Roteador em hardware com políticas de controle de congestionamento reconfiguráveis

Gerferson R. Coelho, José Augusto M. Nacif
Universidade Federal de Viçosa
Florestal, Brasil
gerferson.coelho@ufv.br

Racyus D. G. Pacífico, Marcos A. M. Vieira
Universidade Federal Minas Gerais
Belo Horizonte, Brasil
{racyus,mmvieira}@dcc.ufmg.br

***Resumo*—a demanda por redes de computadores cada vez mais rápidas alavancada pelo uso em massa de dispositivos conectados que requisitam taxas de transferência alta e baixa latência levantam questões como o controle de congestionamento em roteadores. Neste trabalho propomos um roteador implementado em hardware com políticas de controle de congestionamento reconfiguráveis que possibilita a utilização de políticas de congestionamento em tempo de execução sem a necessidade de recompilar ou reiniciar o roteador quando o usuário altera, em tempo de execução, a forma como o congestionamento deve ser tratado. Nosso trabalho foi implementado na plataforma NetFPGA 1Gb e experimentado em um ambiente real. Nossos resultados mostram que nossa arquitetura permite alterar as políticas de congestionamento em tempo de execução controlando o congestionamento de forma dinâmica sem utilizar os nós folhas da rede.**

I. Introdução

Com os recentes avanços tecnológicos na última década, o número de dispositivos conectados aumentaram em níveis nunca vistos antes. Devido a esse aumento problemas referente ao congestionamento tornaram-se frequentes em redes de computadores. Um dos principais protocolos estudados para controle de congestionamento em redes IP foi o protocolo TCP (*Transmission Control Protocol*). No protocolo TCP o congestionamento é controlado nos nós folhas da rede reduzindo o tamanho da janela deslizante dos nós folhas após o congestionamento ocorrer [2].

O principal problema em esperar o congestionamento acontecer para depois controlar está na perda da qualidade de serviço. No protocolo TCP quando os pacotes são perdidos ocorre a retransmissão dos pacotes diminuindo a vazão da rede e aumentando o atraso. Devido as várias limitações do TCP para controlar o congestionamento da rede foram propostos mecanismos mais eficientes com o objetivo de antecipar o congestionamento [12].

Um tipo de mecanismo proposto para prever a ocorrência do congestionamento baseado no controle do tamanho da fila chama-se Gerenciamento ativo da fila (*Active Queue Management* - AQM). AQM consiste em um conjunto de mecanismos utilizados para prever e controlar a ocorrência do congestionamento na rede devido a fila dos elementos de rede estarem totalmente cheias. Neste tipo de mecanismo o controle do congestionamento é realizado monitorando o tamanho médio da fila dos roteadores.

Neste artigo estendemos o projeto de um roteador implementado em hardware para possibilitar a configuração de políticas de controle de congestionamento, sem a necessidade de recompilar ou reiniciar o roteador quando o usuário reconfigura uma nova política.As modificações realizadas foram desenvolvidas sobre a plataforma NetFPGA 1G, que possibilita um alto poder de processamento de pacotes da rede. Os testes foram realizados em um ambiente real.

As principais contribuições deste trabalho são: (i) configurar políticas de congestionamento no roteador sem a necessidade de recompilar ou reiniciar o roteador; (ii) detectar e controlar o congestionamento da rede utilizando aplicações no plano de controle do roteador; (iii) estender o protótipo de um roteador em hardware para utilizar políticas de controle de congestionamento;

Este trabalho tem a seguinte organização: na seção 2 é apresentado uma visão geral sobre o gerenciamento ativo da fila, tipos de congestionamento e mecanismos para controle de congestionamento. Na seção 3 é descrito detalhes de implementação do roteador em hardware com políticas de controle de congestionamento. Na seção 4 são apresentados os resultados dos experimentos que foram realizados em um ambiente real. Na seção 5 são descritos os trabalhos relacionados referentes a outro. Por fim, na seção 6 apresentamos as conclusões e trabalhos futuros.

II. Gerenciamento Ativo da Fila

AQM é um mecanismo implementado em roteadores que detecta a ocorrência do congestionamento de forma antecipada e informa aos nós origens da rede que ajustem a taxa de transmissão de dados. O compromisso deste mecanismo é manter a vazão da rede alta com um tamanho de fila pequeno nos equipamentos da rede [14].

O tamanho da fila de roteadores é um dos principais fatores que cooperam para o surgimento do congestionamento na rede. O tamanho da fila foi projetado para absorver rajadas de dados de pacotes em um prazo curto de tempo, ao invés de ficarem continuamente recebendo dados de modo a preencher a fila totalmente [16]. Aumentar o tamanho da fila não surge como uma solução, pois não existe um tamanho ideal que a fila não seja totalmente preenchida. Além disso, aumentar o tamanho da fila acarreta problemas de configuração no início da comunicação com os transmissores e também contribui para o aumento no custo de fabricação do equipamento [4].

