RACYUS DELANO GARCIA PACÍFICO

ARQUITETURAS EM HARDWARE PARA MONITORAMENTO EXATO E APROXIMADO DO TEMPO DE CHEGADA ENTRE PACOTES EM REDES DE COMPUTADORES

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS – BRASIL
2016

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
P117a
2016

Pacífico, Racyus Delano Garcia, 1990-
Arquiteturas em hardware para monitoramento exato e aproximado do tempo de chegada entre pacotes em redes de computadores / Racyus Delano Garcia Pacífico. – Florestal, MG, 2016.
xiv, 75f. : il. (algumas color.) ; 29 cm.

Inclui apêndice.
Orientador: José Augusto Miranda Nacif.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.60-62.

1. Computadores - Equipamento de entrada e saída. 2. Redes de computadores. 3. Software. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22 ed. 004.2

RACYUS DELANO GARCIA PACÍFICO

ARQUITETURAS EM HARDWARE PARA MONITORAMENTO EXATO E APROXIMADO DO TEMPO DE CHEGADA ENTRE PACOTES EM REDES DE COMPUTADORES

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 05 de outubro de 2016.

Alex Borges Vieira
(Coorientador)

Marcos Augusto Menezes Vieira
(Coorientador)

Fabrício Aguiar Silva

Ítalo Fernando Scotá Cunha

José Augusto Miranda Nacif
(Orientador)

Dedico essa dissertação as pessoas que me ajudaram ao longo da minha trajetória, as pessoas pobres que deixaram sua comunidade assim como eu em busca dos seus sonhos. Dedico também as pessoas que decidiram lutar contra o sistema de maneira justa e honesta perseverando em prol de um mundo melhor. Nunca desista do seu sonho, pois essa dissertação é a maior prova que é possível realizá-lo.

"Ninguém vai bater mais forte que a vida. Não importa como você vai bater e sim o quanto aguenta apanhar e continuar lutando, o quanto pode suportar e seguir em frente. É assim que se ganha."

(Rocky Bolboa)

Agradecimentos

Primeiramente à **DEUS** pois sem ele nada disso seria possível. Ele foi meu fiel amigo nessa caminhada árdua para realização desse sonho.

À minha família (meu pai **Paulo**, minha mãe **Maria Lúcia** e minha irmã **Thalynara**) pelos conselhos, o apoio e incentivo nos meus sonhos. Essas pessoas são o alicerce da minha vida em conjunto com DEUS e minha namorada.

À minha namorada **Pamela** por não ter terminado comigo. Ela teve muita paciência em relação à distância e sempre fez de tudo para nosso relacionamento. Ela é companheira, sete anos de estrada comigo, mesmo estando longe me apoiou com conselhos e conversas em prol do relacionamento. Espero que tenha paciência para os 4 ou 6 anos do doutorado.

À minha ex-orientadora **Alessandra** do IF SUDESTE MG Rio Pomba, que na gradução foi quem me incentivou a fazer o mestrado. Grande amiga que levo como exemplo de vida, espero um dia trabalharmos juntos como professores e pesquisadores.

À família Teodoro e Bragança (**Vó Nenem, Mãe Dona Vera, Pai Binael e irmãos Mariana, Lucas e João Vitor**) por me adotarem e serem minha família quando as coisas ficaram difíceis em Florestal.

Ao meu amigo **Pablo** aluno de graduação da UFMG pela parceria. Aprendi muito com ele sobre coisas da vida, TI e NetFPGA. Quando as coisas estavam difíceis na pesquisa e já tinha feito de tudo, ele foi quem muitas das vezes deu uma luz perante as dificuldades que pareciam não ter solução.

Ao meu orientador **José Augusto Miranda Nacif** pela oportunidade e ensinamentos. Graças a ele e aos desafios impostos tive a oportunidade de melhorar como ser humano, evoluir e valorizar ainda mais as coisas básicas da vida. Por mais difícil que tenha sido tudo, esse processo de aprendizagem foi muito bom para minha formação.

Aos meus coorientadores **Alex Borges Viera** e **Marcos Augusto Menezes Vieira** pela oportunidade. Desde o técnico sonho em trabalhar com pesquisa na área de redes de computadores, só não esperava que teria a oportunidade de trabalhar com lendas da área.

Ao companheiro **Thales** pela parceria nas disciplinas, o apoio e ensinamentos sobre a vida.

Aos meus colegas **Alison, Cristiane, Fábio, Felipe, Paôla, Renan, Marcos, Marques, Tiago e William** pela ajuda nas disciplinas e trâmites do mestrado.

Aos meus amigos do laboratório LESC UFV Florestal (**Danilo (DaniBoy)**, **Kristtoffer** e **compania**) pelos ensinamentos. Aprendi muito com vocês.

Aos técnicos **Celi** e **Renato** do laboratório LESC UFV Florestal por todo o suporte. O Renato tornou-se um grande amigo que vou levar como exemplo de vida. Aprendi com ele que só é possível realizar um sonho com dedicação, perseverança, paciência e fé em DEUS.

Ao **TI** da UFV Florestal, em especial ao Romário, Afrânio e Leo pelos serviços prestados. Eles forneceram todo o suporte necessário para minha pesquisa e atividades do laboratório.

Ao **Prof. Jugurta** (ex-coordenador do mestrado) e ao **Altino** (Secretário do DPI) da UFV por todo o suporte e assistência no mestrado.

Ao **Prof. Ricardo** da UFV pelos ensinamentos em *hardware* e por ter me deixado de recuperação na disciplina de arquitetura de computadores. Graças a aprendizagem obtida nas atividades da recuperação consegui ajudar muitos alunos da UFV Florestal.

Aos **alunos** da UFV Florestal pelo apoio, incentivo e ensinamentos.

À todas as pessoas que de certa maneira contribuíram para realização desse sonho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

PACÍFICO, Racyus Delano Garcia, M.Sc., Universidade Federal de Viçosa, outubro de 2016. **Arquiteturas em hardware para monitoramento exato e aproximado do tempo de chegada entre pacotes em redes de computadores.** Orientador: José Augusto Miranda Nacif. Coorientadores: Alex Borges Vieira e Marcos Augusto Menezes Vieira.

O monitoramento de redes é um campo ativo em pesquisas na área de redes de computadores. Entretanto, com a propagação da *Internet* e a criação de novos serviços de rede, realizar tarefas de monitoramento de um grande volume de tráfego de maneira acurada e em tempo real tornou-se algo complexo e difícil de ser realizado. Isto ocorre devido a maioria das abordagens tradicionais de medição serem implementadas em *software* na extremidade da rede. Esta abordagem gera alto custo e dificulta o monitoramento devido à configuração e implantação do *software* individualmente em cada nó. Medir de maneira acurada e em tempo real é importante, pois é possível detectar falhas na rede, ataques e verificar a qualidade de serviços fornecidos pelos provedores de *Internet* em tempo real. Neste trabalho apresentamos duas arquiteturas de medição em *hardware* para o monitoramento exato e aproximado da métrica tempo de chegada entre pacotes. Ambas arquiteturas realizam tarefas de monitoramento em tempo real independente dos nós hospedeiros da rede e rodam sobre a plataforma *NetFPGA* 1G. O objetivo deste trabalho é comparar a acurácia e a escabilidade de ambas arquiteturas. Para o protótipo de monitoramento exato utilizamos o projeto *OpenFlow*. Estendemos este projeto e realizamos medições que não eram possíveis de ser realizadas em abordagens tradicionais, por exemplo, medição de regras com agregador de fluxo. Para o protótipo de monitoramento aproximado, estendemos o projeto *Reference Nic*. Este projeto foi transformado em um *framework* de medição que escala independente do número de fluxos. Com os resultados obtidos, a arquitetura de monitoramento exato apresenta uma diferença relativa de 28 fluxos quase nula em relação a um *software* na extremidade da rede. Isso significa, que a arquitetura mede de maneira acurada como uma aplicação na folha da rede, com a diferença de não ser necessário instrumentar os nós hospedeiros. A arquitetura de monitoramento aproximado, entretanto, apresenta uma diferença relativa de 1% a 20% de 2.000 fluxos em relação ao *software*. Esta arquitetura mede

um volume de fluxos 70 vezes maior que a arquitetura exata sacrificando a acurácia pela escabilidade.

Abstract

PACÍFICO, Racyus Delano Garcia, M.Sc., Universidade Federal de Viçosa, October, 2016. **Hardware architectures for accurate and approximate monitoring of the packets interarrival time in computer networks.** Adviser: José Augusto Miranda Nacif. Co-Adviser: Alex Borges Vieira and Marcos Augusto Menezes Vieira.

Network monitoring is an active research field in computer networks. However, with the Internet evolution and the adoption of new network services, performing real-time, accurate monitoring of high volumes of network traffic is a challenging task mainly because traditional measurement strategies are implemented in software at the network end hosts. This strategy leads high costs and difficulties related to the deployment and configuration of measurement software in each end host. The ability to perform accurate and real-time network measurement is very important and enables the possibility to detect network faults, attacks, and certify the quality of network service providers in real time. In this work we propose 2 hardware measurement architectures to perform exact and approximate monitoring of the packet interarrival time. Both architectures perform real time monitoring tasks that do not depend on end hosts and are executed at the NetFPGA 1G platform. The objective of this work is to compare accuracy and scalability between the proposed architectures. In our exact measurement prototype we have used the OpenFlow project. We have extended this project to perform measurements that were not possible using traditional strategies as, for example, measuring aggregate flows. For the approximate measurement prototype we have extended the Reference Nic project. This project has been modified to become a measurement framework that scales to thousands of flows. We show that the exact monitoring measurement architecture presents values that are very close to the software measurements performed at the end hosts for 28 flows. Thus, our architecture accuratelly measures the traffic at the network leaf nodes with no need to instrument the end hosts. However, the approximate monitoring architecture presents a relative difference from 1% to 20% when compared to end host software measurement for 2.000 flows. This measurement architecture can handle 70 times more flows than the exact architecuture sacrificing accuracy for scalability.

1. Introdução geral

Tarefas de medição são pontos chave para a engenharia de tráfego moderna [Kekely et al., 2014]. Desde a década passada, o monitoramento tem sido um campo amplamente explorado na área de redes de computadores devido à complexidade em se medir um grande volume de tráfego de maneira acurada em tempo real [Van Adrichem et al., 2014]. Essa complexidade é atribuída à estrutura estática dos elementos de redes tradicionais que ao longo dos anos passaram a não atender à demanda do aumento do tráfego com a propagação da *Internet* e a criação de novos serviços de rede. Neste contexto, a maioria das abordagens de medição são realizadas por *software*, instrumentando a extremidade da rede. Isso gera alto custo e dificulta o monitoramento, pois o *software* deve ser instalado em cada nova máquina a ser monitorada [Kreutz et al., 2015].

Tarefas de monitoramento podem ser classificadas como métodos de medição ativos e passivos. Ambos os métodos de medição são utilizados para monitorar o tráfego da rede e também coletar estatísticas. No método ativo são inseridos pacotes adicionais (sondas) na rede. Estes pacotes sondas são utilizados para estimar as características da rede como, por exemplo, largura de banda ou taxa de perda de pacotes. O método de medição passivo, ao contrário do método ativo, não é necessário enviar pacotes sondas. Este método é mais acurado que o método ativo, pois a medição é realizada localmente por monitores de tráfego instalados na rede. Além disso, estes monitores não interferem na acurácia da medição como os pacotes sondas enviados pelo método ativo. A principal desvantagem deste método está no custo de instalação dos monitores [Ishibashi et al., 2004].

Em se tratando de medições passivas, existem diversas estratégias de medição que podem ser implementadas nos elementos de rede. Cada estratégia é utilizada para uma tarefa de monitoramento específica. Existem três tipos de estratégias que podem ser utilizadas para implementar tarefas de medição: (i) contadores por fluxo; (ii) estruturas de dados baseadas em *hash* e (iii) fragmentos de código na CPU do comutador. Contadores por fluxo permitem uma medição acurada, mas consomem recurso de CPU. Estruturas de dados baseadas em *hash* realizam medições aproximadas de um grande volume de tráfego. Esta estratégia tem como desvantagens o custo de implementação do *hardware* e menor acurácia nas medições em relação aos contadores por fluxo. Fragmentos de código na CPU do comutador possibilitam coletar e analisar medições localmente no comutador com menor granularidade de tempo. Suas principais desvantagens estão relacionadas ao custo de implementa-

ção/configuração e à limitada capacidade de processamento do comutador [Moshref et al., 2013].

Neste trabalho foram desenvolvidos módulos em *hardware* para medição exata e aproximada de métricas de rede. Estes módulos foram implementados sobre os projetos *OpenFlow* [Naous et al., 2008] e *Reference Nic* disponíveis no repositório da plataforma *NetFPGA* 1G [NetFPGA, 2016]. O projeto *OpenFlow* consiste em um comutador com suporte *OpenFlow* que apenas realiza a contagem do número de pacotes e *bytes* por fluxo trafegado. Para realização da medição exata, o projeto *OpenFlow* foi estendido com a implementação de contadores por fluxo. Com a adição de contadores por fluxo e alteração do projeto, este projeto foi transformado em uma API de medição de métricas de rede. O projeto *Reference Nic* configura o *hardware NetFPGA* em uma interface de rede *gigabit*. Este projeto foi utilizado para medição aproximada, que consiste em medir um grande volume de tráfego sacrificando a acurácia da medição. O projeto *Reference Nic* foi estendido com a implementação de estruturas de dados baseadas em *hash*. Utilizamos estruturas de dados *Bloom Filters* armazenadas na SRAM para marcação do tempo de chegada dos fluxos. Além disso, este projeto foi transformado em um *framework* de medição escalável, que é capaz de medir até 2.000 fluxos em intervalos menores ou iguais a 1 segundo.

Para validação dos protótipos foi criado um ambiente de teste realista. Como estudo de caso inicial, a métrica tempo de chegada entre pacotes foi implementada em ambos os protótipos. Em relação à validação dos protótipos, atrasos pré-definidos foram inseridos no envio dos pacotes e comparados com os atrasos medidos por um *software* na extremidade da rede. Todos os testes realizados foram utilizados para validar os protótipos e também apresentar as vantagens e desvantagens das estruturas implementadas.

1.1 Objetivos

O objetivo deste trabalho é comparar a acurácia e a escabilidade de arquiteturas de medição exata e aproximada. Para alcançar o objetivo geral, os respectivos objetivos específicos foram atendidos:

- Desenvolver uma arquitetura em *hardware* para medição aproximada utilizando *hashs* para marcação dos fluxos em estruturas *Bloom Filters*;
- Estender um comutador com suporte *OpenFlow* em uma API de medição de métricas de rede;

- Criar e instrumentar um *software* de medição na folha da rede para se comparar com a medição realizada de ambas arquiteturas.

1.2 Resultados obtidos

Nos capítulos de 2 a 4 são detalhados os resultados obtidos. Estes capítulos foram escritos no formato de artigo. No capítulo 2 e 3 é descrita uma arquitetura de monitoramento exato que utiliza contadores por fluxo. No capítulo 4 é descrita uma arquitetura de monitoramento aproximado que utiliza *hash* e *Bloom Filters*. No apêndice A é apresentado os gráficos CDF (*Cumulative Distribution Function*) da média e variância dos resultados da arquitetura de monitoramento aproximado. O apêndice A é uma estensão do capítulo 4.

1.2.1 Arquitetura de monitoramento exato

Nos capítulos 2 e 3 é apresentada uma arquitetura de medição exata que utiliza do paradigma SDN (*Software Defined Networking*) e de contadores por fluxo para realizar medição acurada dos fluxos trafegados. O objetivo destes trabalhos é modificar um comutador com suporte *OpenFlow* para poder medir métricas de rede. Devido às restrições de recurso do *hardware*, o protótipo desenvolvido em ambos os trabalhos realiza medição de 28 fluxos simultaneamente de acordo com o intervalo definido pelo usuário.

As principais contribuições destes trabalhos são: (i) permitir o monitoramento da rede sem ter que instrumentar os nós folhas; (ii) verificar as condições de uma sub-rede como um todo, sem ter de medir ou saber quais são seus elementos de rede (isto não é possível de ser realizado com abordagens tradicionais); (iii) medir métricas de rede de acordo com as regras especificadas pelo controlador *OpenFlow* (por exemplo, medir apenas os fluxos com determinadas características).

Os resultados obtidos em ambos os trabalhos demonstraram que as medições do protótipo em *hardware* possuem diferença relativa próxima de 0% em relação a medição realizada por *software* na extremidade da rede. Isso implica que a medição desempenhada pelo *hardware* está próxima da medição realizada pelo *software* na extremidade da rede, com uma vazão maior de pacotes, logicamente centralizada e em tempo real.

O artigo do capítulo 2 "Medição de desempenho de rede em *hardware*" foi apresentado no XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2016, [Pacífico et al., 2016a]. No capítulo 3 o artigo "*Hard-*

ware Modules for Packet Interarrival Time Monitoring for Software Defined Measurements" que é uma versão reduzida e melhorada de [Pacífico et al., 2016a] foi publicado no *41st Annual IEEE Conference on Local Computer Networks* - LCN 2016, [Pacífico et al., 2016b].

1.2.2 Arquitetura de monitoramento aproximado

No capítulo 4 é apresentado uma arquitetura de medição aproximada que utiliza de estruturas de dados baseadas em *hash*. Nesta arquitetura é utilizado estruturas de dados *Bloom Filters* armazenadas na SRAM para marcação do tempo de chegada dos fluxos. Os objetivos principais deste trabalho são: (i) realizar a medição aproximada de um grande volume de fluxos utilizando *hashs* e *Bloom Filters* implementados sobre a plataforma *NetFPGA* 1G; (ii) comparar à acurácia e escabilidade da arquitetutura aproximada em relação a arquitetura exata de [Pacífico et al., 2016a].

O protótipo desenvolvido neste trabalho realiza medições de até 2.000 fluxos simultaneamente em tempo real sacrificando a acurácia da medição pela escabilidade. Em relação a [Pacífico et al., 2016a] a arquitetura proposta mede 70 vezes mais fluxos. As principais contribuições deste trabalho são: (i) medir um grande volume de fluxos sem precisar instrumentar as folhas da rede. (ii) implementação de estruturas baseadas em *hash* e *Bloom Filters* em *hardware*.

Este protótipo foi comparado com a abordagem em *software* e também a [Pacífico et al., 2016a]. Com base nessa comparação os resultados obtidos demonstraram que o protótipo proposto escala para um grande volume de fluxos, mas realiza uma medição menos acurada em relação à medição desempenhada por [Pacífico et al., 2016a] e o *software* na folha da rede, devido o método ser aproximado.

No capítulo 4 é descrito o artigo "*BloomTime* - Arquitetura em *hardware* para monitoramento aproximado do tempo de chegada entre pacotes". Os resultados deste artigo foram submetidos no periódico *Computer and Communications Networks* - COMCOM.

2. Medição de desempenho de rede em *hardware*[1]

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 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 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]Neste capítulo é apresentado o artigo "Medição de desempenho de rede em *hardware*". Este artigo foi apresentado no XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2016, [Pacífico et al., 2016a].

2.1 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 consomem 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, 2015] 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 2.2 são apresentadas as principais técnicas de medição em SDN. Em seguida (seção 2.3) são abordados os detalhes da arquitetura proposta e da implementação do comutador *OpenFlow* modificado na *NetFPGA*. Na seção 2.4 é descrito o ambiente de testes e a metodologia. Na seção 2.5 são apresentados os resultados dos testes reais. Na seção 3.5 são abordados os trabalhos relacionados sobre métricas de medição, técnicas e ferramentas para realizar medições em redes. Finalmente, na seção 2.7 apresentamos nossas conclusões e trabalhos futuros.

2.2 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 & 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 3.5.

2.3 Arquitetura proposta

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 em 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 2.1 é apresentado a arquitetura do protótipo implementado.

2.3.1 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*)

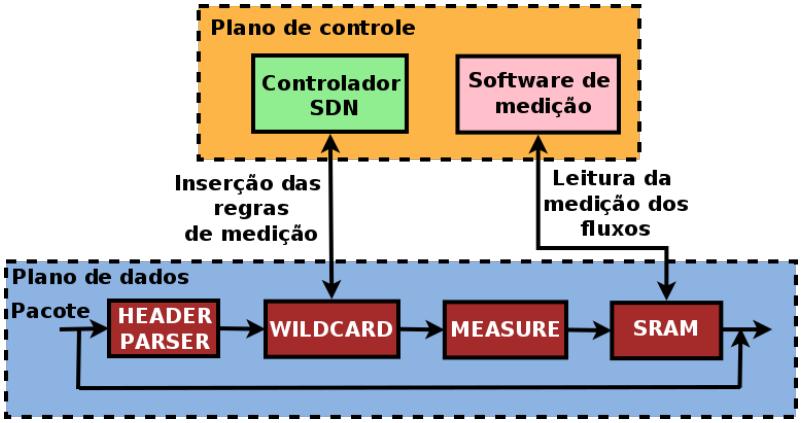


Figura 2.1: Arquitetura *OpenFlow* com suporte à medição do tempo de chegada entre pacotes.

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 2.1) e variância (equação 2.2) foram realizadas no *software* de medição. Na figura 2.2 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*.

$$media_j = \frac{\sum_{i=1}^{N_j} dT_i}{(N_j - 1)}, \quad 1 \leq j \leq 28 \tag{2.1}$$

$$variancia_j = \frac{\sum_{i=1}^{N_j} (dT_i)^2 - (\sum_{i=1}^{N_j} dT_i)^2 / N_j}{(N_j - 1)}, \quad 1 \leq j \leq 28 \tag{2.2}$$

Atualmente existem modelos de plataformas NetFPGA com FPGAs de maior capacidade, por exemplo, a CML [CML, 2015] e a SUME [SUME, 2015]. 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.

