ProViNet: Uma Plataforma para Gerenciamento de Redes Virtuais Programáveis

Wanderson Paim de Jesus[1], **Ricardo Luis dos Santos**[1], **Oscar Maurício Caicedo Rendón**[1]
Lisandro Zambenedetti Granville[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{wpjesus, rlsantos, omrendom, granville}@inf.ufrgs.br

Abstract. *With the evolvement of virtualization and network programming techniques, high-level applications can be used to define the behavior of network traffic while keeping isolation. However, to ensure a harmonious relationship between users, network applications and virtual networks, considerable management efforts are needed. In this paper we propose the ProViNet platform, a solution for managing the deployment of network applications in Programmable Virtual Networks (PVN). In addition to the management facilities, ProViNet contributes with an architecture that allows sharing the control plane of PVN in a scalable way. During the development of this work we identified the need for a standard representation of programmable virtual infrastructures, so it is also proposed an extension of the programmable virtual networks description language VXDL. In order to verify the feasibility of the proposed platform, we implemented a prototype, which is analyzed and evaluated in this paper.*

Resumo. *Ao passo que evoluem as técnicas de virtualização e programação de redes, aplicativos de alto nível podem ser utilizados para definir o comportamento de tráfegos de rede isoladamente. Entretanto, garantir um relacionamento harmônico entre usuários, aplicativos de rede e redes virtuais exige grandes esforços de gerenciamento. Neste trabalho propomos a plataforma ProViNet, uma solução para o gerenciamento da implantação de aplicativos de rede em Redes Virtuais Programáveis (RVP). Além de agregar facilidades no gerenciamento, ProViNet contribui com uma arquitetura que permite o compartilhamento do plano de controle de RVP de forma escalável. Durante o desenvolvimento do trabalho foi identificada a necessidade de uma representação padrão da infraestrutura virtual programável, para tanto propõe-se também, uma extensão da linguagem de definição de redes virtuais VXDL. A fim de verificar a viabilidade da plataforma proposta, foi implementado um protótipo, o qual é analisado e avaliado neste trabalho.*

1. Introdução

Historicamente, o núcleo das redes de computadores, quando comparado com os servidores, *desktops* e dispositivos móveis da borda das redes, é um ambiente hostil à inovação. No contexto específico da Internet, esse fato é geralmente referenciado como ossificação [Hausheer *et al.* 2011]. Para exemplificar, soluções propostas a mais de dez anos, como IPv6 e IPSec, ainda não estão amplamente em uso. São apontadas como possíveis causas: *(i)* a necessidade de modificações globais, ocasionalmente exigindo a substituição de

equipamentos; *(ii)* a lentidão no processo de padronização que trata da interoperabilidade com serviços legados; *(iii)* e a abordagem adotada pelas fabricantes de equipamentos, de implementar e implantar soluções baseadas em seu retorno financeiro.

Na intersecção entre o conceito de Virtualização de Redes [Chowdhury e Boutaba 2010] e o de Programabilidade de Redes [Kanaumi *et al.* 2010], emergem as Redes Virtuais Programáveis (RVP), as quais promissoramente prometem reverter o cenário de lentidão testemunhado nas redes de computadores. Uma das abordagens adotadas pelas RVP segue o formato das Redes Definidas por *Software* (SDN - *Software-Defined Networks*) [Lantz *et al.* 2010], que define o desacoplamento dos planos de controle e de dados. Algumas implementações do plano de controle se baseiam na Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*) para prover a comunicação com aplicativos de rede. Dessa forma, utilizando uma interface padronizada de definição de serviços, os aplicativos de rede podem ser programados em linguagens distintas, se tornam menos dependentes da tecnologia utilizada no plano de controle.

Por um lado, o desacoplamento entre aplicativos de rede, plano de controle e infraestrutura programável torna a arquitetura SDN flexível e escalável [Gutz *et al.* 2012]. Por outro lado, induz uma grande complexidade no gerenciamento. Modelos de gerenciamento utilizados nas redes comuns não são adequados às redes programáveis, uma vez que não tratam da implantação dinâmica de novos serviços. Além disso, dependendo das políticas de acesso às infraestruturas programáveis, um grande número de usuários poderão propor e implantar seus próprios aplicativos de rede. Nesse cenário, harmonizar os aplicativos, usuários e redes virtuais, mantendo a confiabilidade e escalabilidade dos serviços implantados é um problema em aberto.

As propostas para o gerenciamento de RVP variam de acordo com o ambiente de implantação e com os requisitos dos usuários. Nos ambientes de *testbeds*, as propostas focam em prover aos experimentadores e cientistas soluções para o controle do provimento de *Slices*. Entretanto, o gerenciamento da programabilidade que os usuários possuem sobre os *Slices* ainda é incipiente. São exemplos desse tipo de proposta o ProtoGENI [ProtoGENI 2012] e o OFELIA *Control Framework* [Kopsel 2011]. Nos ambientes de *Cloud*, mais direcionados ao mercado, implementam-se soluções para gerenciar os serviços que os provedores de *Cloud* oferecerão aos seus clientes. Ou seja, o foco maior está em prover controle aos gerentes e administradores da *Cloud*, deixando o usuário final sem chance de propor novos aplicativos de rede. As propostas da CITRIX, XenServer *Distributed vSwitch Controller* e da CISCO, OnePK, são exemplos.

O problema de pesquisa deste trabalho está em como prover acesso de múltiplos usuários finais a uma infraestrutura de RVP, agregando facilidades no gerenciamento e implantação de aplicativos de forma a encorajar o desenvolvimento de novas soluções de rede. Propõe-se para isso a plataforma de gerenciamento ProViNet (***Pro****grammable* ***Vi****rtual* ***Net****work Managemet Platform*). ProViNet contribui para o estado-da-arte em quatro pontos: *(i)* na elaboração de uma arquitetura para gerenciamento de RVP que provê escalabilidade e alta disponibilidade de serviços; *(ii)* em uma abordagem para implantação dinâmica de novos serviços no plano de controle; *(iii)* na extensão da linguagem de descrição de rede virtual programável, *Virtual Resources and Interconnection Networks Description Language* (VXDL) [Koslovski *et al.* 2008]; *(iv)* e por fim, no de-

senvolvimento, como parte da plataforma, de um sistema com interface de acesso Web que facilita a compreensão e interação dos usuários finais com ambientes de RVP.

