Revisão envolvendo a viabilidade da utilização de NFV em conjunto com SDN sobre servidores ARM de nova geração

Anderson dos Santos Paschoalon, *Universidade Estadual de Campinas* anderson.paschoalon@gmail.com

***Abstract* —** *Neste trabalho é feita uma revisão a respeito da viabilidade da utilização em conjunto das tecnologias NFV(Network Function Virtualization) e SDN(Software Defined Networks), sobre uma plataforma de hardware ARM. É feita uma introdução teórica sobre o assunto, uma exposição breve de como é a arquitetura das tecnologias NFV e SDN, bem como estudo das motivações, requisitos e desafios, com foco principal em NFV. SDN aqui é visto principalmente como uma possibilidade de que a utilização dessas tecnologias em conjunto, pode levar mais facilmente levar ao cumprimento de requisitos. É feita a análise de alguns resultados envolvendo funções de rede virtualizadas e redes definidas por softwares, que mostram conclusões importantes, como a dimensão dos overheads acrescentados, e pontos onde as tecnologias podem ser otimizadas. É também discutida a viabilidade da utilização de processadores ARM como servidores NFV, ao se analisarem diversos resultados recentes envolvendo desempenho computacional e energético dos mesmos comparados com outras arquiteturas x86. Foi também feita a análise de resultados de desempenho da virtualização sobre esta plataforma. Por fim é brevemente apresentado uma linha de processadores da arquitetura ARMv8 lançados em 2014, que tem grande potencial com novas tecnologias de rede.*

***Index Terms* — NFV, SDN, hipervisor, ARM, RISC**

I. Introdução

A. NFV: motivações e conceitos básicos.

Com o passar dos anos a dimensão e a complexidade das infra-estruturas de redes de computadores vem aumentando progressivamente. Isso se deve tanto ao aumento da demanda, quanto a diversificação dos serviços oferecidos, já que no modelo atual cada nova função de rede requer uma nova gama de equipamentos de hardware dedicado (chamados *middleboxes*), e uma infra-estrutura própria.

Essa dependência entre o serviço e um hardware específico é em grande parte decorrente do modelo clássico da pilha de rede, no qual certas funções acabam sendo dependentes de funcionalidades oferecidas por hardware dedicado e por plataformas proprietárias. Dessa maneira, acaba existindo uma dependência entre o tipo de serviço a ser oferecido e o suporte de hardware, o que faz com que para cada novo serviço, uma nova plataforma tenha que ser desenvolvida. Algumas conseqüências negativas desse paradigma são:

- Espaço físico: o lançamento de um novo serviço torna necessário mais espaço para a acomodação do novo hardware. [12]
- Escalabilidade e operação: devido a quantidade e diversidade desses equipamentos, novas instalações sempre demandam mais mão-de-obra especializada capaz instalar e manter o serviço operacional. Nesse caso, a escalabilidade dos novos serviços se torna cada vez mais difícil e complexa. [10, 12]
- Energia: o aumento da quantidade de equipamentos traz um inerente aumento dos gastos de , energia. Além disso, a ligação indissociável de cada hardware com um determinado tipo de serviço, faz com que este hardware sempre esteja ligado, mesmo que sua demanda seja baixa. Essa granularidade de funcionalidades não permite redistribuição de cargas [5].
- *Time-to-market*: cada novo serviço requer um novo ciclo de desenvolvimento de hardware, com um alto custo de capital, e pouco ou nenhum reaproveitamento da base tecnológica criada.Além disso a demanda por novos serviços está aumentando. Portanto, uma metodologia de redes baseadas no desenvolvimento de hardware não é mais efetiva no encontro das novas demandas. [5,10]
- Longo tempo de serviço: se por um lado, a quantidade de novos serviços aumentam, o tempo de vida de cada tecnologia tende a ser longa. As operadoras tem que manter uma grande infra-estrutura com constante qualidade de serviço, para relativamente poucos usuários, no caso de serviços mais antigos. Um exemplo de consistência de serviços de redes pode ser visto em telefonia. As operadoras tem que manter, ao mesmo tempo, suporte para 2G, 3G, e 4G. [10]

Nesse contexto, surge um novo paradigma que propõe uma modificação do modelo clássico de pilha de rede, realizando um desacoplamento entre o hardware e as funções de rede. Tal modelo é chamado NFV: *Network Function Virtualization*. Assim como definido por Masutani [10], NFV é um conceito que permite funções de rede e aplicações baseadas unicamente em software serem executadas em PC

servidores genéricos, ao invés de ser utilizado hardware dedicado.

Para isso, se torna indispensável o uso de técnicas de virtualização, que será responsável por permitir que diversas funcionalidades antes dependiam de hardware específico, sejam oferecidas pela camada de virtualização.

Na figura 01 temos uma abstração visual dessa modificação realizada na pilha de rede, onde vemos que a infra-estrutura de rede deve passar a ser executada na forma de aplicação sobre o sistema operacional e o hipervisor (também chamado de VMM: *Virtual Machine Monitor*). [3].

Dessa maneira é possível permitir que muitas funções e aplicações como *Firewalls*, *Network Adress Tranlation* (NAT), túneis servidores DHCP e *Cache* sejam executadas como funções de rede virtualizadas [5,10], visando consolidar o equipamento de redes em servidores, *storages* e *ethernet switches* de alto volume, chamados COTS (*Comercial Off-the Shelf*) (Figura 02) [12].

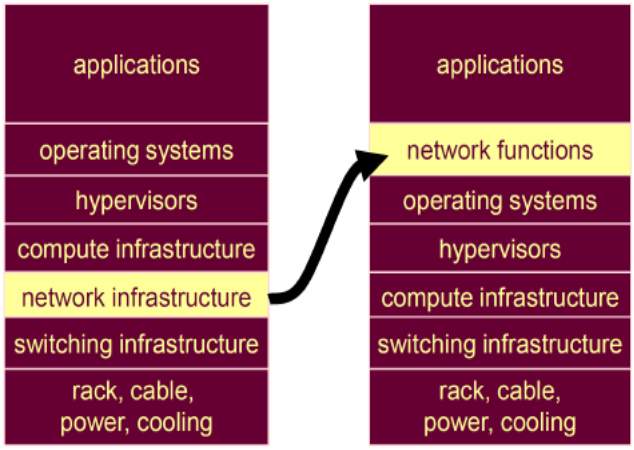

Figura 1: Modificação do modelo de camadas de rede OSI (esquerda), através do modelo NFV(direita).

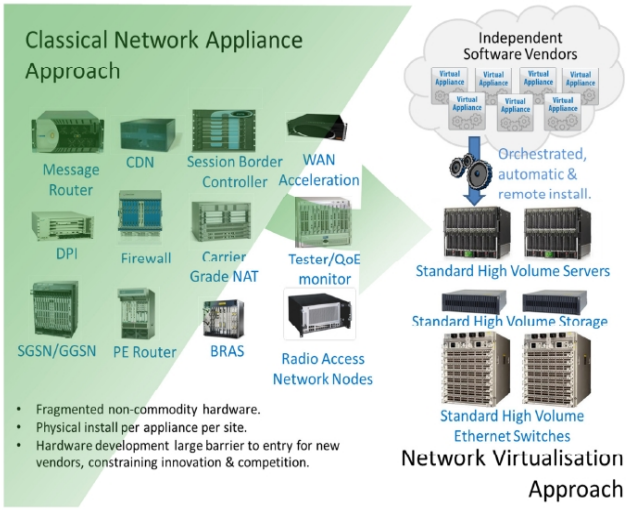

Figura 2: Mudança física do modelo clássico de redes, para o modelo NFV. O hardware passaria a ser do tipo COTS, e as funções de rede, implementadas puramente em software.

Tal mudança do paradigma tradicional para o NFV, traria diversas melhorias em termos financeiros e de inovação. Pode-se citar:

- Melhor aproveitamento do capital, não havendo necessidade de implantação de novo hardware para cada nova funcionalidade;
- Maior flexibilidade na escalabilidade de novas funções e instalações de rede;
- Rápida inovação, com a redução do time-to-market de novos serviços e produtos;
- Implantação rápida de novos serviços;
- Redução de gastos de energia, com a possibilidade de distribuição de cargas, melhor eficiência no aproveitamento do hardware, e utilização de equipamentos que geram maior economia energética, adaptando-se as necessidades elásticas do tráfico, ao invés da rede ser projetada para o pico [11];
- Padronização dos equipamentos de rede, facilitando imensamente a compatibilidade.

Além dessas vantagens, também seria possível desacoplar o software de hardwares proprietários, desvinculando dessa forma o cliente de um único vendedor específico de software. Isso respeita o princípio enunciado por Wolf [16] da "*escolha como um princípio*" para as novas arquiteturas de rede. A virtualização de funções de rede possibilitaria a criação de novas alternativas e soluções de software que fossem completamente independentes de plataforma de hardware proprietárias, restringindo menos as empresas que poderiam desenvolver novas tecnologias, o que uniria mais as áreas de Telecom e TI.

Serão, nas próximas sessões discutidos requisitos para essas metas serem atingidas, bem resultados já obtidos em outros trabalhos. Parte desses objetivos podem ser alcançados não só puramente através da tecnologia NFV como também através da colaboração mútua com a SDN (*Software Defined Network*).

B. Alinhamento da tecnologia SDN com NFV

SDN (*Software Defined Network*) consiste em uma diferente abordagem do tratamento de redes de computadores, rompendo também com o modelo clássico de pilha de protocolos. Suas principais motivações são o aumento da complexidade e tamanho das redes de computadores, e o aumento das taxas de transmissão [13]. Recentemente o protocolo OpenFlow validou e consolidou o modelo SDN [13], e já vem sendo adotado em muitos *datacenters*.

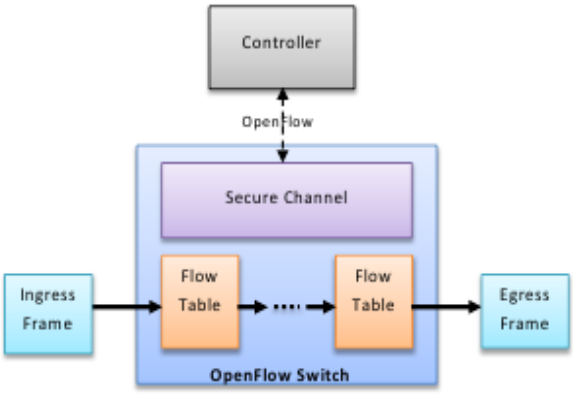

Figura 3: Diagrama lógico simplificado do funcionamento do protocolo

Nessa arquitetura existe uma separação lógica entre o plano de controle e o plano de dados. No protocolo OpenFlow um controlador SDN centralizado trabalha e controla múltiplos *switches*, para os quais envia informações que os permitem construírem suas "tabelas de fluxo". Os *switches* então comparam os *headers* dos pacotes recebidos com as entradas dessa tabela. Quando determinadas comparações são satisfeitas, ações são invocadas, como por exemplo mover o pacote para uma determinada porta, remover ou alterar um cabeçalho, ou move-lo para uma fila de prioridade [14]. Diferentemente dos protocolos pertencentes aos modelos clássicos de redes, o OpenFlow utiliza informações de cabeçalhos de qualquer nível da pilha de protocolos; ou seja pode utilizar cabeçalhos da camada de transporte(TCP, UDP) , IP, Ethernet; podendo emular então ações de qualquer nível de rede, ou mesmo novos tipos de serviços [14].

NFV e SDN são tecnologias independentes, podendo ser implementadas separadamente. Porém, ambas são complementares, possuindo motivações e objetivos semelhantes, as tornando altamente alinhadas. Metodologias que façam a separação do plano de controle e plano de dados como proposto pela SDN pode melhorar o desempenho do sistema como um todo, facilitar a compatibilidade e facilitar operações de manutenção [12]. O serviço SDN pode fornecer maior inteligência para a rede, facilitando a construção de redes mais robustas, e independentes do hardware [11]. Outra grande facilidade é a rede poder ser programada como uma entidade única, e a rápida adição de novas *features* por meio de um controlador central [11].

C. Utilização do processador ARM para NFV: Motivações

Atualmente a maioria dos CPUs de servidores tem arquiteturas baseadas no modelo CISC (*Complex Instruction Set Computer*), sendo a mais utilizada a x86 [2,6]. Porém, como descrito por Stallings[2] que cita trabalhos clássicos de Patterson e Tanembaum (ainda da década de 1980), a prova pratica mostra que arquiteturas RISC (*Reduced Instruction Set Computers*), para a execução de programas escritos em linguagem de alto nível, tendem a ter um desempenho superior [2].

De forma simplificada, podemos explicar isso pelo fato de que após um programa ser compilado, a grande maioria das instruções executadas, mesmo em programas muito complexos, tendem a ser operações simples como operação aritmética e de atribuições. Além disso, características como instruções de tamanho fixo, apenas uma instrução por ciclo de máquina, somente operações entre registradores, modos de endereçamento e formato de instruções mais simples diminuem tempo de ciclo de máquina e de execução de instruções simples (que são maioria), além de diminuir significativamente *overheads* em *pipeline*, facilitando a previsão de *brenchs*. [2]

Com o passar do tempo, arquiteturas CISC como a x86 foram incorporando diversas características da microarquitetura RISC, a fim de tentar se beneficiar das vantagens desse paradigma. O Inverso também é verdadeiro, apesar de na maioria dos casos os estudos apontarem vantagens da arquitetura RISC, a CISC também possuía seus próprios méritos que não podiam ser ignorados. Assim como as arquiteturas CISC tentaram se aproximar da RISC, o contrario também foi até certo ponto verdadeiro. [2]

A arquitetura x86 sempre se manteve no topo do mercado não embarcado (desktops, servidores, e *datacenters*), principalmente por prezar a compatibilidade com softwares anteriores. Sempre que há nova versão da arquitetura, instruções podem são adicionadas, mas nunca são removidas. [2]

A Arquitetura ARM se mantém ainda uma arquitetura RISC [2,15]. Conseqüentemente não possuem muitos dos *overheads* CISC. Ela se consagrou como a dominante em sistemas embarcados, mantendo-se nessa posição ate hoje. Isso lhe concedeu uma grande maturidade em características como operação com baixo gasto de energia, que é uma exigência muito recorrente e natural dessa fatia do mercado [15].