2.3.2 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

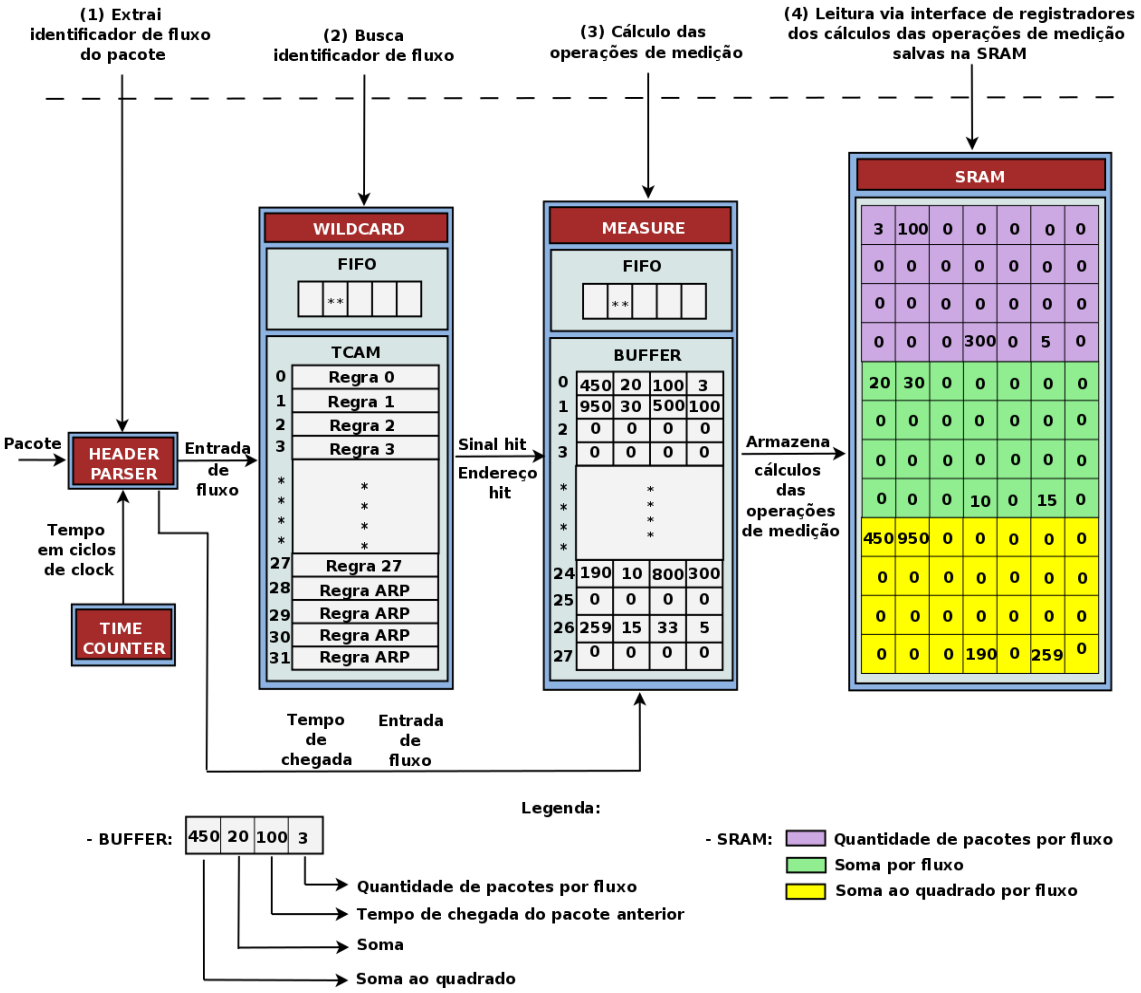


Figura 2.2: Caminho de dados *OpenFlow* com suporte a média e variância do tempo de chegada entre pacotes.

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.

2.4 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.

2.4.1 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 2.3 é apresentado o ambiente de testes.

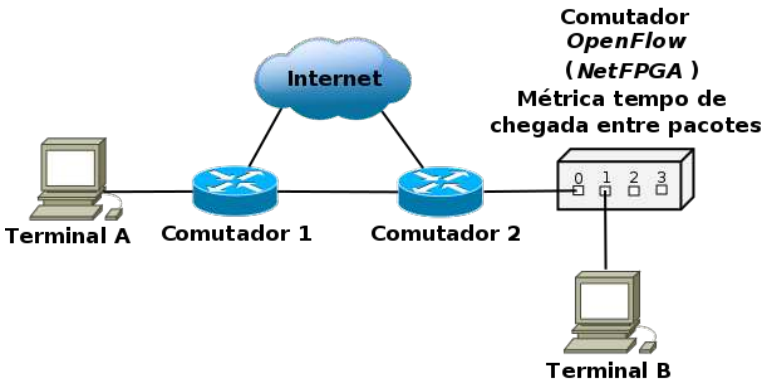


Figura 2.3: Ambiente de testes.

2.4.2 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.

2.4.3 *Warm up*

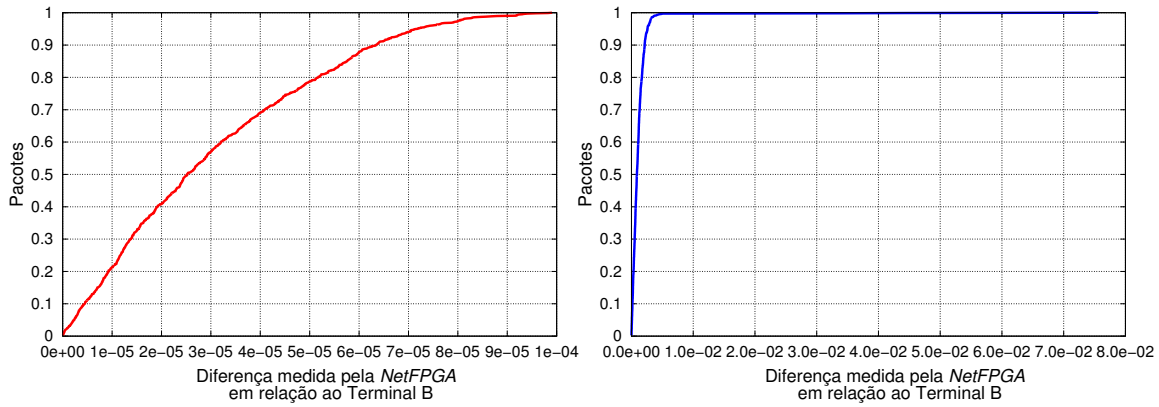
Os experimentos foram compostos por cinquenta iterações, tendo cada iteração a duração de 1 segundo. São gastos 5 segundos para inicializar o controlador, os processos de monitoramento e a transmissão de pacotes, e 1 segundo para finalizar todos os processos. O tempo total do experimento é o tempo de 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.

2.5 Resultados

Todos os experimentos foram configurados de acordo como descrito na seção 2.4.1. Os procedimentos descritos nas seções 2.4.2 e 2.4.3 foram utilizados na preparação do ambiente. Nesta seção são apresentados os gráficos e intervalos de confiança da média e da variança dos atrasos para cada experimento.

2.5.1 Experimento 1

Nesse experimento foram enviados aproximadamente 1.700 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 $10^{-3} \times [1.0043 : 1.3574]$.

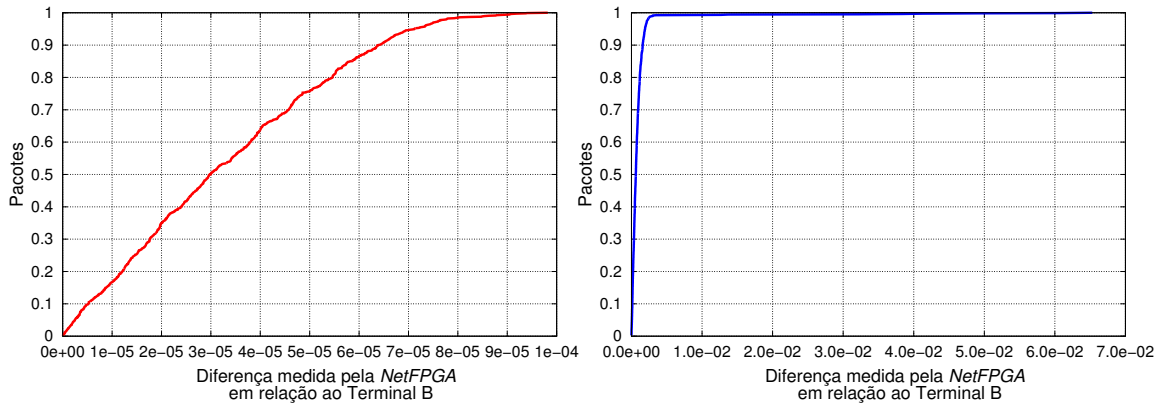


(A) **(B)**

Figura 2.4: Gráficos CDF diferença relativa média(A) e variância(B) atraso 20/40 ms.

2.5.2 Experimento 2

Nesse experimento foram enviados aproximadamente 1.800 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 $10^{-4} \times [7.9642 : 12.4338]$.

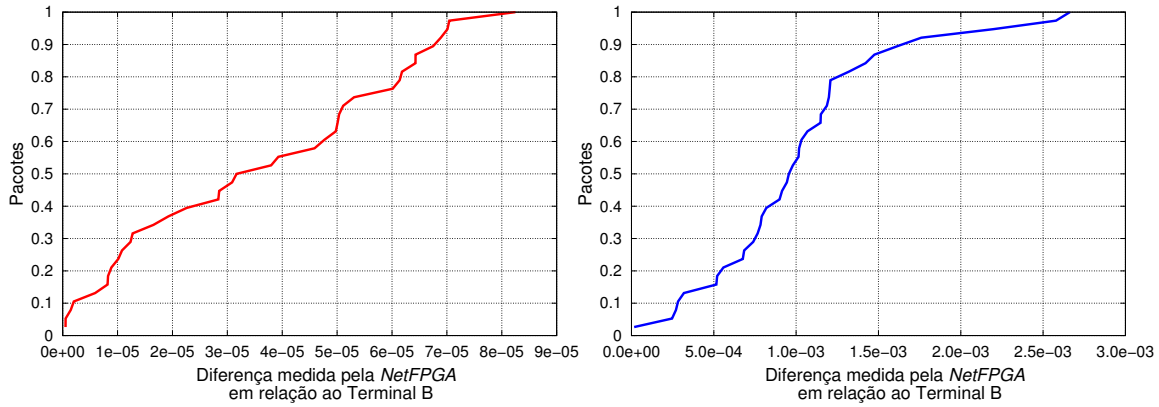


(A) **(B)**

Figura 2.5: Gráficos CDF diferença relativa média(A) e variância(B) atraso variado.

2.5.3 Experimento 3

Nesse experimento foram enviados 1.700 pacotes com atrasos pré-definidos 20/40 ms 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 $10^{-4} \times [8.4223 : 12.072]$.



(A) **(B)**

Figura 2.6: Gráficos CDF diferença relativa média(A) e variância(B) regra wildcard.

2.6 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 & 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.

2.7 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.

3. Hardware Modules for Packet Interarrival Time Monitoring for Software Defined Measurements[1]

Abstract

Measurement and tracking have crucial roles in Software-Defined Networks (SDNs). Unfortunately, most of procedures and techniques to perform measurements and monitoring tasks are implemented in software at network end-hosts. Despite the large use, a software based approach generates high costs and makes monitoring more difficult. In this paper, we extend OpenFlow switch to implement a measurement architecture for SDN networks. Our system performs measurements in a simple and scalable way without depending on end-hosts. It allows monitoring the performance at the granularity of flows. Moreover, our system also enables software-defined measurements since the controller can collect flow's statistics on the fly. We have prototyped our architecture on the NetFPGA platform and, as an initial case study, we have implemented a module to measure packet interarrival time. This module has been validated in a realistic testbed environment. Our results demonstrate that the proposed architecture presents a negligible difference when compared to measurements performed by software at end-hosts.

3.1 Introduction

Software Defined Networks (SDNs) is an emerging network architecture that enables management of computer networks and tracking in programmable way. This paradigm consists of the separation between the control and data planes on switches [ONF, 2012]. In SDN, the control plane manages the network devices centrally by software and the data plane forwards packets according to the actions defined by the controller [Naous et al., 2008]. Several efforts have been made to develop SDN control applications, leaving a gap to be filled in SDN measurement and traffic tracking systems [Yu et al., 2013]. Network measurement and tracking also enables

[1]Neste capítulo é apresentado o artigo "*Hardware Modules for Packet Interarrival Time Monitoring for Software Defined Measurements*". Este artigo foi publicado no *41st Annual IEEE Conference on Local Computer Networks* - LCN 2016, [Pacífico et al., 2016b].

the implementation of traffic engineering techniques. However, developing these techniques for SDN networks is challenging due to the complexity and real-time constraints of these applications.

Traffic tracking plays a crucial role to enable traffic engineering in modern computer networks. Although this area was widely exploited in the past, SDN features require novel traffic engineering techniques to enable performance optimization in SDN networks [Agarwal et al., 2013; Akyildiz et al., 2014]. SDN-based traffic tracking techniques are more feasible to implement because they do not require large investments in hardware resources or network configuration. OpenFlow is an example of SDN and has emerged as a standard for communication between the controller and switches. An OpenFlow switch uses per flow counters of packet and bytes.

In this paper, we extend OpenFlow switch to implement a measurement architecture for SDN networks, enabling Software-Defined Measurements (SDM) since the controller can collect flow's statistics on the fly. We also perform measurement in a simple and scalable way without depending on end-hosts, allowing to track the performance at the granularity of flows. Our architecture has been prototyped extending the OpenFlow switch on NetFPGA 1G platform [Naous et al., 2008; NetFPGA, 2015], enabling accurate and real-time measurements of TCP flows.

As an initial case study, we have implemented a module to measure packet interarrival times. This module has been validated in a realist testbed environment, showing that our architecture has a negligible difference when compared to measurements performed in software at end-hosts. In fact, in our experiments, the difference between the traditional (end-host) measurements and our new proposal is close to 0%. Moreover, we present scenarios where end-host measurement is impracticable as, for example, to measure aggregated flows at a single point.

3.2 Prototype Implementation

In this work, we adopted the open source OpenFlow switch especially designed for the NetFPGA platform [NetFPGA, 2015]. Recall that SDN paradigm migrates network complexity from forwarding network elements (switches/routers) to a centralized controller. As the centralized controller has a complete network view, it may perform fault detection and may improve network measurement tasks. Moreover, controllers allow association of flow rules inside network switches for measurement tasks, enabling to capture metrics and timestamps of the network flows using user-

space programs [Naous et al., 2008].

We have modified the original OpenFlow switch [Naous et al., 2008] to gather flows mean and variance of packet interarrival time. We track packet interarrival times in flow scale, performing measurements for all packets of a flow without sampling, which scales according the number of rules supported in the switch. Currently, we track TCP flows concurrently performing measurements exactly in line-speed without affecting the packets processing rate.

Our architecture supports the description of flows in the flow entry table with the wildcard. The wildcard is useful to aggregate flows. For example, to detect if someone is doing portscan in a machine, the user could measure the packet interarrival time on her machine for any ports (wildcard port rule).

3.2.1 Data plane

Measurement tasks in the data plane begin with incoming TCP packets in *Header Parser* module. This module extracts the flow identifier, saves the timestamp (in clock cycles) that is generated by the module *Time Counter*, and forwards this data to the *Measure* module. The *Measure* module inserts these values inside a queue. It needs to wait an output from the *Wildcard* module to decide if it drops the timestamp and flow identifier or calculation of mean and variance. The module *Wildcard* matches the flow identifier extracted from the packet with the flow entries stored inside the TCAM. After ending the query, the module *Wildcard* sends a signal if it found the flow identifier and the TCAM address.

In next step, the *Measure* module performs calculations of mean and variance of packet interarrival time. The implementation of this metric consists of receiving the arrival time of the first packet (T_1) of the flow F_j, the arrival time of the next packet (T_2) of the flow F_j until the arrival time (T_n) of the nth packet of the flow F_j. F_j refers to the j^{th} measured flow. At each interval among packets, our prototype calculates the difference $dT = T_c - T_p$, being T_c the arrival time of the current packet and T_p the arrival time of the previous packet. Following, the summation of the difference S_j is computed, $S_j = S_j + dT$ and the summation of the squared sum of the difference, $S_j^2 = S_j^2 + dT^2$. Finally, the packet counter N_j is incremented.

After all calculations are performed, the results are temporarily stored in a buffer inside the *Measure* module and also in the SRAM. The buffer has one entry for each TCAM entry. Each buffer entry stores: the number of packet of a specific flow (N_j); the arrival time of the last packet received for that flow (T_p); the sum (S_j) and squared-sum (S_j^2) of the accumulated time intervals of the packet arrival

times. When a matching occurs in the TCAM, the TCAM matched address will also address the buffer as well, where the information will be updated according the to timestamp of the current packet (T_c). The buffer works as a cache. Its main goal is to avoid the time that would be spent with repeated accesses to the SRAM. SRAM spends three clock cycles to read instructions, which may decrease the performance due to the time that the state machine waits for the data from the memory [Goulart et al., 2015].

The user-space program reads, via register interface, the packet interarrival times recorded in the SRAM. Due to the limitation of logic cells number, division operations of the mean and variance are performed by a user-space program. In Figure 3.1, we show the datapath of our proposed architecture that supports measuring the mean and variance of the packet interarrival time. We highlight that our architecture is generic and allows to incorporate other metrics, which we intend for future work.

Currently, the NetFPGA 1G model, which was used in this project, supports twenty-eight TCP flows concurrently due to NetFPGA TCAM size constraints. There are other NetFPGA models with more powerful FPGAs. For comparison effects, the NetFPGA SUME [SUME, 2015] (FPGA Virtex-7) has 13x more resources than the model 1G, allowing to track 416 flows concurrently.

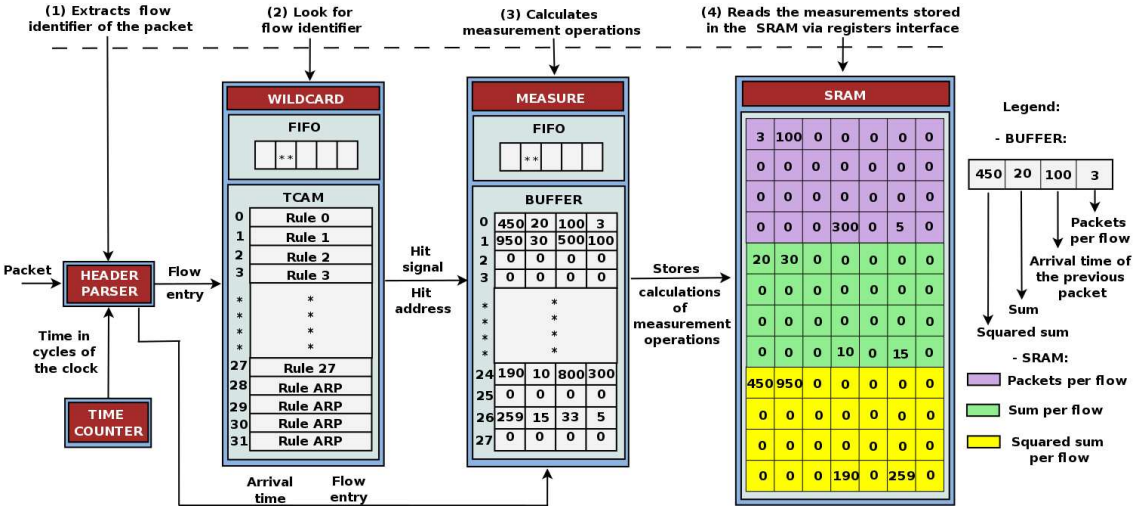


Figure 3.1: Datapath of the implemented NetFPGA switch.

3.2.2 Control Plane

The control plane is divided in two parts: a controller and a measurement software. The controller's function is to insert the rules inside the TCAM and to inform the

switch how the packet will be forwarded. There are several SDN controllers options already implemented, each one with particular characteristics. We used the POX controller. POX is implemented in Python and enables fast prototyping.

We developed an application on top of POX that insert rules containing the desired flows to be monitored on the switch. In this way, it is possible for the switch to monitor the packet interarrival time of the desired flows on the fly.

The measurement software defines the measurement process performing calculations of the mean and variance of each flow. This software communicates with the NetFPGA through the register interface. The software reads the values of mean and variance that are stored in he NetFPGA's SRAM. For each reading, the mean and variance of the requested flows are calculated and stored in a file.

3.3 Evaluation

Figure 3.2 depicts our testbed. Our testbed is composed of two end-hosts: A and B. Host A sends Raw TCP packets at a controlled rate to host B. Our testbed has, between A and B, two traditional switches with real traffic for adding realistic delays among the packets sent by A. At the end point of the network, we inserted the NetFPGA running our prototype. Host B executes a software tool to perform packet interarrival time measurements. In B, the mean and variance were calculated after ending the experiment. In our prototype, the mean and the variance were calculated on the fly. Our goal is to compare the relative difference between the measurements in software (B) and the ones in hardware (NetFPGA) based on the predefined delays inserted among the packets sent by A.

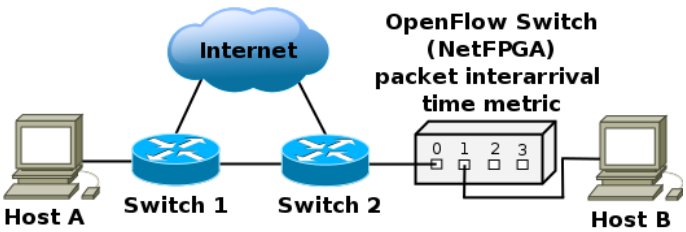


Figure 3.2: Testbed environment.

We developed an user-space application to send TCP packets from the host A. This application reads a file that stores the rules, the same one used by the controller. This program starts a new thread for each flow defined in the file. Each thread sends packets for a specific flow.

We defined two types of interarrival time tests: constant and random. We used constant time intervals in the validation phase for matching with the values given by our prototype. For realistic tests, we used random time intervals. The packets were generated with `Libnet` library of C language. For enabling better control of the number of packets that came from A to B and NetFPGA, we wrote the packets sent by `Libnet` in several pcap files to be transmitted by A. Each pcap file lasts one second. This approach enables performing comparisons easier and accurately. Traces format pcap were also created in the host B (software) and reproduced using DPKT library of Python language.

We performed several iterations of the experiments running the pcap files in the terminal A. Before executing them, we initialized the `tcpdump` in B for capturing packets that came from A in that moment. When ending an iteration, we read the measurements, clean both the internal buffers and the SRAM (first thirty-two lines). Then, the `tcpdump` writes an output file in B with the captured packets. These steps were performed in all iterations.

Initially, we set up five seconds for initializing the controller, the monitoring tools, and the packet transmission. Then, all experiments are performed in fifty iterations, each iteration lasting one second. Finally, we configured one second for ending these processes. Thus, the total experiment time is the summation of the initialization time, with the product of each iteration by the number of iterations, and the ending phase time: $5s + 50 \times 1s + 1s = 56s$. We discarded 10% of the samples due to the warm up period, which means 6 seconds at the beginning and ending. Each second corresponds to twenty-eight samples.

3.4 Results

We designed three types of experiments. First experiment consists of predefined delays. This enables us to have the ground truth and, consequently, to validate our system. The second experiment is configured to have random delays and it represents realistic traffic pattern. Finally, the third experiment evaluates our prototype with the wildcard rules and predefined delays. Figure 3.3 presents mean and variance results for experiments we have performed.