O restante do artigo está organizado conforme segue. Na Seção 2, são descritos os principais trabalhos que envolvem o gerenciamento de RVP. Na Seção 3 é apresentada a plataforma ProViNet, discutindo a arquitetura conceitual e conceitos empregados. Em seguida, na Seção 4 é detalhado o protótipo utilizado como base para a avaliação e análise apresentada na Seção 5. Por fim, a Seção 6 conclui o artigo com as considerações finais e perspectivas para trabalhos futuros.

2. Trabalhos Relacionados e Contextualização

Os conceitos de Virtualização de Redes [Chowdhury e Boutaba 2009] [Chowdhury e Boutaba 2010] e a Programabilidade de Redes [Campbell *et al.* 1999] [Lin *et al.* 2011] são bem discutidos na literatura. A junção desses conceitos formam as Redes Virtuais Programáveis (RVP), as quais são mais comumente aplicadas em dois ambientes, nos projetos de plataformas de testes, também conhecidos como *testbeds* e em Nuvens privadas (*Private Clouds*). Os trabalhos relacionados apresentados neste capítulo são organizados conforme esses dois ambientes, buscando evidenciar em cada um deles o nível de abstração da infraestrutura virtual, a abordagem utilizada e a natureza da licença.

Dentre as propostas no contexto dos *testbeds*, as quais focam em auxiliar os pesquisadores em seus experimentos, destaca-se o *framework* de controle OFELIA (OCF) [Kopsel 2011], o qual é uma derivação da plataforma Expedient proposta pela Universidade de Stanford. Esse *framework* auxilia os pesquisadores na criação de *Slices* utilizando recursos de várias federações. Além disso, lhes oferece também a capacidade de associar esses *Slices* a controladores previamente configurados. Apesar dessa funcionalidade de associação através da interface gráfica, o OCF não provê o gerenciamento da implantação de aplicativos de rede, considerada uma das maiores preocupações da virtualização de redes [Chowdhury e Boutaba 2010].

Outra proposta, ainda no contexto dos *testbeds*, foi criada no projeto GENI [GENI 2011] e é chamada ProtoGENI [ProtoGENI 2012]. Tal proposta também implementa uma interface de acesso via Web que assiste aos pesquisadores na criação de *Slices* com recursos oriundos de diversas federações. Ao interagir com a interface do ProtoGENI, pesquisadores podem instanciar nós virtuais e conecta-los dinamicamente. Assim como a maioria das propostas desenvolvidas nesse mesmo contexto, o foco está no provimento do *Slice* e não na programabilidade do mesmo. Em geral, essas soluções são livres de licença, entretanto existe um conjunto burocrático de regras para utilização e acesso aos recursos geridos pelas ferramentas citadas.

As soluções no contexto de *Cloud Computing* se diferem das propostas de *testbeds*, principalmente pelo foco comercial. Os ambientes de *Cloud* em geral demandam uma grade quantidade de recursos de rede, pois comercializam serviços com alta taxa de disponibilidade e qualidade de serviço. Portanto, a criação de serviços de rede customizados para atender a demandas especiais é visto como um grande atrativo para os provedores de *Cloud*. Para atender esses provedores, surgiram soluções que aumentam o poder de customização dos serviços providos nas redes virtuais de seus *datacenters*. Algumas dessas soluções são comercializadas, tais como Nexus 1000v e OnePK da CISCO e DVSC (*Distributed Virtual Switch Controller*) da CITRIX. Outras são de código aberto,

como os *plugins open-vSwitch*, Ryu Plugin e o *restproxy*, para a plataforma OpenStack [OpenStack 2011].

Tanto em *Cloud* quanto em *testbeds*, é crescente o número de propostas que empregam os conceitos da arquitetura SDN em suas soluções de programabilidade em redes virtuais. Rubio *et al.* [Rubio-Loyola *et al.* 2011], por exemplo, propôs um plano de orquestração, o qual é complementar aos planos originalmente definidos na arquitetura SDN (controle e dados). O propósito desse novo plano é controlar dinamicamente o comportamento das redes virtuais em resposta às constantes variações, gerando assim um sistema de gerenciamento autonômico. Embora esse sistema tenha vantagens, como resposta rápida às falhas geradas por variações no comportamento da rede, requer soluções padronizadas e bem testadas. Em consequência, o usuário final continua distante da criação e implantação de aplicativos na rede virtual programável. Tal fato, contribui para baixa taxa de inovação nas redes, pois mantem a natureza restritiva e a pequena quantidade de pessoas aptas a desenvolver e implantar novas soluções.

Em resumo, apesar de existirem propostas recentes envolvendo redes virtuais programáveis, nenhuma das pesquisadas promove o gerenciamento da programabilidade em uma infraestrutura de RVP, considerando o acesso de múltiplos usuários finais. Além disso, maior parte das propostas analisadas se limitam ao provisionamento da rede virtual (controle de máquinas virtuais e configuração da rede virtual), não oferecendo funcionalidades para o gerenciamento dos aplicativos de rede que podem ser instalados dinamicamente nas redes virtuais programáveis.

3. ProViNet

Uma das abordagens para as Redes Virtuais Programáveis é a utilização da arquitetura de Redes Definidas por *Software*. Tal arquitetura define que os planos de controle e de dados sejam desacoplados. Sendo assim, eles necessitam de um protocolo para comunicação. Recentemente tem se empregado o termo *Southbound API* (SBAPI) para se referir aos protocolos que provejam essa comunicação entre o plano de controle e de dados. Conforme ilustrado na Figura 1 a SBAPI **(2)** é utilizada para comunicação entre o *Pool* de Controle e os *Slices*, que serão descritos mais diante.