Por outro lado, hoje CPUs ARM vem aumentando fortemente seu desempenho, apresentando progressivamente desempenhos melhores. Atualmente as ultimas linhas de processadores ARM, além estarem fornecendo um alto desempenho, a ponto de rivalizar com desktops, também estão incluindo novas *feature*s como suporte à virtualização um set de instruções de 64 bits. Isso torna viável a utilização de processadores da família ARMv8 como servidores de *datacenters*, e conseqüentemente norma possível a utilização deles para implementação de NFVs.[8]

Somados, gastos com energia e com a dissipação de calor representam cerca de 30% de todos os gastos de um *datacenter* [6], e como já foi dito economia energética é um ponto forte dos processadores ARM. Dessa forma, tanto a economia energética quanto o potencial de desempenho da arquitetura ARM-RISC, ainda não explorado no nicho de mercado dos servidores, tende a tornar bastante interessante a prova de conceito da utilização de processadores ARMv8 como servidores NFV, já que tais possíveis benefícios da utilização do ARVv8 também são metas desse nova forma de implementação de funções de rede.

D. Organização do trabalho

O trabalho apresentado a seguir está organizado em mais 5 sessões.

A sessão "Requisitos para redes NFV, desafios e convergência com SDN" é dedicada em apresentar uma introdução teórica básica a arquitetura NFV, expondo também requisitos e desafios a serem alcançados. Também induz o conceito de SDN, expondo de maneira breve o protocolo OpenFlow, e em seguida metas em comum tanto do conceito de NFV quando do SDN.

A sessão seguinte: "Resultados experimentais com NFV e SDN" tem por objetivo contextualizar qual é o desempenho atual alcançado por implementações de SDN, e virtualização de funções de rede, investigar possíveis *overheads*, ou mesmo espaços onde certos pontos podem ser melhorados.

Em "Estudo da virtualização em processadores ARM e x86" é feita análise dos resultados obtidos pelo primeiro hipervisor Linux para processadores ARM multicores implementado. É um tipo de resultado importante já que é capaz de fornecer dados sobre o quão eficiente é a virtualização em um ambiente ARM, e aonde se localizam seus principais *overheads*, além é claro de locais onde novas otimizações podem ser feitas. Tal assunto é de fundamental importância por a implementação de um hipervisor NFV eficiente é de fundamental importância para a viabilidade ou não dessa nova tecnologia. Como as técnicas de visualização são menos maduras em ARM, possivelmente podem haver espaços maiores para otimização de desempenho.

Na sessão seguinte, "Revisitando o debate RISC vs. CISC em arquiteturas atuais" é feito o estudo comparativos de desempenho entre processadores ARM e x86. Também é feita uma análise breve de desempenho de processadores da arquitetura ARMv8, e uma análise das semelhanças dos resultados positivos do processador ARM e no que eles podem beneficiar as metas da metodologia NFV.

II. Requisitos para redes NFV, desafios e convergência com SDN

A. Arquitetura básica de uma infra-estrutura NFV

Antes de analisar os requisitos e desafios de para NFVs, será feita inicialmente uma breve análise da arquitetura definida pelo ETSI [1], e entender a função dos principais componentes da tecnologia.

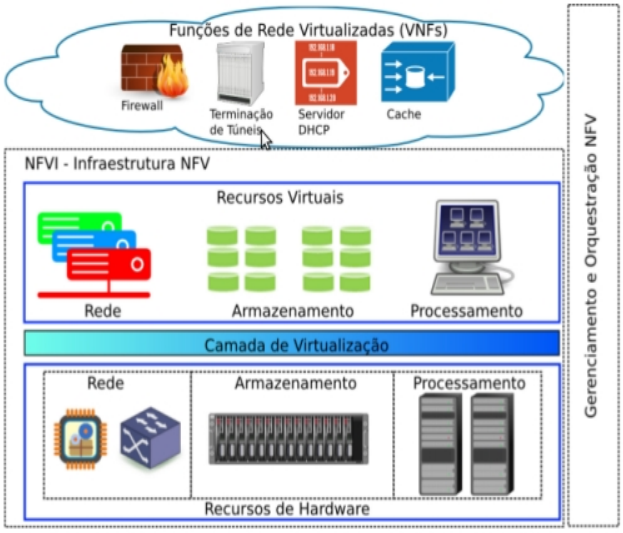

Figura 4: Arquitetura de alto nível de NFV.

De seus conjuntos principais, o de mais baixo nível da arquitetura NFV é chamado de NFVI, ou infra-estrutura NFV. É nada mais do que toda a infra-estrutura necessária, para criação e execução das funções de rede. É composta das seguintes partes:

- Recursos de hardware: recursos de rede, armazenamento e processamento;
- Recursos virtuais: necessários para as funções de rede em software, executadas a um nível acima;
- Camada de virtualização, que faz a função de desacoplar todo o software do hardware, garantindo dessa forma independência das aplicações do hardware, possibilitando o uso de hardware COTS (por exemplo, servidores de alto volume) ao invés de diversos *middleboxes* (ou seja, plataformas físicas dedicadas).

Apesar disso, essa quebra de continuidade do que havia sendo desenvolvido de infra-estrutura é menor do que possa parecer. Há certo tempo já é uma tendência muitas soluções oferecidas pelos vendedores serem baseadas essencialmente em hardware COTS. Porém eram feitas sobre um modelo proprietário, o que acabava impondo da mesma maneira as mesmas restrições que existiriam para um equipamento completamente específico. [12]

Para camada de virtualização não é definida uma solução específica, mas a que mais parece razoável é a utilização de hipervisores ou VMM (*Virtual Machine Monitor*), já amplamente utilizados para a virtualização de sistemas operacionais. Há porém algumas grandes diferenças entre hipervisores de NFV e hipervisores convencionais que precisam ficar bem claras. O hipervisor de uma NFV precisa ter acesso direto a funções de processamento de rede, além de

comunicação direta entre outras máquinas virtuais, para haver a possibilidade de execução de uma função de rede ou VNF (*Virtualizad Network function*) em mais de uma estrutura física. Outra diferença de virtualizações convencionais que pode ser adotada é a utilização de técnicas de virtualização leve; que se por um lado cria um isolamento menor, elimina certos *overheads*, melhorando o desempenho. Não há um nível único de virtualização específico que deverá ser adotado, já que não houve uma definição ainda. Há diversas possibilidades, podendo ela ser feita por meio de um hipervisor que é executado logo acima do hardware, executado acima de um sistema operacional, executado por meio de um contêiner Linux, etc. [9]

O próximo conjunto da arquitetura são as funções de rede propriamente ditas. Elas deverão executar sobre a NFVI, usufruindo, portanto de recursos virtuais de rede, processamento e armazenamento. Toda essa infra-estrutura funciona como uma plataforma unificada onde as aplicações de software vão trabalhar, sem preocupar sobre o que fisicamente está "abaixo". Isso pode ser visto como a consolidação da unificação entre as indústrias de Telecom e TI. Como agora não há mais dependência entre recursos de hardware proprietário e fechados, há a possibilidade de uma colaboração mútua e mais natural entre times dessas áreas. Além disso, o fato do desenvolvimento de novas funções de rede ser puramente baseado em software possibilita um ciclo de desenvolvimento menor, e, portanto com maior inovação, e menores custos. [11]

Omitido na figura 4, mas não menos importante está o nível chamado OSS (*Operational Support System*), que se comunica tanto com as VNFs quanto com o nível de gerenciamento e orquestração. É responsável por suportar processos das operadoras como inventario de rede, configuração de elementos da rede, providenciar serviços e gerenciar falhas. [9]

O nível de gerenciamento e orquestração trabalha sobre todo o sistema instalado, de maneira centralizada, realizando serviços como [11]:

- Computação dos elementos virtualizados e mapeamento da topologia da rede, para uma comunicação com a rede física adequada;
- Um modelo de dados comuns, mantendo uma visão consistente da topologia da rede para todas as NFVI, mantendo controle das conexões virtuais e físicas, e o estado de cada ponto, permitindo o oferecimento de serviços fim-a-fim;
- Gerenciamento dos serviços, mantendo o controle do estado corrente dos serviços e de seus recursos;
- Comunicação e integração com o OSS;
- Política de gerenciamento, com intuito de manter a consistência e segurança da rede.

Uma vantagem que a virtualização fornece é o fato de que também não há necessariamente um vinculo lógico entre um recurso de físico, e a aplicação sendo executada. A camada de virtualização pode (e deverá) ser executada de maneira distribuída possibilitando que uma determinada função de rede esteja desacoplada da localização física dos recursos que ela estará usando. Uma representação ilustrativa disso é apresentada na figura 05, que mostra a comunicação entre dois pontos, "ponto A" e "ponto B", entre os quais há um determinado números de NFVIs (chamadas NFVIs-PoP : *Network Function Vitualization Intrastructure Point of Presence*). Sobre elas há uma camada de virtualização, e nela são executadas as funções de rede VNFs (*Virtual Network Functions*). A comunicação fim-a-fim é feita no plano lógico através da comunicação pelo intermédio de diversas funções de rede VNFs. No exemplo esta comunicação lógica não corresponde a comunicação física que era realizada de fato, podendo, por exemplo, uma única NFVI estar executando mais de uma das VNFs usadas. [5]

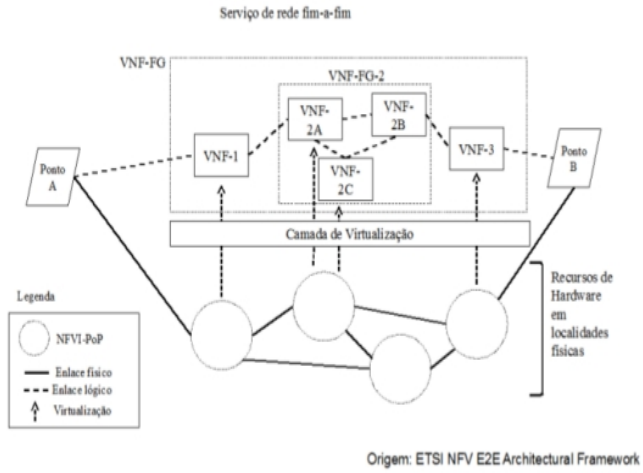

Figura 5: Exemplo de cenário de comunicação fim-a-fim entre um "Ponto A" e um "Ponto B", mostrando que os enlaces lógicos e físicos são desvinculados.

É importante deixar claro que há uma completa independência do tipo de serviço virtualizados oferecido pelo tipo de arquitetura do hardware propriamente dito existente "abaixo" do hipervisor. Os servidores padrões não estão limitados a apenas uma única arquitetura, por exemplo, a x86 (que é majoritária nesse mercado). Havendo uma camada de virtualização que ofereça os recursos necessários para a criação dos recursos virtuais necessários, os servidores podem ser literalmente quaisquer tipos de máquinas, por exemplo, ARM ou PowerPC. Isso é portanto uma grande motivação para a utilização se servidores ARM, que vem ganhando espaço no mercado de servidores (hoje dominado pelo x86), devido a grande experiência dos modelos desta arquitetura em computação de baixo gasto energético. Redução dos gastos energéticos é uma das metas prioritárias do mercados de servidores hoje. [19]

Além disso, o fato de os serviços estarem desvinculados das NFVIs possibilita que em um momento de pouca demanda máquinas sejam desligadas, poupando energia; ou mesmo garantir a possibilidade de ser usada uma grande quantidade de processadores relativamente modestos para execução de uma tarefa que exigira na arquitetura clássica um único equipamento de grande capacidade computacional (dividir para conquistar). [5]

B. Requisitos de software para NFV, e desafios

Para se tornar viável a implantação intra-estruturas NFV, tornando-as competitivas com as redes clássicas já existentes, podemos delinear diversas metas a serem alcançadas. As principais foram definidas em [12] e são:

- Portabilidade e interoperabilidade de softwares NFV;
- Um *trade-off* entre performance e custos que realmente seja vantajoso – já que deve ser levada em consideração uma queda de performance pela utilização de hardware genérico ao invés de um hardware dedicado;
- Migração e coexistência entre as plataformas existentes e as NFV implantadas – todo novo equipamento implantado deve ser compatível e coexistir com o antigo;
- Automatização da implantação da infra-estrutura;
- Orquestração e gerenciamentos consistentes;
- Segurança e resiliência – a existência de virtualização em um servidor causa brechas de seguranças em suas camadas de operação (Sistema Operacional, hipervisor, hardware), e a determinação de uma função executando sobre um conjunto de equipamentos (servidores/*switches* COTS), podem criar outras falhas de seguranças novos níveis de virtualização. E caso alguma NF venha a falhar, há a necessidade de ela ser recriada, sob determinadas métricas;
- Estabilidade de rede.