Experiment 1 checks fixed delay. We send approximately 1700 packets with predefined delays of 20/40 milliseconds (ms) for each one of the 28 flows. Our results show that 95% of the relative difference between hardware and software approaches are close 0.0072% in the mean graph. On the variance graph, 95% of

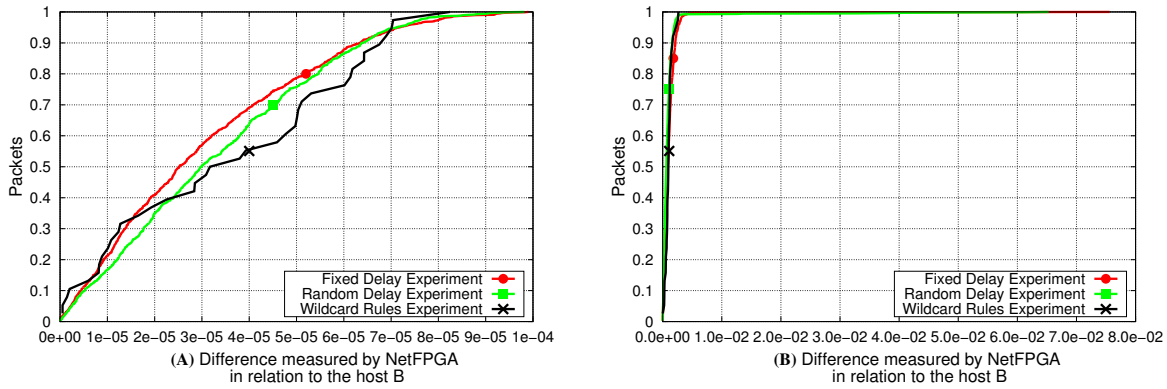


Figure 3.3: CDF plot of the mean **(A)** and variance **(B)**.

the measurements obtained relative difference close to 0.26%. Three values stayed between 3% and 7%, but the absolute differences are small, presenting differences less than 10^{-4}. The expected values of the mean are inside the confidence interval $10^{-5} \times [2.897 : 3.166]$, and the expected values of the variance are inside the confidence interval $10^{-3} \times [1.0043 : 1.3574]$.

Experiment 2 tests random delay. We send approximately 1800 packets with random delays for each one of the 28 flows. We defined three seeds to generate these delays. The graph of mean showed that 95% of the measurements obtained a difference close to 0.0071%. In the graph of the variance, 95% of the measurements obtained a relative difference close to 0.19%. Also, in the graph of the variance six samples were between 3% and 6% of relative difference, but the absolute differences are small (less than 10^{-4}). The confidence interval of the graph of mean is $10^{-5} \times [3.169 : 3.432]$, and the expected values of the variance are inside the confidence interval $10^{-4} \times [7.9642 : 12.4338]$.

Experiment 3 is about wildcard rules. We sent approximately 1700 packets from host A with predefined delays 20/40ms for only four flows. We aimed to measure all these flows inside one wildcard rule. Our goal here is to show that our architecture allows measurements of aggregated flows, allowing us to track entire subnets. The graph of mean shows that 97% of the measurements obtained has a difference close to 0.007%. In the graph of the variance, 97% of the measurements had a relative difference close to 0.257%. The expected values of mean are inside the confidence interval $10^{-5} \times [2.788 : 4.349]$, and the expected values of the variance are inside the confidence interval $10^{-4} \times [8.4223 : 12.072]$.

Summary: The experiments show that the proposed architecture presents a negligible difference when compared to measurements performed by software at end-hosts.

3.5 Related Work

Moshref et. al. Moshref et al. [2013] discuss the tradeoff between amount of network resources and accuracy of different measurement primitives. The paper aims to evaluate measurement techniques based on counters, hashing structures and CPU programming to detect heavy hitter flows. The prototype evaluation uses different time scales and configuration in the resources allocated for each switch. They used CAIDA traces in the evaluation.

OpenSketch Yu et al. [2013] is a Software-Defined Measurement architecture based on the SDN paradigm and uses primitives' measurement structures called sketches. Sketches are data structures that are scalable but can only provide approximation results instead of exact computation. The OpenSketch control plane is composed by one library while the data plane is separated in three stages: hashing, classification, and countering. The validation was performed by comparing the results obtained from NetFlow and the OpenSketch prototype.

OpenNetMon Van Adrichem et al. [2014] is an open source software to perform monitoring metrics per flow, specially delay, throughput, and packet losses in OpenFlow networks. This implementation provides traffic engineering on the controller with online measurements. Packet loss is calculated with the flow statistics from the first and the last switch of each path. They subtract the packet counter from the source and destination switches. Delays and packet losses were statically defined with the Netem Linux tool.

These works illustrate that monitoring the network is important. But, none of them modified the data plane (switches) to collect new metrics beside the ones provided by OpenFlow interface, as we did.

3.6 Conclusion

Traffic tracking is a key point to guarantee the quality in IP networks. Here, we designed and implemented a measurement architecture for SDN. We implemented a module to measure packet interarrival time. Our architecture allows monitoring at the granularity of flows. Our system also enables software-defined measurements since the controller can collect flow's statistics on the fly.

We have prototyped our architecture on the NetFPGA platform. To validate our prototype, we compared the proposed architecture to measurements performed by software at end-hosts. Results show that the relative difference is almost zero,

validating our architecture. Moreover, our architecture has better throughput performance, can collect data on the fly, and enable collecting network's statistics in the middle on the network instead of just at the end-hosts.

4. *BloomTime* - Arquitetura em *hardware* para monitoramento aproximado do tempo de chegada entre pacotes[1]

Abstract

Network monitoring is an active research field in computer networks. However, to perform monitoring tasks for a large volume of tracffic accurately and in real time is complex and challenging task. This occurs because most approaches are implemented in software at the network end hosts. These approaches generate high costs due to the software configuration and implementation in each network node. In this paper, we propose the BloomTime monitoring architecture. BloomTime uses hash-based structures and Bloom Filters implemented in the NetFPGA 1G hardware to perform approximate monitoring. Our architecture measures up to 2.000 flows in a scalable way in a realistic test environment. As a case study we have implemented the packet interarrival time metric and compared the measurements performed by the architecture with two exact approaches (software and hardware). We have used the software approach to validate our prototype measurements. Our results show that the BloomTime architecture measures 70 times more flows than the hardware exact monitoring prototype with a relative difference less than 20%, thus sacrificing accuracy for scalability.

Resumo

O monitoramento de redes é um campo ativo em pesquisas na área de redes de computadores. Entretanto, realizar tarefas de monitoramento para um grande volume de tráfego de maneira acurada em tempo de execução é algo complexo e desafiador. Isto ocorre porque a maioria das abordagens são implementadas em *software* na extremidade da rede. Estas abordagens geram alto custo devido a configuração e implantação do *software* em cada nó da rede. Neste trabalho, apresentamos a arquitetura de monitoramento *BloomTime*. *BloomTime* utiliza estruturas baseadas

[1]Neste capítulo é apresentado o artigo "*BloomTime* - Arquitetura em *hardware* para monitoramento aproximado do tempo de chegada entre pacotes". Os resultados apresentados neste artigo foram submetidos no periódico *Computer and Communications Networks* - COMCOM.

em *hash* e *Bloom Filters* implementados no *hardware NetFPGA* 1G para realizar monitoramento aproximado. Nossa a arquitetura mede até 2.000 fluxos de maneira escalável em um ambiente de teste realista. Como estudo de caso implementamos a métrica tempo de chegada entre pacotes e comparamos a medição realizada pela arquitetura com duas abordagens (*software* e *hardware*) que realizam monitoramento exato. A abordagem em *software* foi utilizada para validarmos a medição do protótipo. Nossos resultados mostram que a arquitetura *BloomTime* mede 70 vezes mais fluxos que o protótipo em *hardware* de monitoramento exato com uma diferença relativa menor que 20%, mas sacrifica a acurácia das medições pela escabilidade.

4.1 Introdução

Tarefas de monitoramento do tráfego são indispensáveis para a engenharia de tráfego moderna [Chowdhury et al., 2014]. Entretanto, realizar tarefas de monitoramento de maneira acurada para um grande volume de tráfego tornou-se algo complexo utilizando os elementos de rede IP tradicionais. Operadores de rede neste contexto se deparam com dificuldades como: detectar falhas na rede, monitorar ataques e realizar medições em escalas de tempo distintas [Yassine et al., 2015].

As tarefas de monitoramento podem ser classificadas com base em dois métodos de medição: ativo e passivo. No método de medição ativo são injetados pacotes extras (sondas) na rede para monitoramento. Este método pode sobrecarregar a rede e prejudicar a acurácia da medição devido ao envio de pacotes extras. No método de medição passivo, ao contrário do método ativo, não é necessário enviar pacotes sondas. Neste método a medição é mais acurada que o método ativo e a medição ocorre localmente por monitores de rede. A principal desvantagem deste método é o grande investimento para instalação e configuração dos monitores de rede [Van Adrichem et al., 2014].

Em abordagens que utilizam medição passiva, as tarefas de monitoramento podem ser implementadas utilizando contadores por fluxo, *hash* ou programas na CPU do comutador [Moshref et al., 2013]. Realizar tarefas de monitoramento usando de contadores por fluxo é algo desafiador devido ao consumo de recursos de CPU em virtude da demanda do monitoramento e granularidade. Outras técnicas como *hash* e programas na CPU do comutador necessitam de grande investimento no projeto do *hardware* e configuração das rotinas de medição [Van Adrichem et al., 2014].

Neste trabalho nós desenvolvemos uma arquitetura de medição aproximada utilizando *Bloom Filters* implementados no *hardware NetFPGA* 1G. A arquitetura

proposta realiza medições em tempo real de até 2.000 fluxos em um cenário de teste realista. Como estudo de caso implementamos a métrica tempo de chegada entre pacotes e comparamos a acurácia *versus* escabilidade em relação a [Pacífico et al., 2016a] que estendeu o comutador *OpenFlow* para medir o tempo de chegada entre pacotes usando de um método exato baseado em contadores por fluxo.

As principais contribuições deste trabalho são: (i) realizar à medição de um grande volume de tráfego de modo escalável e com baixa granularidade de acordo com os recursos disponíveis pelo *hardware*; (ii) implementação de uma arquitetura de medição aproximada utilizando de *Bloom Filters* sendo executados sobre a plataforma *NetFPGA*. Os resultados obtidos demonstram que a arquitetura proposta mede um volume de fluxos 70 vezes maior que [Pacífico et al., 2016a], mas realiza uma medição menos acurada devido ao método de medição ser aproximado.

O artigo tem a seguinte organização: Na seção 4.2 é apresentado uma visão geral sobre estruturas de dados *Bloom Filter*. Em seguida (seção 4.3) é abordado o funcionamento do caminho de dados e o espaço do usuário da arquitetura. Também nesta seção são abordados os detalhes de implementação das estruturas *Bloom Filters* no *hardware NetFPGA*. Na seção 4.4 é descrito o ambiente de testes e os métodos utilizados nos experimentos. Na seção 4.5.2 são apresentados os resultados obtidos em um ambiente de testes realista. Na seção 4.6 são descritas as caraterísticas dos trabalhos relacionados com base na métrica, método e ambiente de medição. Finalmente, na seção 4.7 são apresentadas as conclusões do trabalho em conjunto com os trabalhos futuros.

4.2 Visão geral estrutura *Bloom Filter*

O *Bloom Filter* é uma estrutura de dados probabilística baseada na teoria do conjunto. Essa estrutura é muito utilizada quando o espaço de armazenamento é um problema, pois proporciona de modo compacto o armazenamento dos dados. Em elementos de rede (comutadores e roteadores) *Bloom Filters* são usados para armazenar e processar o resumo dos dados medidos [Tarkoma et al., 2012].

A estrutura *Bloom Filter* é representada por um conjunto S=$\{x_1,x_2,...,\ x_n\}$ de n elementos armazenados em um arranjo de m *bits*. Inicialmente todas as posições do arranjo são inicializadas com zero. O *Bloom Filter* utiliza k funções *hashs* distintas $h_1,...,h_k$ para mapear um item no arranjo. Essas funções também podem ser endereçadas para qualquer posição do arranjo. Um elemento x pertence ao conjunto S, ou seja, está inserido no *Bloom Filter* quando os *bits* do arranjo que $h_i(x)$

endereça são iguais a 1 para $1 \leq i \leq k$.

Em *Bloom Filters* tradicionais apenas operações de inserção e consulta são possíveis de serem realizadas. Um problema geral de estruturas *Bloom Filter* é a geração de falsos positivos. Falsos positivos ocorrem quando é feita a consulta de um item que não foi inserido no conjunto S e essa consulta retorna um valor verdadeiro. A probabilidade da ocorrência de falsos positivos pode ser calculada com base na equação 4.1. Nesta equação, n é definido como o número de fluxos, m o número de posições do *BloomFilter* e k o número de funções *hash* utilizadas no endereçamento do fluxo no *BloomFilter*. Para realizar este cálculo deve-se considerar que a função de *hash* é perfeita, ou seja, que a função endereça todas as posições do arranjo sem colisão. Para encontrar o número ótimo de *hashs* que endereçam as posições do *BloomFilter* é utilizado a equação 4.2. Nesta equação, m e n são os mesmos que na equação 4.1 [Broder & Mitzenmacher, 2004].

$$falsos\ positivos = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-kn/m}\right)^{k} \tag{4.1}$$

$$k = \frac{m}{n} ln(2) \tag{4.2}$$

Na figura 4.1 é apresentado o funcionamento da estrutura *Bloom Filter*. Na letra A o *Bloom Filter* foi inicializado com zeros. Em B é inserido o fluxo X. Os endereços *hashs* gerados para este fluxo são mapeados para as posições 0 e 6. Se estas posições estiverem com 0, será atribuído 1. Na letra C é consultado se o fluxo Y está no *Bloom Filter*. Este fluxo é mapeado para os endereços 2 e 4. Ambas as posições estão com valor um, isso significa que o fluxo está marcado no *Bloom Filter* e que também pode pertencer ao conjunto S ou ser um falso positivo. Neste trabalho utilizamos *Bloom Filters* para termos o resumo do tempo de chegada entre pacotes dos fluxos trafegados.

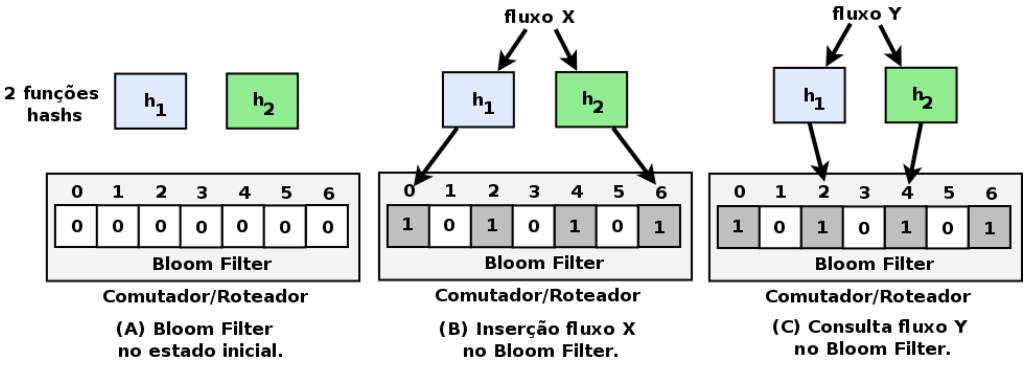


Figura 4.1: Exemplo funcionamento *Bloom Filter*.

4.3 Implementação da arquitetura

O protótipo é composto de 64 *Bloom Filters* armazenados na SRAM do *hardware*. Cada *Bloom Filter* representa uma janela de tempo, onde o tempo de chegada dos fluxos é marcado. Cada estrutura *Bloom Filter* é composta por 8.192 linhas. Definimos 2, 4, 8 e 23 milisegundos o tamanho da janela de tempo dos *Bloom Filters*. Esses tempos foram escolhidos para testarmos a acurácia do protótipo. Os fluxos são marcados nos *Bloom Filters* utilizando de duas funções *hashs*. A justificativa para utilizarmos duas funções *hash* é que esse é o número ótimo de funções para endereçar 8.192 posições do *Bloom Filter*. Este número é calculado com base na equação 4.2. O tempo de monitoramento das marcações dos fluxos nos *Bloom Filters* tem duração de 128 ms e assim por diante com base nas respectivas janelas de tempo. Esses tempos são iguais ao produto do número de *Bloom Filters* e a janela de tempo. Quando esse tempo é atigindo, o registrador em *hardware* é ativado e o *software* de medição realiza a leitura das marcações via interface de registradores. Após a leitura das marcações, os *Bloom Filters* na SRAM e os módulos de medição do *hardware* são inicializados.

Na figura 4.2 é apresentada uma visão geral da estrutura do protótipo. O protótipo é dividido em duas partes: caminho de dados e aplicações que rodam no espaço do usuário. O caminho de dados é composto por módulos que pertencem à biblioteca padrão da *NetFPGA* e módulos desenvolvidos para medição aproximada. O espaço do usuário é constituído pelo *software* de medição. O *software* de medição é responsável pela leitura das marcações dos *Bloom Filters* quando o tempo de monitoramento é alcançado. Em seguida, o *software* realiza o cálculo da média e variância com base nas marcações lidas da SRAM. Os cálculos da média e variância são realizados após as medições serem finalizadas.

Todo o processo de medição no *hardware* baseia-se em cinco estágios: contador de tempo em ciclos de relógio (*Time Counter*), extração do identificador de fluxo do pacote (*Header Parser*), geração de *hashs* (*Hash*), gerenciamento *Bloom Filters* (*Bloom Filters Control*) e armazenamento *Bloom Filters* na SRAM (*SRAM*).

4.3.1 Caminho de dados

O processo de medição no caminho de dados começa com a chegada de um pacote TCP no módulo *Header Parser*. Neste módulo o identificador de fluxo do pacote é extraído e encaminhado para o módulo *Hash*. Com a chegada do identificador de fluxo no módulo *Hash* são gerados dois endereços (*hash1* e *hash2*) que mapeiam

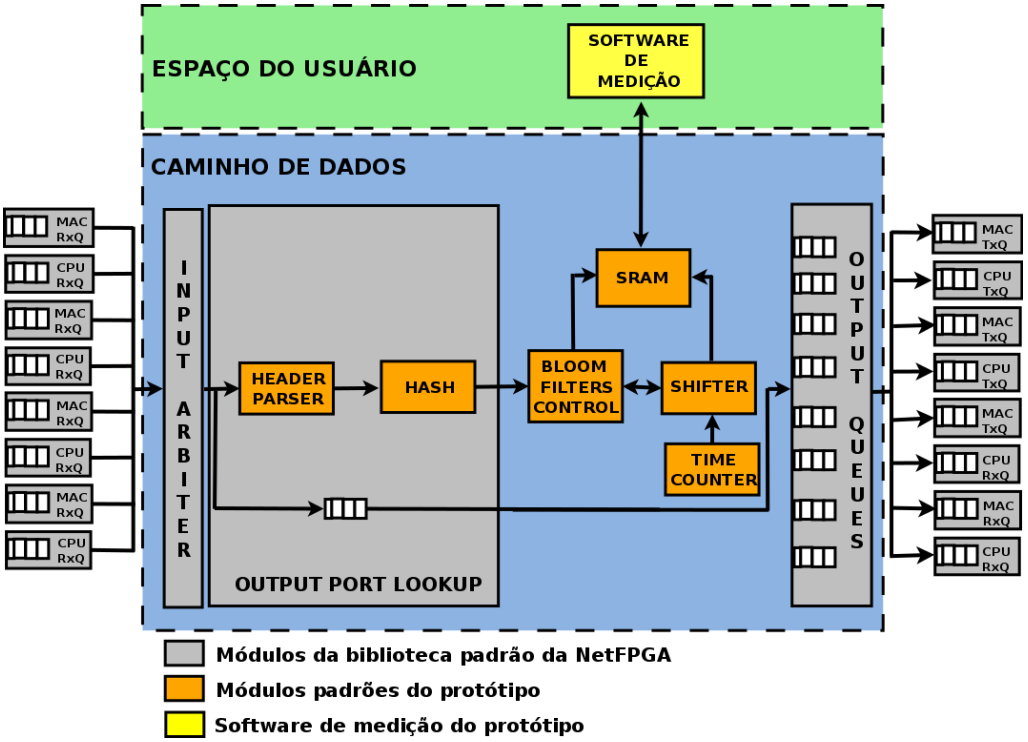


Figura 4.2: Arquitetura *BloomTime* com suporte à medição do tempo de chegada entre pacotes.

o fluxo nos *Bloom Filters* armazenados na SRAM. Ambos endereços têm tamanho de 32 *bits* e são gerados utilizando o algoritmo CRC (*Cyclic Redundancy Check*). Antes desses endereços serem encaminhados para o módulo *Bloom Filters Control*, os endereços são reduzidos de 32 *bits* para 13 *bits*, tal que, 13 *bits* é o número de *bits* necessário para endereçar as linhas dos *Bloom Filters* ($2^{13} = 8.192$ linhas). Em seguida, *hash1* e *hash2* são enviados para o módulo *Bloom Filters Control*.

Os endereços *hash1* e *hash2*, ao chegarem no módulo *Bloom Filters Control*, são inseridos na fila interna do módulo. À partir desse momento inicia-se as operações de leitura e/ou escrita nos *Bloom Filters*. Em seguida, as *hashs* são retiradas da fila e imediatamente são realizados duas requisições de leitura na SRAM para ambos os endereços. Após a chegada dos dados lidos da SRAM a próxima etapa é verificar se a janela de tempo mais atual (*Bloom Filter* B_1) para ambos endereços está marcado. Um fluxo está marcado no *Bloom Filter* quando as duas *hashs* que endereçam o fluxo estão com valor igual a um. Caso contrário, o fluxo não está inserido na estrutura. Se ambos os dados lidos estiverem com valor zero em B_1, será inserido o valor um para marcar o tempo de chegada do fluxo. Se apenas um dos dados lidos estiver com B_1 marcado, apenas o endereço do dado que estiver com valor igual a zero será marcado. Se ambos endereços já estiverem marcados, um novo fluxo será retirado

da fila e todo o processo será repetido.

Todo o controle de tempo é realizado pelo módulo *Time Counter*. Este módulo controla o tempo da janela de tempo e o tempo de monitoramento das marcações nos *Bloom Filters*. Para termos um controle mais preciso da contagem do tempo, foi criado um registrador em *software* para habilitar o início da contagem do módulo *Time Counter*. Quando o tempo da janela de tempo é atingido um sinal é enviado para o módulo *Shifter* para que ocorra as operações de deslocamentos. O processo de deslocamento só ocorre quando o módulo *Bloom Filters Control* não está realizando nenhuma operação na SRAM. Caso alguma operação esteja sendo realizada o módulo *Shifter* espera terminar as operações para que os deslocamentos dos *Bloom Filters* possa começar. Quando as operações de deslocamento começam nenhum fluxo é marcado nas estruturas *Bloom Filters* até que as operações sejam finalizadas. As operações de deslocamentos dos *Bloom Filters* consistem em mover os dados do *Bloom Filter* mais atual para o *Bloom Filter* mais antigo, por exemplo, B_1 para B_2, ..., B_{n-1} para B_n. Quando o tempo de monitoramento é atingido, o registrador em *hardware* é ativo para que as leituras das marcações possam ser lidas pelo *software* de medição.