Ao passo que surgiram diversas soluções para a camada de controle, e cada implementação adota um padrão de linguagem, os aplicativos de rede criados seguiam tal heterogeneidade, ficando assim dependentes das tecnologias dos controladores e isoladas entre si. Atualmente a tendência é que as diferentes implementações do plano de controle ofereçam uma interface padrão de comunicação com aplicativos de rede externos. É comum se referir a esse tipo de interface como *Northbound API* (NBAPI) **(1)**. Em geral, elas independem de linguagem, baseando-se na arquitetura *Representational State Transfer* (REST), por exemplo. Desse modo, novos aplicativos que venham a interagir com a camada de controle necessitam apenas das especificações de serviços providas por cada implementação de controlador.

Conforme ilustrado na Figura 1, a plataforma ProViNet auxilia os usuários finais no gerenciamento e implantação de aplicativos de rede em *Slices* de Redes Virtuais Programáveis. Para isso, se apoia no conceito de separação de planos, sejam eles: o plano de dados, representado pelos elementos que formam a rede virtual nos *Slices* e utilizam a SBAPI para se comunicarem com o plano de controle; o plano de controle, formado

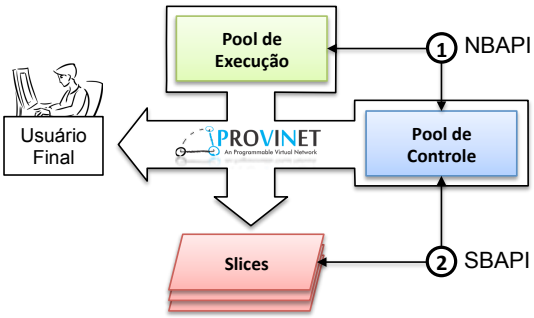

Figura 1. Módulos e relacionamentos

pelos controladores, que são agrupados no *Pool* de Controle para prover alta disponibilidade, conforme será apresentado na Subseção 3.1; e pelo plano de aplicativos, presente no *Pool* de Execução, o qual organiza, armazena e executa os aplicativos de rede que se aproveitam do conceito de NBAPI para comunicação com o plano de controle.

A plataforma ProViNet se aplica a qualquer ambiente que utilize infraestrutura compatível com o conceito de RVP. Em geral, a aplicação se dá em dois níveis, um no nível de rede virtual, no qual o plano de dados seriam representado pelos *vSwitches*, e outro no nível físico, utilizando os *switches* físicos compatíveis como plano de dados. Ou até mesmo em um modelo híbrido, com controle em ambos os níveis.

3.1. Arquitetura Conceitual

A arquitetura apresentada na Figura 2 ilustra os componentes da plataforma ProViNet assim como suas relações em alto nível. O usuário interage com a plataforma através de uma interface Web que expõe graficamente as funcionalidades providas por seus módulos. Conforme ilustrado, essas funcionalidades se subdividem em quatro interações. Na interação **(a)**, o módulo Controle de Usuários atende a requisições de autenticação e cadastro. O processo de gerenciamento de implantação e execução de aplicativos, executado na interação **(b)**, é tratado pelo módulo Controle de Aplicativos. As solicitações de infraestrutura virtual são enviadas na interação **(c)** e tratadas pelo módulo de Controle de *Slices* e Referências. Por fim, as configurações inerentes aos três módulos citados são apresentadas em uma interface de administração **(d)** para o Administrador do ambiente de implantação do ProViNet.

ProViNet fornece escalabilidade utilizando uma arquitetura baseada no conceito chamado de *Resource Pool*. Cada *Pool* representa um conjunto de servidores de virtualização (*hypervisors*) interligados e controlados por uma plataforma de gerenciamento única. O *Pool* de Controle é utilizado para execução de máquinas virtuais com sistemas idênticos, que executam uma implementação de controlador pré-definida e compatível com o conceito de NBAPI. Os aplicativos escritos pelos usuários são executados em máquinas virtuais individuais alocadas no *Pool* de Execução. A comunicação entre os controladores do *Pool* de Controle e as VMs do *Pool* de Execução é provida pela NBAPI. Já a comunicação entre os controladores e os elementos do plano de dados nos *Slices* é provida pela SBAPI. As requisições realizadas pelos módulos do ProViNet aos *Pools* utilizam as interfaces de comunicação providas pela plataforma de virtualização adotada. Finalmente, a requisição ao Provedor de Infraestrutura Virtual ocorre por meio de uma requisição HTTP, enviando um documento de descrição de infraestrutura virtual, a ser

comentado mais adiante.

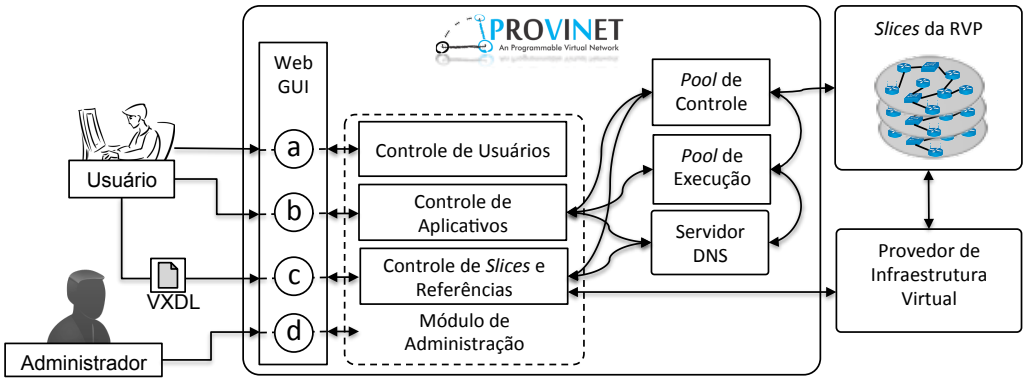

Figura 2. Arquitetura conceitual

Analisando novamente a Figura 2, percebe-se a esquerda o Usuário Final e a direita a Rede Virtual Programável. Entre esses elementos está a plataforma ProViNet, provendo a ligação entre eles. Nessa posição, o ProViNet oferece o gerenciamento da implantação de aplicativos, o controle de acesso para ambientes com múltiplos usuários e os requisitos não funcionais de disponibilidade, confiabilidade e escalabilidade. Com exceção do módulo de Controle de Usuários e de Administração, devido a simplicidade, os outros são detalhados nas subseções seguintes.

