Medição de desempenho de rede em *hardware*

Racyus Pacífico[1], Pablo Goulart[2], Ítalo Cunha[2], Marcos A. M. Vieira[2], Dorgival Guedes[2], Alex B. Vieira[3], José Augusto M. Nacif[1]

[1]Universidade Federal de Viçosa – Florestal, MG – Brasil

[2]Universidade Federal de Minas Gerais – Belo Horizonte, MG – Brasil

[3]Universidade Federal de Juiz de Fora – Juiz de Fora, MG – Brasil

{racyus.pacifico, jnacif}@ufv.br, alex.borges@ufjf.edu.br

{pgoulart, cunha, mmviera, dorgival}@dcc.ufmg.br

Abstract. *Measurement and tracking have crucial roles in Software-Defined Networks (SDNs). Unfortunately, most of these techniques are implemented in software at network end-hosts. This approach generates imprecision, high costs, and makes monitoring more difficult. In this paper, we extend Stanford's OpenFlow switch to implement a measurement architecture for SDN networks. Our architecture performs measurements in a simple and scalable way without depending on end-hosts. It allows monitoring the performance at the granularity of flows. We prototype our architecture on the NetFPGA platform. As an initial case study, we implemented a module to measure packet interarrival times and evaluated it in a the packet interarrival time in a realistic testbed. Our results show that our architecture presents a low relative difference (close to 0%) compared to measurement performed in software at end-hosts.*

Resumo. *Tarefas de monitoramento e medição têm papel crucial em Redes Definidas por Software (SDNs). Infelizmente, a implementação de grande parte destas técnicas é realizada por software nas extremidades da rede (nós hospedeiros). Tal abordagem gera imprecisão, alto custo e dificulta o monitoramento. Neste artigo, nós estendemos o comutador OpenFlow de Stanford para implementar uma arquitetura de medições em redes SDN. Esta arquitetura pode realizar medições das redes independentemente dos nós hospedeiros de maneira simples, escalável e barata. Além disso, ela permite observar o desempenho na granularidade de fluxos. Nosso protótipo foi implementado sobre a plataforma NetFPGA. Como instância inicial, nós implementamos um módulo para medir o tempo entre chegada de pacotes em um ambiente real. Nossos resultados mostram que nossa arquitetura apresenta baixa diferença relativa (próxima de 0%) em relação às medições do software nos nós hospedeiros.*

1. Introdução

O monitoramento do tráfego é indispensável para a utilização de mecanismos de engenharia de tráfego nas redes modernas [Daniel et al. 2003]. Desde a década passada o monitoramento de tráfego é um campo ativo em pesquisas na área de redes de computadores devido à dificuldade em se medir grandes volumes de tráfego de maneira exata por fluxo, além da complexidade no desenvolvimento de estruturas de medição. Tarefas de medição podem ser implementadas utilizando contadores de fluxo,

estruturas de dados *hash* e programação de pequenos fragmentos de código adicionado na CPU do comutador [Moshref et al. 2013]. As tarefas de medição baseadas em contadores de fluxo consumem muitos recursos (largura de banda e CPU) em virtude da demanda do monitoramento e de granularidade. Outros tipos de técnicas como estruturas de dados *hash* e programas de medição na CPU necessitam de grande investimento no desenvolvimento do *hardware* e na configuração dos programas de medição [Van Adrichem et al. 2014].

O paradigma SDN surge como uma arquitetura de rede emergente que tem o objetivo de facilitar o gerenciamento da rede e tarefas de medição de maneira programável. Esse paradigma consiste na separação do plano de dados do plano de controle [ONF 2012]. O plano de controle tem como funcionalidade gerenciar os dispositivos da rede por *software* de modo centralizado e o plano de dados apenas encaminhar pacotes de acordo com a ação enviada pelo controlador [Naous et al. 2008]. Utilizando a estrutura de SDN, as tarefas de medição se tornam mais baratas, pois não é necessário grande investimento no *hardware* ou de configuração na rede. Um exemplo de aplicação SDN é o comutador *OpenFlow*. Esse comutador utiliza contadores de pacotes e de *bytes* por fluxo trafegado. O conceito de fluxo pode ser redefinido dinamicamente de acordo com os requisitos da aplicação [Van Adrichem et al. 2014].

O objetivo desse trabalho é propor uma modificação do comutador *OpenFlow* [NetFPGA 2015c] para medir o tempo de chegada entre pacotes dos fluxos trafegados. Essa métrica do tempo entre pacotes foi implementada no *hardware NetFPGA* 1G, permitindo alto poder de processamento de pacotes da rede e medição de fluxos TCP de maneira acurada. Nossa arquitetura pode ser vista como um concentrador de rede que realiza medições dos fluxos trafegados em *hardware* e, assim, dispensa medições em *software* nos nós hospedeiros da rede. O protótipo foi testado numa rede real. Atrasos variáveis pré-definidos foram inseridos no envio de pacotes e comparados com os atrasos medidos por *software*. Todos os testes foram realizados com o propósito de validar o comutador com suporte à medição do tempo de chegada entre pacotes.

As principais contribuições deste trabalho são: (i) definição da arquitetura de medição distribuída de métricas de desempenho de rede sem a necessidade de instrumentação dos nós hospedeiros; (ii) implementação da arquitetura proposta em um protótipo que utiliza um comutador SDN de código aberto executando na plataforma *NetFPGA*. Os resultados obtidos mostram que as medições realizadas no nosso protótipo apresentam diferenças relativas próximas de 0% em relação às medições em *software*. Ressaltamos que o estudo é inédito e que outras abordagens realizam medições aproximadas ou apenas em *software*, não sendo possível uma comparação direta. Portanto, comparamos com o caso tradicional, medição fim a fim com a geração de carga na origem.