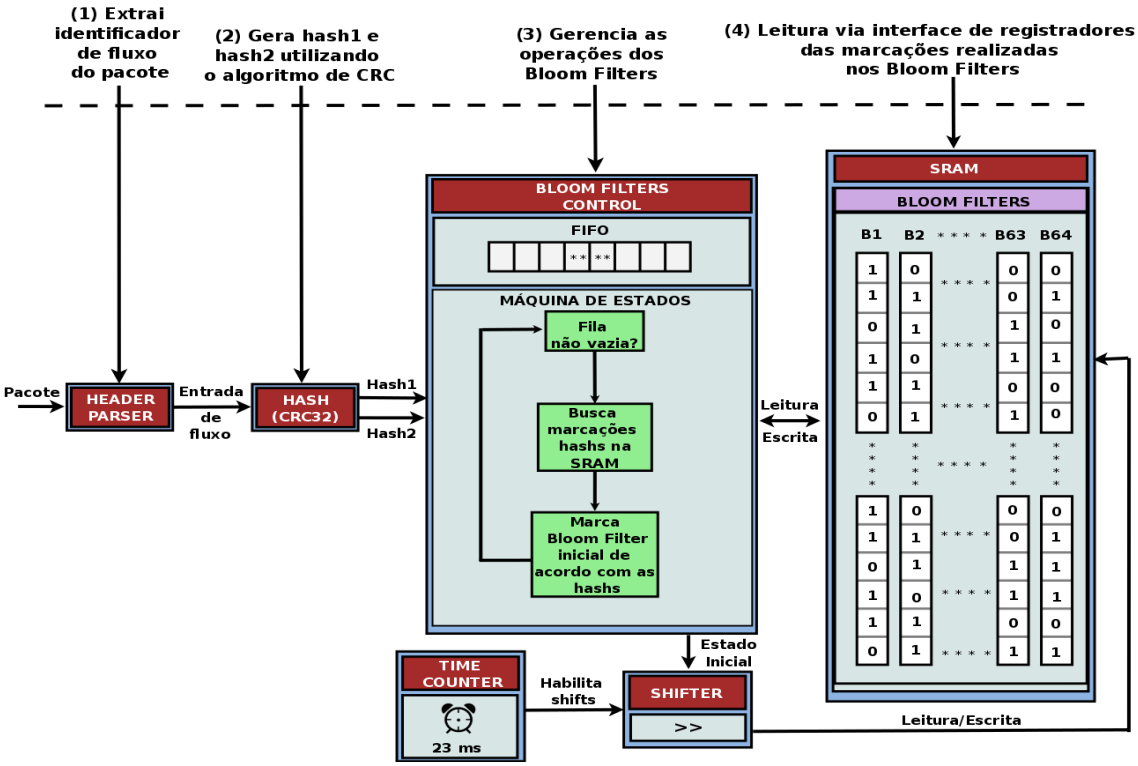


Figura 4.3: Caminho de dados *BloomTime* implementado no *hardware NetFPGA*.

4.3.2 Espaço do usuário

O espaço do usuário do protótipo é composto pelos programas de medição e cálculo da média e variância. O *software* de medição realiza três tipos de funcionalidades: (i) observar o registrador em *hardware* para verificar se está ativo; (ii) realizar a leitura das marcações nos *Bloom Filters*; (iii) inicializar a SRAM após a leitura. O *software* de medição realiza as operações de leitura na SRAM de acordo com o tempo de monitoremento das marcarções nos *Bloom Filters*. Após esse tempo ser atigindo o registrador em *hardware* é ativo e o *software* de medição lê as marcações do tempo de chegada dos fluxos das estruturas *Bloom Filters* via interface de registradores, e salva estas informações em um arquivo no formato texto. Em seguida, os módulos de medição do *hardware* e *Bloom Filters* são inicializados. Os *Bloom Filters* são inicializados automaticamente ao longo do tempo devido aos deslocamentos.

Quando as iterações de medição são finalizadas, o *software* de cálculo da média e variância é executado. Este *software* lê o arquivo de regras dos fluxos medidos e gera a *hash* dos respectivos fluxos. Em seguida, é realizado a diferença do tempo de chegada dos pacotes para cada fluxo com base no arquivo gerado pelo *software* de medição. A diferença é calculada para um determinado fluxo, quando as duas *hashs* do fluxo estão marcadas no mesmo *Bloom Filter*. Caso uma das *hashs* não esteja marcada em um determinado *Bloom Filter*, significa que ocorreu um falso positivo em relação a outro fluxo. Quando ocorre falso positivos a diferença não é calculada. Após as operações de diferença serem efetuadas, a média e variância são calculadas e salvas em outro arquivo no formato texto.

4.4 Avaliação

Nesta seção é apresentado o ambiente dos experimentos realizados com o protótipo *BloomTime*. Nosso objetivo é comparar a escabilidade da arquitetura proposta em relação a [Pacífico et al., 2016a], que estendeu o comutador *OpenFlow* para realizar medição acurada de apenas 28 fluxos. Para compararmos ambos os protótipos, o mesmo cenário desenvolvido por [Pacífico et al., 2016a] foi reproduzido.

4.4.1 Descrição do ambiente de testes

Em [Pacífico et al., 2016a], o cenário de testes é composto de um terminal A enviando pacotes no formato TCP à uma taxa controlada para o terminal B. O terminal A apenas envia 28 fluxos simultaneamente devido à quantidade de fluxos que o

protótipo consegue medir. Entre A e B foram inseridos dois comutadores tradicionais com o objetivo de aumentar o atraso do tempo de chegada dos pacotes ao terminal B. No final da rede foram inseridos o protótipo de medição exata em *hardware* e o terminal B executando o *software* de medição.

O cenário utilizado em [Pacífico et al., 2016a] teve que ser adaptado devido ao grande volume de fluxos a serem gerados. Ao invés de utilizar um terminal gerador de fluxos foram adicionados vinte terminais (TA1, TA2, ..., TA20) para se gerar até 2.000 fluxos simultaneamente na rede. Ambos os comutadores tradicionais foram preservados entre A e B, e no final da rede o protótipo de medição exato foi substituído pelo protótipo do *hardware* aproximado. O *software* de medição executado no terminal B também foi preservado. Este *software* assim como em [Pacífico et al., 2016a] foi utilizado para validar o protótipo e também saber o quanto a medição aproximada está próxima da medição ideal.

As medições em *hardware* foram realizadas em tempo real, sendo o cálculo da média e variância feitos ao final do experimento por uma aplicação em *software*. Esta aplicação demora em torno de 30 minutos para calcular a média e variância de 2.000 fluxos. Em B, tanto as medições e o cálculo da média e variância foram realizadas no final do experimento. O *software* em B leva aproximadamente 1 hora e 30 minutos para calcular a média e variância para a mesma quantidade de fluxos. Na figura 4.4 é apresentado o ambiente de testes.

Figura 4.4: Ambiente de testes.

4.4.2 Verificação do protótipo

Para o envio de pacotes TCP foi utilizado o programa desenvolvido em [Pacífico et al., 2016a]. Este programa lê um arquivo de texto que contêm as regras que devem ser monitoradas pelo protótipo em *hardware*. Cada regra lida é executado

por uma *thread*. Em cada terminal são criados no máximo 100 *threads*, de modo a não prejudicar o envio de pacotes devido ao número de *threads* a serem criadas. O programa para gerar pacotes utilizou da biblioteca `libnet`[2] do C. Para termos um controle maior da quantidade de pacotes transmitidos dos terminais A para B e *NetFPGA*, os pacotes foram escritos em arquivos `pcap`. Os arquivos `pcaps` dos terminais A foram gerados antes do experimento. Cada arquivo tem tempo de duração igual ao tempo total do experimento somado ao tempo de começar e terminar o envio de pacotes. Em B (*software*) também é criado um arquivo `pcap`. Esse arquivo tem duração igual ao tempo total do experimento e é reproduzido utilizando a biblioteca DPKT[3] da linguagem *Python*.

Ao começar a execução do experimento o `tcpdump` é inicializado em B para captura dos pacotes e criação do arquivo `pcap`. Em seguida, os terminais A são inicializados simultamente via ssh para o envio dos pacotes utilizando o programa `bittwist`[4]. No final de cada iteração as marcações de tempo nos *Bloom Filters* são lidas, e os módulos em *hardware* responsáveis pela medição inicializados. Após a última iteração o `tcpdump` em B e `bittwist` são finalizados.

4.4.3 *Warm up*

Os experimentos são compostos de 50 iterações. O tempo de cada iteração varia de acordo com o tempo de monitoramento das marcações nos *Bloom Filters*. Por exemplo, se o tempo da janela de tempo é igual a 23 ms o tempo da iteração é de 1,475 s. Para inicializar os processos de monitoramento e transmissão de pacotes foram gastos 5 s e 3 s para finalizar ambos os processos. O tempo total de duração do experimento é composto pelo tempo de inicializar os processos de medição somado ao tempo total das iterações e o tempo para finalizar todos os processos: 5 s + 50 x 1,475 s + 3 s = 81,75 s (1 minuto e 36 s) utilizando uma janela de tempo de 23 ms. Consideramos o valor de *Warm up* de 10% do tempo total do experimento. Com base no valor de *Warm up*, para uma janela de tempo de 23 ms são retirados as medições das 8 primeiras e últimas iterações (10% de 81,75 s = 8,175 iterações) realizadas por *hardware* e *software*. O número de medições a serem retiradas por iteração varia de acordo com o número de fluxos.

[2]Disponível em: https://sourceforge.net/projects/libnet/files/
[3]Repositório: https://dpkt.readthedocs.io/en/latest/index.html
[4]Repositório: http://bittwist.sourceforge.net/

4.5 Resultados

Nesta seção apresentamos os resultados obtidos da média e variância do tempo de chegada entre pacotes para os experimentos de 28, 1.000 e 2.000 fluxos. Em cada experimento utilizamos 2, 4, 8 e 23 ms o tamanho da janela de tempo do protótipo aproximado. Todos os experimentos foram reproduzidos usando atrasos pré-definidos e aleatórios como descrito em [Pacífico et al., 2016a]. Para os atrasos pré-definidos foi utilizado 20 e 40 ms como atrasos fixos para validar a medição do protótipo. Para os atrasos aleatórios foi utilizado uma distribuição uniforme entre 0 e 50 ms com 3 sementes para representar atrasos distintos como ocorrem em um tráfego realista.

Além disso, apresentamos os resultados obtidos utilizando o gráfico de caixa (*BoxPlot*) com o intervalo de confiança. O gráfico *BoxPlot* representa a distribuição da população dividida em partes chamadas quartis. Este gráfico é composto de quatro quartis. Cada quartil representa uma porcentagem das amostras. O primeiro quartil (Q_1) refere-se a 25%, o segundo quartil (mediana) tem um percentil de 50% e o terceito quartil (Q_3) equivale a 75% das amostras. Q_1 é representado no gráfico como o limite inferior da caixa, a mediana refere-se a linha no meio e o limite superior da caixa representa Q_3. Os traços na extremidade (*whishers*) representam o limite inferior e superior das amostras. Utilizamos este tipo de gráfico para representar a distribuição da diferença relativa e absoluta entre *hardware* e *software*.

4.5.1 Experimento com 28 fluxos

O objetivo deste experimento é verificar o comportamento da medição do protótipo *BloomTime* para uma quantidade pequena de fluxos. Escolhemos 28 fluxos com base no trabalho de [Pacífico et al., 2016a] com o intuito de comparar o protótipo aproximado com o exato.

- **Atraso fixo**

No gráfico da diferença relativa da média (figura 4.5b), 75% das medições estão entre 0 a 23% para as janelas de tempo de 2, 4, 8, 23 ms. Na tabela 4.1 são apresentados a média, o desvio padrão e o intervalo de confiança da média medida. As janelas de tempo de 4, 8, 23 ms tiveram um valor de média e desvio padrão relativamente baixo. O pior resultado encontrado para média foi a janela de tempo de 2 ms. Isso ocorre devido ao aumento no número de deslocamentos realizados. Os deslocamentos com maior frequência podem estar interferindo na acurácia, por

passar a maior parte do tempo deslocando os *Bloom Filters*. No gráfico da diferença absoluta (figura 4.5a), as amostras estão entre 0 a 10 ms para as respectivas janelas de tempo.

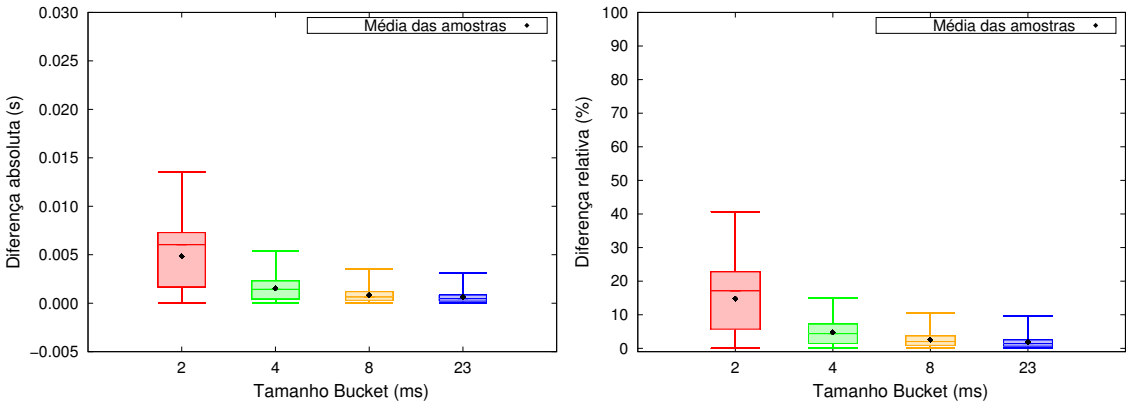
No gráfico da variância (figura 4.6b), 75% das amostras tem diferença relativa entre *hardware* e *software* de 0 a 30%. A variância medida pelo *hardware* tem um valor maior em relação ao *software*, devido as operações internas de deslocamento no *hardware*. Isso acarreta uma maior variação na acurácia das medições. No gráfico (figura 4.6a) é apresentado a diferença absoluta da variância. Os valores da diferença absoluta da variância variam de 0 a 20 μs para as respectivas janelas de tempo. Na tabela 4.2, são apresentados a média, desvio padrão e intervalo de confiança da variância medida.

Tabela 4.1: Dados diferença relativa: média 28 fluxos atraso fixo.

Janela de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	14,75	9,82	[14,15:15,35]
04	4,72	3,52	[4,5:4,93]
08	2,5	2,00	[2,38:2,62]
23	1,8	1,62	[1,73:1,92]

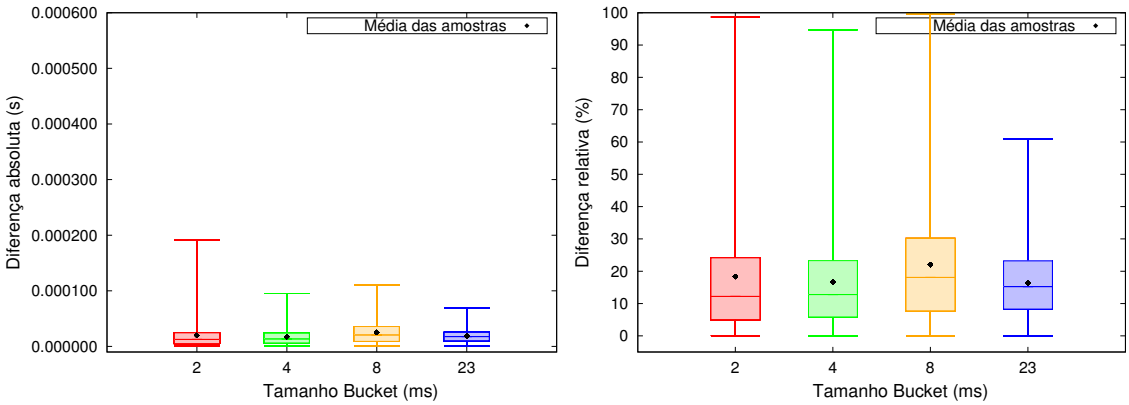
Tabela 4.2: Dados diferença relativa: variância 28 fluxos atraso fixo.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	18,36	19,19	[17,16:19,57]
04	16,69	14,81	[15,79:17,57]
08	22,03	17,99	[20,95:23,11]
23	16,39	10,52	[15,78:17,01]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.5: Gráficos *Boxplot* média: 28 fluxos - atraso fixo.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.6: Gráficos *Boxplot* variância: 28 fluxos - atraso fixo.

■ Atraso aleatório

No gráfico da diferença relativa da média (figura 4.7b), 75% das amostras estão entre 0 a 12% para as janelas de tempo de 4, 8, 23 ms. Entretanto, para a janela de tempo de 2 ms, a diferença relativa média para 75% das amostras é de 30%. Na tabela 4.3 são apresentados a média, o desvio padrão e o intervalo de confiança da média medida. As janelas de tempo de 4, 8 e 23 ms tiveram um valor de média e desvio padrão relativamente baixo. O pior resultado encontrado para média foi a janela de tempo de 2 ms, com um valor de média medida de 16,62%. No gráfico da diferença absoluta (figura 4.7a), as amostras estão entre 0 a 15 ms para as respectivas janelas de tempo.

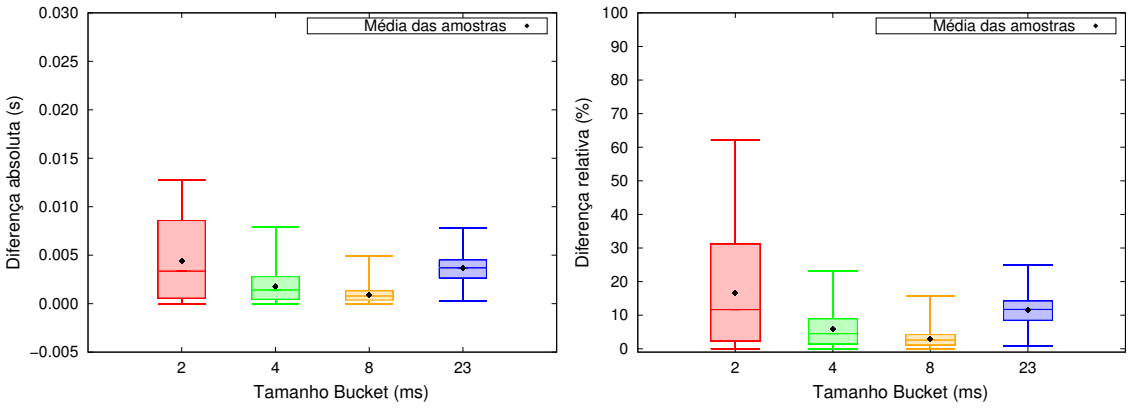
No gráfico da variância (figura 4.8b), 75% das amostras tem diferença relativa entre *hardware* e *software* de 0 a 30%. Os valores da diferença absoluta da variância (figura 4.8a) variam de 0 a 35 μs para as respectivas janelas de tempo. Na tabela 4.4, são apresentados a média, desvio padrão e intervalo de confiança da variância medida.

Tabela 4.3: Dados diferença relativa: média 28 fluxos atraso aleatório.

Janela de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	16,62	16,95	[15,57:17,66]
04	5,80	5,11	[5,48:6,12]
08	2,91	2,18	[2,77:3,04]
23	11,59	4,10	[11,35:11,84]

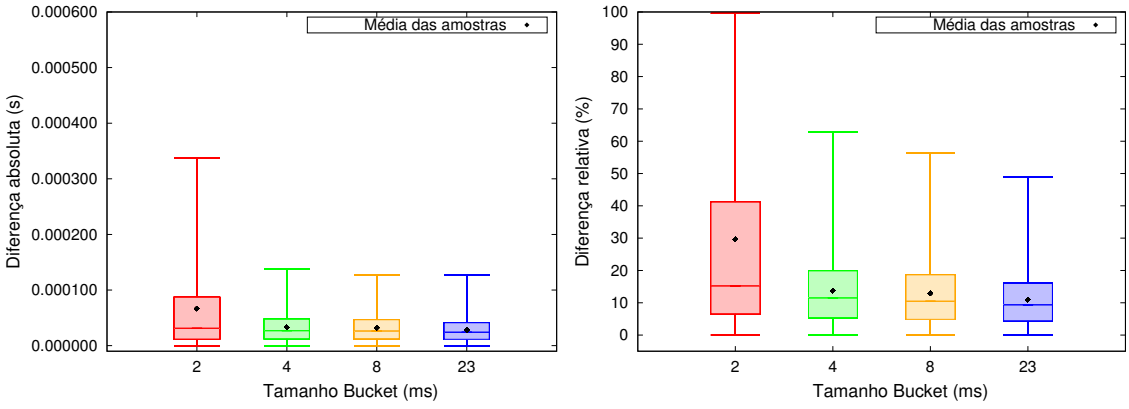
Tabela 4.4: Dados diferença relativa: variância 28 fluxos atraso aleatório.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	29,68	31,45	[27,52:31,86]
04	13,75	10,57	[13,10:14,41]
08	12,96	10,25	[12,35:13,57]
23	10,96	8,02	[10,49:11,43]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.7: Gráficos *Boxplot* média: 28 fluxos - atraso aleatório.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.8: Gráficos *Boxplot* variância: 28 fluxos - atraso aleatório.

4.5.2 Experimento com 1.000 fluxos

Este experimento consiste em verificar se arquitetura escala. Também é observado o impacto da taxa de falsos positivos com o aumento no número de fluxos. 1.000 fluxos equivalem a taxa de 4,69% de falsos positivos.