A. *Tipos de congestionamento*

O preenchimento total da fila em roteadores acarreta o descarte de pacotes prejudicando a qualidade do serviço de aplicações. Em redes de computadores três tipos de congestionamento referente ao tamanho da fila são estudados e pesquisados na área: *Incast*, *Queue Buildup* e *Buffer pressure* [1].

1) Incast: É um padrão de comunicação (muitos-para-um), onde vários nós transmissores enviam tráfego para um único nó receptor ao mesmo tempo [6]. Este padrão é comum em aplicações que demandam taxas de transmissão altas de um fluxo constante. O *Incast* é visto em *datacenters* que possuem aplicações que utilizam operações de *MapReduce* e consultas de dados. O tráfego gerado por esses tipos de aplicações são responsáveis por sobrecarregar a rede provocando o preenchimento total da fila dos roteadores. Quando o preenchimento total da fila acontece o tempo de resposta entre nós transmissões aumenta, prejudicando gravemente a qualidade do desempenho da aplicação [4]. Na Figura 1 é apresentado um exemplo do congestionamento *Incast* na porta do comutador. Vários fluxos pequenos de nós transmissores são enviados em uma única fila de saída referente a um nó transmissor ocasionando o preenchimento total da fila e provocando o congestionamento.

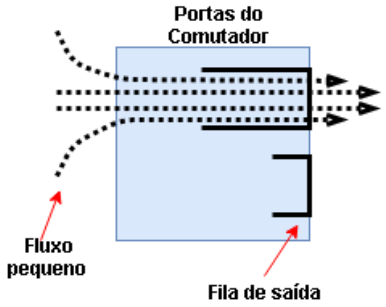

Figura 1: Exemplo congestionamento *Incast*.

2) Queue Buildup: É um tipo de congestionamento caracterizado pelo acúmulo de dados na fila. Esse acúmulo de dados ocorre quando fluxos TCP de longa duração proporcionam o aumento do comprimento da fila até gerar um gargalo na rede e o descarte de pacotes. Neste tipo de congestionamento mesmo que nenhum pacote seja perdido, os fluxos curtos são afetados tendo uma latência alta. Isso ocorre, pois os fluxos pequenos estão na fila de saída atrás de fluxos elefante [1]. Na figura 2, é demonstrado o exemplo do congestionamento *queue build*. Neste exemplo os fluxos longos consumem mais espaço na fila prejudicando os fluxos pequenos.

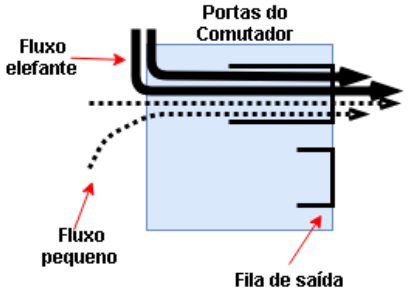

Figura 2: Exemplo congestionamento *Queue buildup*.

3) Buffer Pressure: Em redes de *datacenters* existe uma combinação (mistura) de fluxos elefantes e curtos. No congestionamento *buffer pressure* fluxos elefantes de portas diferentes do comutador afetam o processamento de fluxos pequenos ocasionando a perda destes fluxos. Fluxos pequenos são descartados porque as filas das portas do comutador utilizam um conjunto de memórias compartilhadas. Fluxos elefantes (gulosos) demandam uma quantidade de recurso maior das memórias compartilhadas não restando espaço na memória para absorver fluxos pequenos em rajadas. A perda de fluxos pequenos é baseada no número de fluxos elefantes que chegam no comutador. Os efeitos deste tipo de congestionamento são similares ao *Incast* com a diferença de não existir fluxos sincronizados. Na figura 3, é apresentado um exemplo do congestionamento *buffer pressure*. Os fluxos elefantes são definidos na figura como setas uniforme e fluxos pequenos como setas tracejadas. O objetivo é demonstrar que fluxos elefantes consomem uma quantidade maior de recursos da fila em relação aos fluxos pequenos. [1].

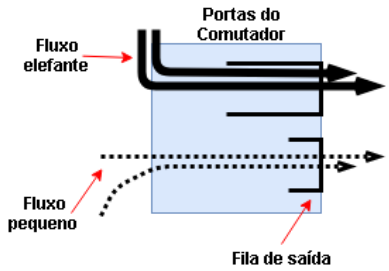

Figura 3: Exemplo congestionamento *buffer pressure*.

B. *Mecanismos para controle de congestionamento*

Nesta subseção são apresentados oito tipos de mecanismos para controle de congestionamento.

