

Universidade Federal do Rio de Janeiro
Escola Politécnica
Departamento de Eletrônica e de Computação

**Um Servidor de Máquinas Virtuais Adaptado a Múltiplas
Pilhas de Protocolos**

Autor:

Rafael dos Santos Alves

Orientador:

Prof. Luís Henrique Maciel Kosmowski Costa, Dr.

Co-orientador:

Prof. Miguel Elias Mitre Campista, D.Sc.

Examinador:

Prof. Marcelo Luiz Drumond Lanza, M.Sc.

Examinador:

Igor Monteiro Moraes, D.Sc.

DEL

Março de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica - Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmear ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do autor e dos orientadores.

À minha família.

Agradecimentos

Em primeiro lugar, agradeço a Deus por me presentear com a vida e por todos os outros dons que recebi. Sem a Sua presença nada disso seria possível.

Aos meus pais, pelo carinho e dedicação durante toda a minha vida; por abrir mão do conforto para garantir minha educação; pelo exemplo de vida e pelo incentivo.

À minha namorada Marcelle pelos momentos felizes e pela compreensão nos momentos mais complicados.

À Congregação Batista de Viçoso Jardim em Papucaia pelo apoio e pelas orações. Em especial, ao meu pastor Hamilton pelos conselhos e pela compreensão.

Aos professores e orientadores Luís Henrique e Miguel pelos conselhos e orientação, por terem me mostrado como pensar e escrever de forma científica e ajudar a derrubar a barreira das apresentações públicas.

Aos amigos e colegas da graduação, Bernardo, Eric, Jefferson e Viviane, pela amizade e pelos momentos divertidos durante a graduação. Aos amigos do GTA, Igor, Marcelo, Natália, Lino, Carlo, Rodrigo, Hugo, Diogo e Pedro, pela disposição em contribuir para este trabalho.

A todos os professores que fizeram e fazem parte da minha vida, em especial os da UFRJ, pela dedicação e excelência. Agradeço em particular aos professores Luís Henrique Maciel Kosmowski Costa, Miguel Elias Mitre Campista e Marcelo Luiz Drumond Lanza e ao doutor Igor Monteiro Moraes pela presença na banca examinadora.

A todas as pessoas que direta ou indiretamente contribuam para que este trabalho se tornasse real.

Ao CNPq, CAPES, FAPERJ e FINEP pelo financiamento de pesquisa.

Resumo

A Internet atual apresenta duas características fundamentais: o argumento fim-a-fim e a pilha de protocolos TCP/IP. Esse modelo permitiu o avanço da Internet durante muitos anos porque novas aplicações podiam ser desenvolvidas sem modificação do núcleo da rede. Entretanto, esse modelo dificulta o atendimento de requisitos cada vez mais importantes como mobilidade, gerenciamento e segurança. Diante das dificuldades apontadas, propostas de novas arquiteturas para a Internet têm aparecido na literatura nos últimos anos. Muitos destes trabalhos apontam a virtualização de computadores como uma ferramenta poderosa nas arquiteturas que vêm sendo propostas, seja como infraestrutura de uma nova Internet ou como ambiente de experimentação. Para a criação de redes virtuais, tanto para a realização de experimentos quanto para oferecer serviços especializados, múltiplas máquinas virtuais com configurações semelhantes devem ser criadas e distribuídas em diferentes máquinas físicas. Um modo de realizar esta tarefa é a criação de servidores com o objetivo de criar e distribuir máquinas virtuais. Nesse sentido, este projeto apresenta um sistema capaz de prover máquinas virtuais sob demanda de forma a atender aos requisitos especificados por uma pilha de protocolos, ou seja, a partir de requisições da criação de novas redes virtuais, o servidor cria o número adequado de máquinas virtuais e as distribui em nós específicos da rede. Além disso, o servidor em questão pode assumir parte das tarefas administrativas da rede. O servidor foi implementado utilizando o conceito de serviços Web, tendo o SOAP como protocolo para requisição de serviços. Esta abordagem facilita a criação de clientes heterogêneos para o servidor de máquinas virtuais, além de reduzir a complexidade de adicionar novas funcionalidades, o que pode ser realizado através da adição de um novo serviço. Além disso, o Xen foi utilizado como ferramenta de suporte à

virtualização de computadores. Um protótipo de um sistema com o servidor de máquinas virtuais proposto neste trabalho foi criado e está em fase de expansão no laboratório do Grupo de Teleinformática e Automação.

Palavras-Chave: virtualização, Xen, SOAP, serviço Web.

Abstract

The Internet has two main characteristics: the end-to-end argument and the TCP/IP stack. Based on this model, the Internet grew significantly for many years. However, this model makes serious barriers to match requirements like mobility, management and security. Because of the pointed challenges, new architectures were proposed in the last few years. Computer virtualization has emerged in the literature as a powerful tool for the proposed architectures, either as new Internet infrastructure or as experimentation environment. To create virtual networks, either for experiments and to offer new specialized services, multiple virtual machines with similar settings must be created and distributed on physical machines. One way to perform this action is developing servers for creation and distribution of virtual machines. In this direction, this project presents a system with the capability of provide virtual routers on demand in order to match specific requirements of a protocol stack, i.e., from requests for new virtual networks creation, the server create the correct number of virtual machines and deploy them in specific nodes of the network. Moreover, this server can take part of the network administrative activities. The server was implemented using the Web services concept, and the protocol used for services requests was SOAP. This approach eases the creation of heterogeneous clients for the virtual machine server, besides decreases the complexity of adding new features. Moreover, Xen was used as tool for computer virtualization support. A system prototype was deployed at the Grupo de Teleinformática e Automação laboratory.

Key-words: virtualization, Xen, SOAP, Web service.

Lista de Acrônimos

ANA - *Autonomic Network Architecture*
BER - *Bit Error Rate*
CABO - *Concurrent Architectures are Better than One*
CIDR - *Classless Inter-Domain Routing*
CORBA - *Common Object Request Broker Architecture*
DHCP - *Dynamic Host Configuration Protocol*
DoS - *Denial of Service*
DNS - *Domain Name System*
DTN - *Delay/Disruption Tolerant Networks*
EPR - *EndPoint Reference*
IDP - *Information Dispatch Point*
IF - *Interstitial Function*
I/O - *Input/Output*
IDE - *Integrated Development Environment*
IP - *Internet Protocol*
IPv4 - *Internet Protocol version 4*
IPv6 - *Internet Protocol version 6*
ISP - *Internet Service Provider*
JMS - *Java Message Service*
JSP - *JavaServer Pages*
NAT - *Network Address Translation*
OM - *Object Model*
OMElement - *Object Model Element*
OMFactory - *Object Model Factory*

ORB - *Object Request Brokers*
RIP - *Routing Information Protocol*
RBA - *Role Based Architecture*
SOAP - *Simple Object Access Protocol*
SMTP - *Simple Mail Transfer Protocol*
SSH - *Secure Shell*
TCP - *Transmission Control Protocol*
URI - *Uniform Resource Identifier*
WSDL - *Web Services Description Language*
XML - *eXtensible Markup Language*

Sumário

Resumo	v
Abstract	vii
Lista de Acrônimos	viii
Lista de Figuras	xii
Lista de Tabelas	xiii
1 Introdução	1
1.1 A Internet Atual	1
1.2 Problemas com a Internet Atual	3
1.3 Novas Arquiteturas Propostas	4
1.3.1 Arquiteturas Puristas	5
1.3.2 Arquiteturas Pluralistas	6
1.4 Objetivos do Projeto	8
1.5 Organização do Texto	9
2 Conceitos Preliminares	10
2.1 Virtualização de Computadores	10
2.1.1 Principais Sistemas de Virtualização	13
2.2 Serviços Web	16
3 O Servidor de Máquinas Virtuais	20
3.1 Arquitetura	21
3.2 Implementação	23
3.2.1 Adição de Novos Serviços	26

3.2.2	Acesso ao Servidor de Máquinas Virtuais	29
3.2.3	Protótipo	31
4	Resultados e Observações	34
5	Conclusão	40
	Referências Bibliográficas	47

Lista de Figuras

1.1	Arquitetura do Projeto Horizon.	8
2.1	Princípio de máquinas virtuais.	11
2.2	Tipos de máquina virtual.	12
2.3	Arquitetura do Xen.	14
2.4	I/O <i>ring</i>	15
2.5	Serviço Web.	16
3.1	Rede de computadores de exemplo.	21
3.2	Arquitetura do servidor de máquinas virtuais proposto.	22
3.3	Evolução do sistema no tempo.	22
3.4	Topologia do <i>testbed</i>	32
4.1	Tela inicial do cliente.	34
4.2	Tela de resposta à requisição ao serviço <code>sanityTest</code>	35
4.3	Tela de resposta à requisição ao serviço <code>getPhysicalServerStatus</code>	36
4.4	Tela de resposta à requisição ao serviço <code>createVM</code>	37
4.5	Tela de resposta à requisição ao serviço <code>getServerStatus</code> após criação de máquina virtual.	38
4.6	Tela de resposta à requisição ao serviço <code>getVirtualMachineStatus</code>	38
4.7	Tela de resposta à requisição ao serviço <code>registerNodes</code>	39

Lista de Tabelas

3.1	Serviços oferecidos pelo servidor de máquinas virtuais.	24
3.2	Computadores e suas funções no protótipo.	32

Capítulo 1

Introdução

O crescimento da Internet deve-se a decisões arquiteturais tomadas durante o seu desenvolvimento inicial e a modificações realizadas ao longo dos anos como, por exemplo, a escolha de comutação de pacotes em detrimento da comutação de circuitos ou a decisão de manter o núcleo simples, deixando tarefas mais complexas para serem realizadas pelas estações finais. Entretanto, algumas aplicações que têm sido propostas sobre a Internet apresentam requisitos diferentes e muitas vezes conflitantes [1]. Devido a estas alterações ocorridas, alguns trabalhos têm apontado a necessidade de uma grande modificação na Internet [2, 3, 4]. Acredita-se que pequenas mudanças podem não ser suficientes para manter a Internet em funcionamento eficiente em alguns anos.