O artigo tem a seguinte organização: Na seção 2 são apresentadas as principais técnicas de medição em SDN. Em seguida (seção 3) são abordados os detalhes da arquitetura proposta e da implementação do comutador *OpenFlow* modificado na *NetFPGA*. Na seção 4 é descrito o ambiente de testes e a metodologia. Na seção 5 são apresentados os resultados dos testes reais. Na seção 6 são abordados os trabalhos relacionados sobre métricas de medição, técnicas e ferramentas para realizar medições em redes. Finalmente, na seção 7 apresentamos nossas conclusões e trabalhos futuros.

2. Medição de Desempenho de Rede em SDN

Desde a década de 1970 as tecnologias de redes têm sofrido grandes modificações, tornando o gerenciamento das redes uma tarefa complexa e contribuindo para o aumento do custo operacional. Essas dificuldades incentivaram os pesquisadores a desenvolverem novas tecnologias, serviços e dispositivos para prover uma forma fácil e barata para gerenciar essas redes. As redes SDN surgiram para preencher esta lacuna no gerenciamento de redes, além de flexibilizar o roteamento de pacotes. Essa tecnologia permite gerenciar ambientes distintos de redes que englobam dispositivos móveis, *data centers*, serviços em nuvem, virtualização [Casado et al. 2012].

Nas redes SDN, o plano de dados é separado do plano de controle. As funções de encaminhamento (plano de dados) e roteamento (plano de controle) dos pacotes são acopladas nos roteadores tradicionais. O plano de dados das redes SDN é composto por tabelas de fluxo. O armazenamento da entrada de fluxo é feito na memória TCAM (*Ternary Content-Addressable Memory*) do comutador. Esse tipo de memória permite a inserção de regras com agregador * (*wildcard*). O plano de dados na arquitetura SDN encaminha os pacotes com base nas regras de fluxo inseridas nas tabelas de fluxo no comutador [Naous et al. 2008]. Ele também pode ser responsável por monitorar os fluxos e coletar estatísticas [Braun and Menth 2014]. O plano de controle é composto por controladores que são totalmente programáveis e permitem administradores de rede definir roteamento de pacotes por fluxo. Toda a complexidade da rede é colocada no controlador central. Esse controle centralizado possibilita aos administradores gerenciarem recursos da rede e decidirem quais fluxos devem ser monitorados. Isso habilita a experimentação de novos protocolos em redes utilizando ambientes reais, pois permite que pesquisadores separem o tráfego real do tráfego experimental [Naous et al. 2008].

Acoplar funcionalidades de medição em redes SDN é uma tarefa que tem interessado a comunidade acadêmica. A medição de tráfego em redes SDN consiste em coletar estatísticas dos fluxos em diversos níveis de granularidade, satisfazendo diferentes tipos de aplicação e maximizando a utilização dos recursos da rede [Yassine et al. 2015]. Um exemplo seria detectar mudanças no tráfego dos fluxos agregados na escala de minutos para realizar engenharia de tráfego em redes de grande escala ou identificar mudanças no tráfego em escala de milisegundos para reduzir a latência em *datacenters* [Moshref et al. 2013].

As técnicas de medição podem ser classificadas como passivas ou ativas. Técnicas passivas são conhecidas por monitorar o tráfego da rede sem injetar novo tráfego ou afetar o tráfego existente. Esse tipo de técnica realiza medições locais e sua principal desvantagem está nos custos de instalação dos monitores de tráfego na rede. [Van Adrichem et al. 2014]. Técnicas de medição ativas, por outro lado, enviam tráfego de teste (sondas) para estimar as características de utilização da largura de banda da rede. As sondas podem afetar o desempenho da rede, o que interfere na acurácia das medições [Prasad et al. 2003].

Recentemente, diversos trabalhos tratam de problemas decorrentes da medição de desempenho em SDN. Algumas dessas soluções serão apresentadas na seção 6.

3. Arquitetura Proposta – Instrumentação do Plano de Dados

O comutador *OpenFlow* utilizado neste trabalho é simples, de código aberto e projetado para ser executado na plataforma *NetFPGA*. A estrutura SDN utilizada por este comutador transfere a complexidade dos nós da rede para um ponto central de controle, contribuindo na detecção de falhas e qualidade da medição. Com essa estrutura é possível inserir regras de fluxo para serem medidas diretamente no comutador e capturar o tempo das medições por programas executados no espaço de usuário [Naous et al. 2008].

Neste trabalho modificamos o projeto do comutador *OpenFlow* de *Stanford* [Naous et al. 2008] para medir o tempo de chegada entre pacotes dos fluxos trafegados. Implementamos as medições da média e variância do tempo de chegada entre pacotes. Nossa implementação mede 28 fluxos TCP concorrentemente em intervalos definidos pelo usuário na velocidade da interface (*line-speed*), sem afetar a taxa de processamento de pacotes. A arquitetura também faz medições para todos os pacotes do fluxo, sem amostragem, e escala de acordo com o número de regras suportadas pelo comutador. Todas as regras de fluxo são inseridas diretamente na memória ternária (TCAM) do comutador. A TCAM do *hardware* permite a inserção de 32 regras de fluxos com campos exatos e coringas (prefixos). Destas 32 regras reservamos 28 para medição e quatro para o encaminhamento de pacotes ARP diretamente sem que eles passem pelo controlador. Nosso trabalho abre a possibilidade de realizar medições que não eram possíveis de maneira fim-a-fim, por exemplo, a medição do tempo de chegada entre pacotes de fluxos de uma subrede utilizando regras com agregador * do *OpenFlow*.

O *pipeline* do nosso plano de dados é dividido em quatro estágios: extração do identificador do fluxo do pacote (*Header Parser*), busca do identificador de fluxo na TCAM (*Wildcard*), cálculo das operações de medição (*Measure*) e armazenamento das operações de medição na SRAM (*SRAM*). Todos os estágios, exceto *Measure*, pertecem ao projeto do comutador *OpenFlow*. Esses estágios foram adaptados para dar suporte à medição do tempo de chegada entre pacotes. O plano de controle da arquitetura é dividido duas aplicações: o controlador SDN que insere as regras dos fluxos a serem medidos e o *software* de medição que realiza a leitura das medições processadas pelo *hardware*. Na figura 1 é apresentado a arquitetura do protótipo implementado.