NFV é uma tecnologia que ainda não foi completamente padronizada, e esse é um processo que leva tempo. No entanto, em uma tentativa de se atingir tais objetivos, Masutani et al [10] delineou 7 requisitos de software para os nós da rede e para o *midleware* (software responsável pela comunicação entre componentes de sistemas distribuídos), menos abstratos que os listados acima em uma tentativa de se alcançar uma rede mais flexível e elástica, possibilitando a operação conjunta com SDN/OpenFlow. Eles estão listados abaixo:

1. Alta performance e menor latência no processamentos de pacotes: servidores padrões não são desenvolvidos pensando-se em desempenho de rede, e sim visando alta performance computacional. Dessa forma se torna necessários desenvolvimento de técnicas que possibilitem uma maior eficiência e performance em processamento de pacotes, e menos latência na comunicação com a interface de rede, para o sistema operacional.
2. Fácil instalação: o processo de instalação deve ser simples, dinamicamente instalado, sem a necessidade de um operador. Para isso de faz necessária independência do sistema com o hardware. Além disso, é necessária compatibilidade comunicação entre softwares fornecidos por diferentes vendedores.
3. Isolação e *multi-tenancy*: *multi-tenancy* consiste em um servidor manter diversas instâncias de um mesmo processo para diversos inquilinos, cada um deles visualizando um mesmo ambiente instanciado. Além disso, isolação de controle, desempenho e falhas deve ser garantida. Esse é um requisito que não é oferecido em servidores padrão, logo deve ser provido via software. Nesse sentido o ultimo kernel do Linux suporta a opção "*isolcpus*" que é capaz de alocar recursos de uma CPU exclusivamente para um processo. Esse tipo de recurso pode ser uma forma de se garantir *multi-tenancy* e desempenho mesmo em um ambiente virtualizados. Da mesma maneira isolação das implementações de segurança também deve ser necessária, para haver a possibilidade de aquisição de funções de diversos vendedores.
4. Controle fino das funções de rede: esse é um dos pontos de encontro entre o paradigma NFV e a tecnologia SDN. Os conceitos de OpenFlow nessa contexto pode garantir um controle abrangente e flexível da rede, e ao mesmo tempo garantir a consistência por ser centralizado. OpenFlow é capaz de isolar o trafego de entrada, em tráfegos de pacotes individuais especificados por serviço.
5. Múltiplas interfaces de controle para sistemas externos: esse requisito é necessário para se garantir a escalabilidade dinâmica de uma rede NFV, e controle cooperativo dos recursos de hardware existentes.
6. Monitoramento: Com servidores padrões são mais frágeis em termos de suporte a falhas do que equipamentos de redes dedicados, é necessário haver funções de software capazes de administras falhas de sistema, seja hardware SO, *middleware* e do software de funções de rede.
7. Tolerância a falhas: além de haver a necessidade de implementação de funções que podem administrar falhas, é necessário haver também um mecanismo de tolerância a sua ocorrência. Assim como aplicado a computação de nuvem, técnicas de redundância e balanceamento de carga podem ser aplicadas.

Ainda assim, é necessário se destacar outros pontos, de maneira mais detalhada, que devem ficar claros como metas, ou como problemas a serem levados em consideração na implementação, e levados em conta dentro dos requisitos de software já definidos, ou mesmo serem acrescentados, descritos por descritos por ROSA et al [5]:

- Semântica de processamento: é um requisito que exige um monitoramento especial, já que alguma forma errada de processamento (alterar de maneira incorreta um cabeçalho, provocando seu descarte) pode tornar não mapeável para o provedor de serviços se o descarte foi causado pela rede (*buffers* cheios) ou pela semântica incorreta do processamento do pacote;
- Formas de se garantir desempenho: caso uma NF esteja operando sob anormalidades geradas por outras NFs, não é possível para um cliente garantir o consumo de recursos utilizado por uma NF. Uma maneira de tentar contornar esse problema, pode ser através da função "*isocpus*".

- Eficiência vs. Complexidade: se por uma lado *chipsets* de propósito único, tendem a ser mais limitados, muitas vezes não podem ser utilizados em cenários de grande complexidade. Isso pode ser um ponto que inclusive favoreça a implantação de NFVs em ambientes onde processamento mais pesado é necessário.
- Interface: a consistência entre PF (ponto fixos) e funções virtualizadas (VNF) é dependente da existência de interfaces apropriadas para a comunicação entre esses dispositivos.
- Resiliência : dentre métricas de resiliência pode ser estabelecidas : taxa máxima de perda de pacotes não intencional, variação máxima de atraso e latência, o tempo máximo de detecção e recuperação de falha, e taxa máxima de falhas de transações validas.

C. Alinhamento da tecnologia NFV com a SDN

Por fim, como dito anteriormente, SDN e NFV são tecnologias independentes, porém que tem objetivos em comum. Sua utilização conjunta, como já mostrado no requisito 4, pode favorecer o alcance em conjunto de certos resultados. Protocolos SDN como OpenFlow podem estabelecer um controle fino e centralizado da rede, possibilitando a criação de algoritmos eficiente de medição de cargas, e balanceá-las através do redirecionamento de pacotes.

O protocolo OpenFlow iria então atuar de maneira paralela ao gerenciador e orquestrador, estando este não somente limitado a comunicação entre entidades virtuais, mas também poderia agir sobre os sistemas operacionais dos servidores, permitindo a obtenção de dados das máquinas físicas, ou mesmo atuar sobre os switches COTS, sendo estes *switchs* OpenFlow. Na Figura 6 é apresentado um esquemático de 3 opções possibilidades de desenvolvimento da estrutura NFV em conjunto com SDN/OpenFlow.

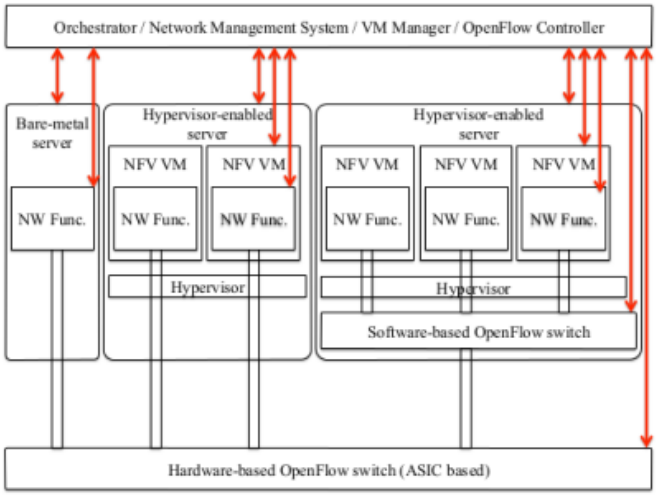

Figura 6: Três opções de desenvolvimento de NFV, utilizando em conjunto um controlador OpenFlow

Além disso, é preciso ressaltar que o protocolo OpenFlow, apesar de ser o padrão SDN mais difundido, não é a única opção existente em desenvolvimento. Por exemplo Hata [14] propõe uma diferente arquitetura do modelo tradicional OpenFlow. Dentre as diferenças estão modificações na estrutura do processador de pacotes, que possibilita a utilização de lógicas condicionais, ou mesmo loops (diferentemente do OpenFlow, que possibilita somente execução de ações em cadeia), que dá ao programador da rede maior flexibilidade, e a leitura de cabeçalhos de protocolos pertencentes a camada de aplicação como por exemplo o HTTP e o SIP. Na sub-sessão 2.B é comparado o desempenho do protocolo ProGFE com o OpenFlow. O ProGFE também é um protocolo SDN, sendo considerado um *superset* do OpenFlow.

III. Alguns resultados experimentais relacionados com NFV e SDN

A. Desempenho do OpenFlow no plano de dados

Nesta sub-sessão será feita uma análise de alguns resultados obtidos por Bianco et al [20], comparando-se o desempenho do protocolo OpenFlow no processamento de pacotes, com uma implementação de repasse nativa do sistema operacional Linux, ambos sendo executados sob uma mesma distribuição de Linux, em um PC desktop.

Pelo fato do desktop utilizado ser um computador arquitetura x86, espera-se que o desempenho medido nos testes seja análogo (ou ao menos não muito diferente) ao que um servidor x86 COTS teria ao executar essas aplicações. Esse resultado é desejável, já que pode indicar como é o desempenho desse protocolo, sem a utilização de uma plataforma dedicada, no caso, um *switch* OpenFlow.

O setup experimental utilizado está apresentado na figura 7.

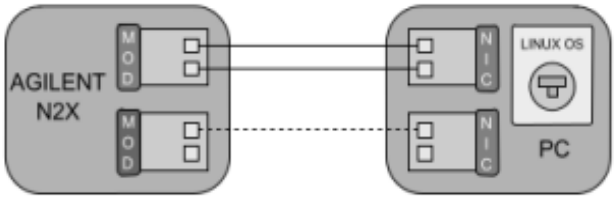

Figura 7: Setup experimental utilizado para teste do OpenFlow no Linux. A esquerda, um roteador de testes da Agilent responsável por gerar tráfego, e à direita um PC padrão Intel, portando o sistema operacional Linux.

Nessa montagem está de um lado um roteador de testes da Agilent N2X responsável por gerar trafego entre suas interfaces, e do outro um PC padrão com o sistema operacional Linux Ubuntu 8.10, kernel versão 2.6.27 (*bridging* e NAPI habilitados), uma placa mãe C2SBX, um processador Intel Core2 Duo e 8 Gbyte de DDR3 1066MHz RAM.

A função de roteamento padrão foi executada com a função de repasse disponível no kernel, e a função de *switch*

foi feita ativando-se o modo *bridge* entre as interfaces do PC (modo *bridge* ativado durante a compilação do kernel).

Na figura 8 vemos a taxa de transferência em Mbits/s de acordo com o tamanho dos pacotes enviados em bytes. Notamos que o OpenFlow tem um desempenho semelhante ao oferecido pela camada de roteamento, sendo na realidade muito próximo da carga total oferecida pelo equipamento Agilent. Tanto que o desempenho dos três (OpenFlow, camada IP e carga oferecida pelo equipamento) se igualam a partir de pacotes maiores que 256 bytes). Todos eles superam em muito o processamento realizado pela camada MAC do Linux.

Realizando agora testes no pior caso de desempenho medido anteriormente, ou seja, com pacotes com apenas 64 bytes de comprimento, foi calculado a taxa de transferência (figura 9) e a latência (figura 10) baseados na relação percentual com a máxima carga que o link suporta. Vemos novamente os resultados do protocolo OpenFlow não tão distantes do máximo fisicamente realizável.

Por fim (Figura 11), é apresentado mais um resultado obtido no estudo, no qual é testado o desempenho do protocolo OpenFlow ao se utilizar como método de busca em sua tabela primária (de 100 entradas) a busca linear e via *hash*. Nota-se que mesmo a tabela sendo bastante pequena a utilização de busca linear (de complexidade $O(n)$) piora em 33% o desempenho, se comparado a busca *hash* (complexidade $O(1)$). Isso mostra que no plano de dados, a busca na tabela primária corresponde a uma parcela muito significativa do processamento total executado por esse protocolo.

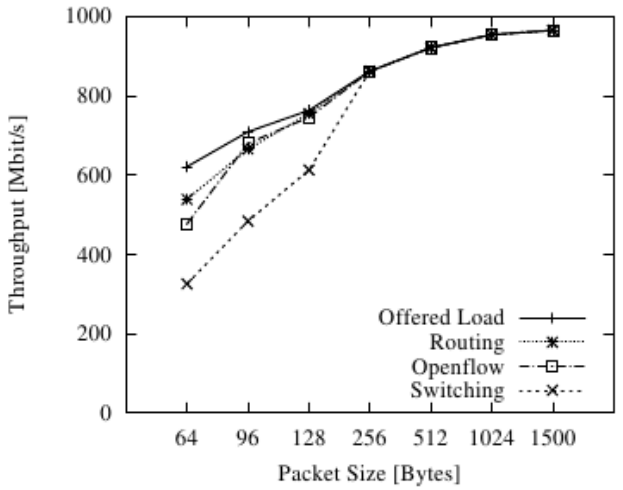

Figura 8: A taxas de transferências obtidas pelo OpenFlow, roteamento oferecido pelo kernel do Linux e switching obtido pela ativação do modo bridge entre as interfaces. É mostrada também a carga oferecida pelo equipamento Agilent.

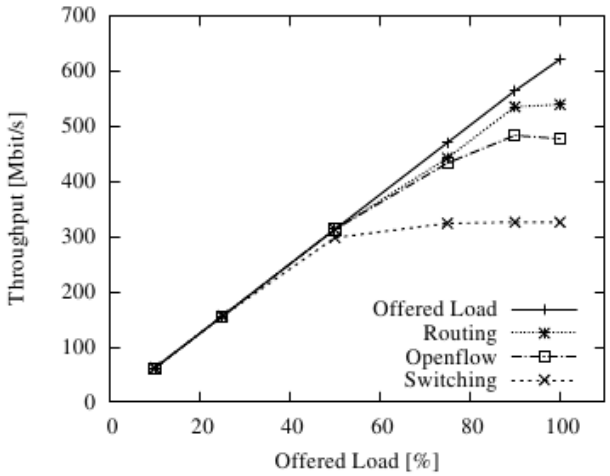

Figura 9: Taxa de transferência obtida variando-se a carga oferecia (100% corresponde a capacidade total do link), para pacotes de 64 bytes.

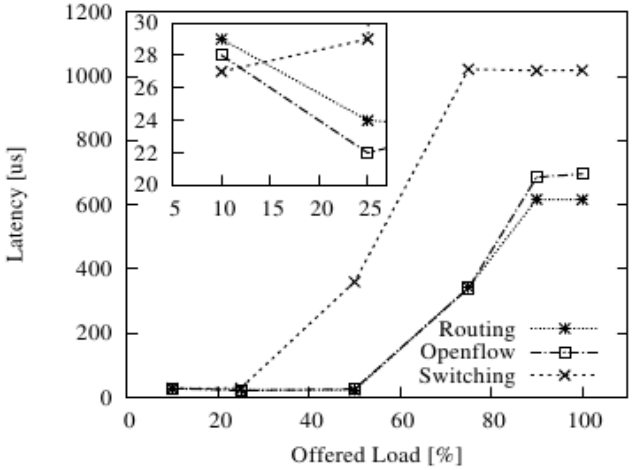

Figura 10: Latência obtida variando-se a carga oferecia (100% corresponde a capacidade total do link), para pacotes de 64 bytes.