1.1 A Internet Atual

A Internet, como é conhecida hoje, tem como origem a ARPANet (*Advanced Research Projects Agency Network*) [5], rede desenvolvida a pedido do Departamento de Defesa dos Estados Unidos. Dentre as características dessa rede, uma bastante importante é que ela deveria ser resistente a ataques, ou seja, mesmo que parte da rede fosse comprometida, o restante deveria continuar funcionando da forma esperada. Esse requisito norteou boa parte das decisões de projeto tomadas durante o desenvolvimento da ARPANet, por exemplo, a sua organização em malha ao invés de níveis hierárquicos como nas redes telefônicas de então.

O modelo adotado para a Internet foi o modelo em camadas, com o obje-

tivo de reduzir a complexidade dos protocolos. A pilha de protocolos utilizada é conhecida como Pilha TCP/IP. Esse nome deve-se ao protocolo da camada de rede (IP – *Internet Protocol*) e a um dos protocolos da camada de transporte (TCP – *Transmission Control Protocol*).

Uma característica importante para a Internet, em seu início, era o suporte à heterogeneidade dos nós e dos serviços. Dessa forma, o núcleo da rede foi mantido simples cabendo aos nós das extremidades as tarefas mais complexas. Essa característica, conhecida como princípio fim-a-fim, permitiu o grande crescimento ocorrido na Internet, já que o núcleo da rede não precisa ser alterado para a implantação de novas aplicações. Entretanto, essa mesma característica tem também causado insatisfação por parte de alguns usuários, já que o núcleo, devido à sua simplicidade, não fornece informações suficientes a respeito das causas das falhas.

Optou-se na Internet pela utilização da comutação de pacotes, em detrimento da comutação de circuitos. Essa característica provê maior robustez à rede, além de eficiência, a partir do compartilhamento da banda [6]. Por outro lado, essa decisão torna mais complexa a tarefa de oferecer garantias mínimas de recursos ao longo da rede para uma determinada aplicação ou usuário, uma vez que os recursos são compartilhados pelas aplicações.

Vale notar que algumas características da Internet foram alteradas durante os anos. Com o crescimento do número de nós, as sub-redes, os sistemas autônomos, o CIDR (*Classless Inter-Domain Routing*) e o DNS (*Domain Name Service*) foram criados com o objetivo de prover escalabilidade à rede [7, 8]. Pode-se citar também a adição de algoritmos para controle de congestionamento no TCP [9] ou ainda a criação do IP *Multicast* [10], com o objetivo de permitir a comunicação de grupos.

Com o surgimento e o crescimento no número de dispositivos móveis um novo desafio surgiu. A forma hierárquica com que os endereços IPv4 foram distribuídos assume que cada endereço IP está associado a um ponto geográfico. Dessa forma se um nó altera seu ponto de ligação, seu endereço IP também é alterado. Nesse contexto, o IP móvel [11] foi criado com o objetivo de dar suporte à alteração da localização de um nó sem que seu endereço IP precise ser alterado, utilizando para isso um esquema de triangulação.

Ainda no contexto das alterações sofridas pela Internet, o Int-serv [12] e o

Diff-serv [13] foram criados com o objetivo de prover qualidade de serviço, ou seja, garantia de recursos mínimos de rede.

1.2 Problemas com a Internet Atual

A alteração de vários pontos da Internet demonstra que os requisitos dos clientes mudaram ao longo dos anos e que essa rede não os atende mais de forma completa. Alguns trabalhos [14, 9] têm apontado, durante os últimos anos, a necessidade de uma alteração mais profunda na Internet, a partir de requisitos diferentes.

O endereçamento IP é a causa de alguns dos principais problemas relacionados à Internet atual. O espaço de endereçamento no IPv4 é limitado pelo tamanho dos endereços, 32 bits. Logo, o número de estações está limitado a aproximadamente 4 bilhões. Atualmente, 92% dos endereços já foram utilizados e estima-se que até setembro de 2011 todos os endereços já tenham sido alocados [15]. O IPv6 possui um espaço de endereçamento de 128 bits, o que solucionaria o problema. Entretanto, a não interoperabilidade entre o IPv4 e o IPv6, além do receio dos administradores dos sistemas autônomos, têm atrasado a implantação do IPv6 por mais de 10 anos [14]. Pode-se ainda citar a criação do NAT (*Network Address Translation*) e a utilização do mecanismo de configuração automática de endereços, DHCP (*Dynamic Host Configuration Protocol*), para aliviar o problema de escassez de endereços IP.

Outro problema relacionado ao endereçamento IP é causado pela semântica sobrecarregada utilizada no IP. Ou seja, um endereço IP serve como identificador e como localizador do nó [16], criando uma séria restrição à mobilidade dos nós. Mesmo a utilização de mecanismos como o IP móvel, não garante a conectividade de nós com alta mobilidade.

Outro problema fundamental é a garantia de requisitos mínimos de segurança. Devido à adoção de um núcleo simples, a tarefa de adicionar segurança foi entregue às camadas superiores. Atualmente, não se pode negar a importância dessa questão devido aos inúmeros ataques de negação de serviço (DoS – *Denial of Service*) [17] e ao crescente número de *spams* na rede [18]. O ataque de negação de serviço consiste no consumo dos recursos de rede e de processamento da máquina atacada e pode ser causado pelo envio de um número excessivo de pacotes pelo atacante. Esse

ataque é possível, pois a rede não filtra os pacotes enviados com base no endereço de origem. Além disso, nenhum mecanismo de autenticação de fonte é utilizado, o que dificulta a descoberta do responsável pelo ataque. O envio de *spams* além de provocar insatisfação dos usuários, devido aos múltiplos e-mails indesejados, tem um efeito mais grave. Mesmo nos casos em que filtros são utilizados pelos clientes evitando que suas caixas de e-mail sejam sobrecarregadas, os recursos ao longo da rede já foram consumidos.

Vale citar também dois tipos de rede que vêm ganhando destaque importante na literatura e que não apresentam bom desempenho quando utilizadas em conjunto com o TCP. As redes sem fio apresentam taxas de perda binárias (BER – *Bit Error Rate*) muito maiores do que as redes cabeadas. Este fato associado à premissa do mecanismo de controle de congestionamento do TCP de que falhas de transmissão devem-se a congestionamentos, levam a subutilização do canal sem fio. Ou seja, sempre que não há recebimento de reconhecimentos positivos para os pacotes, a taxa de transmissão é reduzida. Esse problema já foi abordado em muitos trabalhos na literatura [19, 20, 21], entretanto, uma solução consensual ainda não existe.

Outra classe de redes que não possui bom funcionamento quando em conjunto com a pilha de protocolos TCP/IP são as redes tolerantes a atrasos e desconexões (DTN – *Delay/Disruption Tolerant Networks*) [22]. Os protocolos da Internet assumem que sempre existe um caminho fim-a-fim entre a origem e o destino dos dados. No caso das DTNs isso nem sempre é verdade. Nessa classe de rede uma mensagem pode precisar ser armazenada por algum dos nós responsáveis pelo encaminhamento, até que um caminho, ou parte dele, esteja disponível. Vale notar que algumas redes sem fio, que normalmente não são associadas às DTNs, compartilham a característica de conexão intermitente. Entre elas podem-se citar as redes ad hoc móveis [23] e as redes veiculares [24, 25].

1.3 Novas Arquiteturas Propostas

Diante dos problemas enfrentados pela Internet, um número de projetos surgiu ao redor do mundo com o objetivo de lidar com os desafios encontrados [26, 27, 28, 29, 30]. Essa seção tem como objetivo apresentar as principais soluções propos-

tas. Em geral, essas soluções podem ser divididas em duas categorias: arquiteturas puristas e arquiteturas pluralistas. Na abordagem pluralista, múltiplas pilhas de protocolos são executadas de forma paralela para atender aos diferentes requisitos dos serviços oferecidos. Por outro lado, a abordagem purista assume uma única pilha de protocolos que deve possuir flexibilidade suficiente para atender aos requisitos atuais e futuros dos serviços que utilizam a Internet.

1.3.1 Arquiteturas Puristas

A abordagem purista prevê uma pilha de protocolos monolítica. Vale notar que essa pilha deve ser flexível o suficiente para atender às diferentes demandas das aplicações que utilizam a Internet como meio de comunicação, sendo que algumas dessas demandas podem ainda não ser conhecidas atualmente. A principal desvantagem do modelo purista é sua implantação. Em geral, arquiteturas seguindo essa abordagem têm como característica a não interoperabilidade com os protocolos anteriores, o que torna quase obrigatória sua substituição em grande parte da rede ao mesmo tempo.