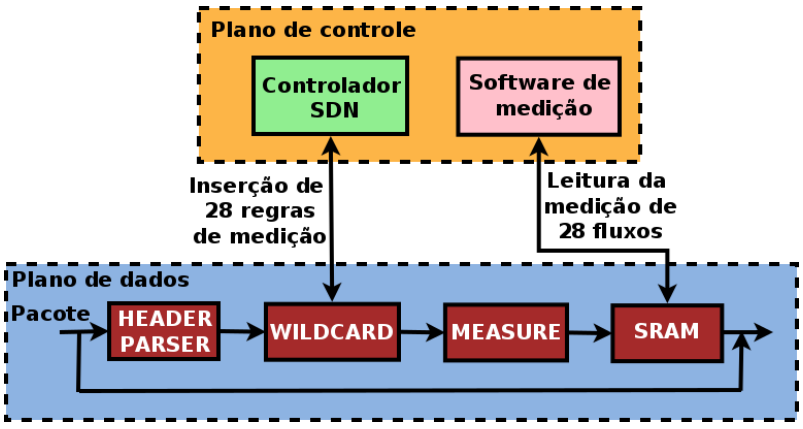

Figura 1. Arquitetura *OpenFlow* com suporte à medição do tempo de chegada entre pacotes.

3.1. Plano de dados

O processo de medição no plano de dados começa com a chegada de um pacote TCP ao módulo que analisa o cabeçalho (*Header Parser*). Este módulo extrai o identificador do fluxo e marca o tempo de chegada do pacote (em ciclos de relógio) gerados pelo módulo contador de tempo (*Time Counter*), e os envia para o módulo de medição (*Measure*). O módulo de medição insere essas entradas em uma fila interna. Este módulo precisa esperar a saída do módulo de busca aproximada (*Wildcard*) para decidir se descarta o tempo de chegada e o identificador do fluxo do pacote ou se calcula a média e a variância. O módulo *Wildcard* compara o identificador do fluxo extraído do pacote com as regras de medição armazenadas na memória do comutador. Após a busca ser completada, o módulo *Measure* envia um sinal (*hit*) indicando se encontrou um identificador de fluxo e a sua posição na memória do comutador.

A partir deste momento iniciam-se os cálculos das operações de medição. A métrica implementada consiste em receber o tempo de chegada do primeiro pacote (T_1) do fluxo (F_j), o tempo de chegada do segundo pacote (T_2) do fluxo (F_j) até o tempo de chegada do enésimo pacote (T_n) do fluxo (F_j). F_j refere-se aos fluxos a serem medidos, tal que, j varia de 1 a 28. A cada intervalo entre pacotes do mesmo fluxo é realizado a operação de diferença $dT = T_c - T_a$, sendo, T_c o tempo de chegada do pacote atual e T_a o tempo de chegada do pacote anterior. Em seguida são realizados os cálculos do acúmulo da soma da diferença, $S_j = S_j + dT$ e o acúmulo da soma ao quadrado da diferença, $S_j^2 = S_j^2 + dT^2$. Por fim, o contador de pacotes fluxo é incrementado (N_j).

Após todos os cálculos terem sido realizados, os resultados obtidos são salvos temporariamente em um *buffer* interno do módulo de medição e na memória SRAM. O *buffer* no módulo de medição tem 28 linhas. Em cada linha é armazenado a quantidade de pacotes por fluxo, o tempo de chegada do último pacote, a soma (S_j) e soma ao quadrado (S_j^2) dos acúmulos dos intervalos dos tempos de chegada dos pacotes. As 28 linhas do *buffer* equivalem às linhas da TCAM às quais são inseridas as regras. Quando ocorre um acerto (*hit*) na busca, o endereço de acerto corresponderá à linha do *buffer* no qual deve ser feita a leitura e a atualização das informações do fluxo do pacote. Por exemplo, se ocorrer um sinal de acerto no endereço cinco da TCAM, as informações da linha cinco do *buffer* são atualizadas. O *buffer* evita que a SRAM seja acessada constantemente. As operações de leitura na SRAM gastam três ciclos de relógio, o que prejudica o desempenho. Isso ocorre devido ao tempo que a máquina de estados espera o dado da memória [Goulart et al. 2015].

Com o resultado das operações salvas na SRAM entra em ação o *software* de medição que é encarregado de terminar a medição e ler via interface de registradores os valores calculados dos fluxos armazenados na SRAM. Devido à limitação do número de células lógicas do *hardware* e a precisão da medição as operações de divisão da média (equação 1) e variância (equação 2) foram realizadas no *software* de medição. Na figura 2 é apresentado o caminho de dados do comutador *OpenFlow* com suporte a média e variância do tempo de chegada entre pacotes. É importante ressaltar que a arquitetura proposta é genérica e permite que outras métricas sejam implementadas, desde que o *hardware* suporte. Um exemplo seria estimar a taxa de transmissão de um fluxo baseado no tamanho do pacote utilizando a média e variância calculada pelo módulo *Measure*.

$$media_j = \frac{\sum_{i=1}^{N_j} dT_i}{(N_j - 1)}, \quad 1 \leq j \leq 28 \tag{1}$$

$$variancia_j = \frac{\sum_{i=1}^{N_j} (dT_i)^2 - (\sum_{i=1}^{N_j} dT_i)^2 / N_j}{(N_j - 1)}, \quad 1 \leq j \leq 28 \tag{2}$$

Atualmente existem modelos de plataformas NetFPGA com FPGAs de maior capacidade, por exemplo, a CML [NetFPGA 2015a] e a SUME [NetFPGA 2015b]. Para efeitos de comparação, o modelo SUME (FPGA Virtex-7) possui 13x mais recursos que o modelo 1G, utilizado nesse trabalho, permitindo monitorar até 416 fluxos concorrentemente.