- **Atraso fixo**

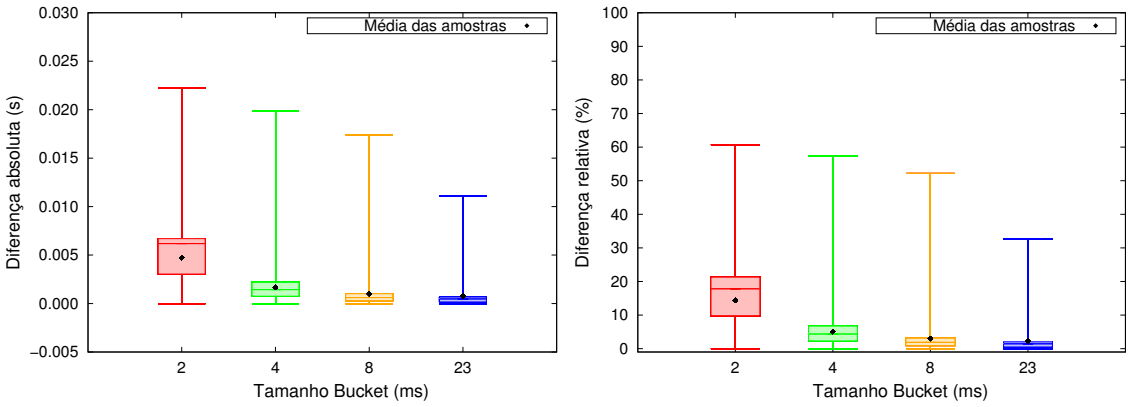
O protótipo se manteve estável com o aumento de 28 para 1.000 fluxos tanto para média e variância. No gráfico da diferença relativa da média (figura 4.9b), 75% das amostras ficaram entre 0 e 10% para as janelas de tempo de 4, 8 e 23 ms. Entretanto, para a janela de tempo de 2 ms, 75% das amostras chegaram 20%. No gráfico da diferença absoluta (figura 4.9a) as amostras estão entre 0 e 23 ms para as respectivas janelas de tempo. No gráfico da diferença relativa da variância (figura 4.10b), 75% das amostras ficaram entre 0 e 20% para as janelas de tempo de 2 e 4 ms. Entretanto, para as janela de tempo de 8 e 23 ms, 75% das amostras chegaram 30%. No gráfico da diferença absoluta (figura 4.10a) as amostras estão entre 0 e 50 μs para as respectivas janelas de tempo. Na tabelas 4.5 e 4.6 são apresentados a média, o desvio padrão e o intervalo de confiança da variância medida.

Tabela 4.5: Dados diferença relativa: média 1.000 fluxos atraso fixo.

Janela de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	14,36	7,99	[14,29:14,44]
04	5,05	4,76	[5,00:5,09]
08	3,02	5,08	[2,96:3,06]
23	2,33	4,35	[2,28:2,36]

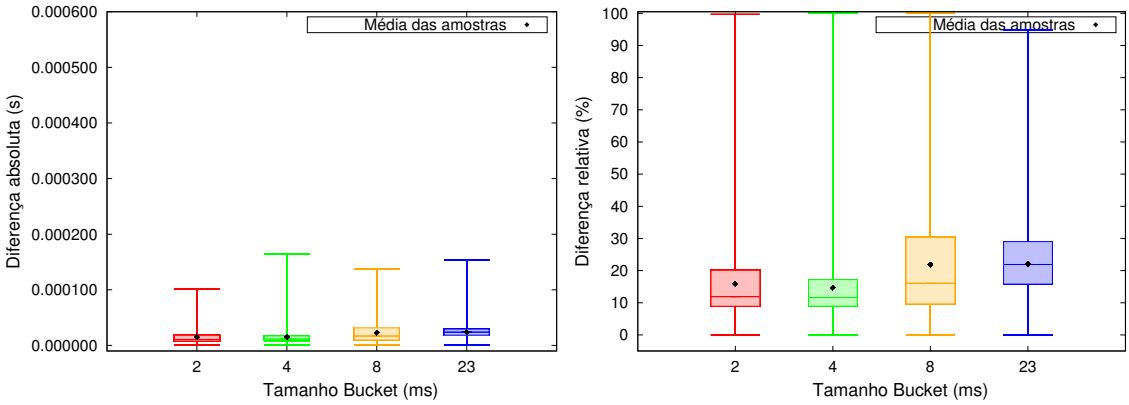
Tabela 4.6: Dados diferença relativa: variância 1.000 fluxos atraso fixo.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	15,86	12,89	[15,73:15,98]
04	14,70	12,78	[14,58:14,83]
08	21,84	18,57	[21,66:22,02]
23	21,99	10,45	[21,89:22,10]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.9: Gráficos *Boxplot* média: 1.000 fluxos - atraso fixo.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.10: Gráficos *Boxplot* variância: 1.000 fluxos - atraso fixo.

■ Atraso aleatório

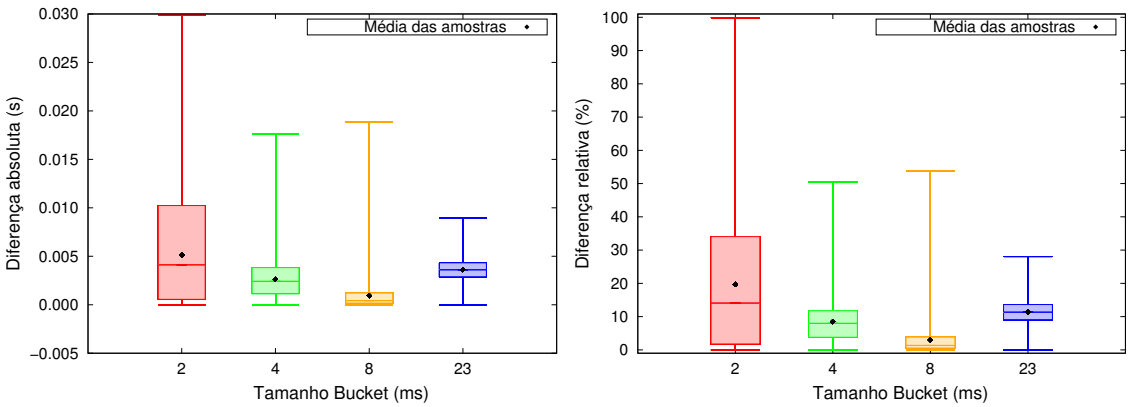
Para as janela de tempo de 4, 8 e 23 ms o protótipo se manteve estável com o aumento de 28 para 1.000 fluxos tanto para média e variância. Entretanto, para a janela de tempo de 2 ms houve uma piora significativa na média e 30% para variância. Nos gráficos (figuras 4.11b e 4.12b) são apresentados a diferença relativa da média e variância para 1.000 fluxos. Nos gráficos (figuras 4.11a e 4.12) são apresentados a diferença absoluta da média e variância. Nas tabelas 4.7 e 4.8 são demonstrados a média e variância medida com o desvio padrão e intervalos de confiança para o atraso aleatório de 1.000 fluxos.

Tabela 4.7: Dados diferença relativa: média 1.000 fluxos atraso aleatório.

Janela de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	19,69	17,93	[19,51:19,87]
04	8,52	0,05	[8,46:8,57]
08	2,97	4,87	[2,92:3,02]
23	11,44	3,59	[11,41:11,48]

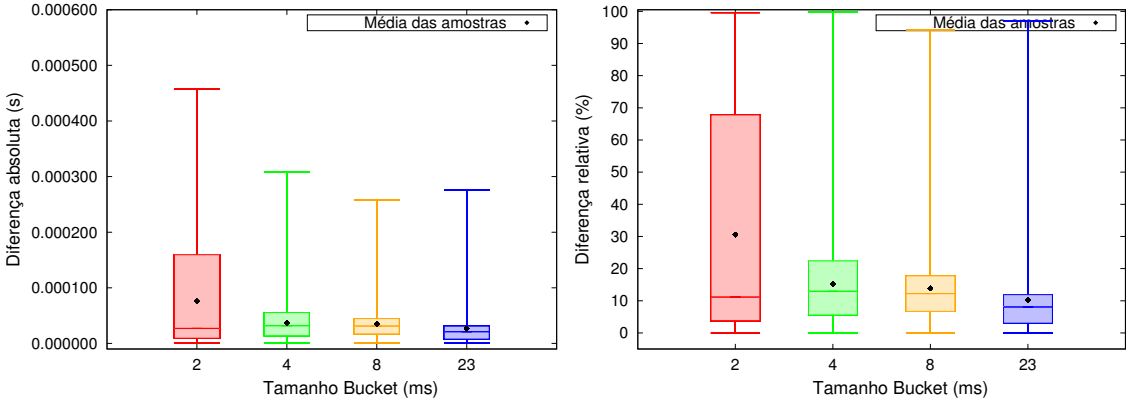
Tabela 4.8: Dados diferença relativa: variância 1.000 fluxos atraso aleatório.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	30,58	35,48	[30,17:30,99]
04	15,22	12,32	[15,10:15,34]
08	13,91	10,75	[13,80:14,01]
23	10,29	13,00	[10,16:10,42]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.11: Gráficos *Boxplot* média: 1.000 fluxos - atraso aleatório.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.12: Gráficos *Boxplot* variância: 1.000 fluxos - atraso aleatório.

4.5.3 Experimento com 2.000 fluxos

Neste experimento são enviados 2.000 fluxos com objetivo de verificar o comportamento do protótipo em relação a escabilidade e a ocorrência de falsos positivos com o aumento de 1.000 para 2.000 fluxos. 2.000 fluxos equivalem a probabilidade de 14,92% de falsos positivos.

- **Atraso fixo**

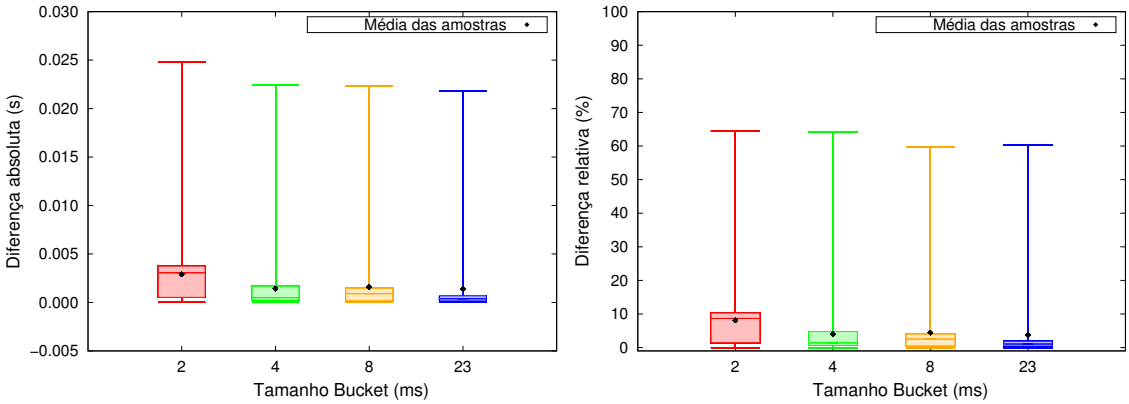
Com o aumento no número de fluxos de 1.000 para 2.000 houve uma melhora na média. Para 75% das amostras todas as janelas de tempo tiveram diferença relativa (figura 4.13b) abaixo de 10%. A diferença absoluta (figura 4.13a) para média está entre 0 e 25 ms. A diferença relativa (figura 4.14b) para variância de 75% das amostras para as janelas de tempo de 2, 4 e 8 ms ficaram abaixo de 30%. Para a janela de tempo de 23 ms, 75% das amostras é igual a 50%. A diferença absoluta (figura 4.14a) está entre 0 e 600 μs. Nas tabelas 4.9 e 4.10 são demonstrados a média e variância medida com o desvio padrão e intervalos de confiança para o atraso fixo de 2.000 fluxos.

Tabela 4.9: Dados diferença relativa: média 2.000 fluxos atraso fixo.

Janela de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	8,13	7,00	[8,08:8,18]
04	3,95	6,83	[3,90:3,99]
08	4,37	7,99	[4,32:4,43]
23	3,78	7,81	[3,73:3,84]

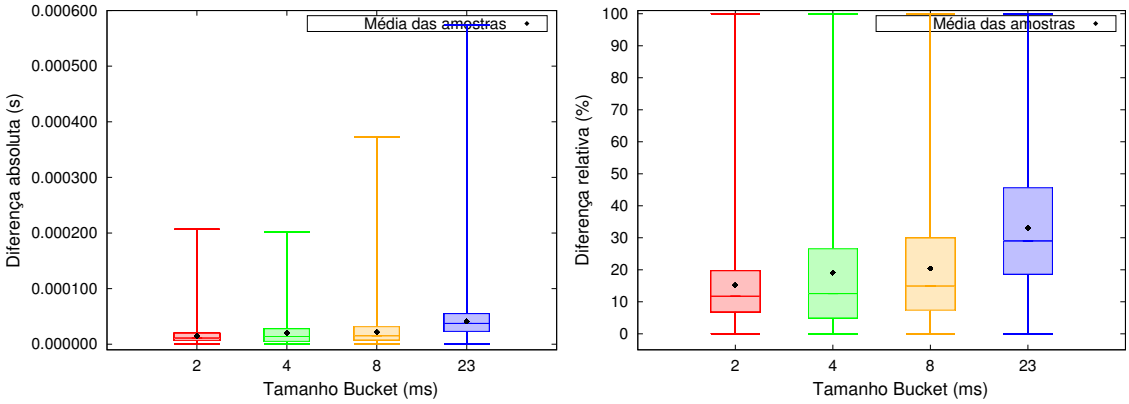
Tabela 4.10: Dados diferença relativa: variância 2.000 fluxos atraso fixo.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	15,23	14,84	[15,13:15,34]
04	19,12	19,56	[18,98:19,26]
08	20,40	16,68	[20,28:20,51]
23	33,01	20,15	[32,87:33,15]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.13: Gráficos *Boxplot* média: 2.000 fluxos - atraso fixo.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.14: Gráficos *Boxplot* variância: 2.000 fluxos - atraso fixo.

■ Atraso aleatório

Com o aumento no número de fluxos de 1.000 para 2.000 houve uma melhora de aproximadamente 13% para média e 4% para variância. Para 75% das amostras todas as janelas de tempo tiveram diferença relativa (figura 4.15b) abaixo de 10%.

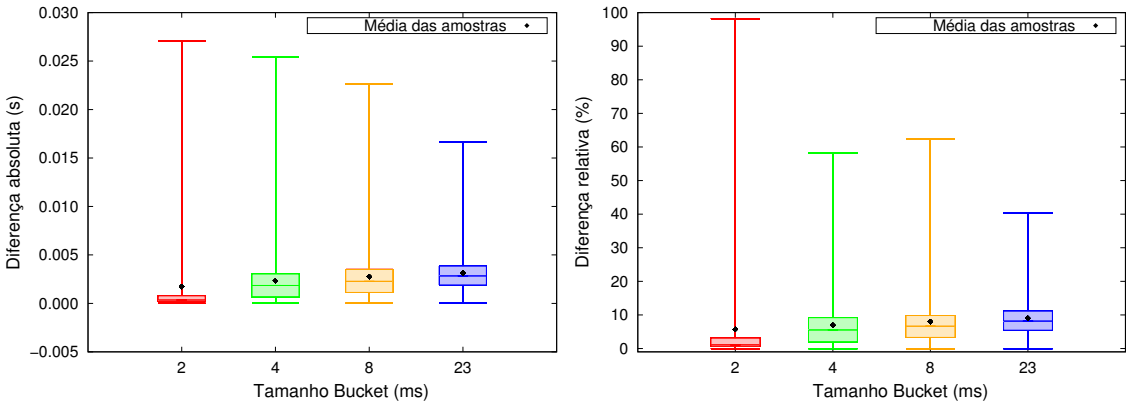
A diferença absoluta (figura 4.15a) para média estão entre 0 e 5 ms. A diferença relativa (figura 4.16b) para variância de 75% das amostras para todas as janelas de tempo ficaram abaixo de 20%. A diferença absoluta (figura 4.16a) ficou entre 0 e 100 μs. Nas tabelas 4.11 e 4.12 são demonstrados a média e variância medida com o desvio padrão e intervalos de confiança para o atraso aleatório de 2.000 fluxos.

Tabela 4.11: Dados diferença relativa: média 2.000 fluxos atraso aleatório.

Janelas de tempo (ms)	Média da média (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	5,73	11,08	[5,65:5,81]
04	6,93	6,70	[6,88:6,98]
08	7,96	7,26	[7,92:8,02]
23	8,98	5,36	[8,95:9,020]

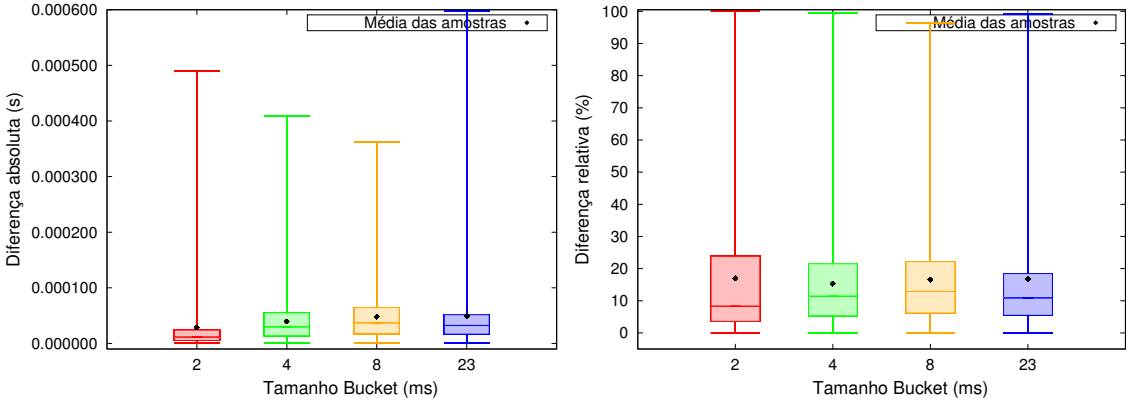
Tabela 4.12: Dados diferença relativa: variância 2.000 fluxos atraso aleatório.

Janela de tempo (ms)	Média da variância (%)	Desvio padrão (%)	Intervalo de confiança (%)
02	16,99	20,88	[16,84:17,14]
04	15,33	13,92	[15,23:15,43]
08	16,52	14,62	[16,45:16,63]
23	16,73	19,05	[16,60:16,86]



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.15: Gráficos *Boxplot* média: 2.000 fluxos - atraso aleatório.



(a) Difereça absoluta **(b) Diferença relativa**

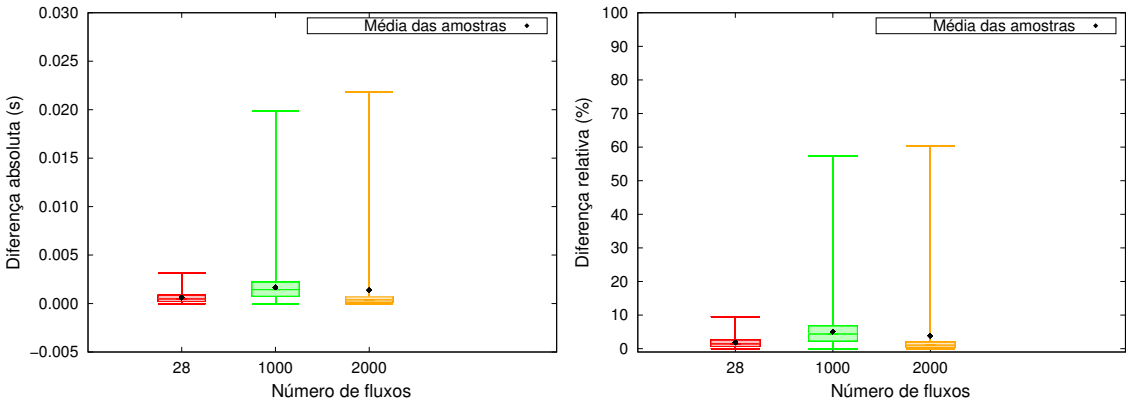
Figura 4.16: Gráficos *Boxplot* variância: 2.000 fluxos - atraso aleatório.

4.5.4 Melhores resultados *BloomTime*

Nesta seção apresentamos os melhores resultados do protótipo *BloomTime* de acordo com o tipo de atraso, número de fluxos e o tamanho da janela de tempo. Os resultados obtidos demonstram que a arquitetura *BloomTime* tem diferença relativa de até 20% em relação ao *software* para 75% das amostras. Entretanto, a diferença absoluta entre *hardware* e *software* está próxima de 0 s para quase 100% das amostras. A medição realizada pela arquitetura é menos acurada que o *software*. Isso ocorre devido a abordagem ser aproximada. Além disso, outro fator que interfere na acurácia da medição é o custo operacional dos deslocamentos periódicos de acordo com o tamanho da janela de tempo.

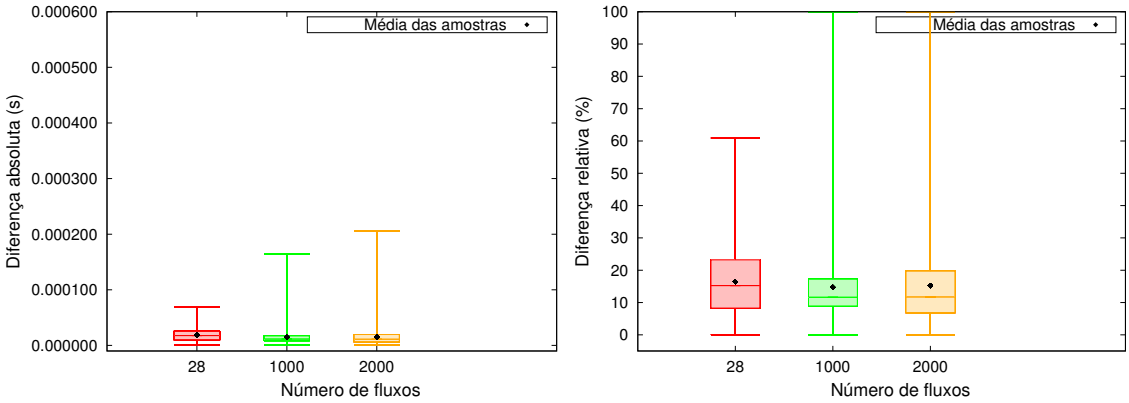
- **Atraso fixo**

No gráfico da média da diferença relativa (figura 4.17b), 75% das medições entre *hardware* e *software* variam de 0 e 10% para 28, 1.000 e 2.000 fluxos. Entretanto, o gráfico da média da diferença absoluta (figura 4.17a) mostra que 75% das medições estão entre 0 e 5 ms para os respectivos fluxos. No gráfico da variância da diferença relativa, 75% das medições estão entre 0% e 20% (figura 4.18b) com uma diferença absoluta de 5 μs (figura 4.18a).



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.17: Gráficos *Boxplot* média: melhores janelas de tempo - atraso fixo.



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.18: Gráficos *Boxplot* variância: melhores janelas de tempo - atraso fixo.