Drop tail utiliza a política FIFO (*First-In, First-Out*) para armazenar os pacotes na fila. Neste mecanismo os pacotes são armazenados na fila a medida que chegam, e retirados de acordo com a ordem de chegada. A principal desvantagem deste mecanismo está em não identificar a ocorrência do congestionamento de forma antecipada. Quando o congestionamento ocorre a fila está cheia causando o descarte de pacotes. Descartar pacotes sem políticas de controle do tamanho da fila reduz a vazão da rede e aumenta o atraso prejudicando a qualidade do serviço. O mecanismo *Drop tail* é implementado nos elementos de rede.

ERD (Early Random Drop) utiliza uma política de descarte de pacotes baseado na probabilidade fixa do tamanho da fila. Neste mecanismo o descarte de pacotes ocorre quando o tamanho da fila excede a probabilidade fixa calculada (limiar máximo). Neste mecanismo é possível prever a ocorrência do congestionamento através do comportamento da fila. A principal desvantagem do ERD está em encontrar uma constante

de probabilidade fixa para descartar pacotes em uma rede com tráfego instável composta por fluxos elefantes e curtos [6].

Random Drop o descarte de pacotes é realizado aleatoriamente monitorando todas as filas de entrada. Neste mecanismo o tráfego gerado pelos usuários recebe uma prioridade. Usuários que geram uma maior quantidade de tráfego na rede têm probabilidade menor de terem os pacotes descartados. A forma como os pacotes são eliminados inviabiliza a implementação desse mecanismo em equipamentos reais [15].

DECbit Gateway [11] tem o objetivo de dividir a responsabilidade do controle de congestionamento entre os roteadores e nós folhas da rede. Neste mecanismo cada roteador monitora a carga do tráfego da rede e notifica explicitamente os nós folhas quando o congestionamento está prestes a ocorrer. A notificação do *DECbit* é implementada definindo um bit de congestionamento binário no campo tipo de serviço (ToS) do cabeçalho IPv4 dos pacotes que trafegam nos roteadores. Os nós destinos ao receberem o pacote indicando que existe congestionamento na rede copiam o bit de congestionamento para o pacote de confirmação (ACK) e enviam o pacote ACK para os nós origens para que ajustem a taxa de transmissão para evitar o congestionamento.

RED (Random Early Detection) [5] foi proposto por Sally Floyd e Van Jacobson no início dos anos 90. Este mecanismo monitora o tamanho médio da fila utilizando a média móvel exponencial. No RED os pacotes que chegam são marcados ou descartados de maneira probabilística. A probabilidade de um pacote ser marcado aumenta quando a média estimada do tamanho da fila aumenta. Neste mecanismo são utilizados dois limiares (mínimo e máximo) de controle do tamanho da fila. Quando o tamanho da fila médio é menor que o limiar mínimo, nenhum pacote é marcado e o pacote que chega é armazenado na fila. Se o tamanho da fila médio é maior que o limiar máximo, todo pacote que chega é marcado indicando que existe congestionamento na rede. Outra condição do RED consiste em verificar se o tamanho médio da fila está entre o limiar mínimo e máximo. Quando essa situação ocorre o pacote que chega é marcado com probabilidade $p_{\mathrm{a}} \in [0, max_{\mathrm{p}}]$, onde p_{a} é uma função do tamanho da fila médio. Esta condição é utiliza como uma forma de prevenção do congestionamento. As principais vantagens do RED são evitar a sincronização global do TCP e não ter problemas de comportamento com fluxos elefantes e pequenos em rajadas.

WRED (Weighted Random Early Detection) é uma variação do RED que utiliza pesos para definir prioridade de quais pacotes serão descartados. No WRED os pacotes são descartados de maneira probabilística utilizando o algoritmo do RED. Neste mecanismo pacotes com prioridade menor definidos pelo usuário ao chegarem na fila são os primeiros pacotes a serem descartados. No mecanismo WRED a ponderação possibilita o desenvolvimento de políticas de qualidade de serviço utilizando um mecanismo de gerenciamento ativo da fila.

DRR (Déficit Round Robin) [13] é um mecanismo caracterizado por ser justo em termos da vazão e ter complexidade de tempo $O(1)$ para processar um pacote. Neste mecanismo o armazenamento do pacote na fila é feito de forma justa utilizando um algoritmo *Round Robin* modificado. O algoritmo *Round Robin* modificado consiste em compensar as próximas rodadas quando não é possível enviar um pacote de tamanho grande de uma fila em uma fatia de tempo.

FRED (Fair Random Early Detection) [8] é uma variação do RED que têm o compromisso de permitir o descarte probabilístico de pacotes do algoritmo RED de forma justa. Neste mecanismo a vazão de uma conexão TCP é determinado pelo tamanho relativo da fila considerando os atrasos da largura de banda. No FRED os pacotes são enviados apenas quando fila tem espaço para armazená-los.