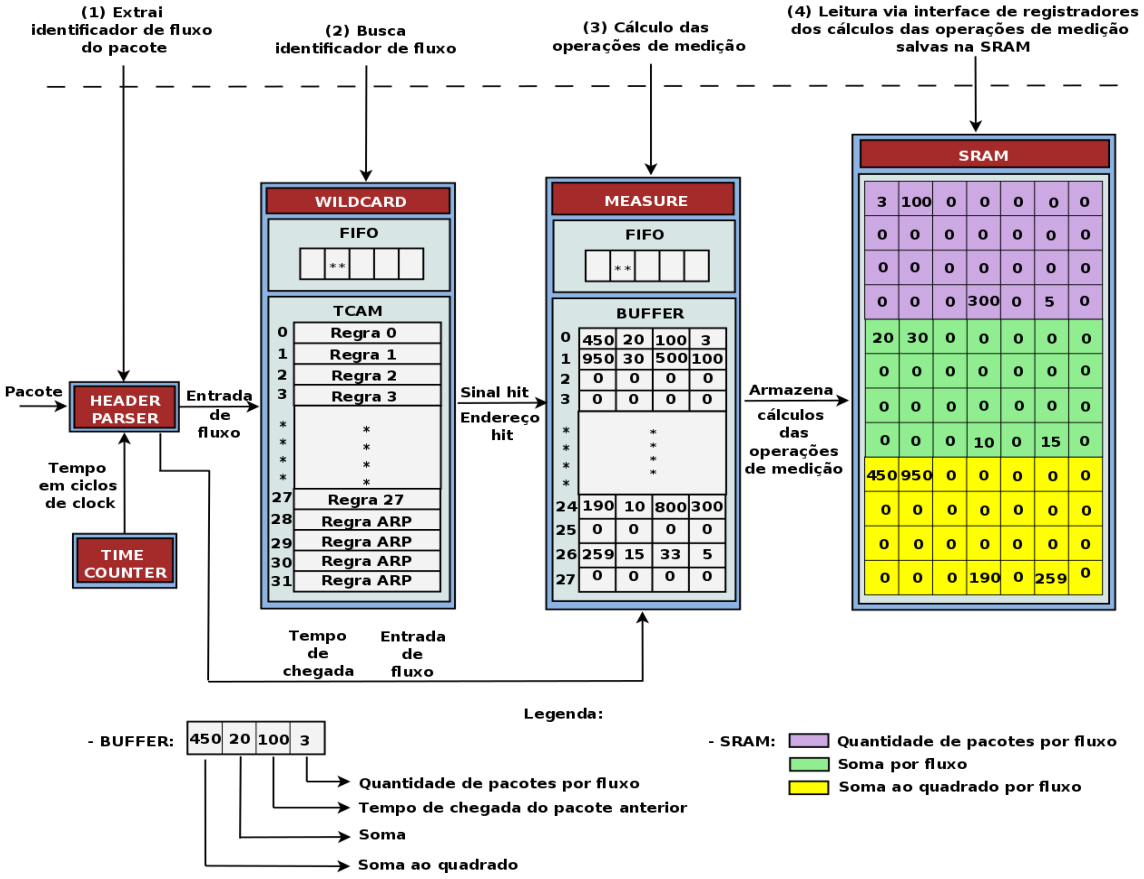

Figura 2. Caminho de dados *OpenFlow* com suporte a média e variância do tempo de chegada entre pacotes.

3.2. Plano de controle

O plano de controle é dividido em duas partes, controlador e *software* de medição. O controlador tem como função inserir na TCAM do comutador as regras que devem ser medidos e dizer ao comutador como os pacotes serão encaminhados. No protótipo desenvolvido o controlador foi programado para medir 28 fluxos TCP e encaminhar os pacotes que pertencem a um fluxo para o terminal B. A especificação das regras dos fluxos a serem medidos são facilmente alteradas. A aplicação do controlador lê um arquivo

com 28 regras e as insere na TCAM. Para modificar as regras dos fluxos que devem ser medidos é necessário apenas alterar o arquivo com as regras e reiniciar o processo responsável pelo controlador. Em SDN existem vários tipos de controladores com suas respectivas características. Utilizamos o controlador POX [OpenNetworkingLab 2015]. Esse controlador é baseado na linguagem *Python* e é utilizado para o desenvolvimento de projetos rápidos de aplicação de rede.

Estendemos o plano de controle criando um *software* de medição. O *software* de medição termina o processo de medição realizando o cálculo resultante da média e variância dos 28 fluxos. Esse *software* lê os valores das operações da média e variância processadas no *hardware* via interface de registradores com base no intervalo definido pelo usuário. A cada leitura, média e variância dos 28 fluxos são calculadas e armazenadas em um arquivo com a data e a hora da medição.

4. Avaliação

Nesta seção apresentamos o ambiente dos experimentos realizados com o protótipo do comutador *OpenFlow* com suporte à medição do tempo de chegada entre pacotes. Nosso objetivo é comparar a acurácia das medições realizadas em *hardware* e *software* com o propósito de validar a medição do protótipo.

4.1. Descrição do ambiente de testes

O cenário de testes consiste em um terminal A enviando pacotes TCP à uma taxa controlada para o terminal B. Os pacotes criados por A não utilizam do protocolo TCP, apenas têm a estrutura do formato do pacote TCP (pacotes TCP RAW). Entre A e B foram inseridos dois comutadores tradicionais contendo o tráfego real com o propósito de aumentar o atraso do tempo de chegada dos pacotes enviados por A e tornar o experimento ainda mais realista. No ponto final da rede foram inseridos a *NetFPGA* rodando o protótipo de medição e o terminal B executando o *software* de medição do tempo de chegada entre pacotes. Em B, a média e a variância são calculadas depois do término do experimento. No protótipo, média e variância são calculadas em tempo real. Nosso objetivo é comparar a diferença relativa entre as medições em B (versão *software*, no hospedeiro) e *hardware* (*NetFPGA*) com base nos atrasos pré-definidos inseridos entre os pacotes enviados por A. Na figura 3 é apresentado o ambiente de testes.

