabela 4. Resumo dos casos de teste A, B e C.

Como é possível observar na tabela 5, podemos constatar que a abordagem desse artigo superou o funcionamento padrão do DCTCP no quesito vazão dos hospedeiros TCP em todos os casos de teste, principalmente no caso (B). Nesse caso, o algoritmo conseguiu controlar melhor o congestionamento, aumentando consideravelmente a vazão do TCP, enquanto diminuiu a do UDP encontrada no hospedeiro híbrido. Com isso, podemos perceber que o uso da histerese favorece os casos em que o congestionamento varia e é mais intenso. Par melhor ilustrar o ganho obtido, mostramos na figura 6 o aumento (em

porcentagem) da vazão atingido pela nossa abordagem em comparação com o DCTCP padrão.

Algoritmo	Caso de Teste	Vazão TCP H1 $\pm \sigma$
1 limiar	A	3,42 Mbps $\pm$ 0,1234
1 limiar	B	0,3848 Mbps $\pm$ 0,060
1 limiar	C	0,1324 Mbps $\pm$ 0,018
2 limiares	A	3,53 Mbps $\pm$ 0,1863
2 limiares	B	0,5220 Mbps $\pm$ 0,0687
2 limiares	C	0,1598 Mbps $\pm$ 0,0651

Tabela 5. Vazão na primeira topologia.

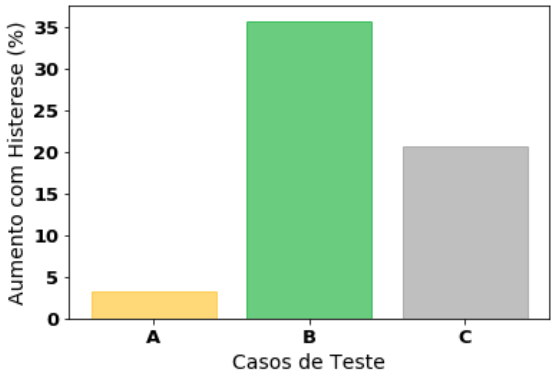

Figura 6. Comparativo entre os resultados da abordagem de Histerese.

5.2. Resultados para sistema com retorno de pacotes

Para realizar os testes da segunda abordagem, mantivemos a máquina real utilizada nos testes anteriores e também as configurações da vazão máxima e atraso das conexões entre os hospedeiros e o switch, ou seja, os valores dos limiares foram mantidos. Assim, criamos uma terceira topologia, como mostra a figura 7. Ela é bastante similar à topologia da figura 5, porém com mais conexões TCP entre os hospedeiros. Isso foi feito para aproveitar melhor o retorno dos pacotes quando acontece um congestionamento, pois é devido à esse retorno que a redução da taxa de envio dos pacotes acontece de forma mais rápida. Além disso, como já mencionado, essa abordagem trabalha com a modificação de campos do cabeçalho TCP, de modo que ter mais pacotes UDP na rede diminui a eficiência do algoritmo. Caso esse aumento de conexões não fosse feito, seria muito difícil de retornar os pacotes TCP, já que a maioria deles seriam UDP, o que aumenta a chance deles serem os causadores do congestionamento. Assim, como não existe algoritmo de controle em hospedeiros UDP, a nossa abordagem seria inutilizada.

Para realizar os casos de teste, desenvolvemos três cenários: i) abordagem padrão do DCTCP (um limiar e sem retorno de pacotes); ii) abordagem com retorno de pacotes; iii) abordagem com retorno de pacotes e histerese. Assim, os casos criados foram os seguintes: (D) criamos três conexões TCP, uma H1 com H2, outra H3 com H4 e por fim H5 com H6. Todas elas enviaram pacotes por 50 segundos. Também produzimos três conexões UDP (H1 com H2, H3 com H4 e H5 com H6) que executaram por 25 segundos cada e a largura de banda individual foi limitada até 1 Mbps. Isso ocasionou um congestionamento moderado durante os primeiros 25 segundos e, após esse intervalo, alterou para