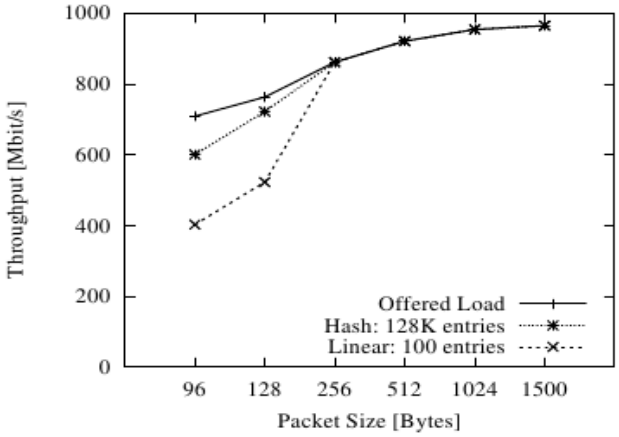

Figura 11: Taxa de transferência obtida pelo OpenFlow ao se utilizar busca linear e por hash na tabela primária de 100 entradas, ao se variar o tamanho dos pacotes enviados. Nota-se uma melhoria substancia na eficiência para pacotes de tamanho pequeno, ao se utilizar busca hash ao invés de busca linear.

Estes resultados são importantes, pois dão uma perspectiva do desempenho do OpenFlow sobre plataformas COTS. E são resultados bastante promissores, já que mostram um desempenho relativo muito bom, nesse ambiente, mesmo sendo uma tecnologia recente.

Além disso, os resultados podem ser uma referência do que se esperar do processamento via software sobre COTS de funções de roteamento e *switch*.

B. Desempenho experimental de um VSR (Virtual Software Router)

Nesta sub-sessão será feita uma análise dos dados obtidos por Rojas-Cessa et al [9], no qual é comparado o desempenho entre Roteadores de Software (SR) e Roteadores de Software Virtualizados (VSR).

Mais especificamente é comparado o desempenho de funções de rede executada sobre uma plataforma COTS, executada sobre um sistema operacional e sobre um ambiente virtualizados. Como estudado na sessão 1, há diversas opções de graus de virtualização a serem utilizados em NFVs. Há tando a opção *bare-metal*, na qual o hipervisor executa diretamente sobre o hardware, quando opções nas quais ele é executado sobre o sistema operacional, ou mesmo implementações em que funções de rede são executadas sobre uma máquina virtual. É claro que quanto mais virtualizações em pilha forem utilizadas mais *overheads* serão acrescentados. Mas é necessário se ter uma idéia sobre até que ponto virtualizações podem prejudicar o desempenho de funções de rede. Está é a importância desse trabalho para essa pesquisa.

Para a realização do experimento, foi utilizada uma topologia como apresentada na figura 13: um gerador de trafego, que poderia produzir até 4 fluxos distintos, conectado a uma *workstation* que executa a função de roteamento.

Foram utilizadas duas máquinas diferentes para efetuação da função de rede: um Gateway 9510, com processador Intel Xeon, com CPU de 3 GHz, e um Dell Optilex 780 com processador Intel Core 2, com CPU de 3.17 GHz. Os dados estão descritos na figura 12.

WORKSTATION SPECIFICATIONS

	Gateway 9510	Dell Optiplex 780
Name	Gateway	Deall
Processor	Intel(R) Xeon(TM)	Intel(R) Core 2(TM)
No. of processor	4	2
CPU speed	3 GHz	3.17 GHz
RAM	2 GB	2 GB
Interface speed	10/100 Mb/s	
Operating systems	Ubuntu 10.10 (kernel 2.6.22-35-generic) and Windows 7	
Virtualization software	VMware Workstation 7.1	

Figura 12: Especificações das workstations utilizadas no experimento

Ambos SR e VSR executaram sobre o sistema operacional Linux Ubuntu 10.10, com kernel 2.6.22-35-generic. Já no caso do VSR, o Linux era executado sobre o sistema operacional Windows 7.

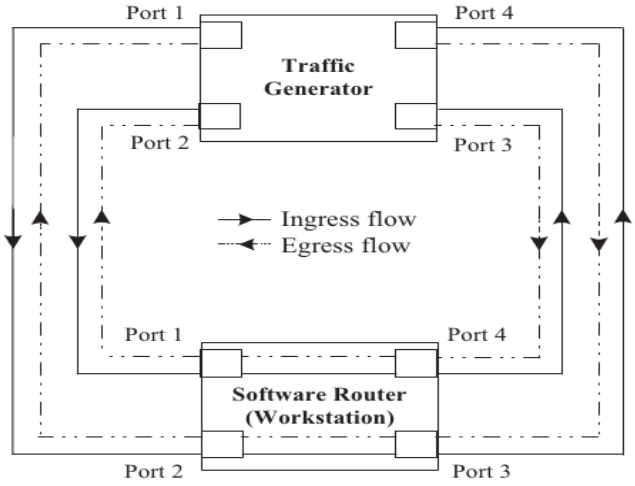

Figura 13: Topologia utilizada na realização do experimento, com 4 repasses.

Na figura 14 estão apresentados os resultados obtidos da taxa de transferência para a operação de repasse com 2 fluxos ativos.

No caso da *workstation* Gateway, os resultados para o SR foram satisfatórios, mas o roteador virtualizados apresentou uma enorme degradação dos resultados.

Para a *workstation* Dell, os resultados da taxa de transferência para o SR foram bastante similares aos obtidos anteriormente, porém ouve um grande aumento da performance do para o VSR. Conclui-se portando que o *overhead* causado pela virtualização para operações de rede, nesse caso, foi consideravelmente inferior. Inclusive para pacotes maiores que 837 bytes, o desempenho do VSR alcançou 90% da capacidade do link.

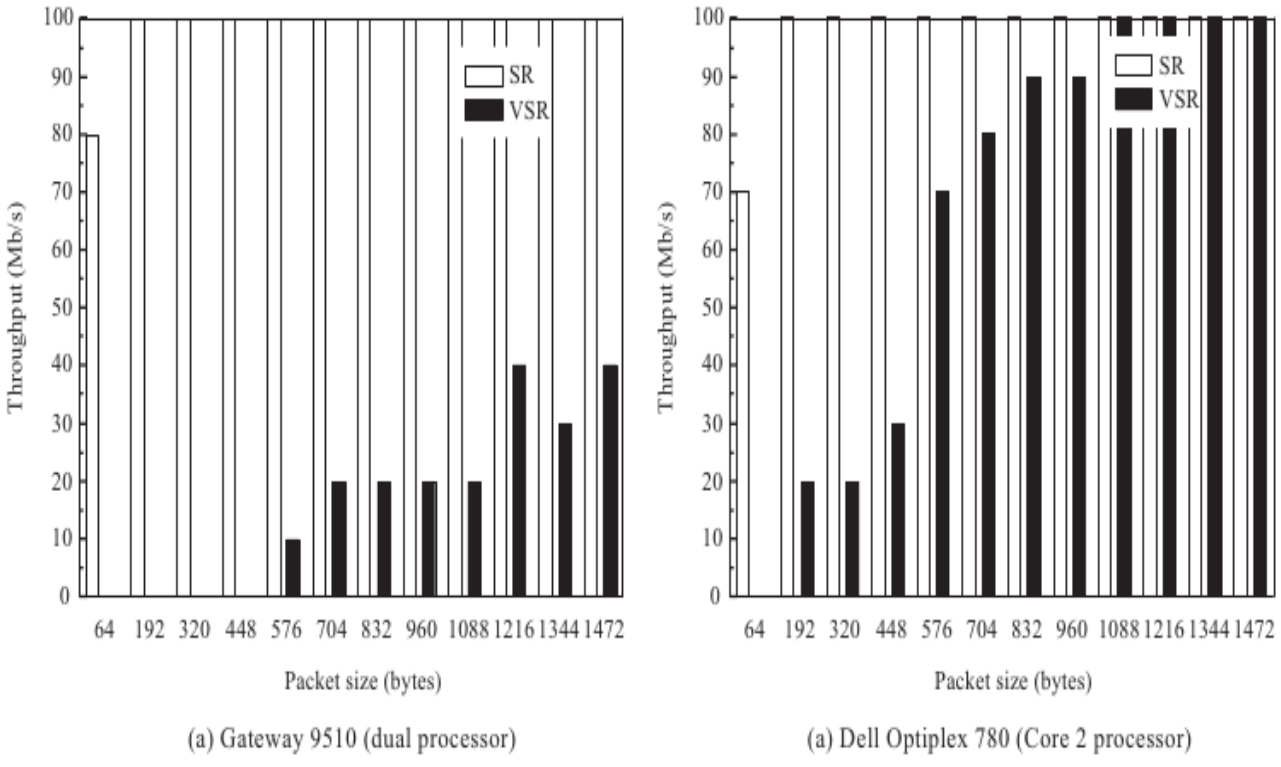

Figura 14: Taxas de transferência para a workstation Gateway e Dell, na operação de repasse com 2 fluxos ativos (topologia da figura 13)

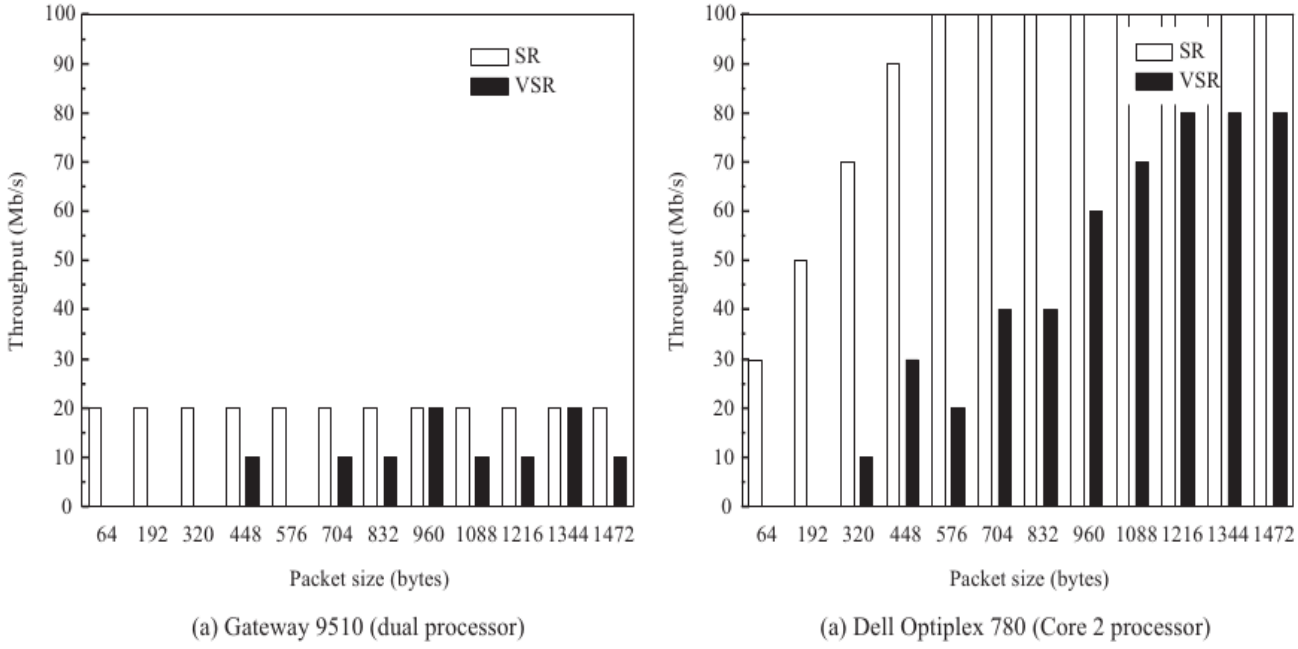

Figura 15: Taxa de transferência a workstation Gateway e Dell, na operação de repasse com 4 fluxos ativos (topologia da figura 13).

Na figura 15 estão apresentados os resultados de desempenho das *workstation* com 4 fluxos ativos.

Vemos que tanto no SR quando no VSR a taxa de transferência alcançado pela *workstation* Gateway foi extremamente penalizado.

Na plataforma Dell, para o SR houve uma degradação considerável para pacotes de comprimento bastante pequenos (até 192 bytes), alcançando bons desempenhos para tamanhos superiores as este. Já o desempenho do SR foi muito penalizado, se comparado ao desempenho anterior. No caso de 2 fluxos de redes simultâneos, o VSR anteriormente alcançava toda a capacidade da rede para grandes pacotes. Agora, mesmo nesses casos o desempenho se limita a 80% da capacidade da rede (80 Mb/s). Para pacotes de tamanho inferior as taxas de transferência alcançadas são bastante baixas. É difícil dizer se esse mesmo tipo de comportamento (saturação do saturação da taxa de transferência) ocorreria caso a capacidade da rede e da interface de rede fossem aumentadas, no caso de 2 fluxos. É algo que precisaria ser avaliado futuramente, e pode inclusive oferecer pistas de como se minimizar os efeitos de *overhead* de virtualização no caso de execução de funções de rede.

Por fim, o experimento é novamente repetido na plataforma Dell, mas desta vez, adota-se um tamanho de pacote fixo (1472 bytes), e varia-se a carga do fluxo oferecida pelo gerador de tráfego, variada de 10 Mb/s até 100 Mb/s. Chega-se a conclusão que tanto o SR quanto o VSR podem suportar 100 Mb/s para 2 fluxos sendo roteados, e para 4 fluxos simultâneos, o VSR pode suportar mais de 70 Mb/s.

Este estudo traz, portanto alguns importantes resultados para execução de funções de redes sob ambientes virtualizados:

- Os *overheads* causados pela virtualização são mais punitivos pelo numero de fluxos por processador do que pela carga suportada em Mb/s
- O processamento de pacotes maiores sofre menores *overheads* decorrentes da virtualização. Para o caso de poucos fluxos simultâneos sendo tratados por processador, podem inclusive se tornarem imperceptíveis, dentro dos limites testados nesse experimento.