1.3.1.1 Redes Ativas

O termo Redes Ativas [31] surgiu como uma contraposição às redes ditas passivas, ou seja, redes que não realizam processamento sobre o conteúdo dos pacotes durante o encaminhamento, entregando-os inalterados ao destino. Nas redes ativas o termo utilizado para o que em uma rede IP é chamado de pacote é cápsula, ou seja, uma cápsula é a unidade de transmissão de dados numa rede ativa. Em cada cápsula em uma rede ativa, um trecho de programa, e em alguns casos dados, é adicionado. Os roteadores podem examinar e executar o código contido em uma cápsula podendo realizar alterações na cápsula.

1.3.1.2 Arquitetura Baseada em Papéis

A arquitetura baseada em papéis (RBA – *Role Based Architecture*) surgiu no contexto do projeto NewArch [9]. Nesse modelo não existe a multiplicidade de camadas evitando, de forma inerente, o problema comum nos protocolos de Internet de violação de camadas. Para substituir as camadas, módulos que os autores

chamam de papéis são utilizados para permitir a modularização do desenvolvimento dos protocolos. O diferencial importante quando comparado ao modelo em camadas é a inexistência de níveis hierárquicos entre os papéis. É importante notar que os papéis devem ser blocos bem conhecidos e padronizados, permitindo assim a criação de serviços bem definidos.

1.3.2 Arquiteturas Pluralistas

Diferente da abordagem purista, a abordagem pluralista permite a coexistência de diferentes pilhas de protocolo. Essa abordagem é interessante, pois permite que novos protocolos sejam testados em um ambiente real sem que a rede de produção seja alterada. Além disso, a migração pode ocorrer de forma gradual, o que é uma característica extremamente desejável no caso de adição de novas tecnologias. A literatura possui casos de técnicas e protocolos como, por exemplo, o IPv6, que não têm sido adotados devido à dificuldade de implantação de mecanismos disruptivos. A seguir, as principais propostas pluralistas para a alteração da Internet são apresentadas.

1.3.2.1 Plutarch

A proposta do Plutarch [32] é permitir a coexistência de múltiplos ambientes heterogêneos através da divisão da rede em contextos homogêneos. Para permitir a comunicação entre contextos são utilizadas funções de mapeamento entre as características dos dois contextos. Cada função de mapeamento, chamada de IF (*Interstitial Function*), deve possuir informações a respeito dos dois contextos envolvidos, além de funções que permitam a tradução entre os dados inerentes aos contextos, como endereçamento, roteamento e transporte.

A motivação por trás do Plutarch é a interconexão de redes com características distintas. Por exemplo, dispositivos com restrição de energia podem não implementar toda a pilha TCP/IP, fazendo necessária a existência de um mecanismo de tradução entre as redes heterogêneas.

É importante ressaltar que do ponto de vista de implantação do Plutarch num ambiente de produção não há complexidade significativa. Por exemplo, o IPv4 seria considerado um dos contextos sendo executados no ambiente. Internamente, o IPv4

não precisaria sofrer qualquer alteração, fazendo com que os usuários não fossem afetados. Além disso, a comunicação com outras tecnologias como, por exemplo, o IPv6 passaria a utilizar uma interface padronizada.

1.3.2.2 *Autonomic Network Architecture – ANA*

A principal contribuição do Projeto ANA [26] é a utilização de uma arquitetura autônoma. As pilhas de protocolos podem evoluir em tempo real para atender a novas demandas através de mecanismos de indireção nas camadas hierárquicas [33]. Os pontos de comunicação pivôs da proposta de Keller *et al.* são os IDPs (*Information Dispatch Points*), que tornam possível o conceito de modularidade em todos os níveis da arquitetura. Em geral, IDPs são ligados a blocos funcionais (FB – *Functional Blocks*), unidades de processamento que executam funções de transmissão de dados ou alguma funcionalidade adicional. Os IDPs são utilizados para realizar a comunicação entre os diversos blocos funcionais tornando possível a criação de sistemas complexos de comunicação a partir de módulos simples. Além disso, a construção dos sistemas não está restrita a uma pilha de protocolos rígida.

1.3.2.3 *Concurrent Architectures are Better than One – CABO*

A arquitetura CABO [34] propõe utilizar máquinas virtuais de modo que em cada rede virtual uma configuração ou até mesmo uma pilha de protocolos distinta seja utilizada. Dessa forma, múltiplas redes, com diferentes características estão disponíveis em um mesmo substrato físico. A principal motivação por trás da arquitetura CABO é permitir que os ISPs (*Internet Service Providers*) ofereçam serviços diferenciados aos seus clientes, o que atualmente não é possível já que nenhum provedor de serviço possui roteadores em todo o percurso fim-a-fim entre todos os usuários da Internet. Nessa arquitetura, cada ISP pode contratar serviços de outros ISPs garantindo um caminho fim-a-fim para seus clientes.

1.3.2.4 *Horizon*

O projeto franco-brasileiro Horizon [27] tem por objetivo desenvolver uma arquitetura para a Internet baseada nos conceitos do pluralismo e de inteligência. Nesse projeto, para cada pilha de protocolos existe um conjunto de máquinas virtuais

instanciadas ao longo da rede executando essa pilha. Além disso, para garantir um desempenho mínimo para as redes propõe-se a criação de um plano de pilotagem (ver Figura 1.1). Esse plano de pilotagem tem por objetivo sensoriar e atuar na rede, ou seja, através de observações colhidas por um conjunto diverso de equipamentos de medidas, o plano de pilotagem deve decidir se alguma mudança de configuração deve ser realizada na rede. Em caso positivo, o plano de pilotagem deve acionar os procedimentos de software necessários para que a tarefa seja realizada.

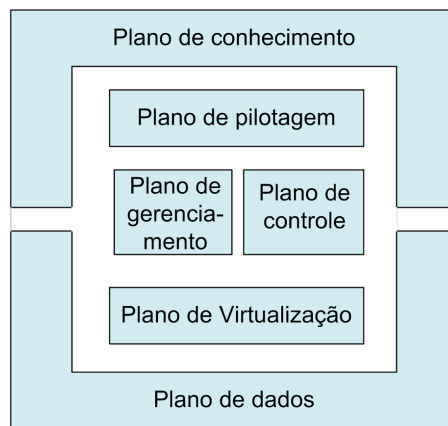


Figura 1.1: Arquitetura do Projeto Horizon.

O plano de pilotagem é responsável por coordenar os outros planos como, por exemplo, através da alteração de parâmetros de configuração de um protocolo de roteamento.

Vale notar que esse trabalho encontra-se no escopo do projeto Horizon, embora, sob o ponto de vista funcional, pudesse ser utilizado por outras propostas, como por exemplo, o CABO.

1.4 Objetivos do Projeto

Este projeto tem como objetivo o desenvolvimento de um sistema capaz de prover roteadores virtuais sob demanda de modo a atender aos requisitos especificados por uma determinada pilha de protocolos, ou seja, a partir de requisições de criação de novas redes virtuais, o servidor deve criar o número adequado de máquinas virtuais e distribuí-las em nós específicos da rede. Além disso, esse sistema pode acumular a execução de algumas das atividades de administração das redes como,

por exemplo, a migração de uma máquina virtual entre máquinas físicas ou ainda a alteração dos recursos de hardware dedicados a um domínio virtual.

1.5 Organização do Texto

Este trabalho está organizado da seguinte forma. No Capítulo 2 são apresentados os conceitos preliminares necessários para o correto entendimento deste trabalho. O Capítulo 3 apresenta o servidor de máquinas virtuais, tanto do ponto de vista conceitual, quanto em sua implementação em estado de protótipo. O Capítulo 4 apresenta os resultados referentes a este trabalho e algumas observações. Finalmente, o Capítulo 5 conclui este trabalho.

Capítulo 2

Conceitos Preliminares

Este capítulo apresenta os conceitos básicos necessários ao entendimento das decisões de projeto tomadas durante o desenvolvimento do servidor de máquinas virtuais e de seu funcionamento.

2.1 Virtualização de Computadores

Como visto na Seção 1.3, a abordagem pluralista introduz propostas para lidar com a multiplicidade de requisitos para a Internet. Entre elas está a utilização de múltiplas redes virtuais em paralelo. Uma ferramenta muito importante nesse sentido é a virtualização de computadores [35, 36, 37]. Mesmo no caso da abordagem purista, a virtualização pode ter um papel importante como plataforma de testes dos novos protocolos sem interrupção da rede de produção. Como um exemplo, pode-se citar o PlanetLab [38], um conjunto de computadores com suporte a virtualização, que atualmente conta com 982 nós distribuídos em 484 locais. Esses computadores fornecem ambientes virtualizados ao longo da rede para a realização de experimentos. Esta seção tem por objetivo apresentar os principais conceitos relativos à virtualização de computadores e as ferramentas relacionadas disponíveis atualmente.

Nos últimos anos, a virtualização de computadores tem ganhado destaque na computação devido, em grande parte, ao aumento do poder computacional disponível. Vale notar que técnicas de virtualização não são uma novidade e remontam à década de 70. Atualmente, a área na qual a virtualização de computadores tem

ganhado destaque é conhecida como consolidação de servidores. Uma situação comum hoje, em muitas organizações, é a existência de um servidor para cada tipo de serviço (e-mail, Web, arquivos etc.). Em geral isso ocorre porque para cada um dos serviços podem ser necessárias diferentes configurações de sistema operacional e em alguns casos são necessários até mesmo sistemas operacionais diferentes. O resultado dessa separação é que alguns servidores passam grande parte do tempo ociosos. Dessa forma, a virtualização de computadores pode ser utilizada para agregar, em um único servidor físico, diversos serviços, cada um utilizando uma máquina virtual.