III. Implementação

A implementação do roteador em hardware é composta de dois componentes: plano de dados e ferramentas no espaço do usuário. O plano de dados contém módulos em *hardware* que pertencem a biblioteca padrão da NetFPGA (módulos na cor cinza) e módulos específicos do roteador em hardware (módulos na cor laranja). Nós estendemos os módulos da biblioteca padrão da NetFPGA (figura 4) para permitir a utilização de políticas de controle de congestionamento. Os módulos modificados foram: *Input Arbiter*, *Output Port Lookup* e *small fifo* (filas). Nossa contribuição em relação a *"Roteador SDN em hardware independente de protocolo com análise, casamento e ações dinâmicas"* [10] foi modificarmos o projeto para permitir o monitoramento do tamanho da fila e implementar políticas de controle de congestionamento utilizando o processador eBPF do projeto. Com as informações armazenadas no metadado do processador, políticas de controle de congestionamento podem ser configuradas no roteador sem realizar nenhuma modificação no plano de dados do hardware.

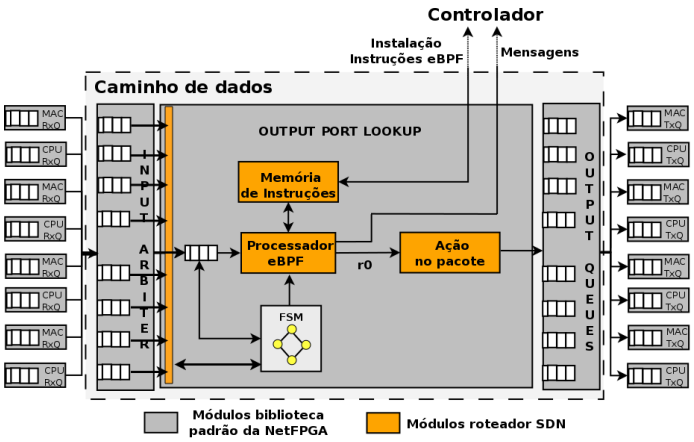

Figura 4: Caminho de dados do roteador implementado na NetFPGA 1G.

A. Random Early Detection (RED)

O mecanismo RED para controle de congestionamento foi utilizado como política de controle de congestionamento para validar as modificações realizadas no roteador. Escolhemos o RED por utilizar a média móvel exponencial no tamanho da

fila para definir o momento que o pacote deve ser descartado. A implementação do RED é composta de três passos. O primeiro passo do RED é calcular a média móvel das filas usando uma média anterior calculada A média móvel **Avg** é calculada utilizando as funções (1), (2).

$$Avg = wq * pkt\_fila + (1.0 - wq) * Avg' \quad (1)$$

$$Avg = (pkt\_fila + (Avg' << 3) - Avg') >> 3 \quad (2)$$

O comprimento médio da fila e calculado desta maneira em vez de se usar um tamanho (fixo) porque devido a natureza inconstante do tráfego da Internet que mistura um conjunto de fluxos longos e curtos, as filas podem ficar cheias muito rapidamente e depois ficar vazia novamente em pouco espaço de tempo. Logo a utilização da média móvel consegue capturar com mais precisão a noção de congestionamento aumentando o desempenho e melhorando a precisão. Na formula (1) temos **wq** que representa um peso da fila definido pelo usuário para para detecção antecipada de congestionamento variando entre $0 < wq < 1$, mas comumente se utiliza valores menores, cerca de **0.002**, pkt_fila é o tamanho da fila quando uma medição de amostra é feita. Na maioria das implementações de software, o comprimento da fila é medido toda vez que um novo pacote chega ao gateway. No hardware, seu cálculo se dá em um intervalo de amostragem fixo, em nossa implementação foi utilizado 0.5 segundos como o tempo de amostragem fixo. Por fim temos a variável Avg'[1] que representa a média móvel calculada anteriormente.

Em segundo lugar, para execução do algoritmo RED e necessário definir dois limites que representam os limiares do comprimento das filas, são eles *MinThreshold* e o *MaxThreshold* e alguns parâmetros como **MAXpb** que é o máximo valor que *Pb* pode assumir.

```
if (Avg <= MinThreshold)
{
    enfileirar o pacote;
}
if (MinThreshold < Avg < MaxThreshold)
{
   calcula probabilidade ( *pacote )
   {
      contador = contador + 1;
      pb = Avg - minThreshold;
      pb = pb * MAXpb;
      pb = pb / (maxThreshold - minThreshold);
      pa = pb / (1 - ( contador * pb ));
   }

   descarta o pacote com probabilidade (pa)
}
if (MaxThreshold <= Avg)
{
    descarta todos os pacotes que chegam;
}
```

[1]Avg' representa a Avg calculada anteriormente

Se o comprimento médio da fila **Avg** for menor ou igual ao limite inferior, o pacote é colocado instantaneamente na fila. Se o comprimento médio da fila **Avg** for maior ou igual ao limite superior, todos os pacotes que chegam serão descartados pois a rede está completamente congestionada. Se o tamanho médio da fila **Avg** estiver entre os dois limites, será calculado uma probabilidade **Pa** acerca do pacote recém chegado, o qual poderá ou não ser o descartado.