C. Análise da performance de Redes Definidas por Software (SDN)

Por fim, nesta sessão serão novamente analisados resultados de desempenho obtidos por redes definidas por software (SDN), retirados de Gelbenger et al [13]. Porém neste estudo há algumas diferenças fundamentais, que podem mostrar resultados importantes:

Foram realizados testes de desempenho com outro protocolo SDN mais complexo, e que inclui as funcionalidades do OpenFlow, chamado ProGFE;

- O desempenho do protocolo SDN foi comparado com o de uma aplicação padrão Linux para repasse, sendo executada tanto em espaço de usuário quanto em nível de kernel, a fim de se explorar melhor os recursos disponíveis no PC. Daí pode-se concluir quanto de *overhead* é adicionado pela execução em espaço de usuário, e se é vantajoso para alguma implementação futura tirar vantagem de uma execução em mais baixo nível.

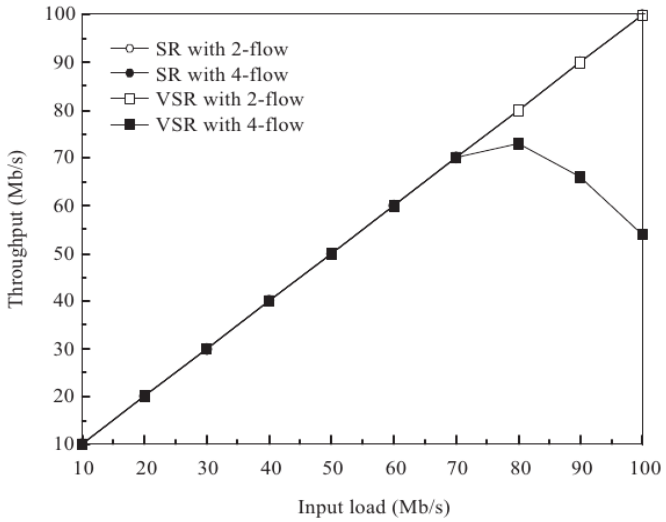

Figura 14: Taxa de transferência medida para o caso de melhor desempenho (trafego de pacotes de 1472 bytes de tamanho), em função da carga oferecida pelo gerador de trafego. Ambas as curvas de desempenho do SR estão juntas com a curva de desempenho do VSR com 2 fluxos.

O protocolo ProGFE (*Programable Generic Fowarding-Element*) é de maneira genérica um *superset* do OpenFlow, já que essencialmente possui todas as funcionalidades deste, e por fornecer também outras funcionalidades mais diversas. Mas o ProGFE e o Openflow, em termos de implementação são tecnologias completamente diferentes.

Neste experimento foi comparada a performance de processamento de diversas *workloads*, tanto das aplicações SDN OpenFLow e ProGFE quanto de não SDN disponíveis no sistema operacional executadas em espaço de usuário e do kernel. Foi analisado somente a performance "crua" das aplicações, sendo ignorados quaisquer benefícios indiretos que o protocolo SDN pudesse oferecer, como por exemplo a redução do número de sistemas de rede utilizados. A seguir foram relacionados os resultados obtidos de operações de roteamento (no artigo [13] também são rescritos resultados de operações de *bridging* e VLAN *tagging*).

Para se avaliar a taxa de transferência (*throughput*) foi utilizada a ferramenta iperf, a fim de se estabelecer um fluxo TCP/IP, com janelas TCP de diferentes tamanhos, variando de 5kbytes até 10Mbytes.

A fim de se avaliar a latência, frames sintéticos foram gerados, variando-se o tamanho dos frames, com um *timestemp* usado para se calcular a latência. O PDV (*packet delay variation*), também chamado de *jitter*, também foi avaliado.

A unidade sob teste, na qual o sistema SDN foi implementado era baseada em um Intel Core2Duo e6600

CPU, operando a 2.4 GHz, com 2 GB de memória DDR2, e três Intel 82572EI gigabit NICs.

Para o teste de taxa de transferência, foram utilizados dois PCs comuns conectados por meio da unidade sob teste. Os dois PCs estabeleceriam um fluxo TCP/IP, usando o iperf, enquanto a unidade sob testes iria realizar as operações de *switch* e roteamento entre eles.

Um embarcado dedicado MPC8360 microcontrolado foi utilizado para traçar frames com a unidade sob testes, para medir a latência e o *jitter*, em microssegundos.

Por fim, para propósitos de comparação, os mesmos valores de taxa de transferência, latência, *jitter* também foram medidos em uma conexão direta entre os dois PCs comuns, sem a unidade sob testes de intermediaria.

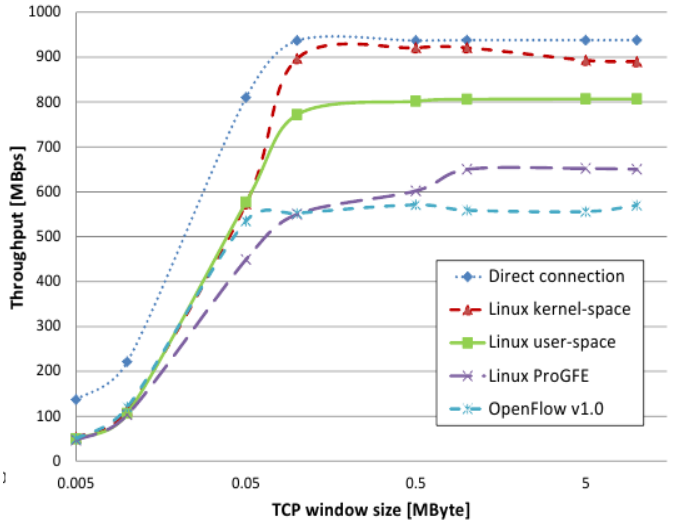

Figura 15: Taxa de transferência (throughput) da conexão TCP/IP medida no setup experimental e ao se variar a janela TCP. Nota-se o desempenho da aplicação de roteamento Linux rodando em espaço de kernel se aproxima bastante do desempenho da conexão direta, e que a performance do ProGFE para janelas TCPs da ordem de 100 kbytes ou maiores é superior ao do OpenFlow.

Por fim nas figuras 17, 18 e 19 são apresentados os valores de taxa de transferência, latência e *jitter* respectivamente para o OpenFlow, ProGFE, e aplicação de roteamento Linux sendo executada em espaço de usuário e kenel, bem como os valores obtidos pela conexão direta.

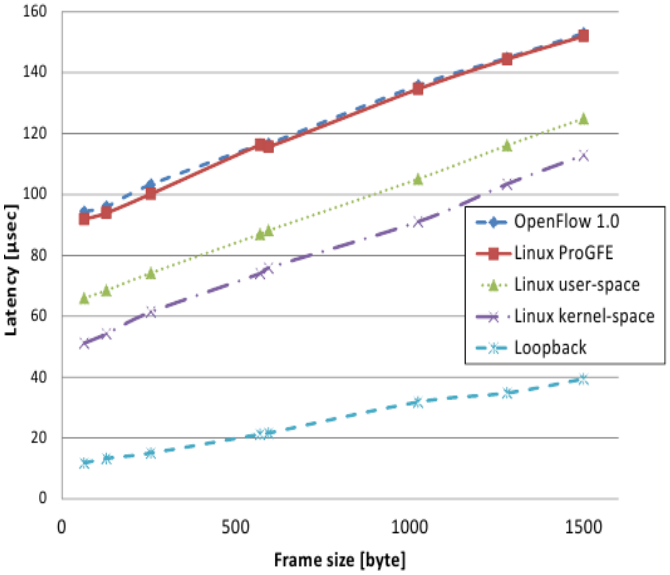

Figura 16: latência da conexão TCP/IP medida no setup experimental e ao ser variado o tamanho dos frames transferidos (em bytes), medida em microssegundos. Nota-se o melhor desempenho da aplicação Linux rodando em espaço de kernel, e a diferença entre os valores de latência do OpenFlow e ProGFE sendo irrelevante.

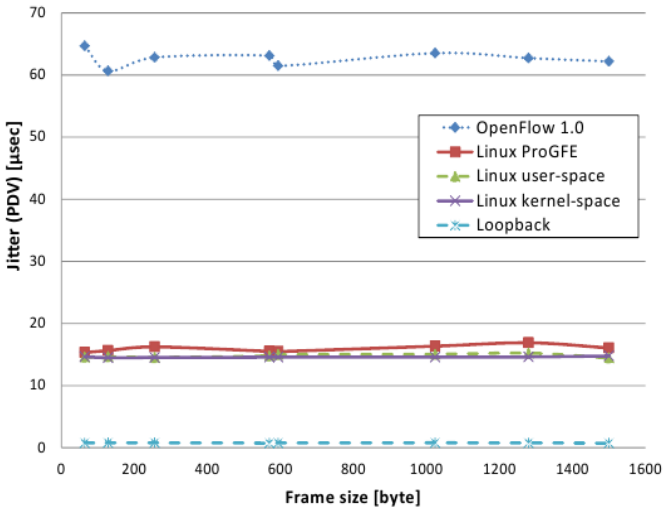

Figura 17: Valores de jitter (em microssegundos) para o experimento. Nota-se valores muito semelhantes entre as aplicações Linux, e o ProGFE, e valores consideravelmente superiores para o OpenFlow.

Um ponto a ser observado, é que na maior parte dos casos o protocolo ProGFE obteve um desempenho superior ao OpenFlow, o que mostra que a complexidade da aplicação não implica em degradação de desempenho no caso de SDN. Uma das possíveis razões para a obtenção de desempenho inferior do OpenFlow pode vir dos altos valores de *jitter*, o que ocasionou uma degradação da taxa de transferência, mesmo a latência de ambos os protocolos sendo semelhantes.

Algo também importante ser observado, que fugia do escopo do trabalho original é o fato de que as implementações das aplicações em espaço de kernel forneceram uma melhora de desempenho em todos os casos, em especial em latência e

taxa de transferência. Esse portanto pode ser uma forma ótima de se obter melhorias e otimizações de desempenho em aplicações futuras.

IV. Estudo da virtualização em processadores ARM

Como descrito anteriormente, o desenvolvimento de funções de rede virtuais NFV, como o nome sugere, está ligada a capacidade de virtualização. Para ser viável o desenvolvimento e implantação de uma NFV, é necessário primeiramente que o hardware e suas instruções tenham certos pré-requisitos, descritos por Popek e Goldberg [4]. Além disso é necessário que ela seja eficiente, já que qualquer *overhead* da virtualização também será sentido na aplicação.

Já a certo tempo os novos modelos da arquitetura x86 possuem suporte de hardware para virtualização, já estando essa tecnologia relativamente madura para essa arquitetura. Porém somente recentemente o suporte a virtualização passou a ser fornecido por processadores ARM, sendo o primeiro o processador com essa característica o Cortex-A7 introduzido em Outubro de 2011, pertencente a família de processadores ARMv7. [15]

Também recentemente foi desenvolvido o primeiro hipervisor Linux para processadores ARM multicores, capaz de executar um sistema operacional sem necessidade de haver modificações em seu código. Ele é chamado KVM/ARM, e foi descrito em por Dall e Nieh em [8]. Ele já está integrado no kernel do Linux, sendo a opção para virtualização padrão do ARM a partir do kernel 3.9. Esta sessão será dedicada a fazer uma análise breve deste trabalho, comentando o hipervisor, e fazendo uma análise dos principais resultados experimentais obtidos no artigo.

Há uma demanda comercial crescente para suporta a virtualização no processador ARM. O processador ARM se destaca tradicionalmente por bastante econômico energeticamente. E melhorias em sua arquitetura o estão o colocando seu desempenho próximo de processadores x86 para muitas aplicações.

Como descrito em [15], o processador ARM para suportar virtualização introduziu um novo modo de operação do processador chamado modo Hyp. Esse modo difere consideravelmente do kernel-mode nível no qual a virtualização é executada no x86. Isso faz com que a implementação do hipervisor para o ARM seja muito diferente da implementada em x86. Dall e Nieh introduzem o chamado *split-mode*, que permite a execução do hipervisor em ambos os níveis. Portanto ele poderá usufruir de recursos já implementados para o kernel, reduzindo a complexidade do desenvolvimento, ao mesmo tempo em que pode usar as *features* de virtualização disponíveis no Hyp-mode.

Os resultados obtidos desta primeira versão de hipervisor são bastante promissores: os *overheads* acrescentados pela virtualização são bastante semelhantes aos obtidos pelo x86 na maioria dos casos, sendo inclusive melhores em testes realizados em duas importantes aplicações: Apache e MySQL. Eles são apresentado para o ARM com e sem dos recursos VGIC (*virtual generic interrupt controller*) e *virtual timers support*.

Os resultados do ARM foram obtidos usando um processador Insignal Arndale Cortex-A15, dual core operando a 1.7 GHz, em uma Sansung Exynos 5250 SoC. Já como plataformas x86 foram utilizados um MacBook Air 2011, dual core, 1.8 GHz, CPU core i7-2677M, e no servidor Intel Xeon, dual core, operando a 3.4 GHz. O sistema operacional utilizado foi um Ubuntu 12.10, com Kernel Linux 3.10. Como hipervisores foram utilizados o KVM/ARM e o KVM x86.

Das figuras 20 a 23 os resultados foram medidos e os desempenhos normalizados de acordo com o desempenho obtido na plataforma sem a utilização de virtualização (ou seja, menor o tamanho da barra, menor o *overhead* acrescentado – ou seja, no limite, tamanho 1 significa mesmo desempenho).

Na figura 20 e 21 foram testados algumas aplicações e eventos (como um *fork* e *pipe*), com multiprocessamento simétrico desabilitado (figura 20) e habilitado (figura 21).