Quando um sistema computacional é virtualizado, a camada responsável por prover a virtualização oferece interfaces similares de parte dos recursos físicos para serem utilizados pelos sistemas hóspedes (sistemas virtualizados). A Figura 2.1 apresenta um exemplo de um sistema computacional que utiliza virtualização. A camada de virtualização imediatamente acima do hardware oferece uma abstração deste para as máquinas virtuais A e B. Nota-se que uma máquina virtual pode conter uma nova camada de virtualização.

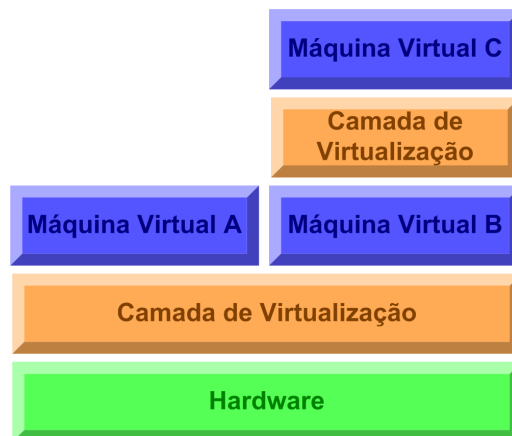


Figura 2.1: Princípio de máquinas virtuais.

A camada de virtualização pode ser feita de duas formas diferentes. Na abordagem que utiliza uma máquina virtual de processo ou de aplicação (Figura 2.2(a)), o sistema operacional que executa sobre o hardware físico não precisa sofrer qualquer alteração, ou seja, pode-se utilizar um sistema operacional de prateleira. A camada de virtualização é vista pelo sistema operacional como um processo comum. A outra abordagem, conhecida como monitor de máquina virtual ou hipervisor (Figura 2.2(b)) utiliza um sistema operacional alterado, que funciona como a camada

de virtualização e todos os sistemas operacionais funcionam a partir dos recursos oferecidos por essa camada. A última abordagem apresenta, em geral, melhor desempenho do que a primeira. Entretanto, em geral, exige-se que o todo o sistema operacional da máquina a ser virtualizada seja reinstalado, o que em muitos casos, pode não ser desejável.

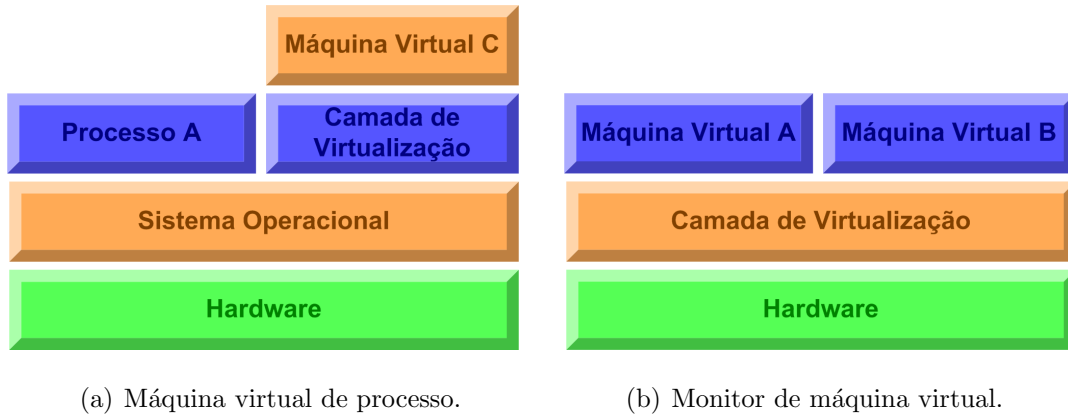


Figura 2.2: Tipos de máquina virtual.

A implementação de monitores de máquinas virtuais pode seguir duas estratégias gerais: virtualização total (ou completa) e paravirtualização.

Na virtualização total as interfaces de hardware oferecidas ao sistema operacional são iguais às oferecidas pelo hardware físico. Dessa forma, o sistema operacional virtualizado pode ser executado sem qualquer alteração. Essa abordagem tem a grande vantagem de se adaptar a qualquer alteração realizada nos sistemas operacionais. Por outro lado, devido ao grande número de dispositivos diferentes desenvolvidos por múltiplos fabricantes, torna-se muito complexo oferecer compatibilidade para todos eles. Em geral, os softwares que oferecem virtualização total apresentam dispositivos genéricos para os sistemas virtualizados. Por exemplo, um mouse padrão PS/2 é exibido ao invés da interface específica. Essa abordagem reduz a complexidade do software de virtualização, porém pode impedir a utilização de determinados recursos de hardware. Outro prejuízo dessa abordagem é que toda instrução realizada pelo sistema hóspede deve ser verificada pelo software de virtualização em busca de instruções sensíveis, o que leva a uma redução de desempenho.

A paravirtualização por sua vez exhibe uma interface alterada, exigindo que o sistema operacional hóspede seja alterado. Ou seja, o sistema hóspede precisa ter

conhecimento da camada de virtualização. Além disso, o software de virtualização não precisa verificar as instruções realizadas pelo sistema hóspede, visto que este realizará uma hiperchamada (*hypercall*) sempre que uma instrução privilegiada precisar ser realizada. Essa abordagem reduz significativamente a complexidade do software de virtualização, permitindo um aumento de desempenho. O custo para o aumento de desempenho é a diminuição da oferta de sistemas operacionais disponíveis para a plataforma. Nesse ambiente, o desenvolvedor do sistema operacional, ou alguém com acesso ao seu código, deve realizar alterações neste sistema para que ele se adapte ao software de virtualização.

2.1.1 Principais Sistemas de Virtualização

O VMware [39] é atualmente um dos principais softwares para virtualização disponíveis. Os produtos oferecidos cobrem boa parte da infra-estrutura de virtualização tanto de *desktops* quanto de *data centers*. Os produtos que oferecem virtualização através de um monitor de máquina virtual utilizam virtualização completa. Um ponto negativo dessa plataforma é o fato de o código ser fechado, não permitindo alterações em seu funcionamento.

O Xen [40], por sua vez, é um monitor de máquina virtual de código aberto. Os pontos essenciais dessa plataforma de virtualização são os domínios e o hipervisor. Existem dois tipos de domínio: domínio privilegiado ou domínio 0 e domínios não-privilegiados ou domínios U. O hipervisor é responsável pelos recursos de memória e de processamento e não possui *drivers* para acesso aos dispositivos. Os domínios virtuais são controlados pelo hipervisor. Ele é responsável por permitir ou não acesso aos recursos de memória e CPU, além de realizar o escalonamento dos domínios virtuais. O acesso aos dispositivos de entrada e saída é realizado pelo domínio 0, que é o domínio criado durante a inicialização de uma plataforma Xen. Através desse domínio a interface de controle do sistema xen pode ser acessada, ou seja, a administração de um sistema Xen é realizada pelo domínio 0. Em geral, o domínio 0 é responsável por realizar o acesso aos *drivers* dos dispositivos. Os sistemas hóspedes são os domínios U. A arquitetura simplificada de um sistema Xen está ilustrada na Figura 2.3.

Inicialmente, as plataformas Xen suportavam somente o paradigma paravir-

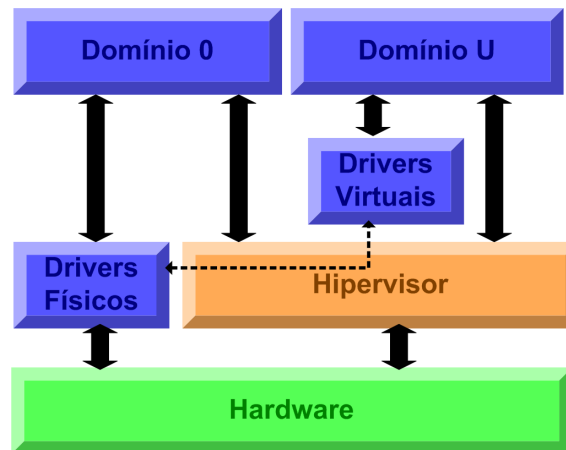


Figura 2.3: Arquitetura do Xen.

tualizado. Essa abordagem foi utilizada devido à complexidade de virtualização das plataformas de hardware existentes. Um problema importante, encontrado na plataforma X86 é a existência de instruções sensíveis que não são privilegiadas. Uma instrução é dita sensível sempre que altera a configuração de recursos do sistema ou quando seu resultado ou comportamento depende da configuração dos recursos [41]. Uma instrução privilegiada é aquela que gera uma exceção sempre que é executada por um processo que não possui privilégios suficientes. Na plataforma X86, algumas instruções que alteram registros importantes não são privilegiadas. Dessa forma, os sistemas operacionais hóspedes tentariam executar essas instruções causando inconsistências no sistema.

Atualmente, os fabricantes de processadores têm adicionado suporte a virtualização em sua arquitetura [42, 43, 44, 45], diminuindo significativamente a complexidade de um sistema com virtualização total. Nesse contexto, os sistemas Xen mais recentes permitem a utilização da virtualização total, desde que exista suporte a virtualização em nível de hardware.