O cálculo da probabilidade **Pa**, se dá por meio da equação (6), mas antes temos o cálculo da variável **Pb** aqui representado por estas três equações (3), (4), (5) para um melhor entendimento.**Pb** leva em conta os valores de máximo e mínimo da fila e o **MAXpb**, O valor de **Pb** sempre e atrelado ao valor da média da fila (Avg) e ao tempo do último pacote descartado. Especificamente, o Pb é uma variável que realiza o controle de quantos pacotes recém-chegados foram enfileirados(não descartados).

Pb aumenta gradativamente a medida que o contador aumenta, tornando cada vez mais provável ter um descarte de pacotes a medida que o tempo desde o último descarte aumenta. Isto faz com que os descartes espaçados sejam relativamente menos prováveis do que descartes menos espaçados [5].

$$pb = Avg - minThreshold \quad (3)$$

$$pb = pb * MAXpb \quad (4)$$

$$pb = \frac{pb}{(maxThreshold - minThreshold)} \quad (5)$$

Com os cálculos de Pb efetuados podemos dar continuidade ao cálculo da probabilidade **Pa**(6), para de fato termos a tomada de decisão descartar o pacote ou enfileira-lo. Para termos está tomada de decisão um número e gerado aleatoriamente em uma semente pré-definida e logo a seguir comparado ao valor de **Pa** caso o valor de Pa seja maior que o valor do número aleatório dividido pelo tamanho da semente o pacote e descartado com a probabilidade *Pa*, caso contrário, o pacote segue o seu ciclo para ser enfileirado.

$$Pa = \frac{Pb}{(1 - contador * Pb)} \quad (6)$$

B. Plano de dados eBPF

Como mencionado as bibliotecas padrões da NetFPGA precisaram de modificações para atender aos requisitos do nosso trabalho. O *input arbiter* recebe os pacotes da rede (via interface MAC) ou do barramento PCI (via CPU) no formato de palavras de 72 bits sendo 8 bits para controle e 64 bits para dados, estas palavras são armazenadas em 8 filas internas **(small fifo)** sendo 4 PCI e 4 MAC. Modificações nestas 8 filas internas se tornou necessário para adicionar registradores responsáveis por contabilizar a quantidade de pacotes que chegam nesta fila bem como variáveis de controle para saber quando a fila estava vazia ou cheia. Pensando um pouco mais a frente estas informações serão utilizadas para a execução do RED e cálculo das médias móveis.

No passo seguinte, as palavras são retiradas das filas utilizando o algoritmo round robin, e as encaminha para o módulo *output port lookup*. O módulo *output port lookup* tem a tarefa de definir qual porta o pacote será enviado baseado nas propriedades do pacote. Devido a necessidade de armazenamento de informações em registradores de hardware, neste módulo foi acrescentado mecanismos para salvar estas informações e redirecionar as demais para o módulo do processador eBPF.

C. Processador eBPF

O roteador em hardware com políticas de controle de congestionamento reconfiguráveis expande o caminho de dados padrão da NetFPGA em três módulos, ação do pacote, processador ebpf e memória de instrução. O processador eBPF é responsável por realizar a análise, execução e ações de acordo com as instruções eBPF geradas a partir do código C ou P4 criados pelo usuário no espaço do usuário. Antes de iniciar o funcionamento do roteador em hardware, o usuário deve carregar as instruções eBPF na memória de instruções para definir qual será o comportamento do plano de dados do roteador bem como suas políticas de controle de congestionamento. O módulo ação no pacote realiza de fato todas as ações para com o pacote sendo ela encaminhamento (empilha) ou descarte do pacote de acordo com o valor armazenado no registrador r0 após o processamento do algoritmo RED (instruções eBPF). A possibilidade de reconfiguração nos possibilita em tempo de execução e com inatividade zero alternar entre várias políticas podendo ser aplicadas a diferentes perfis de tráfego. Para a instalação de instruções e comunicação com o processador foi utilizado o plano de controle implementado utilizando a API soquete que será especificado na seção mais abaixo.