3.2. Controle de *Slices* e Referências

Uma Rede Virtual Programável (RVP) dispõe de recursos computacionais e de rede necessários para a criação de redes isoladas e programáveis. Para isso, se apoiam em tecnologias de virtualização (tais como XenServer, VMWare e Virtualbox) e de programabilidade (como o OpenFlow, empregado neste trabalho). Seja qual for a tecnologia e a abordagem de provimento da RVP, o resultado é que o usuário terá uma rede virtual programável que interliga exclusivamente suas máquinas virtuais. Por compartilharem os recursos do provedor (ou *datacenter*), é conveniente e usual se referir ao subconjunto de recursos de rede, computacionais e de armazenamento pertencentes a um usuário, de *Slice*.

Não é função da plataforma ProViNet o provimento do *Slice*, ou seja, inicializar máquinas virtuais e configurar a rede virtual que interliga as mesmas. Portanto, o módulo Controle de *Slices* e Referências tem, entre outras, a função de receber e encaminhar um documento de descrição de infraestrutura virtual especificando a topologia e os recursos a serem alocados pelo Provedor de Infraestrutura Virtual (PIV). Existem diversas propostas para esse tipo de documento, tais como Rspec (GENI), NDL-OWL(RENCI), NMC (OGF) e *Virtual Resources and Interconnection Networks Description Language* (VXDL) do projeto INRIA [Koslovski *et al.* 2008]. Neste trabalho propomos uma extensão para a linguagem VXDL, tornando-a compatível com arquiteturas de programabilidade de redes em que haja separação de planos (de controle e de dados).

Originalmente, a linguagem VXDL, apresentada no trabalho de Koslovski *et.al.* [Koslovski *et al.* 2008], define que os recursos possuem um nome e podem ter uma lista de funções, parâmetros, softwares e uma localização conforme se nota no trecho destacado de sua definição e apresentado abaixo:

```
<resource> ::= "resource" "(" <name> ")" "{"
               ["function" <elementary-functions>]
               ["parameters" <resource-parameters>]
               ["software" <software-list>]
               ["anchor" <location>] "}"
```

O atributo *function* pode assumir diversos valores, tais como *endpoints*, *aquisition* e *router*. Propõe-se a adição de um novo atributo inerente ao valor *router*, o qual nomeia-se <*controller-list*>. Esse valor representa uma lista de controladores. Foi adotado uma lista pois alguns roteadores ou *switches* compatíveis com SDN aceitam redundância de controladores. O *switch* virtual Open vSwitch por exemplo aceita a configuração de um *master* e diversos *slaves*. Cada elemento <*controller*> é composto por atributos que definem o tipo de conexão (ftp, ptcp e ssl são exemplos), o endereço IP e a porta em que o controlador remoto está configurado.

```
<elementary-functions> ::= <function> ("," <function> )*
<function> ::= "endpoint" | "aquisition" | "storage"
    | "computing" | "visualization" | "network_sensor"
    | "router" "(" "ports" <ports> [<controller-list>] ")"
<ports> ::= <number>
<controller-list> ::= <controller> ("," <controller> )*
<controller> ::= "(" <connection-type> <ip-address> <port> ")"
```

O usuário inicialmente deve elaborar este arquivo sem as informações sobre o controlador, pois estas serão adicionadas automaticamente pelo ProViNet durante o processo de requisição de infraestrutura virtual. Isso se justifica pelo fato do usuário não ter informações sobre os endereços de IP dos controladores, até mesmo porque eles são dinâmicos, conforme será apresentado adiante.

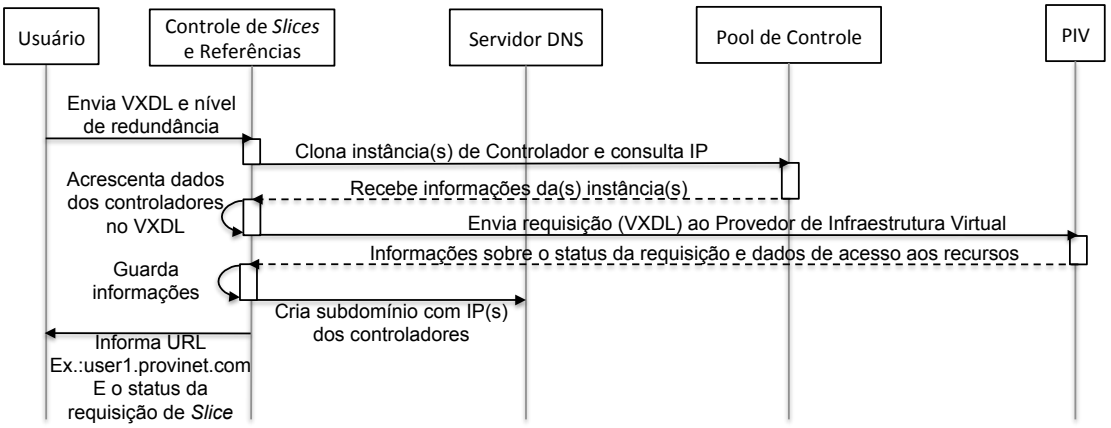

Figura 3. Diagrama de sequência da abordagem de criação de *Slices*

De acordo com o diagrama da Figura 3, quando o usuário já elaborou o documento VXDL, ele o envia ao módulo Controle de *Slices* e Referências via formulário de *upload*. No mesmo formulário, o usuário deve definir um nível de redundância de controladores no *Pool* de Controle que pretende ser associado ao *Slice* requerido. Após a instanciação dos controladores, o *Pool* de Controle reponde a requisição informando o IP dos mesmos. Esses IPs são adicionados ao VXDL juntamente com o tipo de conexão e porta. Em se-

guida, o documento é enviado ao PIV por uma requisição HTTP, gerada automaticamente pela plataforma.