O Xen realiza procedimentos de entrada e saída (*I/O - Input/Output*) através de uma região de memória compartilhada chamada de *I/O ring*. Todas as transferências de dados entre os domínios U e o domínio 0 são realizadas através dessa região de memória. Na Figura 2.3, pode-se observar a utilização de *drivers* virtuais para a comunicação dos domínios virtualizados. A Figura 2.4 representa um *I/O ring*. Sempre que o sistema hóspede deseja receber algum dado, a referência apon-

tada pelo produtor de requisições deve ser atualizada com a localização do dado requerido e o apontador deve avançar no anel. De forma assíncrona, o domínio 0 avançará o consumidor de requisições atendendo aos pedidos do domínio hóspede. Ao atender um hóspede, o domínio 0 deve atualizar a referência apontada pelo produtor de respostas e avançar este ponteiro. Também de forma assíncrona, o domínio hóspede deve obter os dados apontados pelo consumidor de respostas e atualizar seu valor.

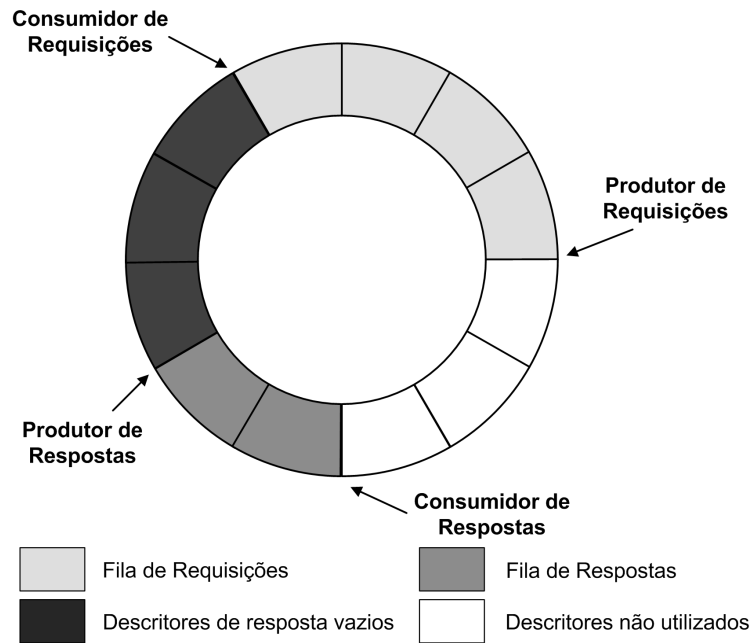


Figura 2.4: I/O ring.

Caso uma grande demanda seja requerida do domínio 0, por exemplo, múltiplas placas de rede operando a altas taxas, este pode tornar-se um gargalo de desempenho para o sistema. Nesses casos é possível criar um novo domínio chamado de domínio de *driver*. Um domínio de *driver* é um domínio virtual que possui acesso direto e exclusivo a um ou mais dispositivos físicos. Em essência, o domínio 0 pode ser chamado de domínio de *driver* já que este é um domínio responsável pela comunicação com dispositivos físicos através de seus *drivers* reais. Entretanto, em alguns casos pode ser positivo utilizar um domínio de *driver* específico para alguns dispositivos. Esta estratégia reduz a sobrecarga sobre o domínio 0, permitindo ao sistema maior desempenho.

Neste projeto, o sistema de virtualização adotado foi o Xen. Esse sistema

possui um grande número de trabalhos relacionados na literatura [40, 46, 47, 48]. Além disso, o sistema é feito em código aberto, o que permite a análise e extensão do sistema de acordo com as demandas.

2.2 Serviços Web

Os Serviços Web [49] são programas que podem ser acessados e executados via Web. Um serviço Web é uma aplicação modular autocontida e autodescrita. Existe um conjunto de ferramentas com o objetivo de publicar, localizar e invocar esses serviços na Web. Além disso, os serviços provêm um método padronizado de integrar sistemas de software executados em plataformas heterogêneas. Vale notar que essa técnica é caracterizada por ser altamente extensível. Outra forma de entender um serviço Web é assumir que um conjunto de serviços forma uma interface de programação de aplicativo (API - *Application Programming Interface*) que pode ser acessada via Web.

O tipo mais comum de serviço Web funciona a partir da troca de mensagens XML (*eXtensible Markup Language*) utilizando o protocolo SOAP (*Simple Object Access Protocol*) [50]. Logo, as requisições de serviços e suas respostas são descritas em mensagens XML, que são trocadas entre clientes, os que requisitam o serviço, e servidores, os que executam os serviços. Com o objetivo de facilitar a descoberta de serviços é comum a utilização de mensagens contendo descrições de serviços Web. O padrão utilizado para isso é o WSDL (*Web Services Description Language*) [51]. A Figura 2.5 apresenta uma visão conceitual de um serviço Web simples.

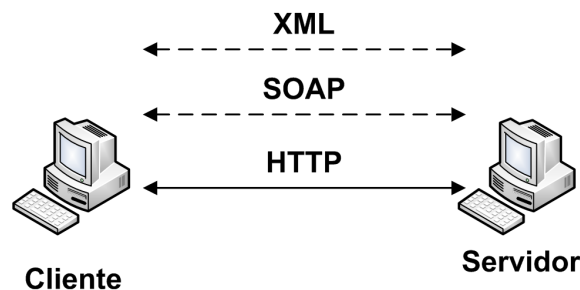


Figura 2.5: Serviço Web.

O SOAP é um protocolo para troca de informações estruturadas desenvolvido

dentro do contexto de serviços Web. Através da utilização do SOAP, serviços Web autodescritos e de fácil descoberta podem ser invocados. O SOAP utiliza XML como padrão para estrutura das mensagens e algum protocolo de transmissão da camada de aplicação (em geral HTTP - *HyperText Transfer Protocol*) para a transmissão das mensagens.

Uma mensagem SOAP contém dentro de seu elemento raiz, chamado de `env:Envelope`, dois elementos `env:header` e `env:body`. O conteúdo desses elementos é dependente da aplicação. Entretanto, as especificações do SOAP definem que o elemento `env:header` é opcional e deve ser utilizado como forma de transmissão de informações que não fazem parte dos dados da aplicação como, por exemplo, dados que permitam identificar a organização que fornece o serviço. As informações passadas dentro deste elemento devem ser diretivas ou informações contextuais para a aplicação.

O elemento `env:body` é obrigatório e os dados da aplicação devem ser passados dentro desse elemento. Logo, no caso do servidor de máquinas virtuais as mensagens identificando o serviço requisitado e os parâmetros necessários estarão contidos dentro deste elemento. A Listagem 2.1 apresenta a estrutura de uma mensagem SOAP.

Listagem 2.1: Mensagem SOAP.

```
<env:envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <env:header>
    <!-- Content of header goes here >
  </env:header>
  <env:body>
    <!-- Content of body goes here >
  </env:body>
</env:envelope>
```

O WSDL por sua vez é definido como um formato XML para a descrição de serviços de rede como um conjunto de estações finais operando em mensagens que contêm informações orientadas a documentos ou orientadas a procedimento. Logo, um conjunto de serviços Web oferecido por uma entidade pode ser publicado através de um arquivo WSDL. A Listagem 2.2 apresenta a estrutura de um arquivo WSDL.

Listagem 2.2: Estrutura de um arquivo WSDL.

```
<definitions >
  <types >
    definition of types .....
  </types >
  <message >
    definition of a message ....
  </message >
  <portType >
    definition of a port .....
  </portType >
  <binding >
    definition of a binding ....
  </binding >
</definitions >
```

O elemento `types` define os tipos de dados utilizados pelo serviço Web. O elemento `message` define os elementos de dados de uma operação. Cada mensagem pode conter uma ou mais partes. Essas partes podem ser vistas como parâmetros de uma chamada de procedimento em uma linguagem de programação tradicional. O elemento `portTypes` descreve o serviço Web, as operações que podem ser realizadas e as mensagens envolvidas. Em comparação com uma linguagem de programação orientada a objetos, o elemento `portTypes` equivale a uma classe. Finalmente, o elemento `binding` define o formato de mensagem e os detalhes de protocolo para cada porta. A Listagem 2.3 apresenta o arquivo WSDL de um serviço Web de exemplo.

A utilização de um serviço Web pode em muitos casos aumentar o tempo de resposta, já que as mensagens são enviadas em forma de texto ao invés de serem codificadas. A utilização de outra plataforma como o CORBA (*Common Object Request Broker Architecture*) [52] poderia prover menor tempo de resposta. Entretanto, o CORBA obriga que todas as partes do sistema distribuído suportem todas as bibliotecas ORB (*Object Request Brokers*), o que exige que todo cliente CORBA consuma mais recursos, como memória, processamento etc. [53]. No Projeto Horizon propõe-se que o plano de pilotagem seja implementado com agentes inteligentes.

Listagem 2.3: Exemplo de arquivo WSDL.

```
<definitions >
  <message name="getTermRequest">
    <part name="term" type="xs:string"/>
  </message>
  <message name="getTermResponse">
    <part name="value" type="xs:string"/>
  </message>
  <portType name="glossaryTerms">
    <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
    </operation>
  </portType>
  <binding type="glossaryTerms" name="b1">
    <soap:binding style="document" transport="http://schemas.xmlsoap.
      org/soap/http" />
    <operation>
      <soap:operation soapAction="http://example.com/getTerm"/>
      <input<×soap:body use="literal"/></input>
      <output<×soap:body use="literal"/></output>
    </operation>
  </binding>
</definitions
```

Esses agentes devem ser simples, ou seja, devem requerer o mínimo possível de recursos. Nesse cenário, a utilização de serviços Web é desejável, já que cada cliente só precisa dar suporte a um pequeno número de serviços.