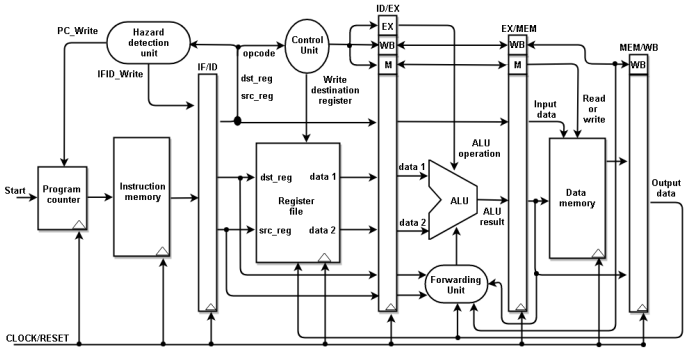

Figura 5: Caminho de dados e de controle do processador eBPF.

Na Figura 5 temos uma de nossas contribuições para a comunidade científica, o processador eBPF multiciclo juntamente com um pipeline. Em [10] foi utilizado o processamento monociclo o que nos leva a ter um tempo de vazão **5x** mais lento do que com o pipeline. Para o nosso trabalho o processador eBPF em [10], foi melhorado e expandido para fazer uso da técnica de implementação de processadores que permite a sobreposição temporal das diversas fases de execução das instruções eBPF, mais conhecido como processador pipeline. Com o uso do pipeline em nosso trabalho tivemos uma taxa de vazão 5x maior em relação ao processador monociclo podendo processar o pacote 5 vezes mais rápido e aumentando o paralelismo podendo executar várias tarefas simultaneamente que utilizam recursos diferentes.

D. Hardware

Neste trabalho, foi implementado o roteador com políticas de controle de congestionamento reconfigurável no hardware NetFPGA 1G. NetFPGA é uma plataforma aberta que combina um FPGA (field-programmable gate array, ou arranjo de portas programável em campo, especificamente para nosso trabalho foi utilizado uma FPGA Vertex-II Pro 50 com aproximadamente 53.000 células lógicas, uma SRAM de 4 KiB com 2^{19} linhas e ciclo de relógio de 8 ns (125 MHz)) memória (SRAM e DRAM) e processadores de sinais numa placa com quatro portas Ethernet de 1 Gbps. A NetFPGA é particularmente útil como uma alternativa de hardware flexível e de baixo custo para avaliação de protótipos de pesquisa.

E. Controlador SDN

O controlador SDN (Software Defined Networking) e o ponto de comunicação entre o espaço de usuário e o plano de dados. Neste trabalho utilizamos um controlador implementado em python que utiliza API sockets para se comunicar em ambas as direções enviando mensagens do controlador para NetFPGA e vice versa. Primeiramente uma mensagem e enviada para confirmar, se a conexão está estabelecida entre a NetFPGA e o controlador. Em seguida, a mensagem "Instalar"é enviada contendo as instruções eBPF que serão instaladas na memória de instruções do processador eBPF. As mensagens "Notifica"e "Estado"são utilizadas para ver o real estado do roteador podendo obter informações dos principais registradores da NetFPGA. Em [10] apenas as instruções "Oi"e "Instalar"estavam devidamente implementadas mas, devido a necessidade de verificar o estado e o valor de alguns registradores foi preciso adicionar esta duas mensagens ao nosso escopo.

Tabela II: Mensagens Controlador SDN

Tipo da mensagem	Direção	Descrição
Oi	$C^1 \leftrightarrow R^2$	Comunicação estabelecida
Instalar	$C^1 \rightarrow R^2$	Instalar instruções eBPF no roteador
Notificar	$C^1 \leftrightarrow R^2$	Ler um registrador específico
Estado	$C^1 \leftrightarrow R^2$	Mostra o estado do roteador

[1]Controlador, [2]Roteador.

F. Metadados

O plano de dados recebe o pacote pela interface de entrada e armazena o pacote na fila de entrada com algumas informações adicionais chamadas de metadados, na Tabela I e mostrado os metadados com as modificações realizadas, distintivamente do [10], foi acrescentado 8 campos ao metadado com o intuito de contabilizar o número de pacotes nas 8 filas. A Tabela I apresenta a estrutura armazenada, os metadados atualmente definidos são a porta de destino, o tamanho do pacote em múltiplos de 64 bits, a porta de origem, o tamanho do pacote

Tabela I: Metadado: Informações retiradas do pacote e armazenadas na memória de dados do processador eBPF.