Ao receber a resposta sobre o status da requisição e com informações sobre o acesso aos recursos do *Slice*, a plataforma registra no Servidor DNS os IPs dos controladores (os mesmos adicionados no documento VXDL). Esse registro é a adição do nome do usuário como subdomínio do domínio *www.provinet.local*. Ou seja, cada usuário tem um subdomínio no qual são associados os IPs dos controladores por ele requeridos. Caso o usuário tenha definido grau e redundância maior que um, o Servidor DNS aplicará o algoritmo Round Robin entre as referências. Abaixo segue um exemplo de configuração de uma domínio em que o "user1" possui dois controladores e o "user2" tem um.

```
...
                NS      servidor.provinet.local.
servidor        A       xxx.xxx.xxx.xxx
user1           A       <IP-controlador-1>
user1           A       <IP-controlador-2>
user2           A       <IP-controlador-3>
...
```

Por fim, o módulo apresenta ao usuário a URL a ser utilizada em seus aplicativos para fazer chamadas aos serviços (Ex.: http://user1.provinet.local). São fornecidos, também, os dados de acesso aos recursos virtuais recebidos do PIV. Dessa forma, o aplicativo do usuário poderá enviar requisições aos serviços providos pelos controladores instanciados no *Pool* de Controle.

3.3. Controle de Aplicativos

Uma vez que a estrutura de programabilidade já está estabelecida, ou seja, o usuário possui um *Slice* e um plano de controle devidamente configurado, resta ao usuário desenvolver e executar os aplicativos de rede. Para que o desenvolvimento seja possível, o usuário precisa saber quais são os serviços a sua disposição. Por isso, é disponibilizado através da interface Web do ProViNet, uma documentação completa sobre tais serviços, os quais dependem da tecnologia utilizada no *Pool* de Controle.

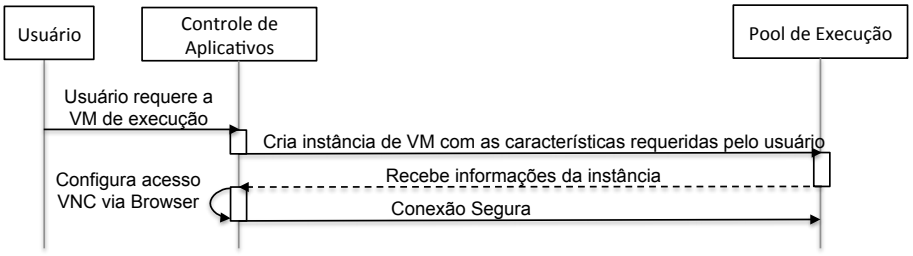

Figura 4. Abordagem de referências para provimento de escalabilidade e disponibilidade

De acordo com o diagrama apesentado na Figura 4, no primeiro passo o usuário acessa a plataforma e requer a VM que rodará seus aplicativos. Nesta requisição são apresentados perfis de VM, variando o Sistema Operacional e quantidade de recursos, tais como memória, processamento e armazenamento. Após a escolha de um perfil, a plataforma, por meio de uma interface de comunicação com o *Pool* de Execução requer uma VM com tais características. Para evitar qualquer tipo de bloqueio por *firewall*, ao

receber os dados da VM criada, o módulo Controle de Aplicativos configura uma interface de acesso remoto via *Browser*. Ou seja, o usuário é capaz de visualizar e interagir com a VM criada por meio de um terminal apresentado em seu *Browser*.

Como ilustrado na Figura 5, o aplicativo do "user1" rodando no *Pool* de Execução poderia fazer a chamada *http://user1.provinet.local/getTopology* para consultar a topologia da rede virtual disponibilizada em seu *Slice*. Se o usuário tem mais de um controlador, as requisições dos aplicativos serão direcionadas para os controladores conforme o algoritmo Round Robin, implementado pelo Servidor DNS. Assim, existirá um balanceamento de carga entre os controladores. Além disso, em caso de falha de um controlador, terá outro para atender as requisições.

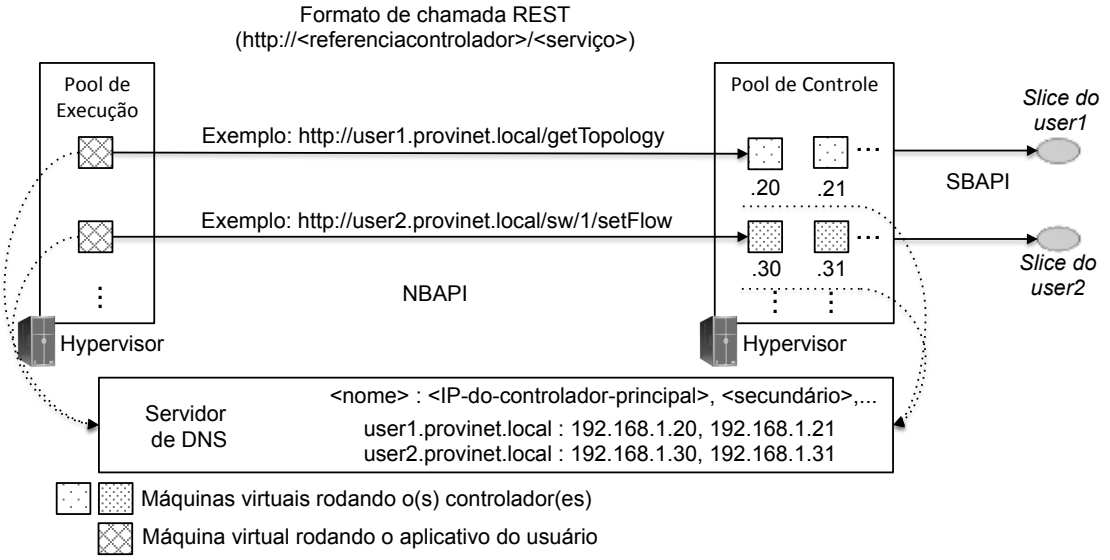

Figura 5. Abordagem de referências para provimento de escalabilidade e confiabilidade