Outras razões para a escolha de serviços Web em detrimento da tecnologia CORBA incluem a facilidade de desenvolvimento de clientes Web para o caso de serviços Web; a probabilidade de não ter suas mensagens filtradas por um *firewall*, já que um serviço Web em geral utiliza HTTP, um protocolo que normalmente é aceito pelos firewalls; e a possibilidade de criação de clientes móveis, que no caso do CORBA seria uma tarefa complexa [53].

Capítulo 3

O Servidor de Máquinas Virtuais

O servidor de máquinas virtuais definido no projeto Horizon consiste em um conjunto de serviços Web oferecidos. Além das tarefas de criação de máquinas virtuais e de redes virtuais, inerentes ao servidor, um conjunto de serviços extras foi adicionado ao servidor tornando-o um controlador de redes virtuais. Dessa forma, esse servidor pode ser utilizado como ponto de partida para a realização das tarefas definidas pelo plano de pilotagem. Por exemplo, caso o plano de pilotagem perceba um gargalo de desempenho de uma rede virtual em um determinado nó, uma requisição para o aumento de recursos (memória, CPU etc.) pode ser feita através de um serviço provido pelo servidor.

Considere a rede física apresentada na Figura 3.1(a). Nesse exemplo são exibidos os roteadores e os enlaces físicos que os conectam. De acordo com a proposta deste projeto, cada um dos roteadores é equipado com alguma tecnologia, como o Xen ou VMWare, que os torna capazes de hospedar sistemas virtualizados.

A Figura 3.1(b) apresenta a mesma rede com um conjunto de roteadores virtuais. Note que as linhas pontilhadas representam enlaces virtuais criados para conectar os roteadores virtuais. A função básica do servidor de máquinas virtuais é, sob demanda, criar máquinas virtuais nos nós físicos da rede e configurar seus enlaces de forma que a rede esteja ativa após essa operação.

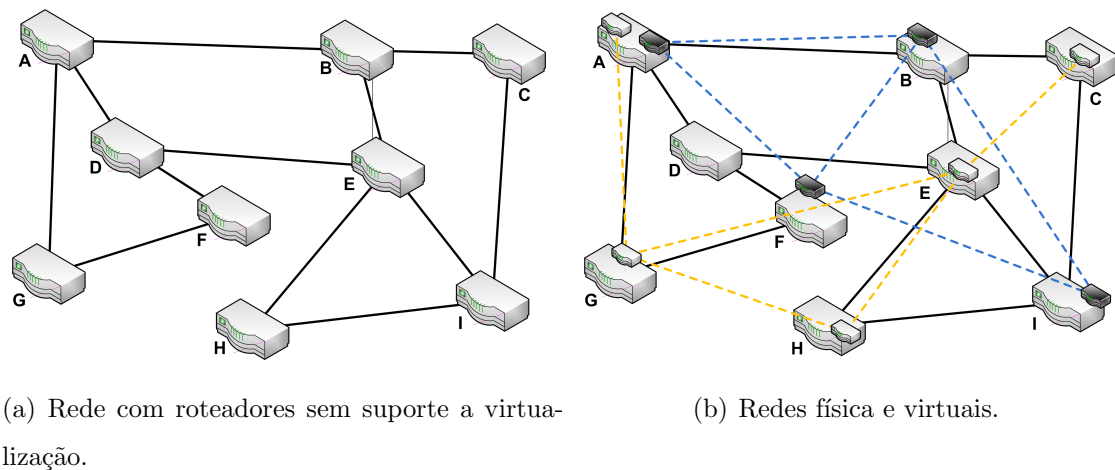


Figura 3.1: Rede de computadores de exemplo.

3.1 Arquitetura

A Figura 3.2 apresenta a arquitetura de um sistema virtualizado contendo o plano de pilotagem proposto no projeto Horizon e o servidor de máquinas virtuais, tema deste trabalho. Sempre que o plano de pilotagem requisitar um dos serviços providos pelo servidor de máquinas virtuais, uma mensagem deve ser enviada utilizando o protocolo SOAP sobre HTTP para o servidor. O servidor de máquinas virtuais, por sua vez, comunica-se com as máquinas físicas que hospedam as máquinas virtuais indicadas na mensagem SOAP e que serão alcançadas através de uma API de gerenciamento como, por exemplo, a Libvirt [54] ou a XenAPI [55]. É importante observar que para o plano de pilotagem a forma como o serviço vai ser realizado é totalmente transparente, permitindo que o servidor de máquinas virtuais seja atualizado, mudando a implementação de seus serviços, sem alterar a interface com seus clientes.

Na maioria dos serviços oferecidos o cliente deve esperar uma mensagem de retorno informando o resultado da operação. Em caso de falha, um relatório com os motivos da falha é enviado ainda por serviço Web ao cliente.

É importante notar que não existe qualquer restrição conceitual ao tipo de protocolo de comunicação instalado na máquina virtual. A limitação que pode existir é do ponto de vista da implementação. Um determinado protocolo pode não estar disponível para o sistema operacional desejado, por exemplo.

A Figura 3.3 mostra como o sistema evolui ao longo do tempo. Nesse exem-

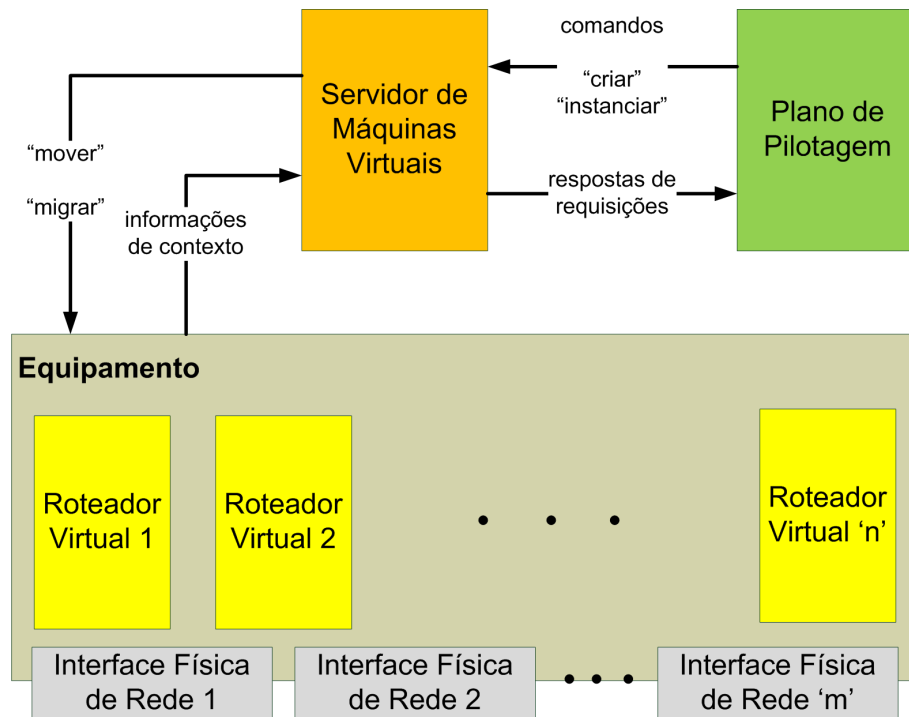


Figura 3.2: Arquitetura do servidor de máquinas virtuais proposto.

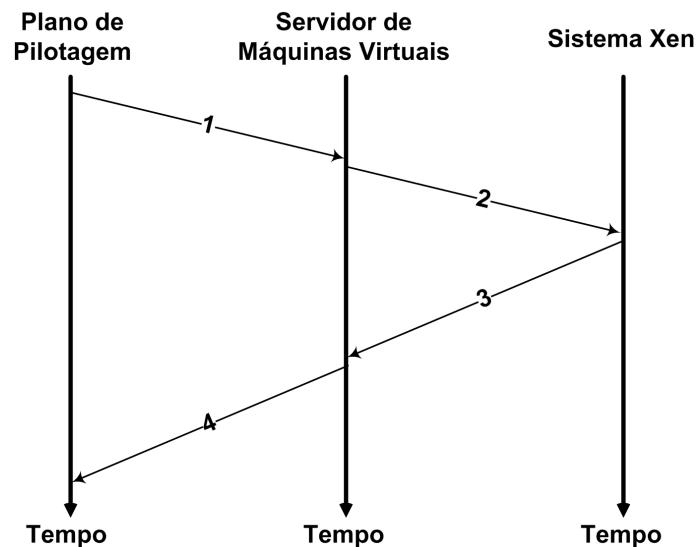


Figura 3.3: Evolução do sistema no tempo.

plo, assume-se que o serviço requisitado é o de criação de uma máquina virtual (`createVirtualMachine`). Na primeira interação do sistema (identificado por 1 na figura), o plano de pilotagem - o cliente - envia uma mensagem SOAP contendo uma requisição do serviço. Ao receber a mensagem, o servidor de máquinas virtuais identifica o sistema Xen indicado na mensagem e envia (passo 2), através de uma

biblioteca de gerenciamento de sistema virtualizados, uma requisição para a criação de uma máquina virtual com as características definidas na mensagem que o plano de pilotagem o enviou.

Em seguida, o sistema Xen tenta criar a máquina virtual solicitada e envia o resultado da operação ao servidor de máquinas virtuais (passo 3). O termo “criar máquina virtual” aqui utilizado refere-se ao processo de definição de um arquivo que será utilizado como disco virtual para a máquina hóspede e outros parâmetros, como tamanho da memória, número de interfaces de rede etc. O servidor por sua vez, cria uma mensagem XML com o resultado da operação e o envia ao plano de pilotagem (passo 4).