63:48		**47:32**		**31:16**		**15:0**	
Porta destino em codificação one-hot		Tamanho do pacote em palavras de 64 bits		Porta de origem em binário		Porta destino em codificação one-hot	
Timestamp (nanosegundos)				Timestamp (segundos)			
Quantidade de pocotes na fila 0	Quantidades de pacotesna fila 1	Quantidade de pacotes na fila 2	Quantidades de pacotes na fila 3	Quantidades de pacotesna fila 4	Quantidades de pacotes na fila 5	Quantidades de pacotes na fila 6	Quantidade de pacotes na fila 7
...							

em *bytes*, o *timestamp* em segundos e nanosegundos e logo abaixo o conjunto de 8 campos que representam as 8 filas, estes campos possuem 8 bits e são utilizados para contabilizar o número de pacotes para aquela fila em específico. Os campos sobre o tamanho do pacote em 64 bits e em *bytes* foram incluídos porque o módulo da fila de entrada já tinha essa informação.

IV. Resultados

A Figura 6 apresenta a topologia do experimento. O roteador em hardware interliga dois computadores hospedeiros pelas portas 0 e 1 e conecta com o controlador pela porta 3.

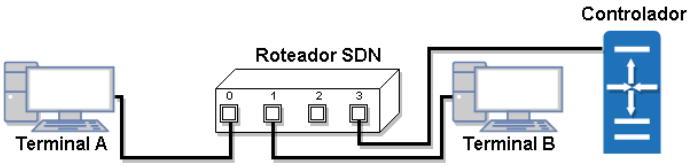

Figura 6: Topologia do experimento de validação do roteador em hardware com politicas de congestionamentos.

Foram criadas 2 aplicações em código assembly para realizar o controle de congestionamento: Uma aplicação contendo o **RED** e outra o **ERD**. Ambos os códigos possuem instruções já suportadas pelo nosso processador eBPF.

No passo seguinte enviamos este conjunto de instruções por meio do controlador SDN (espaço de usuário) para a nossa plataforma dando início ao processamento de pacotes com a nova política de congestionamento contida nas instruções. O controlador SDN envia a mensagem "Instala" com as instruções para a NetFPGA cuja finalidade e instalar o conjunto de instrução na memória de instrução. As instruções são inseridas na memória de instruções através da interface de registradores da NetFPGA para serem executadas.

Utilizando a ferramenta *iperf*, foram criados conexões UDP entre os terminais A e B. As Figuras 6 7 mostram a vazão por tamanho do pacote e a quantidade de pacotes descartados. Neste experimento medimos a vazão do roteador para tamanho de pacotes: 64, 128, 256, 512, 1.024 e 1.500 *bytes* e a quantidade de pacotes descartados. A vazão máxima obtida foi de 635 Mbps com 69759 pacotes processados em 1 s para pacotes de 1500 *bytes*.

A. Experimento 1

O experimento 1, Figura 7 utilizamos um limiar de 15 pacotes para o tamanho fixo da fila baseado no ERD [6]. Os resultados mostram uma taxa de descarte de aproximadamente 80% em relação a taxa de envio/taxa recebida, mostrando que o quando o limiar e atingido um congestionamento eminente está ocorrendo e para controla-lo o algoritmo descarta todos os pacotes diminuindo a janela de transferência.

B. Experimento 2

O experimento 2 figura 8 e mostrado limiar sendo modificado para 75 pacotes se adequando ao tamanho real da fila da plataforma NetFPGA, foi gerado uma taxa de descarte de aproximadamente 35% em relação a taxa de envio/taxa recebida independente do tamanho do pacote.

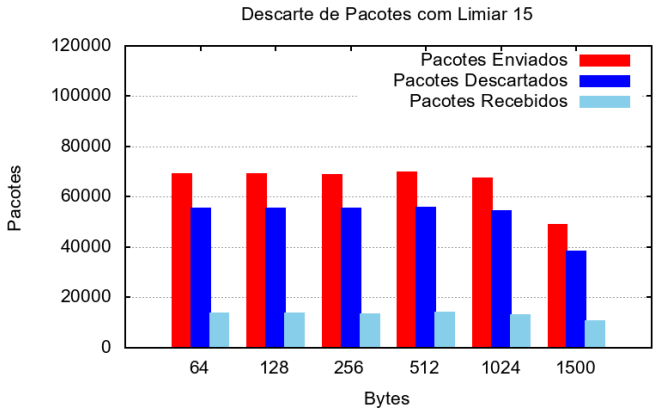

Figura 7: Descarte de pacotes com limiar 15.

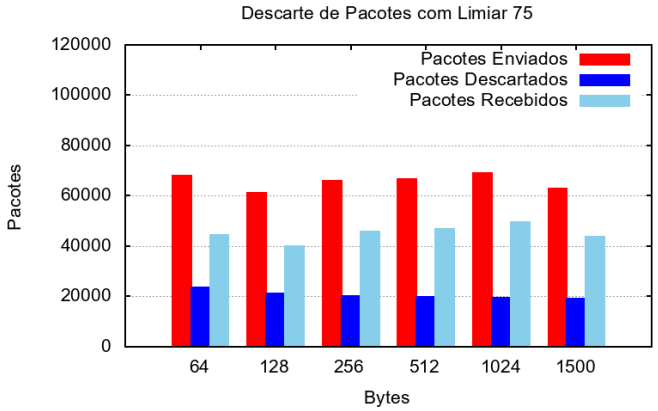

Figura 8: Descarte de pacotes com limiar 75.