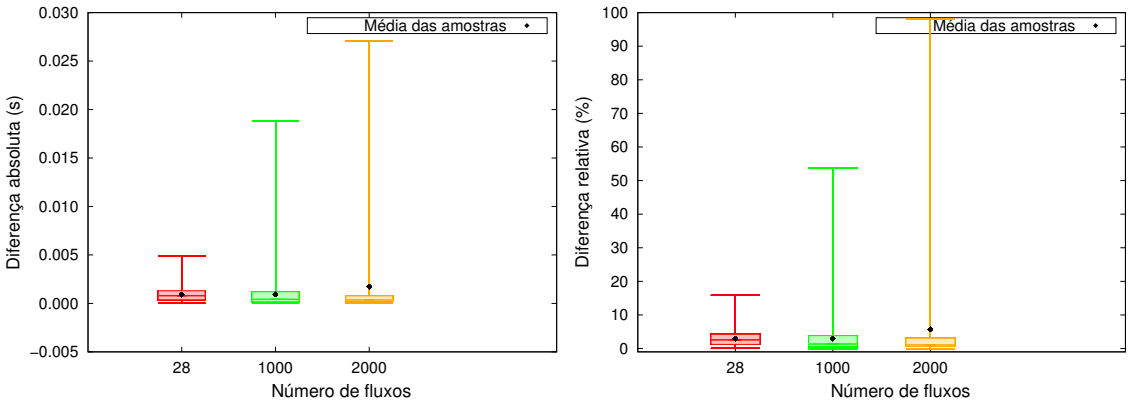
Na tabela 4.13 é apresentado as melhores janelas de tempo para o atraso fixo da média e variância com base no número de fluxos.

Tabela 4.13: Melhores janela de tempo - atraso fixo.

Fluxos	**Média**	**Variância**
28	23 ms	23 ms
1.000	23 ms	04 ms
2.000	23 ms	02 ms

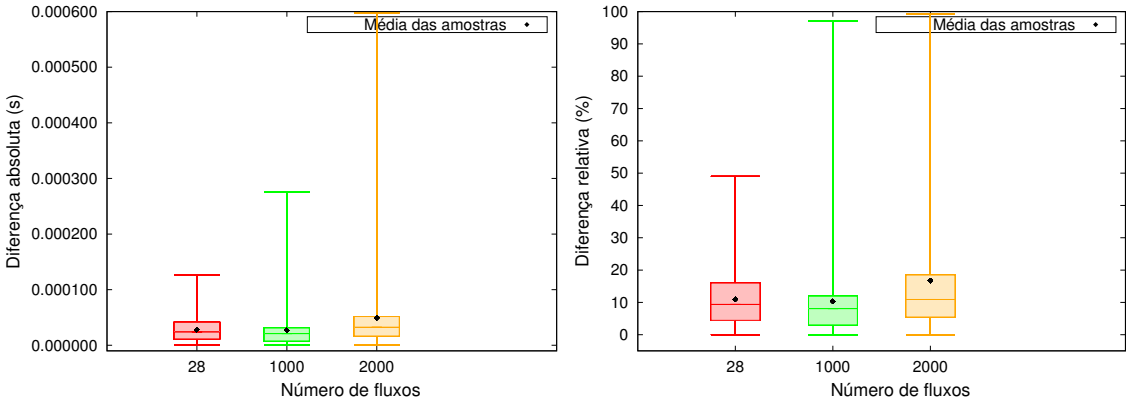
- **Atraso aleatório**

No gráfico da média da diferença relativa (figura 4.19b), 75% das medições entre *hardware* e *software* variam de 0 e 8% para 28, 1.000 e 2.000 fluxos. Entretanto, o gráfico da média da diferença absoluta (figura 4.19a) mostra que 75% das medições estão entre 0 e 25 ms para os respectivos fluxos. No gráfico da variância da diferença relativa (figura 4.20b), 75% das medições estão entre 0% e 20% com uma diferença absoluta de 90 μs (figura 4.20a).



(a)Diferença absoluta **(b) Diferença relativa**

Figura 4.19: Gráficos *Boxplot* média: melhores janelas de tempo - atraso aleatório.



(a) Difereça absoluta **(b) Diferença relativa**

Figura 4.20: Gráficos *Boxplot* variância: melhores janelas de tempo - atraso aleatório.

Na tabela 4.14 é apresentado as melhores janelas de tempo para o atraso aleatório da média e variância com base no número de fluxos.

Tabela 4.14: Melhores janela de tempo - atraso aleatório

Fluxos	Média	Variância
28	08 ms	23 ms
1.000	08 ms	23 ms
2.000	02 ms	23 ms

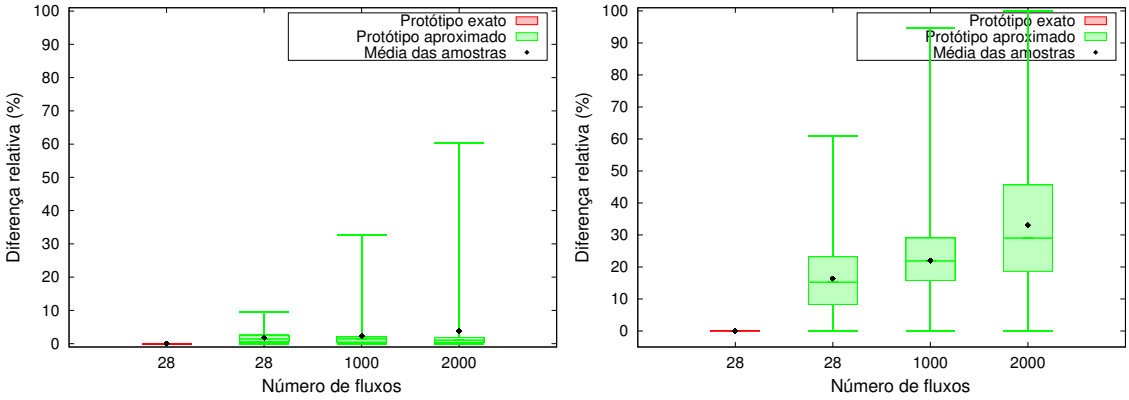
Com base nos gráficos apresentados nesta seção, a arquitetura *BloomTime* realiza uma medição menos acurada, mas escala independente do número de fluxos.

4.5.5 *Hardware* exato x *Hardware* aproximado

Nesta subseção apresentamos os resultados obtidos entre a comparação da medição do *hardware* exato em relação ao aproximado. Para comparação, escolhemos o número máximo de fluxos que o protótipo exato consegue medir (28 fluxos). Para o protótipo aproximado, escolhemos o experimento de 28 fluxos com janela de tempo de 23 ms, por ser o experimento com melhor resultado em relação aos demais. Além disso, inserimos os experimentos 1.000 e 2.000 com janela de tempo de 23 ms para compararmos com a medição de 28 fluxos exato e aproximado.

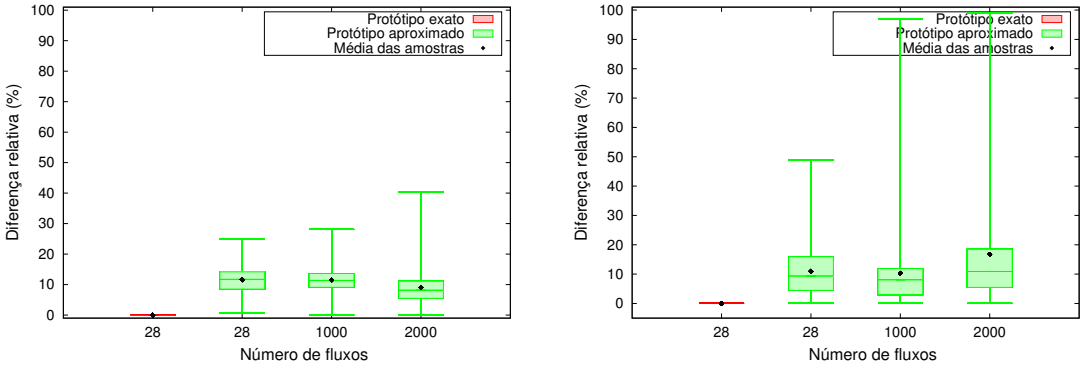
Na figura 4.21 são apresentados os gráficos da média(a) e variância(b) do atraso fixo. Em ambos os gráficos é possível observar que o protótipo do *hardware* exato para 100% das medições tem diferença relativa quase nula em relação ao *software* na extremidade da rede. O protótipo aproximado varia de 0 a 60% em relação ao *software*. No experimento com atraso aleatório (figura 4.22), a diferença relativa da média e variância do *hardware* exato em relação ao *software* também é quase nula. No protótipo aproximado a diferença relativa da média e variância estão entre 0 e 100% em relação ao *software*.

Em ambos experimentos (atraso fixo e aleatório), o *hardware* exato é mais acurado e estável que o protótipo aproximado. Entretanto, o *hardware* exato é limitado em relação ao número de fluxos que consegue medir. O protótipo aproximado ao contrário do exato, mede uma quantidade maior de fluxos sacrificando a acurácia.



(a) Média **(b) Variância**

Figura 4.21: Gráficos *Boxplot* média e variância: exato x aproximado - atraso fixo.



(a) Média **(b) Variância**

Figura 4.22: Gráficos *Boxplot* média e variância: exato x aproximado - atraso aleatório.

A figura 4.23 apresenta as linhas de tendência das médias das diferenças relativa das medições aproximadas da média e variância em função do número de fluxos. Pode-se observar que as soluções aproximadas não apresentam grandes variações em função do número de fluxos.

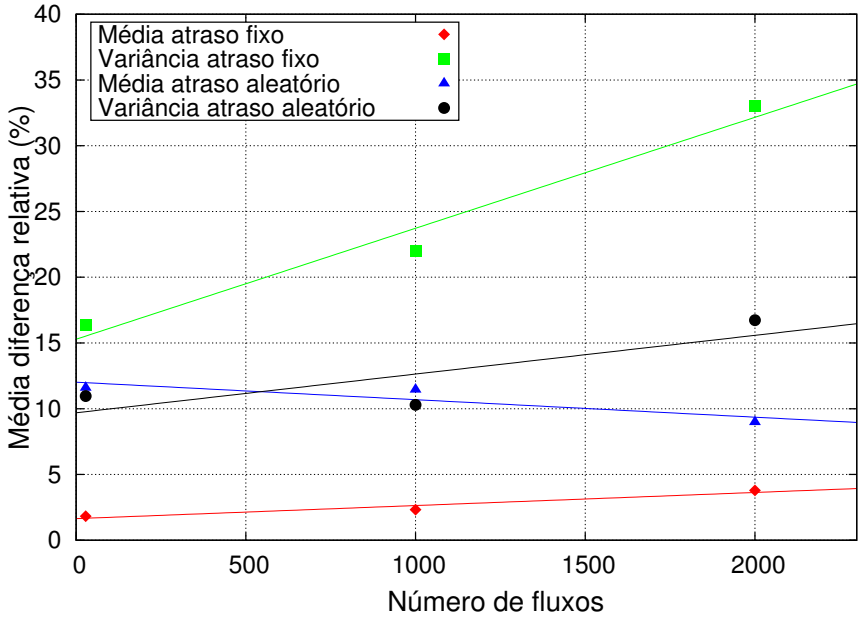


Figura 4.23: Gráfico tendência medição protótipo aproximado.

4.5.6 Limitações da arquitetura

O protótipo *BloomTime* mede até 2.000 fluxos de maneira escalável sacrificando a acurácia das medições. Isto ocorre devido a três fatores: operações de deslocamento dos *BloomFilters*, tamanho da janela de tempo e ocorrência de falsos positivos (colisões).

O protótipo *BloomTime* é composto de 64 *BloomFilters* com 8.192 linhas. Os *BloomFilters* são armazenados na SRAM. 64 é o número de *bits* da linha e 8.192 correspondem ao número de linhas (figura 4.3). Quando o tempo da janela é atingido, as operações de deslocamento são ativadas. O processo de deslocamento consiste em ler, esperar o retorno do dado da memória, realizar a operação de deslocamento e salvar o dado atualizado na SRAM. Esse processo é repetido 8.192 vezes. Esse tempo é igual a 400 μs da janela de tempo, o que em escalas de tempo pequenas (ms) afeta a acurácia da medição.

Além das operações de deslocamento, a janela de tempo é outro fator crítico do protótipo. As marcações do tempo de chegada dos fluxos são realizados com base no tempo da janela. Janelas de tempo grandes prejudicam a acurácia da medição. Janelas de tempo menores, por exemplo, 2 ms, aumentam a acurácia da medição. Entretanto, com aumento no número de deslocamentos periódicos ocorre uma variância maior na medição e uma piora na acurácia.

A arquitetura *BloomTime* utiliza duas funções *hashs* para endereçar os fluxos

nas estruturas *BloomFilter*. Nesta abordagem diferentes fluxos podem endereçar as mesmas posições acarretando o surgimento de falsos positivos (colisões). Quando o *software* de medição lê as marcações da SRAM e encontra falsos positivos para um determinado fluxo a medição é descartada. O descarte de medições prejudica a acurácia das medições. No gráfico número de fluxos x falsos positivos (figura 4.24) é apresentado a relação diretamente proporcional do aumento no número de fluxos e falsos positivos. Este gráfico foi construído utilizando a equação 4.1, e como entrada foi fornecido os valores de m, n e k da arquitetura *BloomTime*, $m = 8.192$, $n = 2.000$ e $k = 2$. Para 2.000 fluxos a taxa de ocorrência de falsos positivos no protótipo é de 14,92%.

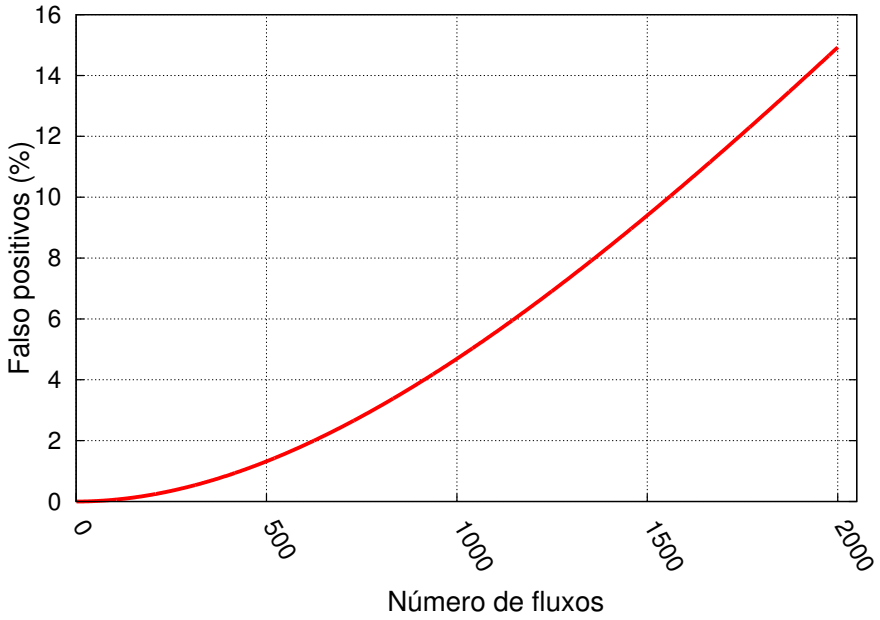


Figura 4.24: Gráfico número de fluxos x falsos positivos.

Abordagens de monitoramento exato medem de maneira mais acurada e os fatores descritos acima não existem ou afetam a medição desses tipos de abordagens. Essas abordagens utilizam de contadores por fluxo que consomem recursos de CPU. Além disso, medem um número pequeno de fluxos estando limitadas ao tamanho da memória. Entretanto, abordagens de monitoramento aproximado permitem medir um volume maior de fluxos de maneira aproximada sacrificando a acurácia pela escabilidade.

4.6 Trabalhos relacionados

Nesta seção são descritos os trabalhos relacionados de acordo com a métrica, método e ambiente de medição.

[Pacífico et al., 2016a] estendeu o projeto em *hardware* do comutador *OpenFlow* e transformou este comutador em uma API de medição de rede. O protótipo desenvolvido realiza medição acurada de 28 fluxos simultaneamente da métrica tempo de chegada entre pacotes. O cenário de experimentação foi composto de um terminal A enviando pacotes com taxa controlada para B. Entre A e B foram inseridos dois comutadores tradicionais para aumentar o atraso e o protótipo em *hardware*. Os resultados obtidos demonstraram que o *hardware* logicamente centralizado mede de maneira acurada como um *software* na extremidade da rede. A principal limitação do protótipo está no número de fluxos que é possível medir.

[Van Adrichem et al., 2014] propôs o *software OpenNetMon*. *OpenNetMon* é uma aplicação de código aberto que tem como funcionalidade monitorar métricas por fluxo, especialmente, atraso e perda de pacotes em redes *OpenFlow*. Esta aplicação fornece engenharia de tráfego ao controlador com medições de monitoramento *online*. A perda de pacotes foi calculada com base na diferença das estatísticas coletadas do primeiro e último comutador da rede. O atraso de pacotes foi calculado utilizando o método ativo RTT (*Round Trip Time*). Nos testes foram utilizados vídeo *stream* entre dois servidores na extremidade da rede ligados por comutadores com atrasos e perda de pacotes estabelecidos pela ferramenta `Netem` do linux. Os resultados do *OpenNetMon* foram comparados com o *software tcpstat*.

Em [Moshref et al., 2013] são discutidos os diferentes tipos de primitivas de medição. O artigo avalia o desempenho das primitivas contadores por fluxo, *hash* e programação da CPU para detecção de fluxos pesados (*Hierarquical Heavy Hitters*). Na avaliação foram utilizados várias escalas de tempo e diferentes configurações. Os testes foram realizados utilizando *traces* obtidos do projeto CAIDA.

[Yu et al., 2013] desenvolveu o protótipo de medição *OpenSketch*. *OpenSketch* é uma arquitetura baseada no paradigma SDN e utiliza de estruturas de medição *sketches*. *Sketches* são escaláveis mas fornecem um resultado aproximado. No plano de controle do *OpenSketch* é utilizado uma biblioteca padrão do protótipo e o plano de dados é divido em três estágios: *hashing*, classificação e contagem. Os resultados do *OpenSketch* foram validados comparando o protótipo com a arquitetura *NetFlow*.

Na tabela 4.15 é apresentado um resumo das características dos trabalhos apresentados.

Tabela 4.15: Comparação características trabalhos relacionados.

Artigo	Tipo de medição	Ambiente	Limitação
[Pacífico et al., 2016a]	Passivo	*Hardware*	Número de regras limitado ao tamanho da TCAM.
[Van Adrichem et al., 2014]	Ativo	*Software*	Os pacotes extras injetados na rede afetam a acurácia da medição.
[Moshref et al., 2013]	Ativo	*Software*	As tarefas de medição dependem de prefixos IPs dos nós folhas.
[Yu et al., 2013]	Passivo	*Hardware*	*Sketch* é restrito apenas a um tipo de medição.

4.7 Conclusões

Tarefas de monitoramento do tráfego são indispensáveis para a engenharia de tráfego moderna. Infelizmente, medir um grande volume de tráfego de maneira acurada e em tempo real tornou-se algo desafiador com os métodos e estruturas tradicionais das redes IP. Neste trabalho estendemos o projeto *Reference Nic* para medir de maneira aproximada o tempo de chegada entre pacotes dos fluxos trafegados em um ambiente de teste realista. Nós também comparamos a medição aproximada do protótipo com uma arquitetura de monitoramento exato com propósito de apresentar os benefícios e desvantagens das estruturas de medição exata e aproximada.

Com os resultados obtidos concluímos que nossa arquitetura mede até 2.000 fluxos com uma taxa de 14,92% de falsos positivos. Mas, realiza uma medição menos acurada que um protótipo exato. Como trabalhos futuros, pretendemos implementar os *Bloom Filters* fora da SRAM usando de registradores e implementar outros tipos de métricas de desempenho de rede disponíveis na literatura.

5. Considerações finais e trabalhos futuros

Medir de maneira acurada em tempo de execução um grande volume de tráfego é algo complexo e desafiador. Isso ocorre, devido a maioria das abordagens tradicionais serem implementadas em *software* na extremidade da rede. Tais abordagens geram alto custo e dificultam o monitoramento devido à configuração e implantação individual do *software* nos nós hospedeiros da rede.

Nos capítulos 2 e 3 foi apresentada uma arquitetura de medição exata que utiliza do paradigma SDN e contadores por fluxo para realizar medição acurada dos fluxos trafegados. O objetivo do trabalho desenvolvido nos capítulos 2 e 3 foi estender um comutador com suporte *OpenFlow* para medição de métricas de rede. O protótipo desenvolvido em ambos os capítulos realiza medição de apenas 28 fluxos devido à limitação do tamanho da memória TCAM, onde são inseridas as regras de monitoramento.

No capítulo 4 foi apresentada uma arquitetura de medição aproximada que utiliza *hash* para marcação do tempo de chegada dos fluxos em estruturas de dados *Bloom Filters* armazenadas na SRAM. Os objetivos do trabalho desenvolvido no capítulo 4 foram realizar medição aproximada de um grande volume de tráfego utilizando *hashs* e *Bloom Filters* implementadas sobre a plataforma *NetFPGA* 1G e comparar à acurácia e escabilidade em relação à solução exata proposta anteriormente.

Os resultados obtidos na pesquisa demonstraram que a arquitetura de monitoramento exato mede de maneira acurada e centralizada como um *software* na extremidade da rede. Isso implica em uma medição barata, precisa, sem depender dos nós folhas. Além disso, este protótipo realiza medições que não são possíveis de serem realizadas em abordagens tradicionais em *software*, por exemplo, medir uma sub-rede utilizando regra com agregador de fluxo. Os resultados obtidos com o protótipo de monitoramento aproximado demonstraram que o protótipo mede um volume 70 vezes maior de fluxos em relação ao protótipo exato, devido a abordagem ser aproximada e utilizar endereços *hash* para marcar os fluxos em estruturas *BloomFilters*. No protótipo exato, as regras de monitoramento são armazenadas na memória TCAM, de modo que o número de regras a serem monitoradas é limitado de acordo com o tamanho da memória.

Como trabalhos futuros pretende-se implementar outros tipos de métricas disponíveis na literatura, por exemplo, perda de pacotes e latência para ambos os protótipos. Além disso, adaptar e implementar ambos os protótipos em modelos de *NetFPGA* mais novos, SUME e CML. Devido ao recurso disponível pelo *hardware NetFPGA* 1G, muitas operações foram realizadas externamente no espaço do usuário por aplicações. Com estes modelos as limitações encontradas poderiam ser solucionadas e todas as operações poderiam ser realizadas internamente no *hardware*. Por exemplo, para o protótipo de monitoramento exato, mais contadores por fluxo poderiam ser implementados na memória TCAM. Para o protótipo de monitoramento aproximado, o processamento da medição poderia ser realizado internamente no *hardware*.

Referências Bibliográficas

Agarwal, S.; Kodialam, M. & Lakshman, T. (2013). Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pp. 2211--2219. IEEE.

Akyildiz, I. F.; Lee, A.; Wang, P.; Luo, M. & Chou, W. (2014). A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1--30.

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

Broder, A. & Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485--509.

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

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

CML, N. (2015). Netfpga cml. http://netfpga.org/site/#/systems/2netfpga-1g-cml/details/. Accessed on: 27/12/2015.

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, pp. 1738--1742. IEEE.

Goulart, P.; Cunha, Í.; Vieira, M. A.; Marcondes, C. & Menotti, R. (2015). Netfpga: Processamento de pacotes em hardware. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2015*.