O conjunto de serviços disponíveis deve variar de acordo com a tecnologia utilizada no plano de controle. Todavia, diante da necessidade de um novo serviço, o mesmo pode ser desenvolvido e instalado. Em geral, os controladores seguem uma arquitetura que permite o acoplamento de módulos com a implementação de novos serviços. Baseado nas informações de quais controladores pertencem a quais usuários, a plataforma ProViNet auxilia o usuário na instalação desses módulos.

Para implementar o módulo a ser instalado, o usuário deve seguir os tutoriais[1] disponibilizados pela fabricante do controlador adotado no plano de controle. O processo de instalação de um novo módulo se inicia pelo envio do módulo compactado para a plataforma. O módulo responsável então o encaminha a todos os controladores do usuário que estão associados a um determinado *Slice*. Em cada controlador, um *deamon* configurado para fazer a implantação de módulos executa uma rotina de instalação. Uma vez que os módulos estão instalados, os aplicativos dos usuários podem enviar requisições a ele (Ex.: *http://user1.provinet.local/modules/newModule/service*).

[1]Tomando como exemplo o controlador Floodlight, estão disponíveis no endereço *http://www.openflowhub.org/display/floodlightcontroller/How+to+Write+a+Module*, um conjunto de instruções para o desenvolvimento de novos módulos.

4. Protótipo

Com o objetivo de demonstrar a viabilidade da arquitetura conceitual detalhada na Figura 2 do Capítulo 3, foi desenvolvido um protótipo. Sua implementação baseia-se no desenvolvimento dos cinco principais módulos, a Interface Web, o Controle de Usuários, o Controle de Aplicativos, o Controle de *Slices* e Referências e a Administração do ProViNet. Tais módulos foram implementados utilizando o *framework* Django 1.4.3, a linguagem Python 2.7.3 e o sistema gerenciador de banco de dados PostgreSQL 9.1.6. A fim de fornecer maior compatibilidade do sistema, foi utilizado o servidor Web Apache 2.2.23.

O módulo Controle de Usuários fornece na interface do ProViNet, formulários de registro e login. O Controle de Aplicativos apresenta na interface uma área especial para apresentação dos serviços disponibilizados no plano de controle e uma área para requisição, controle e interação com as VMs no Pool de Execução. É disponibilizado pelo módulo de Controle de *Slices* e Referências um formulário para *upload* do documento VXDL e definição do nível de redundância. Finalmente, a área de Administração apresenta um conjunto de funcionalidades de configuração, que incluem o endereço de IP e dados de acesso dos *Pools*, do Servidor DNS, do Provedor de Infraestrutura Virtual e ainda a definição dos perfis de máquinas virtuais que serão disponibilizadas aos usuários.

Os *Pools* de Execução e de Controle são, na prática, servidores com alguma plataforma de virtualização instalada. No protótipo desenvolvido foi utilizado o *hypervisor* XenServer da CITRIX. Sendo assim, a comunicação entre os módulos da plataforma ProViNet e os Pools se dão pela utilização do XenServer SDK. Com o SDK é possível controlar e monitorar o *hypervisor* através de chamadas XML-RPC. O último módulo da plataforma é o Servidor DNS, o qual foi instalado em uma máquina com os sistemas Bind9 e Apache2. Para o controle dinâmico de configuração do DNS, um serviço web foi implementado para que, a partir de chamadas HTTP a uma URL específica, seja feita a adição e remoção de entradas no arquivo de configuração do bind9.

A implementação de controlador utilizada foi o Floodlight, por prover uma API RESTfull que possibilita o consumo dos serviços disponibilizados no controlador por meio de chamadas HTTP. Outras implementações poderiam ser utilizadas, desde que seja possível a instalação de módulos para provimento de serviços customizados e a disponibilização de serviços utilizando a arquitetura REST.

5. Caso de Estudo e Análise de Resultados

Para avaliar a solução apresentada foi elaborado um caso de estudo que aborda cada um dos diagramas de sequência apresentados no Capítulo 3. O cenário de execução é composto por servidores Intel Xeon CPU E3-1220 3.1GHz, 4GB RAM, com o *hypervisor* XenServer 6.1 representando o *Pool* de Controle e de Execução. O controlador OpenFlow utilizado foi o Floodlight v0.90, e é iniciado, por *script* durante a inicialização da VM (Ubuntu 12.04 com 1 vCPU e 384MB RAM) no *hypervisor*. Para executar o *framework* Django com o ProViNet foi utilizado um *laptop* Intel Core i7 2.8GHz e 4GB RAM. Por fim, o Servidor DNS foi instalado e configurado em um terceiro PC (Intel Core 2 Duo 2.33GHz e 4GB RAM) na mesma rede local que os outros PCs. Vale ressaltar que os tempos apresentados neste trabalho foram obtidos após 30 execuções, e representam a média uma vez que o coeficiente de variação foi muito próximo de zero.