Já as figuras 22 e 23 mostram os resultados obtidos com alguns benchmarks envolvendo aplicações como Apace e MySQL, e compilação de kenel por exemplo. Da mesma forma que anteriormente, os testes foram feitos com o multiprocessamento simétrico desabilitado (figura 22) e habilitado (figura 23).

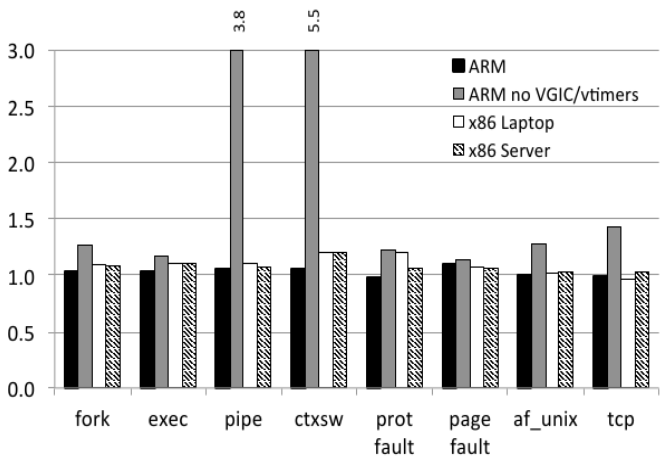

Figura 18: Desempenho de alguns eventos sobre uma máquinas virtuais (usando-se o hipervisor KVM/ARM para o processador ARM e KVM x86 para ambos processadores x86), normalizados sobre o mesmo desempenho obtido executado diretamente no host nativo, com SMP (multiprocessamento simétrico) desabilitado.

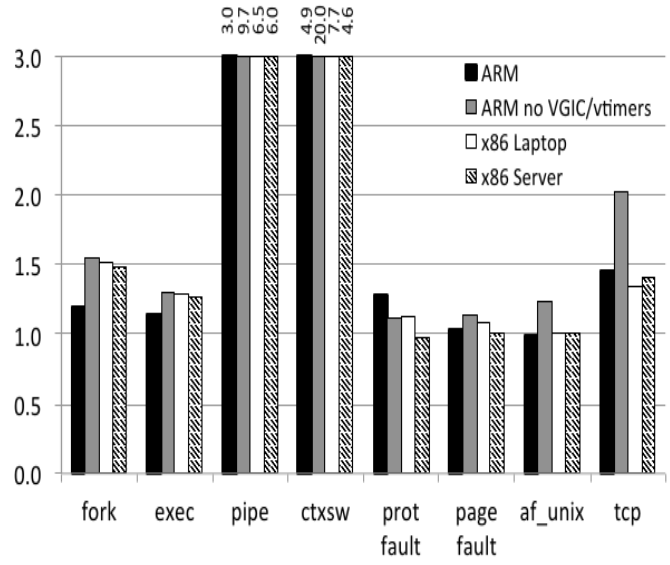

Figura 19: Desempenho de alguns eventos sobre máquinas virtuais (usando-se o hipervisor KVM/ARM para o processador ARM e KVM x86 para ambos processadores x86), normalizados sobre o mesmo desempenho obtido executado diretamente no host nativo, com SMP (multiprocessamento simétrico) habilitado.

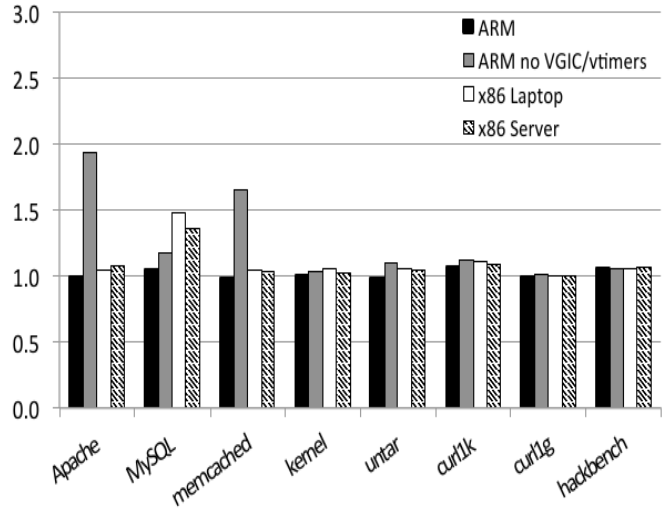

Figura 20: Desempenho de alguns benchmarks sobre um máquinas virtuais (usando-se o hipervisor KVM/ARM para o processador ARM e KVM x86 para ambos processadores x86), normalizados sobre o mesmo desempenho obtido executado diretamente no host nativo, com SMP (multiprocessamento simétrico) desabilitado.

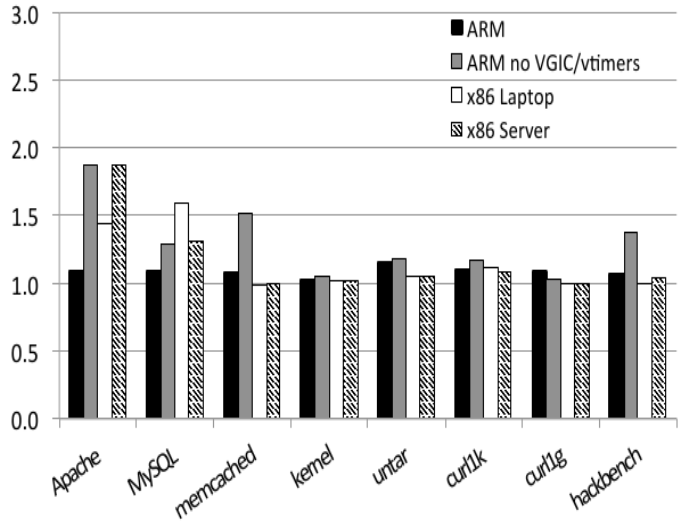

Figura 21: Desempenho de alguns benchmarks sobre um máquinas virtuais (usando-se o hipervisor KVM/ARM para o processador ARM e KVM x86 para ambos processadores x86), normalizados sobre o mesmo desempenho obtido executado diretamente no host nativo, com SMP (multiprocessamento simétrico) habilitado.

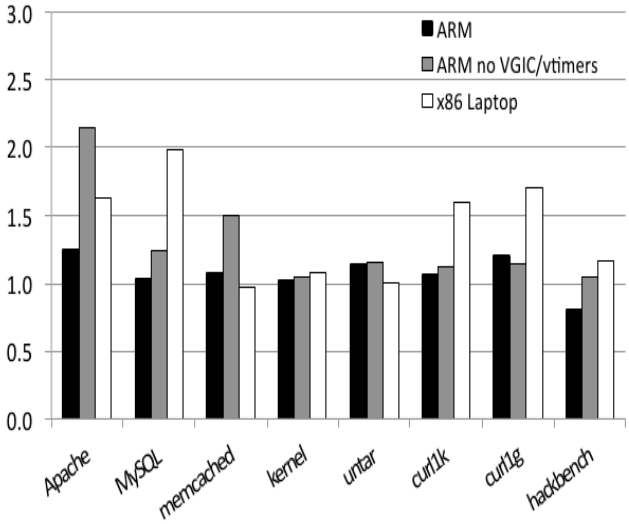

Figura 22: Desempenho obtido de gasto de energia dos benchmaks sobre máquinas virtuais, normalizados pelos gastos obtidos com as mesmas aplicações sendo executadas diretamente no host nativo.

Primeiramente nota-se que na maioria dos casos o uso de VGIC e *vtimers* melhorou o desempenho. Além disso, um resultado bastante promissor foi que na maioria em grande parte dos casos o overhead causado pelo KVM/ARM foi bastante pequeno, ou mesmo menor que o gerado pelas plataformas x86 (mesmo sendo hipervisores mais maduros).

Por fim, na figura 24 está apresentado o desempenho de consumo energético normalizado da execução das aplicações virtualizadas versus as execuções sem uso de virtualização. Nesse caso, não foram medidos os resultados do servidor. Nota-se que, novamente, na maioria dos casos, o desempenho do KVM/ARM enquanto hipervisor foi superior ao obtido pelo já maduro KVM x86.

Este é um estudo importante com relação a viabilidade da implementação de NFVs em ARM, já que mostra o quão eficiente a execução de aplicações em ambiente virtualizados em ARM. Tais resultados não mostram o desempenho absoluto das mesmas, mas mostram claramente os *overheads* acrescentados pela virtualização. Os resultados são muito animadores, já que o primeiro hipervisor Linux para ARM implementado apresenta excelente resultados, acrescentando poucos *overheads*, e em grande parte dos casos, menores que o gerado por hipervisores maduros x86. É uma prova clara do grande potencial que a processadores ARM tem para virtualização. Como descrito no trabalho, há ainda diversos pontos onde a aplicação poderia ser melhorada e otimizada. Por exemplo, muitos dos recursos utilizados foram os já fornecidos pelo kernel, não foram reimplementados e otimizados para a operação do hipervisor.

V. Revisando o debate CISC vs. RISC em arquiteturas atuais

Como descrito anteriormente na introdução deste trabalho, já existe um debate histórico do uso das arquiteturas RISC e CISC. Como conseqüência dos resultados clássicos desse debate, Blem et al [19] cita na introdução de seu estudo que houve um consenso que as diferenças fundamentais entre as ISAs RISC e CISC que levavam aos gaps de performance, fez com que houvesse uma profunda otimização da microarquitetura CISC, o que levou a tais diferenças fossem em sua maioria corrigidas.

Como já dito, atualmente a arquitetura CISC x86 domina o mercado de servidores. O fato de a arquitetura RISC ARM ser conhecida por sua eficiência energética, colocou em pauta novamente essa discussão. E assim como o paradigma NFV se propõe a reconstruir funções de redes sobre plataformas de servidores padrões, a viabilidade da utilização de servidores ARM, com as suas possíveis vantagens em termos energéticos (que é uma meta da NFV) e possivelmente ate mesmo vantagens computacionais (em termos de desempenho RISC, ou na virtualização como descrito na sessão anterior).

Nas sessões seguintes serão feitas de algumas análises de resultados atuais obtidos a respeito desse debate, e uma rápida apresentação do processador ARMv8.

A. Estudo do desempenho de processadores Cortex-A8 e A9 comparados processadores x86

Será feita uma breve análise de alguns resultados obtidos obtidos por Blem et al [19]. Neste trabalho são analizados e comparados resultados entre o processador ARM Cortex-A8, Cortex-A9 (ambos pertencente a família de processadores ARMv7), e os processadores x86 Intel Atom e o Sandybridge i7. Segundo os autores, a partir dos resultados encontrados se conclui que:

- Grandes *gaps* de performance existem entre as implementações, porém o *gap* de contagem de ciclos para cada aplicação é sempre menor do que 2.5 vezes;
- As diferenças de desempenho são geradas por diferenças de microarquiteturas independentes da ISA adotada;
- O consumo de energia é independente da ISA;
- As utilizações de ISAs diferentes causam certas implicações, mas em arquiteturas modernas uma ISA não é fundamentalmente mais eficiente que a outra;
- ARM e x86 são simplesmente desenhados para serem otimizados em diferentes níveis de desempenho.

Essencialmente o estudo sustenta a tese que hoje, com as técnicas maduras de desenho de processadores, a utilização de uma ISA ou outra tem na realidade peso irrelevante para o quão eficiente será o processador.

Um sumário dos *benchmarks* utilizados está apresentado na figura 25, com testes de desempenhos para Mobile Client, Desktop (SPEC INT e EPEC FP), e para servidores.

Domain	Benchmarks	Notes
Mobile client	CoreMark	Set to 4000 iterations
	WebKit	Similar to BBench
Desktop	SPECCPU2006	10 INT, 10 FP, test inputs
Server	lighttpd	Represents web-serving
	CLucene	Represents web-indexing
	Database kernels	Represents data-streaming and data-analytics

Figura 23: sumário dos benchmarks utilizados nesse estudo.

Para se eliminar influência do compilador utilizado, para todos os testes foi utilizado o gcc 4.4. Mas compiladores fornecidos pelos vendedores são capazes de fornecer códigos melhores para cada uma das plataformas.

Na figura 26 está apresentado um sumário de cada plataforma de hardware utilizada nos testes.

	32/64b x86 ISA		ARMv7 ISA	
Architecture	Sandybridge	Atom	Cortex-A9	Cortex-A8
Processor	Core 2700	N450	OMAP4430	OMAP3530
Cores	4	1	2	1
Frequency	3.4 GHz	1.66 GHz	1 GHz	0.6 GHz
Width	4-way	2-way	2-way	2-way
Issue	OoO	In Order	OoO	In Order
L1 Data	32 KB	24 KB	32 KB	16 KB
L1 Inst	32 KB	32 KB	32 KB	16 KB
L2	256 KB/core	512 KB	1 MB/chip	256 KB
L3	8 MB/chip	—	—	—
Memory	16 GB	1 GB	1 GB	256 MB
SIMD	AVX	SSE	NEON	NEON
Area	216 mm^2	66 mm^2	70 mm^2	60 mm^2
Tech Node	32 nm	45 nm	45 nm	65 nm
Platform	Desktop	Dev Board	Pandaboard	Beagleboard
Products	Desktop	Netbook	Galaxy S-III	iPhone 4, 3GS
		Lava Xolo	Galaxy S-II	Motorola Droid

Data from TI OMAP3530, TI OMAP4430, Intel Atom N450, and Intel i7-2700 datasheets, `www.beagleboard.org` & `www.pandaboard.org`

Figura 24: Sumario das plataformas de hardware utilizadas nos testes.

Nas figuras 27 e 28 estão apresentados os resultados de tempo de execução para cada uma das plataformas e de contagem de ciclos, normalizados com o desempenho do processador i7.

Como era de se esperar no teste de tempo, houveram grandes *gaps* de performance, no qual o i7 acaba levando uma enorme vantagem. Mas é claro, isso se deve principalmente a freqüência de operação mais alta, além da quantidade maior de *cores*.