Hu, Z. & Luo, J. (2015). Cracking network monitoring in dcns with sdn. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 199--207. IEEE.

Ishibashi, K.; Kanazawa, T.; Aida, M. & Ishii, H. (2004). Active/passive combination-type performance measurement method using change-of-measure framework. *Computer Communications*, 27(9):868--879.

Kekely, L.; Pus, V. & Korenek, J. (2014). Software defined monitoring of application protocols. In *INFOCOM, 2014 Proceedings IEEE*, pp. 1725--1733. IEEE.

Kreutz, D.; Ramos, F. M.; Verissimo, P. E.; Rothenberg, C. E.; Azodolmolky, S. & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. In *Proceedings of the IEEE*, volume 103, pp. 14--76. IEEE.

Lombardo, A.; Reforgiato, D.; Riccobene, V. & 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.

Moshref, M.; Yu, M. & 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*, pp. 73--78. ACM.

Naous, J.; Erickson, D.; Covington, G. A.; Appenzeller, G. & 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*, pp. 1--9. ACM.

NetFPGA (2015). Project switch openflow netfpga. https://github.com/NetFPGA/netfpga/wiki/OpenFlowNetFPGA100. Accessed on 22/11/2015.

NetFPGA (2016). Site netfpga. http://netfpga.org/site/#/systems/4netfpga-1g/applications/. Accessed on 22/08/2016.

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

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

Pacífico, R.; Silva, P.; Cunha, I.; Vieira, M.; Vieira, A. B.; Guedes, D. & Nacif, J. A. (2016a). Medição de desempenho de rede em hardware. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2016.*

Pacífico, R.; Silva, P.; Vieira, A. B.; Vieira, M. A. M. & Miranda Nacif, J. A. (2016b). Hardware modules for packet interarrival time monitoring for software defined measurements. *41st Annual IEEE Conference on Local Computer Networks - LCN 2016.*

Prasad, R.; Dovrolis, C.; Murray, M. & 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. & Fonseca, R. (2014). Planck: millisecond-scale monitoring and control for commodity networks. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pp. 407--418. ACM.

SUME, N. (2015). Netfpga sume. http://netfpga.org/site/#/systems/1netfpga-sume/details/. Accessed on: 27/12/2015.

Tarkoma, S.; Rothenberg, C. E. & Lagerspetz, E. (2012). Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131--155.

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*, pp. 1--8. IEEE.

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

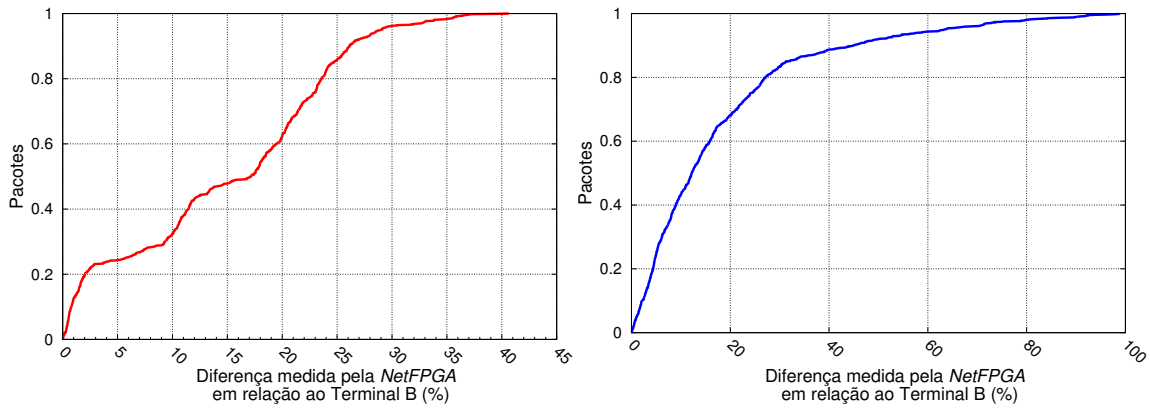
Yu, M.; Jose, L. & Miao, R. (2013). Software defined traffic measurement with opensketch. In *Symposium on Networked Systems Design and Implementation - NSDI*, volume 13, pp. 29--42.

A. Gráficos CDF - média e variância arquitetura aproximada

A.1 Experimento com 28 fluxos

- **Atraso fixo**

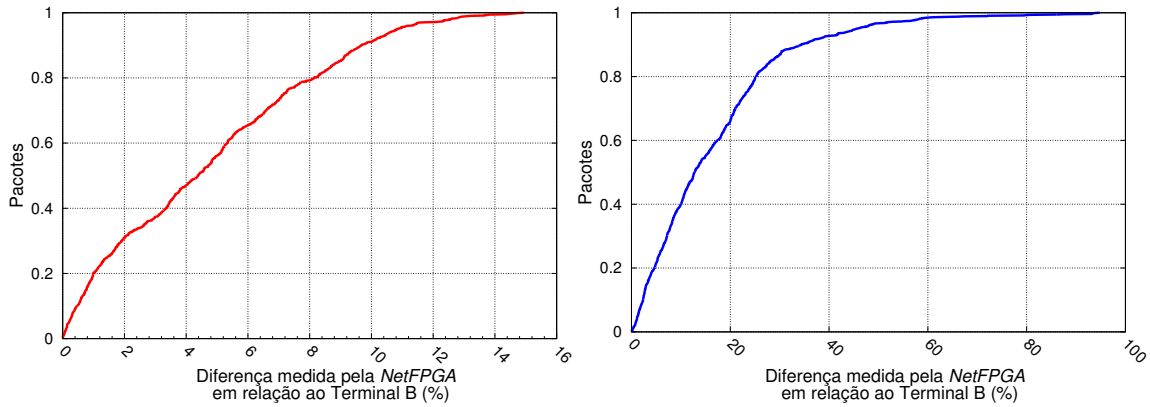
No gráfico da média (figura A.1a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 29,10%. No gráfico da variância (figura A.1b) 95% das medições possuem diferença relativa próxima de 63,53%.



(a) Média **(b) Variância**

Figura A.1: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso fixo.

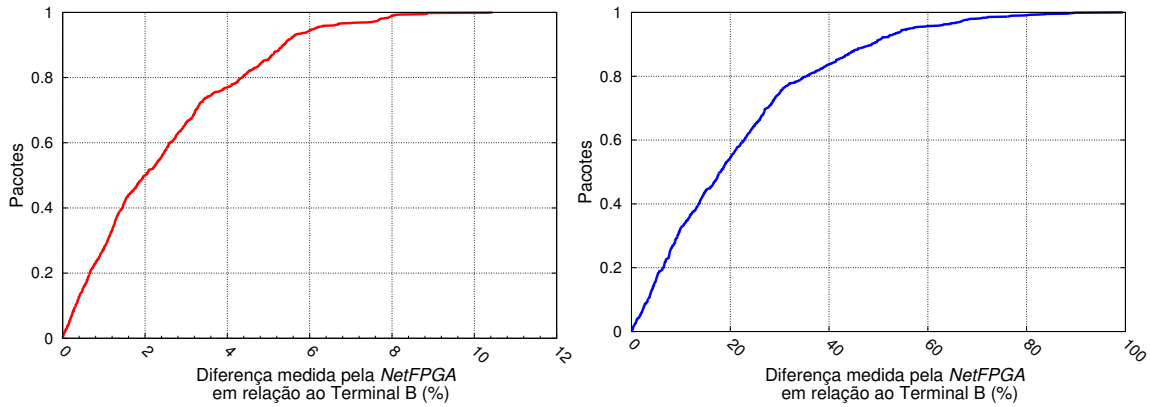
No gráfico da média (figura A.2a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 10,82%. No gráfico da variância (figura A.2b) 95% das medições possuem diferença relativa próxima de 45,57%.



(a) Média **(b) Variância**

Figura A.2: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso fixo.

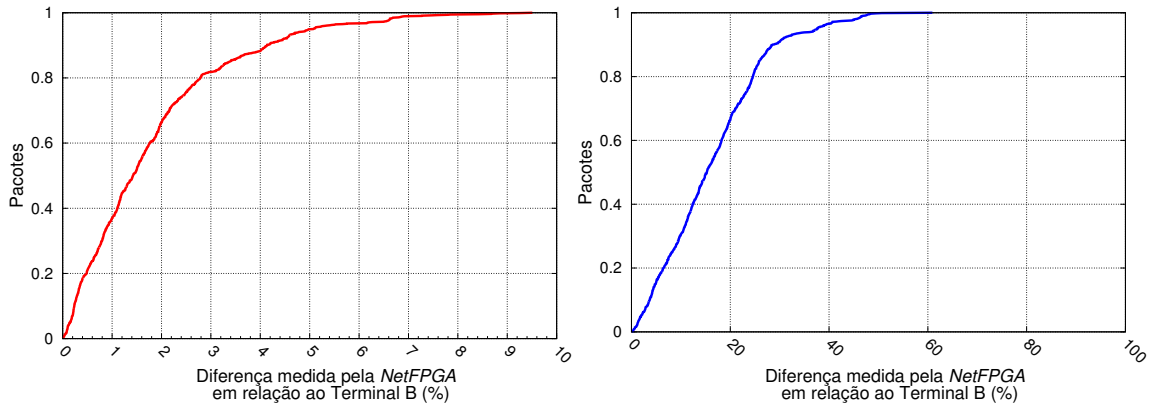
No gráfico da média (figura A.3a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 6,13%. No gráfico da variância (figura A.3b) 95% das medições possuem diferença relativa próxima de 56,55%.



(a) Média **(b) Variância**

Figura A.3: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso fixo.

No gráfico da média (figura A.4a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 5,07%. No gráfico da variância (figura A.4b) 95% das medições possuem diferença relativa próxima de 37,62%.

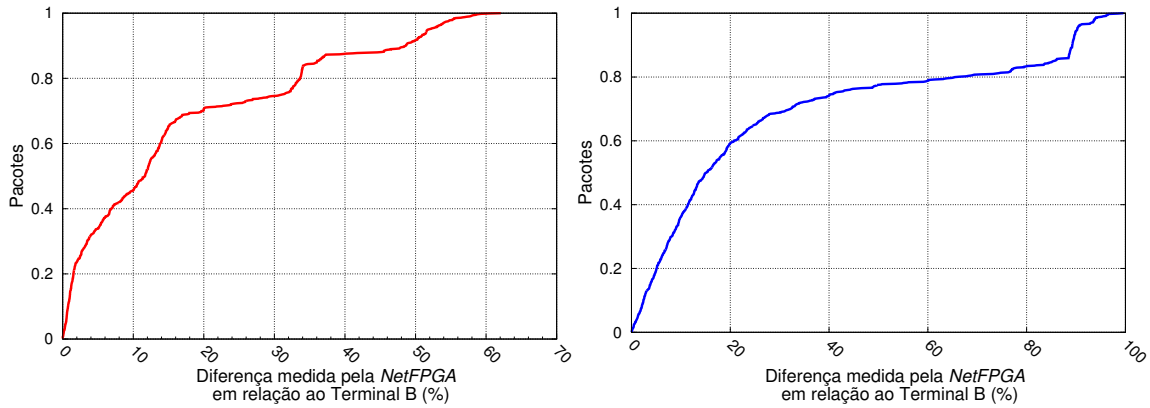


(a) Média **(b) Variância**

Figura A.4: Gráficos CDF média e variância: janela de tempo de 23 ms - atraso fixo.

■ Atraso aleatório

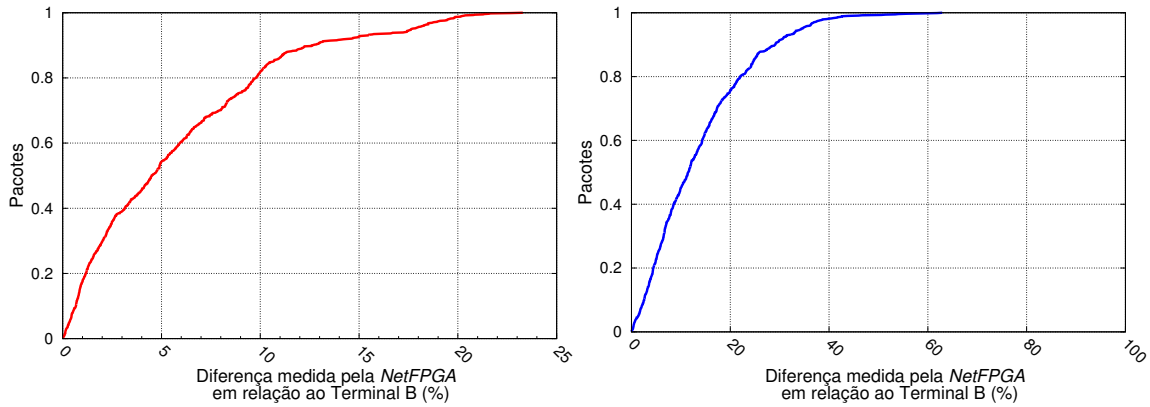
No gráfico da média (figura A.5a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 51,85%. No gráfico da variância (figura A.5b) 95% das medições possuem diferença relativa próxima de 90,24%.



(a) Média **(b) Variância**

Figura A.5: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso aleatório.

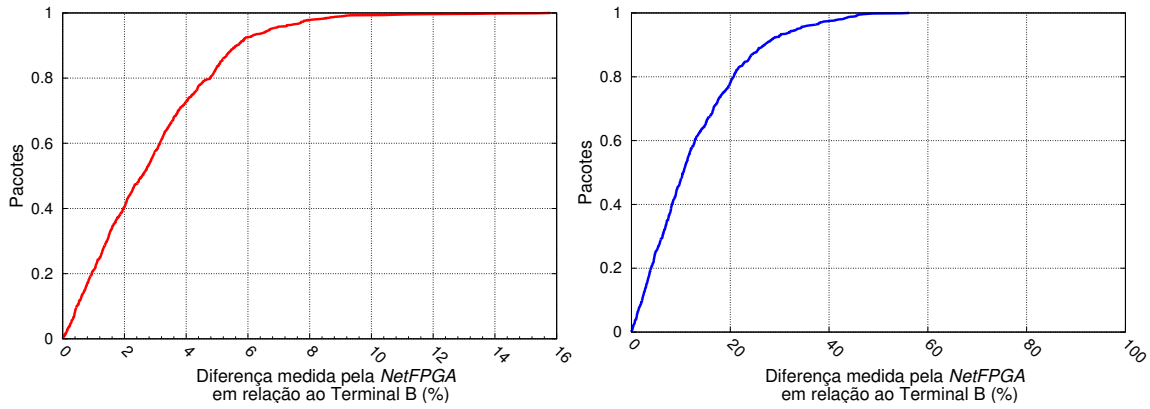
No gráfico da média (figura A.6a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 17,66%. No gráfico da variância (figura A.6b) 95% das medições possuem diferença relativa próxima de 34,43%.



(a) Média **(b) Variância**

Figura A.6: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso aleatório.

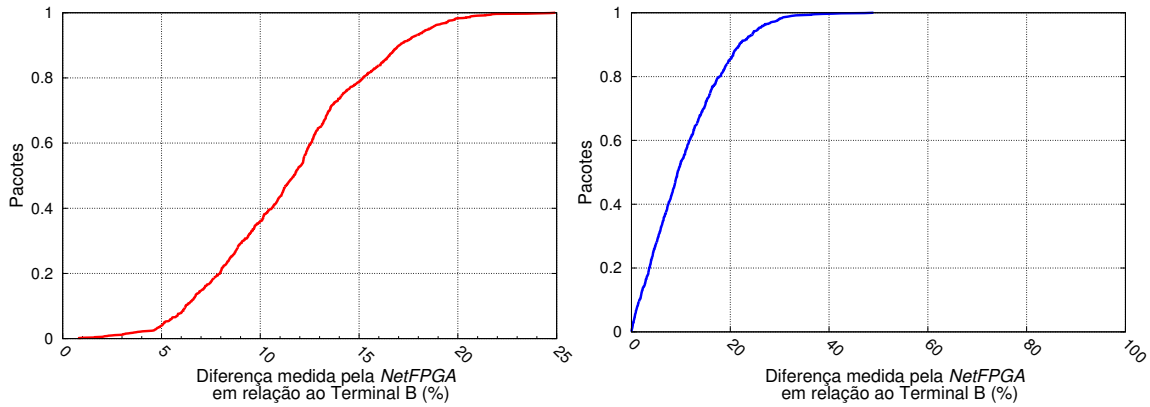
No gráfico da média (figura A.7a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 6,72%. No gráfico da variância (figura A.7b) 95% das medições possuem diferença relativa próxima de 33,54%.



(a) Média **(b) Variância**

Figura A.7: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso aleatório.

No gráfico da média (figura A.8a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 18,53%. No gráfico da variância (figura A.8b) 95% das medições possuem diferença relativa próxima de 25,72%.



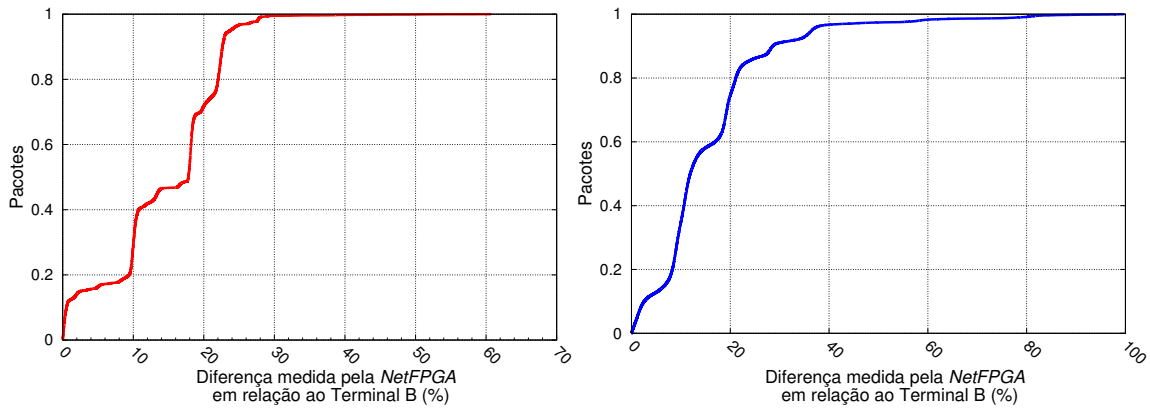
(a) Média **(b) Variância**

Figura A.8: Gráficos CDF média e variância: janela de tempo de 23 ms - atraso aleatório.

A.2 Experimento com 1.000 fluxos

- **Atraso fixo**

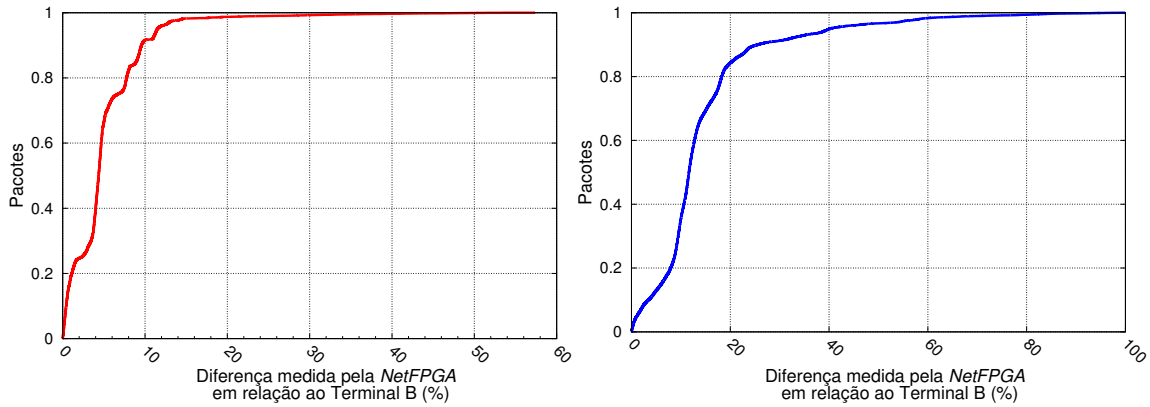
No gráfico da média (figura A.9a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 23,73%. No gráfico da variância (figura A.9b) 95% das medições possuem diferença relativa próxima de 36,56%.



(a) Média **(b) Variância**

Figura A.9: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso fixo.

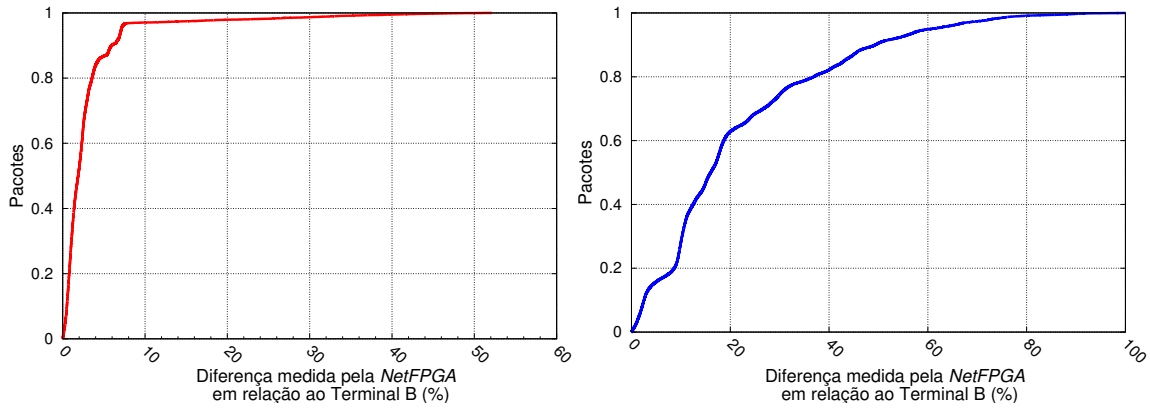
No gráfico da média (figura A.10a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 11,48%. No gráfico da variância (figura A.10b) 95% das medições possuem diferença relativa próxima de 40,17%.



(a) Média **(b) Variância**

Figura A.10: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso fixo.

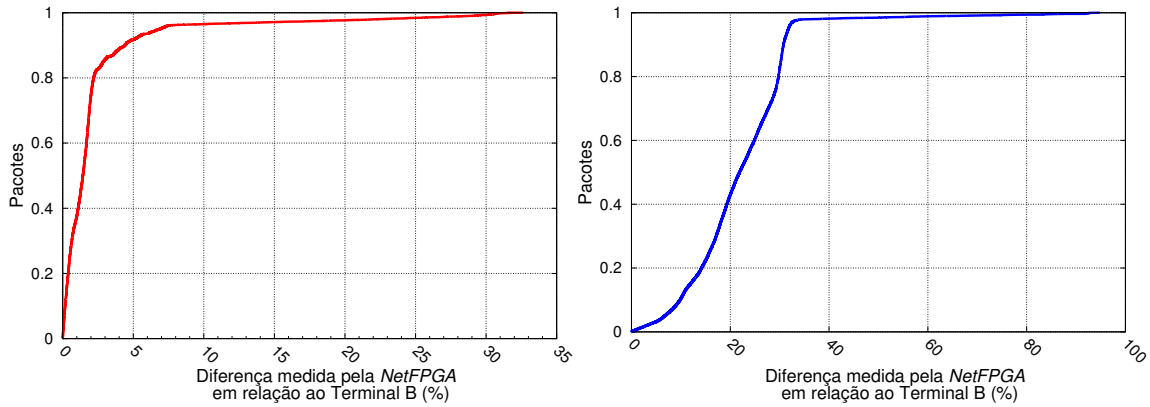
No gráfico da média (figura A.11a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 7.12%. No gráfico da variância (figura A.11b) 95% das medições possuem diferença relativa próxima de 60.69%.