Instanciar dois controladores	Adicionar informações no VXDL	Provimento da infraestrutura virtual (PIV)	Configuração de subdomínios	Total
12,824s	0,003s	57,81s	0,04s	75,677s

Tabela 1. Tempos para requisição de *Slice* e configuração de subdomínios

Atraídas pelo baixo custo de manutenção, segurança e armazenamento, é cada vez mais comum que empresas migrem parte de sua infraestrutura de computação para ambientes de *Cloud*, seja ela pública, privada ou híbrida. Para representar esse caso de estudo, consideramos uma rede composta por 7 *switches* e 4 *hosts*, organizados em uma topologia de árvore. Consideramos também que o usuário responsável pela migração já está registrado na plataforma ProViNet. Inicialmente, o usuário faz a requisição da infraestrutura virtual enviando um arquivo VXDL com a descrição da topologia (*switches*, *hosts* e *links*). A avaliação desse processo é apresentada na Tabela 1 e pode ser acompanhada pelo diagrama da Figura 3. Os tempos apresentados consideram nível de redundância igual a dois, ou seja, o *Slice* terá dois controladores associados.

Vale ressaltar que o tempo gasto para instanciar a infraestrutura virtual deve variar de acordo com a abordagem de mapeamento utilizada pelo PIV. A utilização de máquinas virtuais ao invés de *bridges* para representar os *switches* virtuais, por exemplo, certamente acarreta em maior custo de tempo. Os tempos apresentados consideram um sistema desenvolvido em um trabalho anterior chamado HyFS [Wickboldt *et al.* 2012] como provedora da infraestrutura virtual programável. O HyFS é atualmente a única plataforma de *private Cloud* capaz de receber requisições de rede virtual programável com topologias variadas. A topologia virtual é criada utilizando o modelo *overlay*, no qual *switches* e *hosts* são implantados em máquinas virtuais. Uma vez instanciados, os *software switches* são configurados para receberem informações de controle de controladores externos, estabelecidos pela plataforma ProViNet.

Após o provimento da infraestrutura virtual, o usuário responsável pela migração deve avaliar os serviços presentes no cenário anterior a migração, tais como *Firewall*, Balanceadores de carga e outros serviços necessários. Uma vez que a rede virtual a ser criada é programável, esses serviços podem ser implementados em forma de aplicativos de rede. Utilizando uma notação simplificada, um exemplo de aplicativo para tratamento de ataques de negação de serviço (DDoS) é apresentado no Algoritmo 1. Esse aplicativo roda no plano de execução, em uma VM que o usuário acessa através do *Browser*.

O algoritmo apresentado analisa o tráfego recebido por um *switch* a cada 10 segundos, se esse tráfego for superior a um limite pré definido o sistema toma alguma ação. Essa ação depende do número de ocorrências em que se detectou tráfego superior ao limite definido. Na segunda ocorrência o sistema bloqueia o tráfego ICMP, na terceira, o tráfego Web (porta 80) é bloqueado, e por fim, na quarta ocorrência, todo tráfego é bloqueado e um gerente é contactado. Certamente essa não é a melhor solução, mas é um exemplo de um possível aplicativo de rede que pode ser desenvolvido pelo usuário e implantado no *Pool* de Controle para proteger seu *Slice* desse tipo de ataque.

Para medir desempenho das requisições do aplicativo do usuário, foram realizados experimentos com varições no nível de redundância no plano de controle. Espera-se uma variação no desempenho das chamadas, pois ao utilizar uma maior quantidade de controladores, o balanceamento de carga realizado pelo Servidor DNS torna o processo

Algorithm 1 Exemplo de Aplicativo: Solução ProViNet para DDoS

Require: limite - Limite de tráfego considerado normal
Require: swdpid - DPID do switch de entrada de tráfego rede

```
1: limite ← 2000MB
2:
3: anterior ← httpRequest(user1.provinet.local/readstatus/swdpid)
4: ocorrencias ← 1
5: while True do
6:     atual ← httpRequest(user1.provinet.local/readstatus/swdpid)
7:     if (atual − anterior) ≥ limite then
8:         if ocorrencias == 2 then
9:             httpRequest(user1.provinet.local/addFlow/swdpid/blockICMP)
10:        else if ocorrencias == 3 then
11:            httpRequest(user1.provinet.local/addFlow/swdpid/blockWeb)
12:        else if ocorrencias ≥ 3 then
13:            httpRequest(user1.provinet.local/addFlow/swdpid/blockAll)
14:            CallManager()
15:        end if
16:        ocorrencias + +
17:    end if
18:    anterior ← atual
19:    sleep(10)
20: end while
```

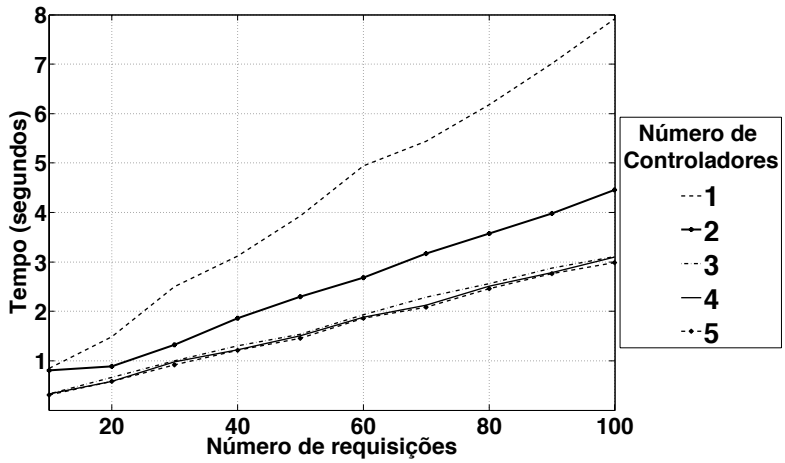

Figura 6. Performance do plano de controle com balanceamento de carga

mais rápido. Tal fato pode ser acompanhado no gráfico da Figura 6, que mostra o tempo gasto para concluir X requisições, sendo X, valores entre 10 e 100 com intervalos de 10 unidades. A requisição executada para obtenção dos valores apresentados foi *provinet.local/wm/core/controller/switches/json*, a qual retorna a lista de *switches* presentes no *Slice*.