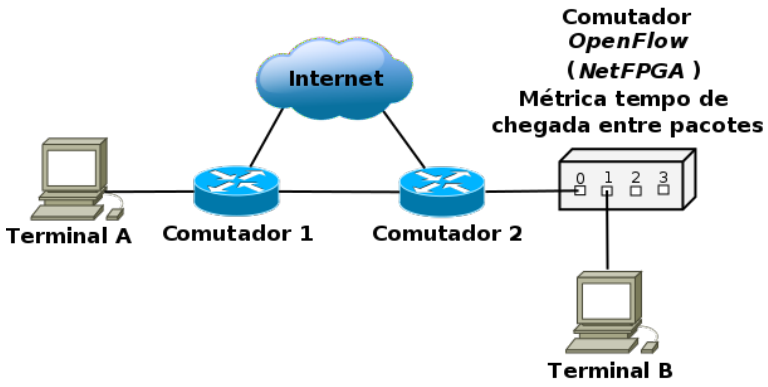

Figura 3. Ambiente de testes.

4.2. Verificação do protótipo

Desenvolvemos um programa para enviar pacotes TCP a partir do terminal A. Esse programa lê um arquivo de texto que contêm as regras de monitoramento definidas no controlador. As regras referem-se aos fluxos monitorados no comutador. O programa executa uma *thread* para cada fluxo lido nesse arquivo. Cada *thread* envia pacotes que correspondem a apenas um fluxo monitorado. Esse programa envia pacotes em intervalos aleatórios ou constantes. Utilizamos intervalos constantes na fase de validação do protótipo para conferir os valores capturados no *hardware*. Para os testes reais, utilizamos intervalos aleatórios para observar os valores medidos. Os pacotes foram gerados utilizando a biblioteca `libnet` do C. Para termos um controle maior do número de pacotes vindos de A para B e *NetFPGA*, escrevemos os pacotes enviados pelo `libnet` em vários arquivos `pcap` de duração de 1s para serem transmitidos por A. Esta abordagem permite uma comparação mais acurada. Arquivos `pcap` em B (*software*) também foram criados e reproduzidos utilizando a biblioteca DPKT da linguagem *Python*.

Foram realizadas várias iterações a partir dos arquivos `pcap` criados no terminal A. Antes de cada execução, o programa `tcpdump` é inicializado em B para capturar os pacotes enviados por A. No fim da iteração, as medições são lidas e a SRAM e o *buffer* da NetFPGA são inicializados. Após essa etapa, o `tcpdump` escreve um arquivo de saída com os pacotes capturados. Esses passos são repetidos em todas as iterações.

4.3. *Warm up*

Os experimentos foram compostos por cinquenta iterações, tendo cada iteração a duração de 1 segundo. São gastos 5 segundos para inicializar o controlador, os processos de monitoramento e a transmissão de pacotes, e 1 segundo para finalizar todos os processos. O tempo total do experimento é o tempo de inicialização somado ao tempo gasto pelas iterações e o tempo da finalização: $5s + 50 \times 1s + 1s = 56s$. Foram descartados 10% das amostras devido ao período de *warm up*, que referem-se à 6 segundos no começo e no fim do experimento. Cada segundo corresponde à 28 amostras.

5. Resultados

Todos os experimentos foram configurados de acordo como descrito na seção 4.1. Os procedimentos descritos nas seções 4.2 e 4.3 foram utilizados na preparação do ambiente. Nesta seção é apresentado os gráficos e intervalos de confiança da média e da variança dos atrasos para cada experimento.

5.1. Experimento 1

Nesse experimento foram enviados aproximadamente 1700 pacotes com atrasos pré-definidos de 20/40 milisegundos (ms) para cada um dos 28 fluxos. Os resultados mostram que 95% da diferença relativa entre as abordagens em *hardware* e *software* estão próximas de 0.0072% no gráfico da média. No gráfico da variância, 95% das medições possuem diferença relativa próximas à 0.26%. Três valores ficaram entre 3% e 7%, mas as diferenças absolutas são menores que 10^{-4}. Os valores esperados para a média estão dentro do intervalo de confiança $10^{-5} \times [2.897 : 3.166]$, e os valores esperados para a variância estão dentro do intervalo de $10^{-3} \times [1.0043 : 1.3574]$.

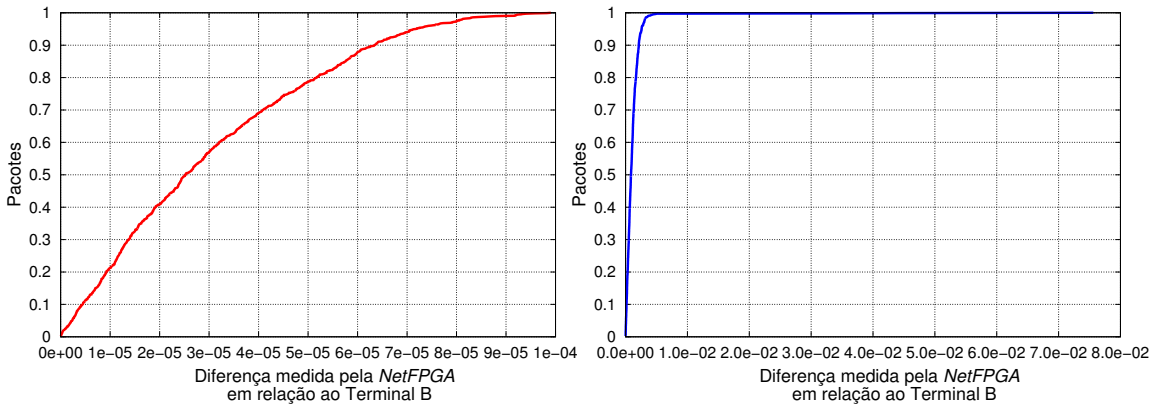

Figura 4. Gráficos CDF diferença relativa média(A) e variância(B) atraso 20/40ms.