Um ponto importante a ser ressaltado é a localização do arquivo que será utilizado como disco virtual. Esse arquivo deve conter o sistema operacional desejado, com as funções requeridas pelo serviço. Por exemplo, caso a máquina virtual requisitada seja um roteador IPv4, o arquivo deve conter um sistema operacional Linux, com suporte a IPV4 e o protocolo de roteamento RIP (*Routing Information Protocol*), por exemplo. Esse arquivo pode ser armazenado em locais diferentes. Por exemplo, o servidor de máquinas virtuais pode armazenar diferentes imagens de sistemas operacionais e transferi-las para as estações físicas no momento em que máquinas virtuais forem criadas.

3.2 Implementação

Um protótipo do servidor de máquinas virtuais foi implementado e implantado dentro do laboratório do Grupo de Teleinformática e Automação com o objetivo de realizar análises de prova de conceito. Os serviços atualmente disponíveis estão descritos na Tabela 3.1.

O serviço de criação de máquinas virtuais (`createVirtualMachine`) utiliza os parâmetros passados pela requisição do serviço para criar um nó virtual no nó físico da rede especificado. O serviço de criação de redes virtuais (`createVirtualNetwork`) deve criar um conjunto de nós virtuais em estações físicas da rede. Além disso, para que os nós criados formem uma rede, deve-se realizar o mapeamento entre a interface física indicada e a interface virtual criada, além da configuração

Tabela 3.1: Serviços oferecidos pelo servidor de máquinas virtuais.

Serviço	Descrição
<code>createVirtualMachine</code>	serviço básico para a criação de máquinas virtuais
<code>createVirtualNetwork</code>	criação de redes virtuais
<code>destroyVirtualMachine</code>	exclusão de máquinas virtuais
<code>getPhysicalServerStatus</code>	informações básicas sobre um servidor físico
<code>getRegisteredNodes</code>	lista de nós registrados
<code>getVirtualMachineStatus</code>	informações básicas sobre determinado domínio virtual
<code>migrateVirtualMachine</code>	migração de máquinas virtuais
<code>registerNodes</code>	registro de novo nó físico na rede
<code>sanityTest</code>	teste de sanidade para observar funcionamento do servidor
<code>shutdownVirtualMachine</code>	desligar máquina virtual
<code>topologyDiscover</code>	descoberta de topologia

dos endereços de rede.

Além das tarefas inerentes ao servidor de máquinas virtuais, tarefas adicionais foram implementadas. O serviço `destroyVirtualMachine` pode ser utilizado para destruir uma máquina virtual, por exemplo, quando a rede para a qual ela foi criada não é mais necessária. Em alguns casos a máquina virtual deverá ser desligada para ser religada depois de um tempo. O serviço `shutdownVirtualMachine` pode ser utilizado com esse intuito. Os serviços `getPhysicalServerStatus` e `getVirtualMachineStatus` têm por objetivo obter algumas informações gerais, como memória e número de processadores, acerca de um nó físico e de uma máquina virtual, respectivamente. Essas informações podem ser utilizadas pelo plano de pilotagem no processo de tomada de decisões. O serviço de migração (`migrateVirtualMachine`) pode ser utilizado para mover uma máquina virtual de um nó físico para outro, por exemplo, quando um determinado nó físico encontra-se sobrecarregado.

Em algumas ocasiões pode ser importante para um administrador conhecer a topologia da rede. O serviço de descoberta de topologia (`topologyDiscover`) provê as topologias da rede física e das redes virtuais. Inicialmente, o servidor de máquinas virtuais não tem conhecimento sobre as máquinas físicas as quais possui

acesso. Para garantir que o servidor de máquinas virtuais tenha conhecimento desses nós, o serviço `registerNodes` permite que um determinado nó seja registrado no servidor para futura administração, ou seja, o nome, a chave pública e os endereços IP são enviados ao servidor que os armazena localmente. Os nós registrados podem ser consultados pelo serviço `getRegisteredNodes`.

O protótipo foi implementado utilizando a linguagem de programação Java [56]. A biblioteca Libvirt [54], versão 0.7.5 foi utilizada para a realização de tarefas administrativas e a biblioteca Axis2 [57] em sua versão 1.5.1 para a construção dos serviços Web. O servidor Web utilizado foi o Tomcat [58], versão 6. Completando o ambiente de desenvolvimento, a IDE (*Integrated Development Environment*) utilizada foi o NetBeans [59] versão 6.7.1. Adotou-se como plataforma de virtualização o Xen. Essa plataforma possui grande apoio da comunidade acadêmica, o que a torna mais confiável, já que um grande número de pesquisadores e usuários comuns tem utilizado essa tecnologia ao redor do mundo. Além disso, o Xen é distribuído sob uma licença de código livre, permitindo que o grupo de pesquisa envolvido com o projeto Horizon proponha alterações em seu funcionamento.

A linguagem de programação escolhida foi Java [56], uma linguagem orientada a objetos com suporte aos principais sistemas operacionais disponíveis. A comunidade de desenvolvedores Java produziu ao longo dos anos uma grande quantidade de bibliotecas publicamente disponíveis. Essa característica foi um dos fatores determinantes para a escolha dessa linguagem para o protótipo. Além disso, o amplo suporte a serviços Web em Java contaram pontos a favor nessa decisão. Finalmente, um programa Java é em princípio multiplataforma, ou seja, depois de compilado, o código Java pode ser executado em qualquer estação desde que esta possua uma máquina virtual Java.

A Libvirt [54] é uma biblioteca para gerenciamento de sistemas virtualizados de código livre. A biblioteca foi originalmente desenvolvida em C e atualmente provê suporte para um grande conjunto de linguagens, a destacar Java e Python. A Libvirt é uma ferramenta de gerenciamento genérica, ou seja, provê funções comuns à maioria dos sistemas de virtualização, por exemplo, Xen, VMware, OpenVZ, QEMU etc. Esta generalidade da Libvirt permite que mesmo que o sistema de virtualização utilizado no projeto Horizon seja alterado, a maior parte do código do

servidor de máquinas virtuais permaneça inalterado. Essa propriedade é altamente desejável para tornar o servidor de máquinas virtuais útil em cenários mais amplos do que seria se comparado ao uso de uma biblioteca de administração específica para sistemas Xen. Além disso, vale notar que a Libvirt é hoje uma das principais bibliotecas de gerenciamento existentes, com uma ampla comunidade de usuários e desenvolvedores.

A biblioteca Axis2 [57] desenvolvida pela Apache Foundation implementa o protocolo SOAP. É importante observar que essa biblioteca também é publicada sob uma licença de código livre. Atualmente o Axis2 está implementado em C e em Java, sendo a última a implementação utilizada neste protótipo.

A principal tarefa do servidor de máquinas virtuais é a criação de domínios virtuais ao longo da rede. Por simplicidade de implementação, no protótipo atual, uma máquina virtual com determinada pilha de protocolos e certo sistema operacional só pode ser criada se a imagem estiver disponível na máquina física. Esta imagem é compartilhada por todos os domínios virtuais com a mesma configuração que são executados num determinado nó físico. Num ambiente de produção essa abordagem poderia ser utilizada, dado que sempre que o nó hospedeiro não possua a imagem desejada, o servidor de máquinas virtuais pode enviar a imagem ao destino. Note que essa alteração é transparente para o usuário, já que afeta somente a implementação do serviço de criação de máquinas virtuais. A premissa de compartilhamento é válida para a maioria dos casos. Em princípio, o servidor proverá máquinas virtuais para o núcleo da rede, composta basicamente de roteadores. Nestes casos, uma imagem na qual o domínio virtual só possui direito de leitura é suficiente.

3.2.1 Adição de Novos Serviços

Os serviços implementados até o momento não cobrem todas as possibilidades de tarefas administrativas que podem ser necessárias. Por exemplo, pode ser necessária a coleta de estatísticas mais detalhadas a respeito de um determinado nó da rede ou a alteração de parâmetros do algoritmo de escalonamento de máquinas virtuais de um sistema de máquinas virtuais. Nesses casos, faz-se necessária a adição de novos serviços. Esta seção tem por objetivo detalhar o método de implantação

de novos serviços.

Cada serviço no servidor de máquinas virtuais é implementado como um método na classe principal do servidor de máquinas virtuais (`VirtualMachineServer`). A Listagem 3.1 apresenta a implementação do método de desligamento de máquina virtual (`shutdownVirtualMachine`). Todo serviço deve ser implementado como um método público que recebe um objeto da classe `OMElement` (*Object Model Element*) e retorna outro objeto da classe `OMElement`. Essa classe é oferecida pela biblioteca Axis2 e tem por objetivo armazenar um elemento XML, ou seja, depois de transformado em uma *string*, um objeto `OMElement` torna-se uma *tag* de uma mensagem XML. Neste caso, o elemento recebido como parâmetro é o conteúdo do elemento `body` de uma mensagem SOAP, e o `OMElement` retornado será também o conteúdo do elemento `body` da mensagem SOAP enviada como resposta pelo servidor de máquinas virtuais.