V. Trabalhos relacionados

Nesta seção são descritos protótipos de roteadores em hardware em software.

BPFabric [7] foi proposto como uma plataforma em *software* que permite o processamento de pacotes independente do protocolo. BPFabric utiliza instruções eBPF para definir como o processamento e o encaminhamento dos pacotes no plano de dados serão realizados. BPFabric foi implementado sobre uma interface *socket raw* do Linux e o *framework* Intel DPDK.

Click modular router [9] é apresentado uma arquitetura em software para se criar roteadores configuráveis. Roteadores *Click* podem ser construídos a partir de componentes sendo cada componente formado por módulos de processamento de pacotes chamados elementos. Cada elemento é capaz de estender e implementar novas funcionalidades dos roteadores. Clique foi implementado em um computador de uso geral como uma extensão do *kernel* do *Linux*.

Data Center TCP (DCTCP) [3] é um mecanismo para controle de congestionamento TCP do tráfego em redes de data center. O (DCTCP) altera o processamento tradicional da Explicit Congestion Notification **(ECN)** estimando a fração de *bytes* que se encontra no congestionamento em vez de simplesmente detectar que algum congestionamento ocorreu. O DCTCP, em seguida, dimensiona a janela de congestionamento do TCP com base nessa estimativa.

VI. Conclusão e Trabalhos Futuros

Foi implementado um roteador em *hardware* na plataforma NetFPGA capaz de alternar entre diferentes políticas de controle de congestionamentos utilizando o processador eBPF. Nossa arquitetura permite fazer a configuração de políticas de controle de congestionamento, casamento e ações de forma dinâmica através de instruções eBPF. O sistema é independente de protocolo e permite utilizar novos conjuntos de instruções, para empregar novas políticas de congestionamento.As modificações no roteador implementado em hardware permite alterar a imagem do programa eBPF em tempo de execução, permitindo alterar como o roteador deve controlar os congestionamentos, com tempo zero de inatividade. Para trabalho futuro, é pretendido aplicar novos casos de uso buscando atingir novas vertentes para controlar congestionamentos aplicando nosso mecanismo em redes de alta taxa de vazão.

Referências

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM computer communication review*, vol. 40, no. 4. ACM, 2010, pp. 63–74.

[2] J. Aweya, M. Ouellette, and D. Y. Montuno, "Dred: a random early detection algorithm for tcp/ip networks," *International Journal of Communication Systems*, vol. 15, no. 4, pp. 287–307, 2002.

[3] S. Bensley, P. Balasubramanian, G. Judd, L. Eggert, and D. Thaler, "Data center tcp (dctcp): Tcp congestion control for data centers," 2017.

[4] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[5] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[6] E. S. Hashem, "Analysis of random drop for gateway congestion control," MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, Tech. Rep., 1989.

[7] S. Jouet and D. P. Pezaros, "Bpfabric: Data plane programmability for software defined networks," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. IEEE Press, 2017, pp. 38–48.

[8] W.-J. Kim and B. G. Lee, "Fred-fair random early detection algorithm for tcp over atm networks," *Electronics Letters*, vol. 34, no. 2, pp. 152–154, 1998.

[9] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 217–231, 1999.

[10] R. D. Pacífico, G. R. Coelho, M. A. Vieira, and J. A. Nacif, "Roteador sdn em hardware independente de protocolo com análise, casamento e ações dinâmicas," in *Simpósio Brasileiro de Redes de Computadores (SBRC)*, vol. 36, 2018.

[11] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer," in *Symposium Proceedings on Communications Architectures and Protocols*, ser. SIGCOMM '88. New York, NY, USA: ACM, 1988, pp. 303–313. [Online]. Available: http://doi.acm.org/10.1145/52324.52355

[12] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," Tech. Rep., 2001.

[13] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on networking*, vol. 4, no. 3, pp. 375–385, 1996.

[14] G. Thiruchelvi and J. Raja, "A survey on active queue management mechanisms," *International Journal of Computer Science and Network Security*, vol. 8, no. 12, pp. 130–145, 2008.

[15] L. Zhang, "A new architecture for packet switching network protocols," Ph.D. dissertation, Massachusetts Institute of Technology, 1989.

[16] W. Zhao, D. Olshefski, and H. Schulzrinne, "Internet quality of service: An overview," *Columbia University, New York, New York, Technical Report CUCS-003-00*, 2000.