(a) Média **(b) Variância**

Figura A.11: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso fixo.

No gráfico da média (figura A.12a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 6,88%. No gráfico da variância (figura A.12b) 95% das medições possuem diferença relativa próxima de 31,77%.

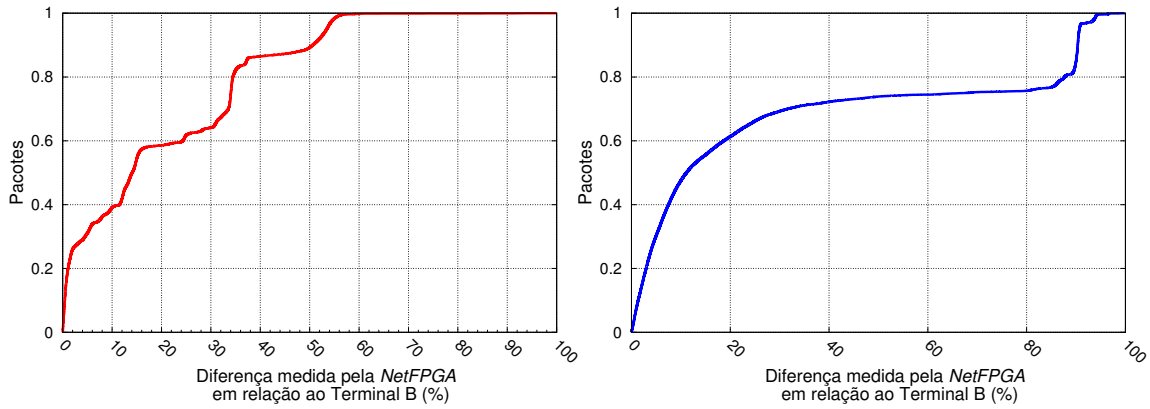


(a) Média **(b) Variância**

Figura A.12: Gráficos CDF média e variância: janela de tempo de 23 ms - atraso fixo.

■ Atraso aleatório

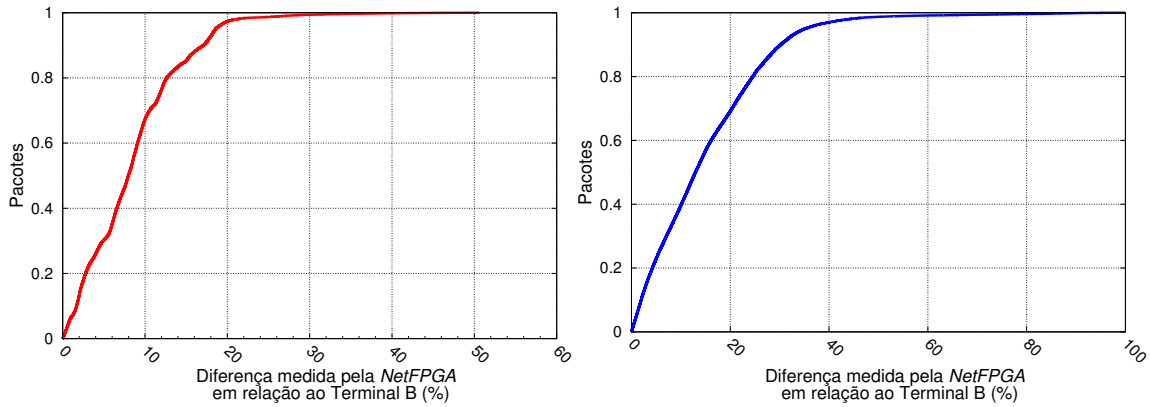
No gráfico da média (figura A.13a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 53,44%. No gráfico da variância (figura A.13b) 95% das medições possuem diferença relativa próxima de 90,67%.



(a) Média **(b) Variância**

Figura A.13: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso aleatório.

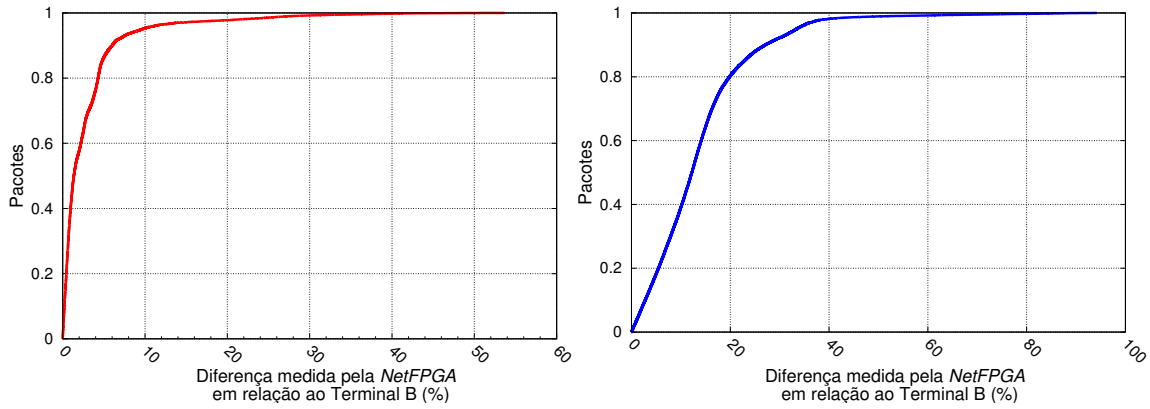
No gráfico da média (figura A.14a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 18,57%. No gráfico da variância (figura A.14b) 95% das medições possuem diferença relativa próxima de 35,15%.



(a) Média **(b) Variância**

Figura A.14: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso aleatório.

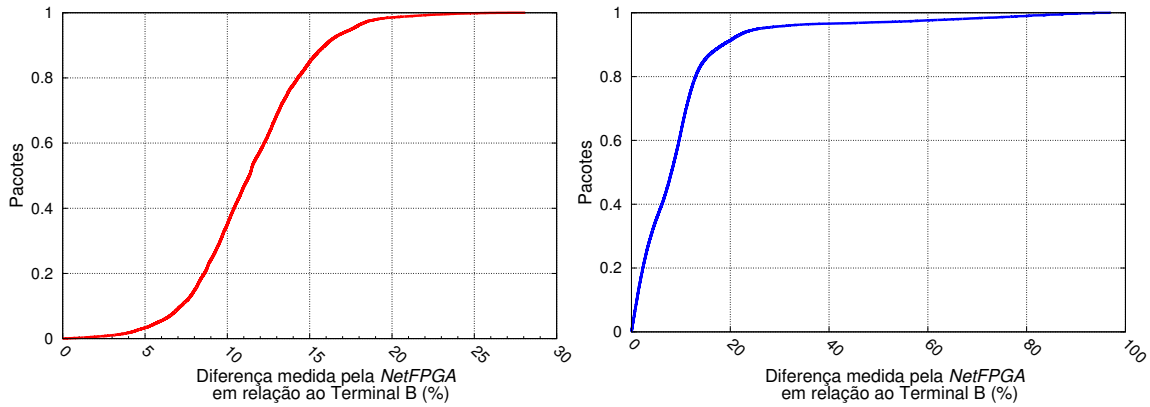
No gráfico da média (figura A.15a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 9,65%. No gráfico da variância (figura A.15b) 95% das medições possuem diferença relativa próxima de 33,42%.



(a) Média **(b) Variância**

Figura A.15: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso aleatório.

No gráfico da média (figura A.16a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 17,54%. No gráfico da variância (figura A.16b) 95% das medições possuem diferença relativa próxima de 25,73%.



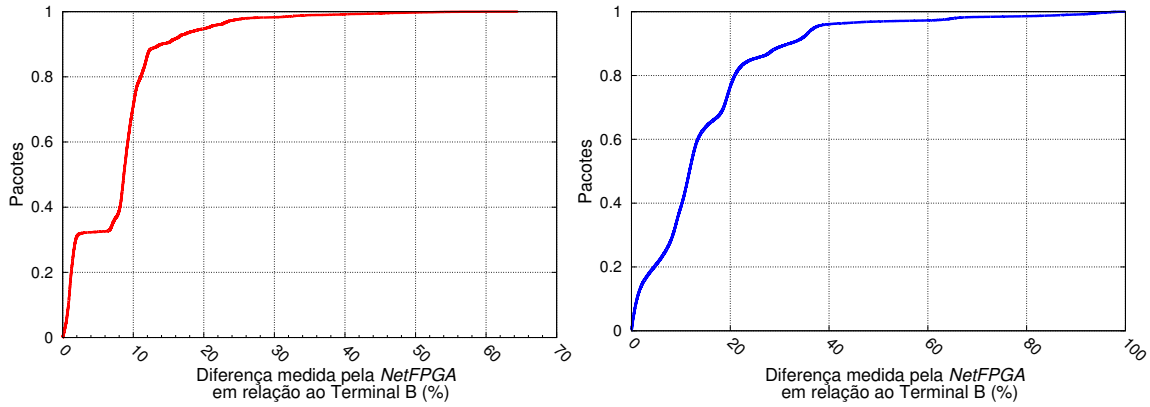
(a) Média **(b) Variância**

Figura A.16: Gráficos CDF média e variância: janela de tempo de 23 ms - atraso aleatório.

A.3 Experimento com 2.000 fluxos

- **Atraso fixo**

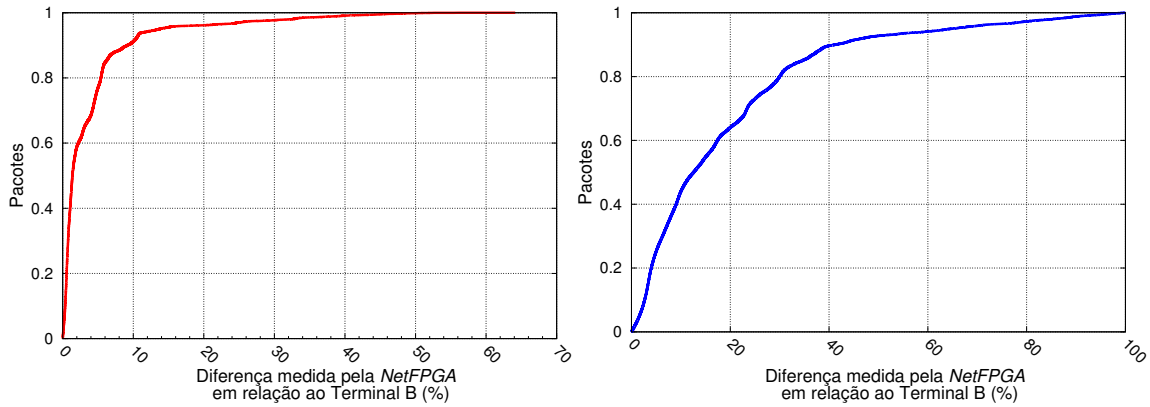
No gráfico da média (figura A.17a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 20,42%. No gráfico da variância (figura A.17b) 95% das medições possuem diferença relativa próxima de 37,08%.



(a) Média **(b) Variância**

Figura A.17: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso fixo.

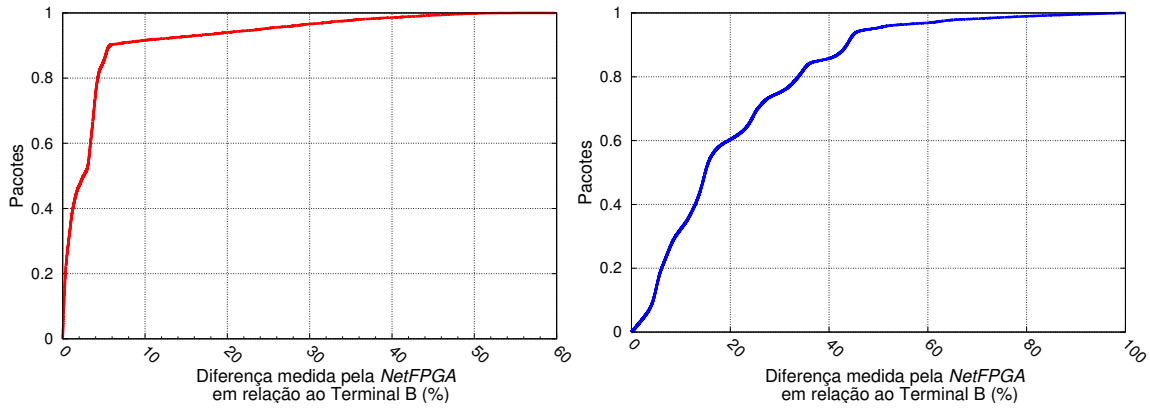
No gráfico da média (figura A.18a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 13,73%. No gráfico da variância (figura A.18b) 95% das medições possuem diferença relativa próxima de 64,80%.



(a) Média **(b) Variância**

Figura A.18: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso fixo.

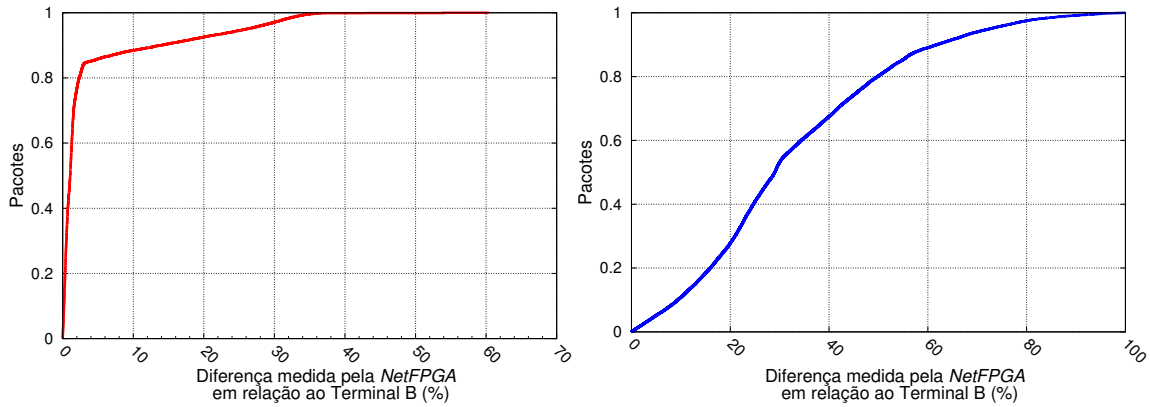
No gráfico da média (figura A.19a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 23,95%. No gráfico da variância (figura A.19b) 95% das medições possuem diferença relativa próxima de 48,57%.



(a) Média **(b) Variância**

Figura A.19: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso fixo.

No gráfico da média (figura A.20a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 26,02%. No gráfico da variância (figura A.20b) 95% das medições possuem diferença relativa próxima de 72,65%.

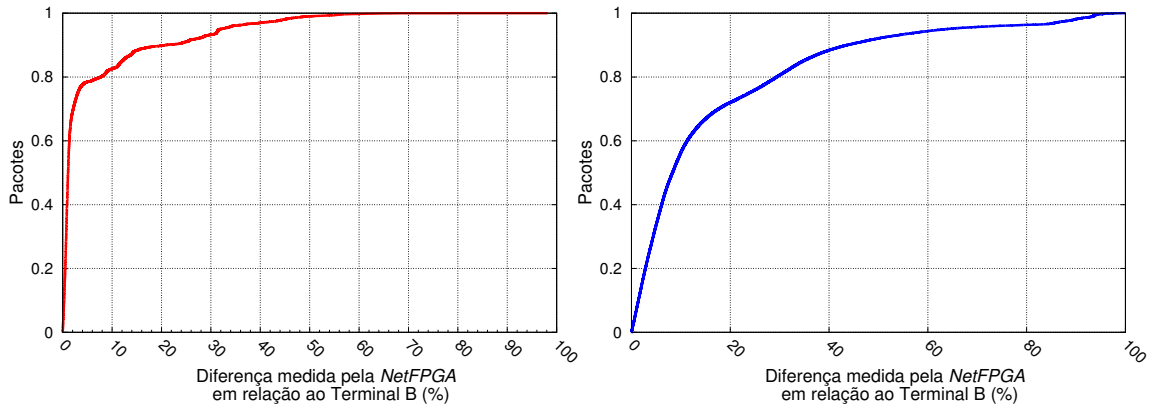


(a) Média **(b) Variância**

Figura A.20: Gráficos CDF média e variância: janela de tempo de 23 ms - atraso fixo.

■ Atraso aleatório

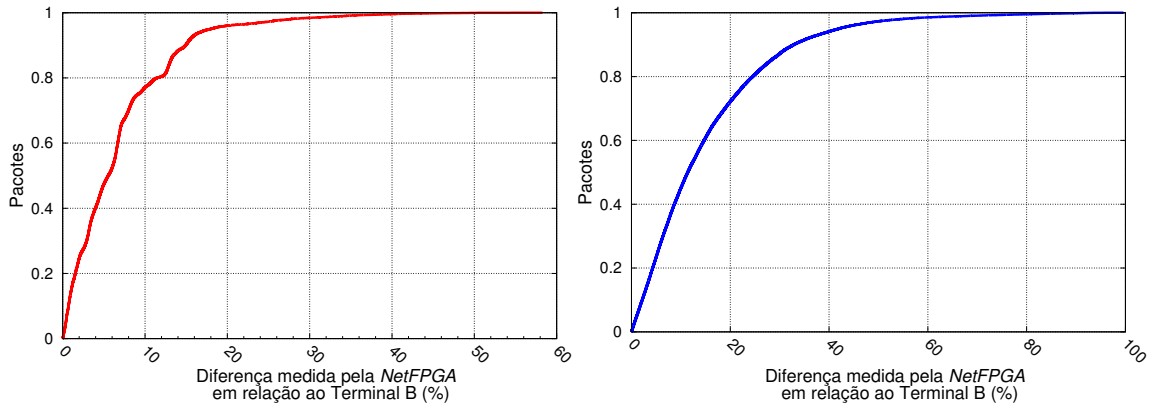
No gráfico da média (figura A.21a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 31,87%. No gráfico da variância (figura A.21b) 95% das medições possuem diferença relativa próxima de 63,78%.



(a) Média **(b) Variância**

Figura A.21: Gráficos CDF média e variância: janela de tempo de 2 ms - atraso aleatório.

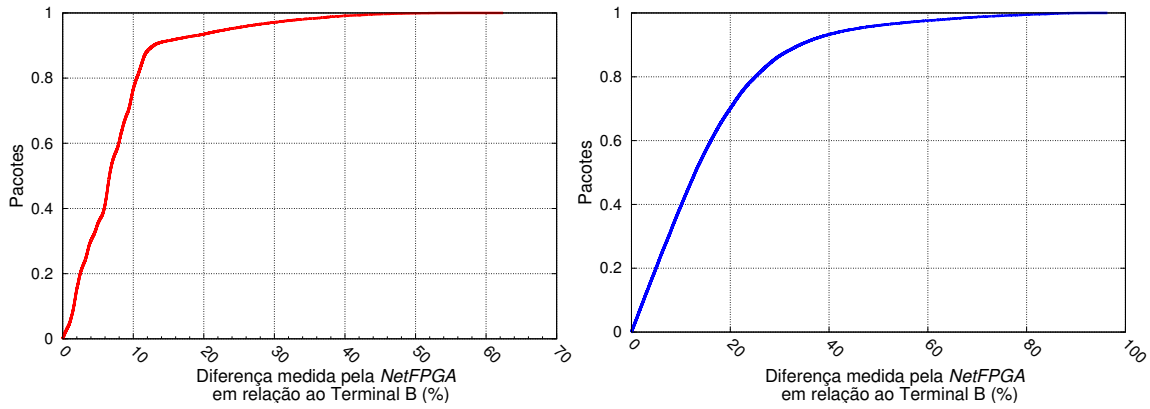
No gráfico da média (figura A.22a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 17,95%. No gráfico da variância (figura A.22b) 95% das medições possuem diferença relativa próxima de 41,99%.



(a) Média **(b) Variância**

Figura A.22: Gráficos CDF média e variância: janela de tempo de 4 ms - atraso aleatório.

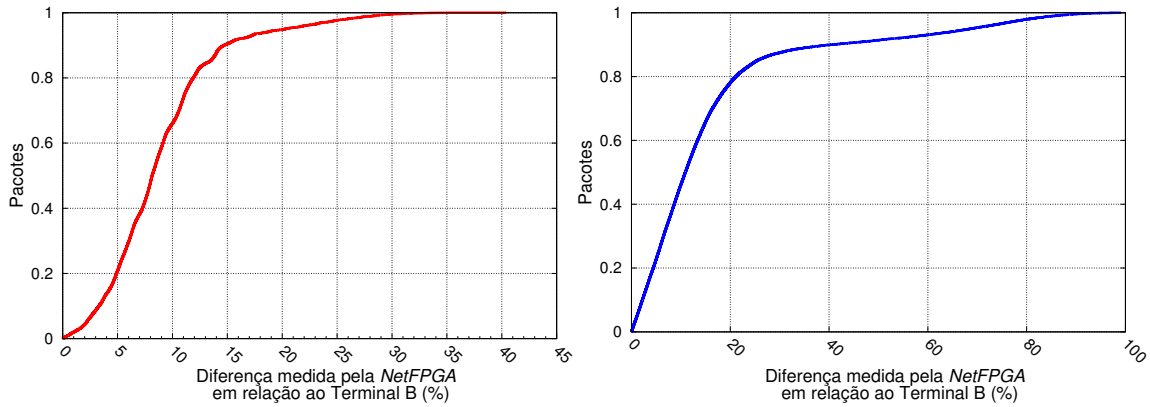
No gráfico da média (figura A.23a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 23,44%. No gráfico da variância (figura A.23b) 95% das medições possuem diferença relativa próxima de 45,27%.



(a) Média **(b) Variância**

Figura A.23: Gráficos CDF média e variância: janela de tempo de 8 ms - atraso aleatório.

No gráfico da média (figura A.24a) 95% da diferença relativa entre *hardware* e *software* estão próximas de 20,25%. No gráfico da variância (figura A.24b) 95% das medições possuem diferença relativa próxima de 68,66%.



(a) Média **(b) Variância**

Figura A.24: Gráficos CDF média e variância: janela de tempo 23 ms - atraso aleatório.