5.2. Experimento 2

Nesse experimento foram enviados aproximadamente 1800 pacotes com atrasos aleatórios para cada um dos 28 fluxos. Foram definidas três sementes para gerar os atrasos. O gráfico da média mostra que 95% das medições obtidas com diferença próxima de 0.0071%. No gráfico da variância, 95% das medições obtiveram diferença relativa próximas de 0.19%. Nesse gráfico, seis amostras ficaram entre 3% e 6% de diferença relativa, mas as diferenças absolutas foram menores que 10^{-4}. O intervalo de confiança do gráfico da média é $10^{-5} \times [3.169 : 3.432]$, e os valores esperados da variância estão dentro do intervalo de confiança $10^{-4} \times [7.9642 : 12.4338]$.

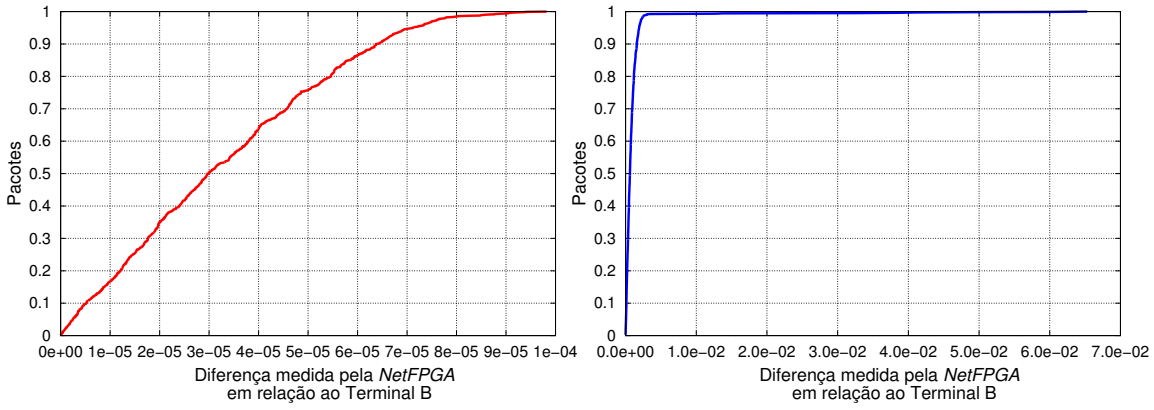

Figura 5. Gráficos CDF diferença relativa média(A) e variância(B) atraso variado.

5.3. Experimento 3

Nesse experimento foram enviados 1700 pacotes com atrasos pré-definidos 20/40ms para quatro fluxos. Objetivo deste experimento foi medir todos esses fluxos numa regra agregada com o propósito de mostrar que a arquitetura permite monitorar o tráfego de subredes inteiras. O gráfico da média mostra que 97% das medições obtidas com diferença próxima de 0.007%. No gráfico da variância, 97% das medições obtiveram diferença relativa próximas de 0.257%. Os valores esperados da média estão dentro do intervalo

de confiança $10^{-5} \times [2.788 : 4.349]$, e os valores esperados da variância estão dentro do intervalo de confiança $10^{-4} \times [8.4223 : 12.072]$.

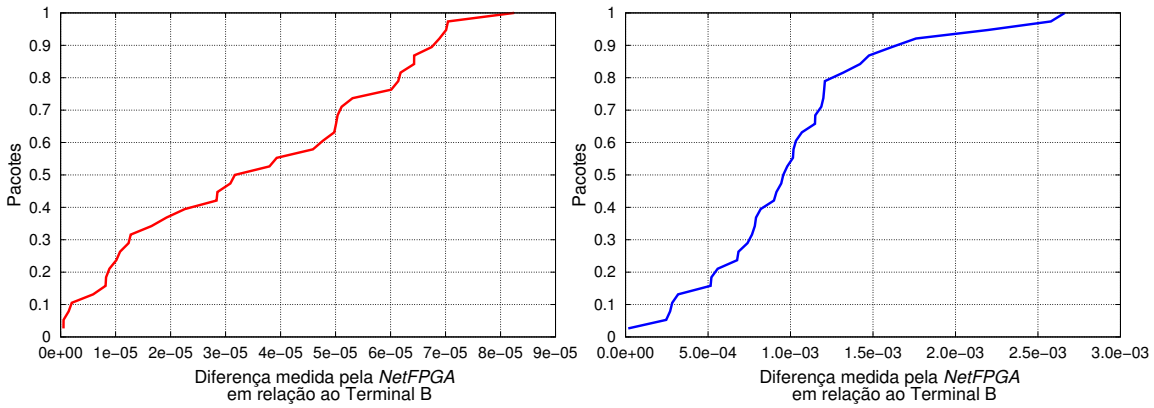

Figura 6. Gráficos CDF diferença relativa média(A) e variância(B) regra wildcard.

6. Trabalhos Relacionados

Nesta seção será apresentado uma visão geral dos trabalhos relacionados com base nas métricas, técnicas e ferramentas utilizadas em medição de redes. Realizamos o levantamento de onze trabalhos de acordo com os seguintes critérios: métricas de medição, a plataforma e o ambiente de implementação.

[Lombardo et al. 2012] apresenta a implementação de quatro módulos que estendem o roteador de referência implementado na *NetFPGA*. Dois desses módulos monitoram a taxa de *bits* nas filas de entrada e saída. Um terceiro módulo realiza a previsão da taxa de bits de entrada a partir de um filtro de médias móveis exponencialmente ponderadas (EWMA), o quarto módulo é uma extensão do limitador de taxa da biblioteca da *NetFPGA*. Um programa no espaço de usuário lê os registradores no *hardware* que armazenam as taxas de *bit* calculadas nas filas de entrada e saída. As diferentes escalas de tempo da amostragem (em microsegundos) e do ciclo de relógio da *NetFPGA* prejudicam a acurácia da medição. O cenário de testes foi composto por duas *NetFPGAs* ligadas entre si: a primeira contendo o projeto do *reference router* com os quatro módulos implementados e a segunda com o projeto *packet generator*, um projeto de referência na *NetFPGA* que gera e transmite pacote a partir de arquivos `pcap`.