Ao serem comparados o número de ciclos de maquina necessários para a execução de cada um dos benchmarks, nota-se uma diferença muito menos significativa entre os resultados. Comparados os processadores Cortex-A9 com o Sandybridge i7 e o Atom com o Cortex-A8, por terem arquiteturas proporcionalmente mais semelhantes, nota-se que a razão entre o número de instruções executadas é de no máximo 2,5 (as razões estão apresentadas na figura 29).

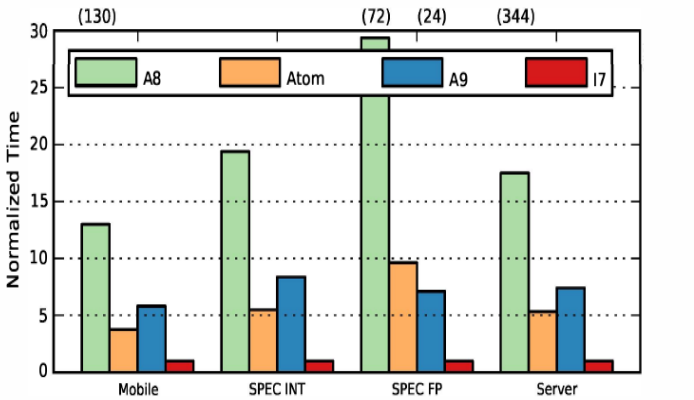

Figura 25: Tempos de execução dos benchmaks, normalizadas com os valores obtidos no processador i7.

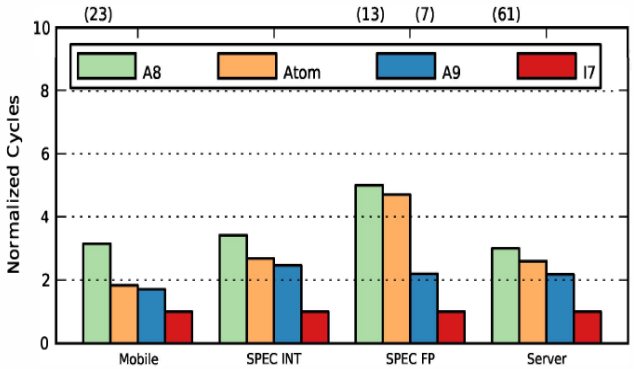

Figura 26: Contagem de ciclos de instruções executadas nos benchmaks, normalizadas com os valores obtidos no processador i7.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8 to Atom	1.2 (12)	1.2	1.5 (2.7)	1.3 (23)
A9 to i7	1.7	2.5	2.1 (7.0)	2.2

Figura 27: Razões encontradas entre o número de instruções executadas entre os processadores nos benchmarks.

Em seguida foram feitos testes com sobre o desempenho de energia. Aqui estão apresentados dois dos resultados colhidos no estudo: a potência dissipada normalizada com a área do chip, e o total de energia gasto para cada teste.

No teste de potência dissipada(figura 30), o processador i7, por uma grande margem teve os piores resultados. Porém, ao se checar o total de energia gasto no processo (figura 31), verifica-se que ele em geral não apresentou os piores resultados. Conclui-se portanto que os altos gastos com potência se devem a freqüência de operação mais alta. Nos testes relacionados aos gastos de energia, o processador Cortex-A9 mostrou os resultados mais satisfatórios, e o processador A8 os mais pobres.

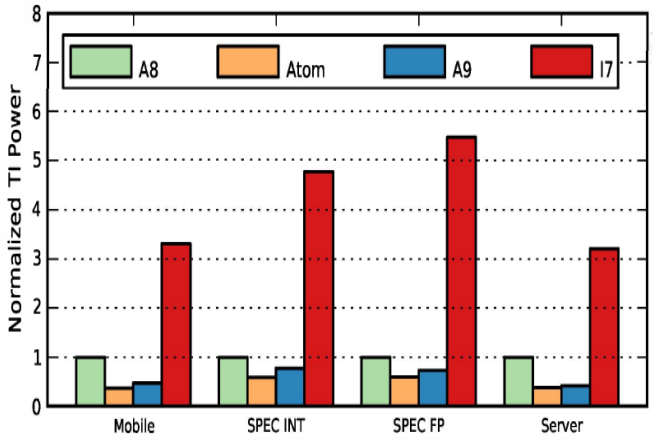

Figura 28: Potência dissipada pelos processadores, normalizada de acordo com os valores obtidos pelo processador A8.

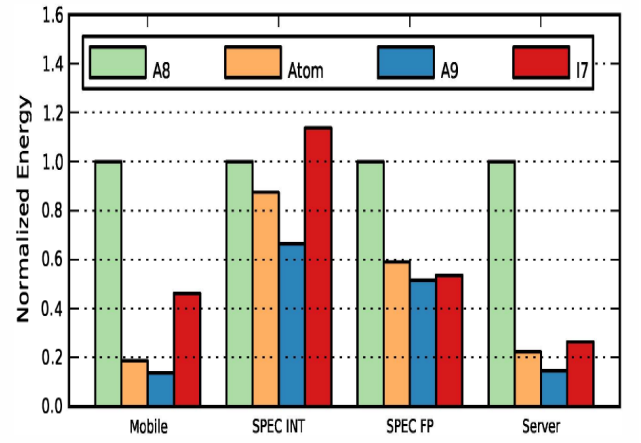

Figura 29: Energia dissipada pelos processadores, normalizada de acordo com os valores obtidos pelo processador A8.

Por fim, os autores chegam a conclusão que a curva de *trade-off* potência por performance obedece uma cúbica.

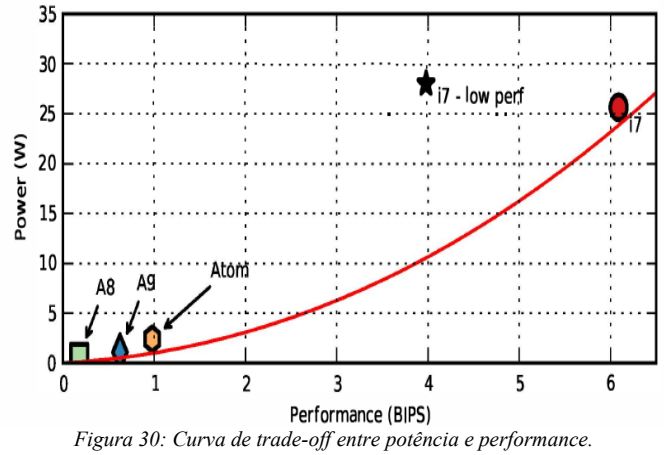

Figura 30: Curva de trade-off entre potência e performance.

B. Estudos envolvendo o desempenho do Cortex-A15 contra o Intel i3

Alguns estudos informais também foram feitos nesse sentido, tentando se comparar o desempenho de processadores atuais ARM e x86. Aqui também serão descritos alguns dos resultados obtidos por Michael Larabel [22,23], em duas baterias de benchmarks comparando-se arquiteturas ARM com x86.

Os testes a seguir foram realizados no sistema operacional Ubuntu 12.04. As especificações das montagens estão apresentadas na figura 33 e 34.

Processor	Cortex-A9 ARMv7 (2 cores)
Frequency	1.20GHz
Motherboard	OMAP4 Panda board
Memory	912MB
Disk	16GB SD16G
Kernel	Linux 3.2.0-1405-omap4
Compiler	gcc 4.6
File System	ext4
Processor	Intel Pentium M (1 core)
Frequency	1.86GHz
Motherboard	IBM 18494WU
Memory	2048MB
Disk	80GB Hitachi HTS541080G9AT00
Kernel	Linux 3.2.0-11-generic-pae
Compiler	gcc 4.6
File System	ext4
Processor	Intel T2400 (1 cores)
Frequency	
Motherboard	LENOVO 2613EJU
Memory	2048MB
Disk	80GB Hitachi HTS541080G9SA00
Kernel	Linux 3.2.0-11-generic-pae
Compiler	gcc 4.6
File System	ext4
Processor	Intel Atom N270 (2 cores)
Frequency	1.60GHz
Motherboard	SANSUNG NC10
Memory	2048MB
Disk	32GB OCZ CORE_SSD
Kernel	Linux 3.2.0-11-generic-pae
Compiler	gcc 4.6
File System	ext4

Figura 31: Dados dos benchmarks realizados na referência [22].

Processor	Cortex-A15 Exynos ARMv7 (2 cores)
Frequency	1.70GHz
Motherboard	SAMSUNG EXYNOS5
Memory	2048MB
Disk	16GB SEM16G
Kernel	Linux 3.4.0 (armv7l)
Compiler	GCC 4.6
File System	ext4
Processor	Intel Atom D525 (4 Cores)
Frequency	1.80GHz
Motherboard	FOXCONN NETBOX nT-435/535
Memory	2048MB
Disk	500GB Western Digital WD5000BEVT-2
Kernel	Linux 3.2.0-29-generic (x86_64)
Compiler	GCC 4.6
File System	ext4
Processor	Intel Core i3 330M (4 Cores)
Frequency	2.13GHz
Motherboard	ASRock HM55-HT
Memory	4096MB
Disk	500GB Seagate ST9500325AS
Kernel	Linux 3.2.0-29-generic (x86_64)
Compiler	GCC 4.6
File System	ext4

Figura 32: Dados dos benchmarks realizados na referência [23]. Nesta tabela foram somente expressos os dados dos processadores Intel de melhor desempenho e do Cortex-A15 Exynos, o de maior interesse.

Nas figuras 35 e 36 estão apresentados dois dos resultados obtidos nos estudos. Nos outros testes realizados, as razões entre os valores encontrados não foram muito diferentes dos apresentados. Em ambos os benchmarks apresentados aqui o C-Ray v1.1 foi o utilizado.

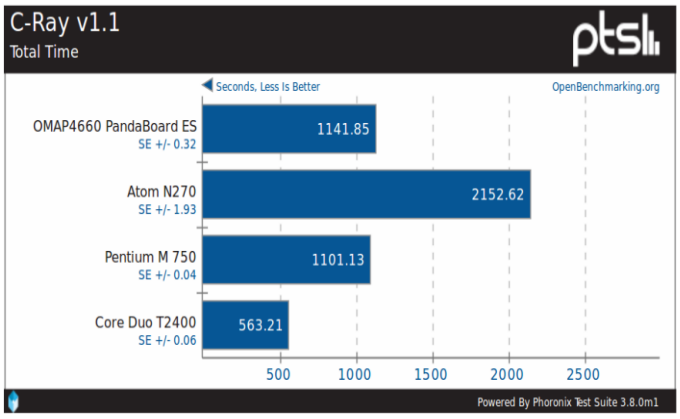

Figura 33: Resultados do benchmark C-Ray envolvendo o processador ARMv7 Cortex-A9 sobre a placa mãe OMAP4660 PandaBoard ES.

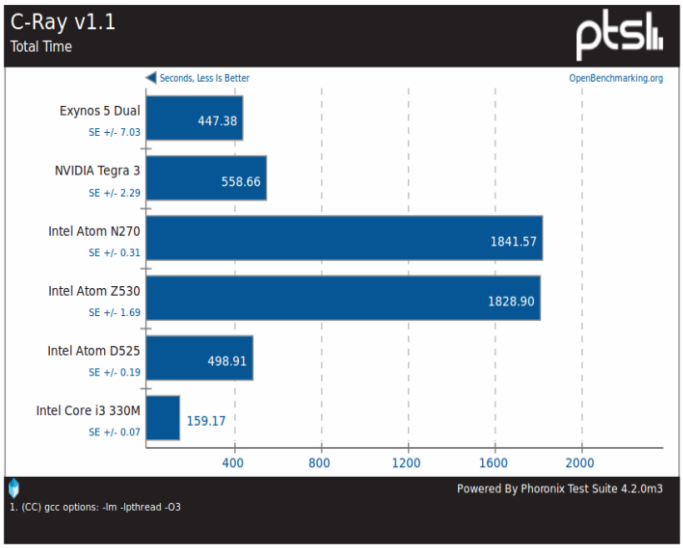

Figura 34: Resultados do benchmark C-Ray envolvendo o processador ARMv7 Cortex-A15 Exynos 5 Dual.

Nota-se claramente uma agressiva evolução de desempenho do processador Cortex-A9 para o Cortex-A15. Em desempenho absoluto, em quase todos os resultados obtidos em [23] o processador Cortex-A15 fica na frente do Intel Atom D525, mesmo este possuindo freqüência de operação maior e o dobro de *cores*. Esses resultados, ao menos em parte parecem dizer o contrário dos resultados obtidos por Blem et al [19], ao menos a partir do processador Cortex-A15, já que o estudo foi dedicado ao Cortex-A8 e A9. Mais estudos podem se mostrar necessários para reavaliar se com as novas otimizações lançadas nos processadores ARM mais recentes (em especial com o lançamento da nova arquitetura ARMv8), os resultados obtidos para o Cortex-A8 e A9 continuam valendo. Inclusive é importante ressaltar que as mudanças mais recentes, como descritas em [15] visavam justamente realizar uma grande revisão na microarquitetura, buscando ao mesmo tempo realizar uma grande otimização, mantendo a eficiência energética.

C. Processador ARMv8

Nesta subseção será analisando brevemente um pouco do que há sobre a nova arquitetura ARMv8.

Dentre os processadores da arquitetura ARMv8, temos os processadores Cortex-A53 e Cortex-A57. Os dados a segui foram retirados diretamente do site da ARM (www.arm.com) [24,25]. Nele estão disponíveis dados de *benchmarks* comparando esses processadores com desenhos anteriores, dentre eles o Cortex-A9, analisado por Blem et al [19] anteriormente. Nota-se uma grande melhora de desempenho, apesar de que a freqüência de operação do processador também aumentou (2GHz).