Analisando os valores apresentados no gráfico 6, percebe-se que o balanceamento de carga provido pela abordagem proposta e gerenciado pelo ProViNet é efetivo e implica

em uma redução do tempo gasto para execução das requisições. Entretanto, o ganho se torna menos significativo para um número de controladores maior que 3. Ou seja, utilizando apenas 1 controlador, foram gastos em média 7,91 segundos para concluir 100 requisições, ao passo que com 2 controladores esse valor caiu para 4,45. O ganho ao aumentar o número de controladores de 4 para 5, não é tão expressivo quanto de 1 para 2, saindo de 3,09 para 2,98 segundos nesse caso. Discussões mais aprofundadas nesse contexto foram apresentadas por Heller *et.al.* [Heller *et al.* 2012].

Para avaliar a funcionalidade de implantação de módulos sob demanda, desinstalou-se do Floodlight um módulo que originalmente já vem instalado, o módulo de *firewall*. Compactou-se tal módulo em um arquivo e, através da interface do ProViNet, foi feita a requisição de instalação. Uma vez que o módulo Controle de Aplicativos possui cadastrado o endereço IP de todos os controladores e seus respectivos usuários, após receber o arquivo por *upload*, um *script* de envio é disparado. O papel desse *script* é acessar via ssh a VM de cada controlador, fazer a cópia do novo módulo para uma pasta específica na VM e ativar um segundo *script* na VM que faz a instalação do mesmo. Esse segundo *script* segue as informações disponibilizadas no site do Floodlight para instalação de módulos. O tempo médio gasto nesse processo foi de 23,435 segundos.

6. Conclusões e Trabalhos Futuros

A diversidade de ambientes computacionais requerem distintos serviços de comunicação em redes. As soluções desenvolvidas no passado, e implementadas de acordo com as vontades das fabricantes de equipamentos de redes, podem não ser mais suficientes. Entretanto, com o surgimento de propostas abertas de virtualização e programabilidade, como as Redes Definidas por *Software*, a criação de novas soluções se torna, de certa forma, mais democrática. Ou seja, depende menos dos anseios financeiros das grandes fabricantes.

Todavia, a complexidade inerente ao gerenciamento de ambientes de Rede Virtual Programável (RVP) ainda representa um grande desafio. Neste trabalho, propomos a plataforma ProViNet para o gerenciamento da implantação de aplicativos de rede em ambiente de RVP. A plataforma ProViNet contribui com uma arquitetura escalável, utilizando o conceito de *Resource Pool*, com uma abordagem para a instalação dinâmica de novos módulos no plano de controle, com a extensão da linguagem de definição de infraestrutura virtual de rede virtual programável, chamada VXDL, e por fim, com o desenvolvimento de um sistema com interface de acesso Web, facilitando a compreensão e interação com usuários finais.

Como trabalhos futuros pretende-se investigar mais precisamente, como os ambientes de *Cloud* poderiam prover pratilheiras de serviços de rede dinâmicas. As quais seriam ocupadas por soluções desenvolvidas e consumidas pelos próprios usuários de *Cloud*.

Referências

Campbell, A. T., Meer, H. G. D., Kounavis, M. E., Miki, K., Vicente, J. B., e Villela, D. (1999). A survey of programmable networks. *Computer Communication Review*, 29:7–23.

Chowdhury, N. e Boutaba, R. (2009). Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20 –26.

Chowdhury, N. M. K. e Boutaba, R. (2010). A survey of network virtualization. *Computer Network*, 54(5):862–876.

GENI (2011). Global Environment for Network Innovations. Disponível em: http://www.geni.net/. Acessado em: Julho 2012.

Gutz, S., Story, A., Schlesinger, C., e Foster, N. (2012). Splendid isolation: a slice abstraction for software-defined networks. Em *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, páginas 79–84, New York, NY, USA. ACM.

Hausheer, D., Parekh, A., Walrand, J., e Schwartz, G. (2011). Towards a compelling new internet platform. Em *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, páginas 1224 –1227.

Heller, B., Sherwood, R., e McKeown, N. (2012). The controller placement problem. Em *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, páginas 7–12, New York, NY, USA. ACM.

Kanaumi, Y., Saito, S., e Kawai, E. (2010). Deployment of a programmable network for a nation wide randd network. Em *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, páginas 233 –238.

Kopsel, W. (2011). Ofelia - pan-european test facility for openflow experimentation.

Koslovski, G. P., Primet, P. V.-B., e Charão, A. S. (2008). Vxdl: Virtual resources and interconnection networks description language. volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, páginas 138–154. Springer.

Lantz, B., Heller, B., e McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. Em *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, páginas 19:1–19:6, New York, NY, USA. ACM.

Lin, P., Bi, J., Hu, H., Feng, T., e Jiang, X. (2011). A quick survey on selected approaches for preparing programmable networks. Em *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, páginas 160–163, New York, NY, USA. ACM.

OpenStack (2011). Open source software for building private and public clouds. Disponível em: http://www.openstack.org/. Acessado em: Julho 2012.

ProtoGENI (2012). Control Framework for GENI Cluster C. Disponível em: http://www.protogeni.net/trac/protogeni. Acessado em: Dezembro 2012.

Rubio-Loyola, J., Galis, A., Astorga, A., Serrat, J., Lefevre, L., Fischer, A., Paler, A., e Meer, H. (2011). Scalable service deployment on software-defined networks. *Communications Magazine, IEEE*, 49(12):84 –93.

Wickboldt, J. A., Granville, L. Z., Schneider, F., Dudkowski, D., e Brunner, M. (2012). A new approach to the design of flexible cloud management platforms. Em *8th International Conference on Network and Service Management (CNSM)*, páginas 155–158, Las Vegas, USA.