No trabalho de [Moshref et al. 2013] é discutido o compromisso entre utilização de recursos da rede com a exatidão das diferentes primitivas de medição. O artigo tem como propósito avaliar a utilização de técnicas baseadas em contadores, estruturas de *hash* e programação da CPU para detectar fluxos pesados (*Hierarquical Heavy Hitters*), que consiste na identificação dos prefixos de IP mais longos cuja utilização da rede supere um limiar pré-estabelecido. Na avaliação do protótipo foram utilizadas várias escalas de tempo e diferentes configurações da utilzação dos recursos dos comutadores. O protótipo foi avaliado utilizando-se *traces* de pacotes obtidos do projeto CAIDA.

Em [Yu et al. 2013] é apresentado a arquitetura de medição OpenSketch. OpenSketch baseia-se na estrutura SDN do comutador *OpenFlow* e utiliza como primitiva

de medição estruturas baseadas a *sketches*. O plano de controle é composto por uma biblioteca de medição enquanto o plano de dados é dividido em três estágios: *hashing*, classificação e contagem. Neste trabalho foram implementados sete *sketches* que possibilitam a medição de cinco diferentes tipos de métricas. A arquitetura OpenSketch foi implementada no *hardware NetFPGA* 1G. A validação deste trabalho ocorreu por meio da simulação de pacotes *trace* CAIDA e pela comparação de duas métricas implementadas no OpenSketch com a amostragem de pacotes do NetFlow.

[Chowdhury et al. 2014] é proposto um *framework* de monitoramento de rede. Esse *framework* fornece uma visão geral e estatísticas sobre a utilização de recurso da rede. O PayLess é construído acima do controlador *OpenFlow* e fornece uma API RESTful para o desenvolvimento de aplicações de monitoramento. A validação do PayLess foi realizada com o monitoramento de utilização de *link* entre comutadores. Toda a topologia foi desenvolvida com base na plataforma Mininet e utilizado o programa *iperf* para o envio de pacotes UDP durante 100s.

[Kekely et al. 2014] apresenta uma implementação de SDM (*Software-Defined Monitoring*) para monitorar o tráfego na camada de aplicação. Os autores implementaram o protótipo sobre o projeto NetCOPE, utilizando FPGAs para trabalhar à taxas de até 100Gb/s. O *hardware* implementado coleta informações do cabeçalho dos pacotes HTTP dos fluxos monitorados e os envia para um programa em espaço de usuário utilizando um formato unificado de pacotes através das interfaces PCI utilizando DMA. As tarefas mais complicadas, como *deep packet inspection* são realizadas em *software* devido as dificuldades em se implementar módulos de análise de pacotes em HDLs. Essa estratégia possibilita alcançar alta vazão no processamento e flexibilidade.

O artigo Planck [Rasley et al. 2014] apresenta uma ferramenta de medição para redes SDN que tem o objetivo identificar as condições da rede com baixa latência. Ambientes como *datacenters* ou de computação em nuvem são caracterizados pela dinâmica do tráfego, o que exige tempos de respostas muito pequenos para controlar o congestionamento ou dar manutenção em falhas na rede. O Planck oferece uma solução eficiente para realizar as tarefas de identificação dos fluxos pelo espelhamento do tráfego numa porta de monitoramento. Como desvantagem, o tráfego através do comutador frequentemente excederá a largura de banda máxima, o que poderá resultar em descartes de pacotes utilizados no monitoramento.

[Van Adrichem et al. 2014] propôs uma implementação de *software* de código aberto para monitorar métricas por fluxo, especialmente, atraso e perda de pacotes em redes *OpenFlow*. Essa implementação fornece engenharia de tráfego ao controlador com medições de monitoramento *online*. A perda de pacotes foi calculada com base nas estatísticas do fluxo do primeiro e último comutador de cada caminho da rede subtraindo o aumento do contador de pacotes do comutador origem com o aumento do contador de pacotes do comutador destino. Essa técnica permite uma medição exata da perda de pacotes. O atraso de pacotes foi calculado por RTT com a injeção de pacotes na rede. Esses pacotes viajaram pela rede e retornaram ao controlador determinando assim os atrasos. Os testes basearam em executar vídeo *stream* entre dois servidores na extremidade da rede ligados por comutadores com atrasos e perda de pacotes estabelecidos com `netem`. Os resultados medidos pelo OpenNetMon foram comparados com *software tcpstat*.

O artigo descreve uma ferramenta de monitoramento de tráfego utilizando técnicas de tomografia de redes para identificar grandes fluxos em redes DCN [Hu and Luo 2015]. O protótipo foi concebido em dois passos. O primeiro passo consistiu na formulação de um modelo de otimização baseado nos valores de uma matriz de tráfego TM preenchida com os valores dos contadores SNMP e SDN dos comutadores, e da definição de uma heurística para solucionar esse modelo eficientemente. O segundo passo consistiu na identificação de grandes fluxos entre os servidores através desses contadores e do modelo de tomografia de rede proposto. A qualidade da ferramenta na escala de *datacenters* foi avaliada por simulações no *ns-3*. Seus resultados mostraram que a identificação utilizando essa abordagem é superior às tradicionais baseadas em contadores.

Em [Mittal et al. 2015] foi implementado um protocolo de controle de congestionamento para *datacenters* que controla a taxa de transmissão usando a técnica de gradiente do RTT. Essa técnica elimina as trocas de contexto do espaço usuário *kernel* contribuindo para uma redução significativa do consumo de CPU e uma baixa latência na rede com alta largura de banda. O gradiente do RTT foi implementado em um *software* rodando sobre uma placa NIC em um sistema operacional com suporte à *OS-bypass*. Foram realizados dois tipos de testes. O primeiro examinou propriedades básicas do controlador de congestionamento tal como: *throughput*, *fairness*, latência e acurácia de tempo. Para o primeiro teste utilizou-se de um pequeno *testbed*. No segundo o protocolo TIMELLY foi executado em um *testbed* de grande escala com 100 máquinas ligadas em uma rede clássica Clos.