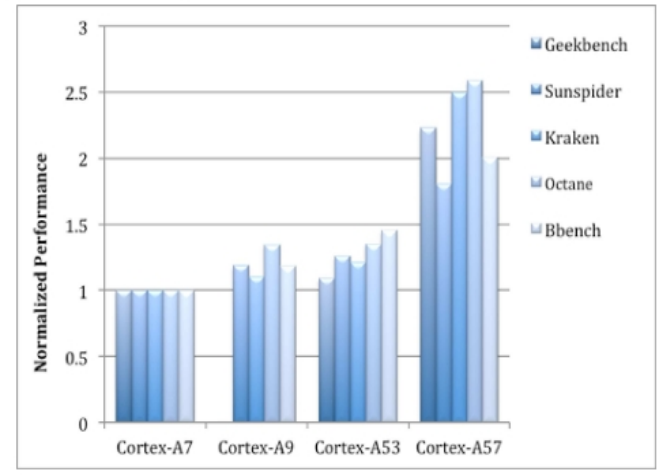

Figura 35: Benchmarks normalizados, comparando-se o desempenho de processadores ARMv7 e ARMv8.

Por fim, toda esse evolução recente da arquitetura ARM descrita nesse trabalho, chega em um ponto crucial, que pode potencializar muito a utilização de plataformas baseadas na arquitetura ARM como servidores de propósito geral, e conseqüentemente serem uma solução extremamente interessante para implementação de NFVs: o processador baseado em ARMv8 ThunderX, um processador de 2.5GHz com até 48 cores. Sua apresentação pode ser encontrada em [26].

Dentre variações do processador ThunterX lançado, para a implementação de NFVs o mais interessante é o chamado ThunderX_NT™ , que possui algumas *features* interessantes e desejadas para alta performance de processamento de redes e virtualização. Dentre elas, se encontram:

- Até 100GbE de conectividade;
- Múltiplas portas PCIe e Gen3;
- Alta banda de memória;
- Aceleradores de hardware para alta taxa de transferência de pacotes, virtualização de rede, e monitoramento de dados

D. Convergência entre resultados do ARM e requisitos de NFV

Como fim da discussão a respeito dessa breve revisão do debate ARM vs. x86 atualizado, serão analisados alguns pontos de encontro entre resultados positivos que as arquiteturas ARM vem mostrando e requisitos de NFV.

Em primeiro lugar, e o ponto de encontro mais natural é que processadores ARM via de regra possuem um ótimo desempenho energético, e economia de energia é um dos grandes pontos motivacionais para o desenvolvimento de NFVs. Não será descrito aqui, mas já existem estudos, como o realizado por Saponara et al [6] recentes envolvendo propostas de arquiteturas para servidores envolvendo justamente visando redução dos gastos de energia.

É importante se destacar também os ótimos resultados de virtualização realizados em ARM, mostrando baixíssimos *overheads* na maioria das aplicações testadas. Inclusive, havendo muitas possibilidades de otimização em aberto. É claro que o estudo mostrado aqui se referia a virtualização convencional. O tipo de virtualização requerido para NFVs é diferente, e requer diversos tipos de privilégios a acesso a recursos, além de executar de maneira distribuída. Mas ao que esses resultados indicam há uma grande possibilidade de servidores de propósito geral ARM conseguirem atende os requisitos de virtualização para NFV inclusive de maneira superior aos x86.

Por fim, com o lançamento do processador baseado em ARMv8 ThunderX, em especial o ThunderX_NT, otimizado justamente para a execução de funções de redes virtualizadas, se abre uma janela de estudos promissora.

VI. CONCLUSÃO

Neste trabalho foi feita uma revisão em algumas novas tecnologias de rede, com foco principalmente em NFV. Foi feita uma revisão teórica sobre o assunto, e foi discutindo também porque é interessante o assunto NFV ser trabalhado de maneira colaborativa com SDN. Ambas tecnologias possuem motivações em comum, dentre elas o aumento da complexidade da rede de computadores, necessidade maior eficiência e de mais rápida inovação. E apesar das vantagens que haveriam com a implantação de sistemas NFV, ao menos de início uma perca de desempenho seria inevitável. Este também é um ponto de encontro entre NFV e SDN, pois mesmo que em sistemas individuais protocolos SDN também acrescentam sobrecargas, ao se analisar o sistema com um todo, em geral os resultados são melhores. Foram discutidos alguns experimentos envolvendo conceitos de NFV e SDN, onde foi possível concluir qual é o peso pratico de certos overheads como por exemplo a execução de certas funções de rede em um ambiente virtualizados.

Outro ponto também discutido neste trabalho foi a viabilidade da utilização de servidores baseados na arquitetura ARM como máquinas de propósito geral para a execução de funções de rede. Foi estudado o peso da virtualização de aplicações sobre um processador ARM, e os resultados foram bastante promissores, pois mostram acréscimos de poucas sobrecargas, mesmo se comparado o desempenho com o de hipervisores de máquina virtual mais maduros. Também foi analisadas a viabilidade em termos de desempenho do processador ARM, em comparação com outras arquiteturas x86. Para testes com processadores ARMs lançados à pouco tempo os resultados foram atraentes, já que mostram um trade-off que pode ser interessante, principalmente no que se diz respeito a economia de energia, que é inclusive uma das grandes motivações para implantação de sistemas NFV. Processadores da mais nova família ARM lançada, chamada ARMv8 mostram desempenho consideravelmente melhores. Em especial o processador ThunderX_NT™ especialmente otimizado para processamento em ambientes virtualizados e

de rede se mostra como uma forte opção de hardware para esse propósito.

Por fim, se fazem hoje necessários estudos envolvendo a combinação das tecnologias aqui citadas, e se fazer provas de conceito se esta combinação traria de fato benefícios.

Fontes de Imagens

Figura 1: Rosa et al [5], p. 8.
Figura 2: [2], p. 5
Figura 3: Gelberger, A. [16], p. 390.
Figura 4: Rosa et al [5], p. 10.
Figura 5: _____, p. 16.
Figura 6: Masutani et al [10], p. 260.
Figura 7: Bianco et al [20], p. 2.
Figura 8: _____, p. 2.
Figura 9: _____, p. 3.
Figura 10: _____, p. 3.
Figura 11: _____, p. 4.
Figura 12: Roja-cessa et al [9], p.1.
Figura 13: _____, p. 1.
Figura 14: _____, p. 2.
Figura 15: _____, p. 2.
Figura 16: _____, p. 2.
Figura 17:Gelberger, A. [16], p. 393.
Figura 18: _____, p. 393.
Figura 19: _____, p. 393.
Figura 20: Dall, Christoffer; Nieh, Jason, p. 344.
Figura 21: _____, p. 344.
Figura 22: _____, p. 344.
Figura 23: _____, p. 344.
Figura 24: _____, p. 344.
Figura 25: Blem et al [19], p. 4
Figura 26: _____, p. 4
Figura 27: _____, p. 7
Figura 28: _____, p. 7
Figura 29: _____, p. 10
Figura 30: _____, p. 10
Figura 31: _____, p. 10
Figura 32: _____, p. 10
Figura 35: Larabel, Michael [22]
Figura 36: _____, [23]
Figura 37: [24]

Referencias Bibliográficas

[1] “Network Function Virtualization (NFV); Architectural Framework”. ETSI, 2013.

[2] STALLINGS, WILLIAM. *Computer Organization and Architecture* : *Designing for Performance*. Oitava edição. Prentice Hall, Upper Saddle River, New Jersey, 2006. pp 774.

[3] CARDOSO, Eleri. *Introdução aos Sistemas Operacionais*. 2012. 121p. Apostila do curso EA876 – Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas

[4] POPEK, GERALD J.; GOLBERG, ROBET P.; “Formal Requiriments for Virtualizable Third Generation Architectures ” *Communications of the ACM* vol. 17 numb. 7, pp. 412 – 421 , Julho de 1974

[5] ROSA, Raphael Vicente; SIQUEIRA, Marcos Antonio; ROTHENBERG, Christian Esteve; BOREA, Emerson; MARCONDES, Augusto Cavalheiro. *Network Function Virtualization* : *Perspectivas, Realidades e Desafios*. Disponível em: <https://dl.dropboxusercontent.com/u/15183439/PPTs/NFV-short-course-SBRC14-full.pdf>. Acesso em : 20/10/2014.

[6] SAPONARA, S.; FANUCCI, L.; COPOLA, M. “Many-core platform with NoC interconnect for low cost and energy sustainable cloud server-on-chip ”. *Sustainable Internet and ICT for Sustainability (SustainIT),* pp. 1–5, Pisa, Italia , 4-5 Oct. 2012.

[7] GELBERGER, A. ; YEMINI, N. ; GILADI, R. "Network Hypervisors: Managing the Emerging SDN Chaos" Computer Communications and Networks (ICCCN), 2013 22nd International Conference on. pp. 1-7. Nassau, Bahamas, July 30 2013-Aug. 2 2013

[8] DALL, Christoffer; NIEH, Jason. “KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor”. *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 333–348, Salt Lake City, Utah, USA, Março 1-5, 2014.

[9] ROJA-CESSA, R; SALEHIN, K.M; “Experimental performance evaluation of a virtual software router”. *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. pp. 1-2 Dept. of Electr. & Comput. Eng., New Jersey Inst. of Technol. Newark, NJ, USA , Outubro 13-14 2011.

[10] MASUTANI, H.; NAKAJIMA, Y. ; KINOSHITA, T. ; HIBI, T. ; TAKAHASHI, H. ; OBANA, K.; SHIMANO, K. ; FUKUI, M. “Requirements and design of flexible NFV network infrastructure node leveraging SDN/OpenFlow”. *Optical Network Design and Modeling, 2014 International Conference on*, pp. 258 - 263, Stockholm, Sweden, Maio 19-22 2014.

[11] *Technical white paper Network functions virtualization*. Disponível em : <http://www.hp.com/hpinfo/newsroom/press_kits/2014/MWC/White_Paper_NFV.pdf>. Acesso em 20/10/2014.

[12] *Network Functions Virtualisation – Introductory White Paper*: *An Introduction, Benefits, Enablers, Challenges & Call for Action*. Disponível em :

<http://portal.etsi.org/NFV/NFV_White_Paper.pdf>. Acessado em: 20/10/2014.

[13] GELBERGER, A. ; YEMINI, N. ; GILADI, R. "Performance Analysis of Software-Defined Networking (SDN)". *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*. pp. 389 - 393, San Francisco, CA, USA, Agosto 14-16 2013.

[14] HATA, HIROAKI; "A study of requirements for SDN switch platform". Intelligent Signal Processing and Communications Systems (ISPACS), 2013 International Symposium on . pp 79 - 84, Naha, Japão, Novembro 12-15 2013.

[15] GOODACRE, JOHN. *White Paper - Technology Preview: The ARMv8 Architecture*, November 2011. Disponível em : <http://www.arm.com/files/downloads/ARMv8_white_paper_v5.pdf>. Acessado em 20/10/2014.

[16] WOLF, TILMAN; GRIFFIOEN, JAMES; CALVERT, KENNETH L.; DUTTA, RUDRA; ROUSKAS, GEORGE N.; ROUSKAS, ILIA X; NAGURNEY, ANNA "Choice as a principle in network architecture" *ACM SIGCOMM Computer Communication Review - Special october issue SIGCOMM 12 archive*, pp .105-106, Volume 42 Issue 4,New York, NY, USA October 2012.

[17] COPPOLA ST, MARCELLO; FALSAFI, BABAK; GOODACRE, JOHN; KORNAROS, GEORGE "From embedded multi-core SoCs to scale-out processors" *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 947-951.

[18] NGUYEN, TUNG; BUI, DUY-HIEU; PHAN, HAI-PHONG; DANG, TRONG-TRINH; TRAN, XUAN-TU. "High-performance adaption of ARM processors into Network-on-Chip architectures" *SOC Conference (SOCC), 2013 IEEE 26th International.* pp. 222 – 227. Erlangen, Alemanha, Setembro 4-6 2013.

[19] BLEM, E.; MENON, J.; SANKARALINGAM, K. "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures" *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 1 – 12, Shenzhen, Fevereito 23-27 2013.

[20] BIANCO, A.; BIRKE, R; GIRAUDO, L.; PALACIN, M. "OpenFlow Switching: Data Plane Performance" Communications (ICC), 2010 IEEE International Conference on, pp. 1 – 5, Cape Town, Maio 23-27 2013.

[21] PATTERSON, David A.; HENNESSY, John L. *Arquitetura de Computadores* : *Uma abordagem quantitativa*, 4a Edição. Rio de Janeiro : Elsevier, 2008 494p.

[22] LARABEL, MICHAEL; *ARM On Ubuntu 12.04 LTS Battling Intel x86?*; Publicado em 17 Fevereiro 2012. Disponível em: <http://www.phoronix.com/scan.php?page=article&item=ubuntu_1204_armfeb&num=1>. Acessado em 31/10/2014.

[23] LARABEL, MICHAEL; *ARM Cortex-A15 vs. NVIDIA Tegra 3 vs. Intel x86*; Publicado em 29 Novembro 2012. Disponível em: <http://www.phoronix.com/scan.php?page=article&item=samsung_exynos5_dual&num=1>. Acessado em 31/10/2014.

[24] Cortex-A53 Processor. Disponível em: <http://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php> Acessado em 20/10/2014.

[25] Cortex-A57 Processor. Disponível em: <http://www.arm.com/products/processors/cortex-a/cortex-a57-processor.php>. Acessado em 20/10/2014.

[26] Cavium Introduces ThunderXTM: A 2.5 GHz, 48 Core Family of Workload Optimized
Processors for Next Generation Data Center and Cloud Applications. Disponível em: <http://www.cavium.com/newsevents_Cavium_Introduces_ThunderX_A_2.5_GHz_48_Core_Family_of_Workload_Optimized_Processors_for_Next_Generation_Data_Center_and_Cloud_Applications.html>. Acessado em 20/10/2014.