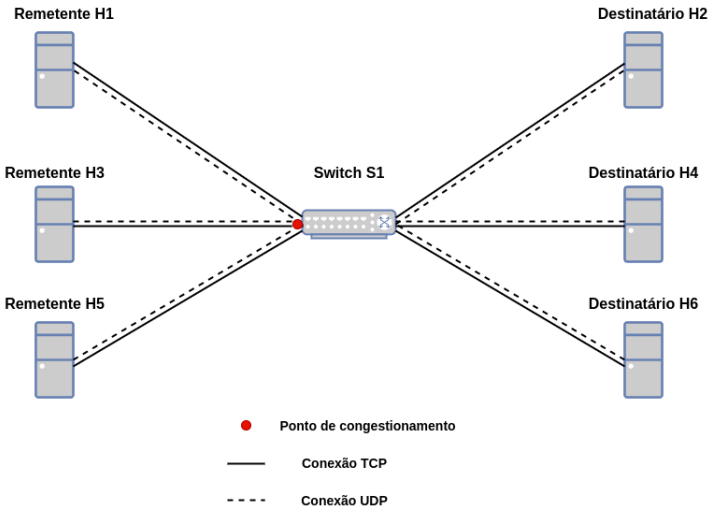

Figura 7. Topologia utilizada no segundo experimento.

congestionamento leve. Em (E), as conexões criadas foram as mesmas do caso de teste anterior. Exceto pelas seguintes mudanças nas conexões UDP: elas executaram por 50 segundos e a largura de banda foi limitada até 5 Mbps. Como as conexões consumiram a banda durante todo o período de execução do teste, o congestionamento causado foi intenso durante todo o tempo. A tabela 6 mostra o resumo dos cenários criados.

Caso de Teste	Conexões TCP	Conexões UDP	Congestionamento
D	50 s	25 s e 1 Mbps	Moderado, depois leve
E	50 s	50 s e 5 Mbps	Intenso todo o período

Tabela 6. Resumo dos casos de teste D e E.

Os resultados obtidos podem ser observados na tabela 7. A partir dela, podemos fazer duas comparações no caso de teste (D): i) DCTCP padrão com retorno e ii) DCTCP padrão com retorno utilizando o conceito de histerese. Em i), as vazões em H1 foram iguais, em H3 o retorno de pacotes teve menor vazão, mas em H5 foi maior. Já em ii), o retorno com histerese proporcionou a maior vazão do cenário e a menor também, porém a diferença entre ela e a outra menor (encontrada no DCTCP padrão) é muito baixa.

Já em (E), os resultados obtidos pela nossa abordagem foram melhores. No comparativo entre retorno de pacotes e DCTCP padrão, a diferença entre as vazões de H1 foi de apenas 0,0001 Mbps, enquanto o aumento das vazões para os outros dois hospedeiros híbridos foram mais significativos, como mostra a figura 8. Com relação ao retorno de pacotes com histerese, a diferença no caso do H1 foi maior, porém ele também aumentou a vazão para os outros hospedeiros híbridos, como é possível observar na figura 9. Isso mostra que utilizar o retorno de pacotes TCP com ou sem histerese funciona bem em situações de congestionamento intenso e variável em relação à intensidade.

6. Conclusão

Este trabalho apresentou o desenvolvimento de uma nova política para marcar pacotes que indicam congestionamento no meio da rede. Ela foi implementada em linguagem P4 e é executada internamente nos switches. A nova política utiliza do conceito de histerese para

Algoritmo	Caso de Teste	Vazão TCP H1 $\pm \sigma$	Vazão TCP H3 $\pm \sigma$	Vazão TCP H5 $\pm \sigma$
1 limiar	D	1,56 Mbps $\pm$ 0,0264	1,56 Mbps $\pm$ 0,0964	1.33 Mbps $\pm$ 0,0763
1 limiar	E	0,1596 Mbps $\pm$ 0.0250	0,1019 Mbps $\pm$ 0,0084	0,0748 Mbps $\pm$ 0,0084
Retorno	D	1,56 Mbps $\pm$ 0,1386	1,41 Mbps $\pm$ 0,0802	1,41 Mbps $\pm$ 0,0650
Retorno	E	0,1474 Mbps $\pm$ 0,0173	0,1126 Mbps $\pm$ 0,0250	0,0821 Mbps $\pm$ 0,0091
Retorno com histerese	D	1,68 Mbps $\pm$ 0,01527,	1,37 Mbps $\pm$ 0,0435	1,30 Mbps $\pm$ 0,0251
Retorno com histerese	E	0,1590 Mbps $\pm$ 0,0084	0,1364 Mbps $\pm$ 0,0336	0,0779 Mbps $\pm$ 0,0139

Tabela 7. Vazão na segunda topologia.

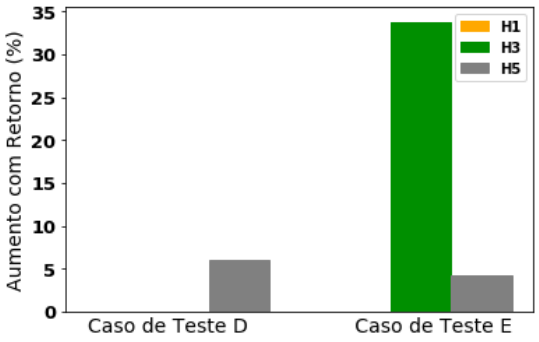

Figura 8. Comparativo entre DCTCP padrão e retorno de pacotes.