[Wellem et al. 2015] aborda uma arquitetura de medição baseada em *sketches* diferente do OpenSketch. Os autores desenvolveram um plano de dados com duas tabelas de fluxo para armazenar identificadores de fluxo e contadores de *bytes*/pacotes sem usar um método de classificação de fluxos que utilize memória TCAM. O protótipo desenvolvido não tem uma estrutura dinâmica, não permitindo a medição de outros tipos de métricas que não sejam baseadas em fluxos. Todo o processamento é realizado no *hardware*, armazenado nas memórias e a medição lida por *software* via interface de registradores. O plano de dados foi implementado na *NetFPGA* 10G e o protótipo validado a partir de pacotes *trace* do CAIDA.

[Zhu et al. 2015] assim como [Mittal et al. 2015] implementou um protocolo de controle de congestionamento para *datacenters*. O protocolo proposto (DCQCN) controla a taxa de transferência dos *hosts* que acessam à memória utilizando RDMA (*Remote Direct Memory Access*). RDMA é um mecanismo de acesso direto à uma memória pré-registrada de outra máquina. Esse mecanismo pode sofrer congestionamento quando a memória compartilhada é acessada por diversos outros *hosts* ao mesmo tempo. A implementação foi totalmente desenvolvida sobre placas NIC contribuindo para redução de consumo de CPU. Os autores validaram o protocolo usando tráfego *benchmark* derivados de tráfegos *trace* de seus próprios *datacenters*.

7. Conclusões

O monitoramento de tráfego é o ponto chave para se garantir a qualidade de redes IPs. As redes SDN e o padrão *OpenFlow* surgem como tecnologias que facilitam as tarefas de gerenciamento e medição dos fluxos. Modificamos o comutador *OpenFlow* de *Stanford* para medir o tempo de chegada entre pacotes dos fluxos trafegados em uma rede realista.

Com base nos resultados obtidos concluímos que a diferenca relativa entre a medição em *hardware* e *software* é quase nula. Isso implica, que a medição do protótipo em *hardware* tem uma acurácia próxima à medida no nodo folha da rede em *software*, com uma maior vazão de pacotes, centralizada e em tempo real.

Como trabalhos futuros, pretendemos implementar as métricas de desempenho tempo de chegada entre pacotes e latência utilizando de estrutura *hash* (*Bloom Filters*) para medir com baixa granularidade o atraso entre pacotes. Pretendemos também implementar outras métricas de desempenho de rede disponíveis na literatura.

Agradecimentos

Este trabalho foi parcialmente financiado por Fapemig, CAPES, CNPq, e pelos projetos MCT/CNPq-InWeb (573871/2008-6), FAPEMIG-PRONEX-MASWeb (APQ-01400-14), e H2020-EUB-2015 EUBra-BIGSEA (EU GA 690116, MCT/RNP/CETIC/Brazil 0650/04).

Referências

Braun, W. and Menth, M. (2014). Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet*, 6(2):302–336.

Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: a retrospective on evolving sdn. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 85–90. ACM.

Chowdhury, S. R., Bari, M. F., Ahmed, R., and Boutaba, R. (2014). Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE.

Daniel, E. J., White, C. M., Teague, K., et al. (2003). An interarrival delay jitter model using multistructure network delay characteristics for packet networks. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1738–1742. IEEE.

Goulart, P., Cunha, Í., Vieira, M. A., Marcondes, C., and Menotti, R. (2015). Netfpga: Processamento de pacotes em hardware.

Hu, Z. and Luo, J. (2015). Cracking network monitoring in dcns with sdn. In *Proc. IEEE INFOCOM*.

Kekely, L., Pus, V., and Korenek, J. (2014). Software defined monitoring of application protocols. In *INFOCOM, 2014 Proceedings IEEE*, pages 1725–1733. IEEE.

Lombardo, A., Reforgiato, D., Riccobene, V., and Schembra, G. (2012). Netfpga hardware modules for input, output and ewma bit-rate computation. *International Journal of Future Generation Communication and Networking*, 5(2):121–134.

Mittal, R., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., Zats, D., et al. (2015). Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 537–550. ACM.

Moshref, M., Yu, M., and Govindan, R. (2013). Resource/accuracy tradeoffs in software-defined measurement. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 73–78. ACM.

Naous, J., Erickson, D., Covington, G. A., Appenzeller, G., and McKeown, N. (2008). Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9. ACM.

NetFPGA (2015a). Netfpga cml. Disponível em: http://netfpga.org/site/#/systems/2netfpga-1g-cml/details/. Acessado em: 27/12/2015.

NetFPGA (2015b). Netfpga sume. Disponível em: http://netfpga.org/site/#/systems/1netfpga-sume/details/. Acessado em: 27/12/2015.

NetFPGA (2015c). Project switch openflow netfpga. Disponível em: https://github.com/NetFPGA/netfpga/wiki/OpenFlowNetFPGA100. Acessado em: 22/11/2015.

ONF (2012). Software-defined networking: The new norm for networks. Disponível em: http://www.opennetworking.org. Acessado em: 23/11/2015.

OpenNetworkingLab (2015). Controller pox. Disponível em: https://openflow.stanford.edu/display/ONL/POX+Wiki. Acessado em: 27/12/2015.

Prasad, R., Dovrolis, C., Murray, M., and Claffy, K. (2003). Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6):27–35.

Rasley, J., Stephens, B., Dixon, C., Rozner, E., Felter, W., Agarwal, K., Carter, J., and Fonseca, R. (2014). Planck: millisecond-scale monitoring and control for commodity networks. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 407–418. ACM.

Van Adrichem, N. L., Doerr, C., Kuipers, F., et al. (2014). Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8. IEEE.

Wellem, T., Lai, Y.-K., and Chung, W.-Y. (2015). A software defined sketch system for traffic monitoring. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, pages 197–198. IEEE Computer Society.

Yassine, A., Rahimi, H., and Shirmohammadi, S. (2015). Software defined network traffic measurement: Current trends and challenges. *Instrumentation & Measurement Magazine, IEEE*, 18(2):42–50.

Yu, M., Jose, L., and Miao, R. (2013). Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42.

Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M. H., and Zhang, M. (2015). Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 523–536. ACM.