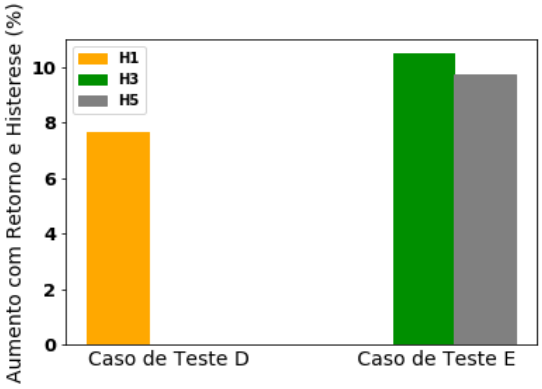

Figura 9. Comparativo entre DCTCP padrão e retorno de pacotes com histerese.

marcar os pacotes com o uso de dois limiares. Ela é compatível com o protocolo DCTCP e foi avaliada em conjunto com ele, mostrando melhoras de desempenho no quesito vazão. A solução proposta é prática, pois funciona nos equipamentos físicos e é vislumbrado que a contribuição poderá ser, futuramente, integrada nos centros de dados.

Além disso, também desenvolvemos um sistema capaz de notificar ao emissor a ocorrência de congestionamento na rede, fazendo com que a taxa de envio seja reduzida mais rápido, o que evita a sobrecarga na rede. Esse sistema funciona bem tanto de forma independente (apenas um limair) tanto com a política criada nesse artigo. Além disso, ele é compatível com o DCTCP, apresentando resultados promissores para aumento da vazão dos hospedeiros.

Sobre trabalhos futuros, pretendemos desenvolver um modelo matemático para indicar qual deve ser os valores dos dois limiares para atingir uma configuração ótima. Além disso, queremos mudar os limiares de acordo com o tipo do fluxo (qualidade de serviço), como feito em [Chen et al. 2019]. Também desejamos estudar, avaliar e testar o sistema com placas de redes mais rápidas e modernas, que operem em torno de 100 Gbps.

Agradecimentos

O presente trabalho foi realizado com o apoio das agências de fomento à pesquisa CNPq, CAPES, FAPESP e FAPEMIG. Agradecemos pelo suporte oferecido.

Referências

Afanasyev, A., Tilley, N., Reiher, P., and Kleinrock, L. (2010). Host-to-host congestion

control for tcp. *IEEE Communications Surveys & Tutorials*, 12(3):304–342.

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 40(4):63–74.

Alizadeh, M., Javanmard, A., and Prabhakar, B. (2011). Analysis of dctcp: Stability, convergence, and fairness. *SIGMETRICS Perform. Eval. Rev.*, 39(1):73–84.

Bas, A. (2020). The bmv2 simple switch target.

Bertotti, G. (1998). *Hysteresis in Magnetism: For Physicists, Materials Scientists, and Engineers*. Academic Press.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. (2016). Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53.

Chen, X., Feibish, S., Koral, Y., Rexford, J., Rottenstreich, O., Monetti, S., and Wang, T.-Y. (2019). Fine-grained queue measurement in the data plane. pages 15–29.

Dong, M., Meng, T., Zarchy, D., Arslan, E., Gilad, Y., Godfrey, B., and Schapira, M. (2018). PCC vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA. USENIX Association.

Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413.

Floyd, S., Ramakrishnan, D. K. K., and Black, D. L. (2001). The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168.

Geng, J., Yan, J., and Zhang, Y. (2019). P4qcn: Congestion control using p4-capable device in data center networks. *Electronics*, 8(3):280.

Ha, S., Rhee, I., and Xu, L. (2008). Cubic: a new tcp-friendly high-speed tcp variant. *Operating Systems Review*, 42:64–74.

Kaur, K., Singh, J., and Ghumman, N. S. (2014). Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)*, pages 139–42.

Kfoury, E. F., Crichigno, J., and Bou-Harb, E. (2021). An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends.

Malhotra, R., Mandjes, M., Scheinhardt, W., and van den Berg, H. (2009). A feedback fluid queue with two congestion control thresholds. *Mathematical methods of operations research*, 70(1):149–169.

Ramakrishnan, K. K. and Jain, R. (1990). A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. Comput. Syst.*, 8(2):158–181.

Tirumala, A. (1999). Iperf: The tcp/udp bandwidth measurement tool. *http://dast. nlanr. net/Projects/Iperf/*.