Ainda com relação à Listagem 3.1, na linha 2 um iterador é criado para a lista de filhos do `OMElement` passado como parâmetro. Na linha 3 um `OMElement` é criado para armazenamento temporário. Em seguida, na linha 4, duas *strings* são criadas. A primeira para armazenar o nome do servidor físico que hospeda a máquina virtual a ser desligada, e a segunda para armazenar o nome da máquina virtual.

Em seguida, na linha 5, um objeto da classe `OMFactory` (*Object Model Factory*) é criado. Na linha 6, um *namespace* é criado a partir de uma URI (*Uniform Resource Identifier*) e de um prefixo previamente definidos. Na linha 7, um objeto da classe `OMElement` é criado. Este será o objeto retornado pelo método. Na linha 8 é iniciado um ciclo que realiza iterações pelos filhos do `OMElement` passado como parâmetro para o método utilizando o iterador anteriormente criado. Para cada um dos elementos da iteração, verifica-se o nome do elemento. O serviço em questão espera dois filhos. Um indicando o nome do servidor físico e outro que indica o nome da máquina virtual. Nas linhas 11 e 13, os nomes do servidor físico e da máquina virtual são ajustados, respectivamente.

Na linha 16, uma conexão é iniciada com o servidor físico em questão utilizando a Libvirt. Na linha 17, um objeto indicando a máquina virtual desejada é criado utilizando a classe `Domain`, provida pela Libvirt. Na linha 18, o comando

Listagem 3.1: Exemplo de método da classe `VirtualMachineServer`.

```

1 public OMElement shutdownVirtualMachine(OMElement element){
2     Iterator it = element.getChildElements();
3     OMElement ele = null;
4     String phyServer = null, vmName = null;
5     OMFactory fac = OMAbstractFactory.getOMFactory();
6     OMNamespace omNs = fac.createOMNamespace(URI, PREFIX);
7     OMElement retElement = fac.createOMElement("
            shutdownVirtualMachineResponse", omNs);
8     while(it.hasNext()){
9         ele = (OMElement) it.next();
10        if(ele.getLocalName().equals("phyServer")){
11            phyServer = ele.getText();    }
12        if(ele.getLocalName().equals("vmName")){
13            vmName = ele.getText();    }
14    }
15    try {
16        Connect Conn = new Connect("xen+ssh://" + phyServer + "/", true);
17        Domain domain = Conn.domainLookupByName(vmName);
18        domain.shutdown();
19        OMElement value = fac.createOMElement("result", omNs);
20        value.addChild(fac.createOMText(value, "SUCCESS"));
21        retElement.addChild(value);
22        return retElement;
23    } catch (LibvirtException ex) {
24        Logger.getLogger(VirtualMachineServer.class.getName()).log(Level.
            SEVERE, null, ex);
25        OMElement value = fac.createOMElement("result", omNs);
26        value.addChild(fac.createOMText(value, "Error:␣"+ex.getMessage()));
27        retElement.addChild(value);
28        return retElement;
29    }
30 }

```

para desligamento da máquina é enviado ao servidor físico através da conexão previamente estabelecida. Nas linhas 19 e 20, um elemento é criado para indicar o sucesso da operação. Na linha 21, este elemento é adicionado como filho do elemento de

retorno previamente criado e que é retornado na linha 22.

Caso ocorra alguma exceção da Libvirt (`LibvirtException`) entre as linhas 16 e 22, o bloco de linhas de 24 a 28 captura a exceção, criando um `OMEElement` contendo a indicação de que a operação não foi realizada com sucesso e com o resumo da exceção. O elemento é adicionado como filho do elemento `retElement` que é retornado na linha 28.

3.2.2 Acesso ao Servidor de Máquinas Virtuais

Com o objetivo de facilitar a criação de sistemas de software que acessem o servidor de máquinas virtuais, uma classe foi desenvolvida de forma conjunta ao projeto. Para cada serviço oferecido, a classe `HorizonXenClient` possui um método para a criação do conteúdo da mensagem. A Listagem 3.2 apresenta a API oferecida por essa classe.

Listagem 3.2: API oferecida pela classe `HorizonXenClient`.

```
1 public OMElement createVirtualMachinePayload(String phyServer, String
   vmName, String vmIP, String vmRAM);
2 public OMElement createVirtualNetworkPayload(Vector<String> phyServers,
   Vector<String> VMNames, Vector<String> IPs, Vector<String> RAMs,
   Vector<String> netInterface);
3 public OMElement destroyVirtualMachinePayload(String phyServer, String
   vmName);
4 public OMElement getPhysicalServerStatusPayload(String phyServer);
5 public OMElement getRegisteredNodesPayload();
6 public OMElement getVirtualMachineStatusPayload(String phyServer,
   String vmName);
7 public OMElement migrateVirtualMachinePayload(String sourcePhyServer,
   String destPhyServer, String vmName, String live);
8 public OMElement registerNodesPayload(Vector<PhysicalServer> phyServers
   );
9 public OMElement sanityTestPayload(String testString);
10 public OMElement shutdownVirtualMachinePayload(String phyServer, String
   vmName);
11 public OMElement topologyDiscoverPayload();
```

Observa-se que todos os métodos retornam um objeto da classe `OMEElement`

que serão utilizados como conteúdo do elemento `env:body` de mensagens SOAP.

Ainda com relação à Listagem 3.2, as ocorrências `vmName` referem-se ao nome esperado para a máquina virtual. Esse nome será o nome acessível através dos recursos de administração de sistemas Xen e para interações futuras com o servidor de máquinas virtuais. O parâmetro `phyServer` refere-se ao nome DNS externamente acessível do nó físico a que se está referindo ou o seu endereço IP. Os parâmetros `vmIP` e `vmRAM` apontam, respectivamente, o endereço IP e o tamanho da memória RAM desejados para o novo domínio virtual.

Existem ainda parâmetros específicos para a função de migração de máquinas virtuais: `sourcePhyServer` e `destPhyServer` definem, respectivamente, os nós físicos de origem e de destino do domínio virtual a ser migrado; o parâmetro `live`, que pode receber os valores `true` ou `false`, define se a migração deve ser realizada ao vivo, ou seja, sem interrupção do funcionamento do domínio virtual.

Alguns serviços podem atuar sobre um conjunto de máquinas físicas e virtuais. Nesses casos os parâmetros esperados são vetores e a semântica é similar aos casos já apontados. Existe ainda o parâmetro `testString` do método de teste de sanidade. Esse parâmetro define a *string* que formará o corpo da mensagem de teste e que será retornada pelo servidor, caso este esteja funcionando corretamente.

A Listagem 3.3 apresenta um exemplo de utilização da classe `HorizonXenClient`. Na linha 3 um objeto da classe `HorizonXenClient` é instanciado. Este objeto será utilizado para a criação da mensagem de requisição do serviço. Nas linhas 5, 6 e 7 as opções referentes à busca do serviço são ajustadas. Mais especificamente, na linha 6 o destino da requisição (EPR – *EndPoint Reference*) é ajustado, ou seja, o endereço do servidor de máquinas virtuais (armazenado no campo `hxc.targetEPR`) será utilizado como destino da mensagem SOAP. Em seguida, na linha 7 o protocolo de comunicação a ser utilizado é definido. Nesse caso o protocolo escolhido foi o HTTP. No caso da biblioteca Axis2, poderia-se escolher entre SMTP (*Simple Mail Transfer Protocol*), TCP, HTTP e JMS (*Java Message Service*). Na linha 9, um novo cliente para serviços Web é criado utilizando a classe `ServiceClient`, oferecida pela biblioteca Axis2. Em seguida, esse cliente recebe as opções anteriormente ajustadas. Na linha 12 a mensagem é criada utilizando um método da classe `HorizonXenClient`, nesse caso, o serviço requisitado será o de criação de

Listagem 3.3: Utilização da API da classe `HorizonXenClient` para a criação de uma máquina virtual.

```
1 public static void main(String [] args) {
2     try {
3         HorizonXenClient hxc = new HorizonXenClient ();
4         // set options to send message
5         Options options = new Options ();
6         options.setTo(hxc.targetEPR);
7         options.setTransportInProtocol(Constants.TRANSPORT.HTTP);
8         // create a Web Service client
9         ServiceClient sender = new ServiceClient ();
10        sender.setOptions(options);
11        // creating message payload
12        OMElement messagePayload = hxc.createVirtualMachinePayload("
            phyServer", "vmName", "10.0.0.1", "65536");
13        // send message and wait for the server response
14        OMElement result = sender.sendReceive(messagePayload);
15    } catch (Exception e) {
16        e.printStackTrace();
17    }
18 }
```

uma máquina virtual. Finalmente, na linha 14 a mensagem de requisição é enviada ao servidor e a resposta é armazenada em um novo objeto da classe `OMElement`.

É importante observar que a utilização da classe `HorizonXenClient` não é obrigatória. O cliente pode ser construído sem fazer uso dessa biblioteca. Não existe sequer limitação quanto à linguagem de programação, já que a utilização de um serviço Web permite essa flexibilidade. O único requisito é a necessidade de utilização do protocolo SOAP e de que o conteúdo da mensagem seja um XML válido com os campos esperados pelo servidor.

3.2.3 Protótipo

Para a realização de testes, um protótipo foi implantado no laboratório do Grupo de Teleinformática e Automação. Este protótipo serve como prova de conceito para o servidor de máquinas virtuais. A Tabela 3.2 apresenta os computadores

utilizados e suas respectivas funções.

Tabela 3.2: Computadores e suas funções no protótipo.

Computador	Função
lido	cliente para o servidor de máquinas virtuais / servidor DHCP
debian04	servidor de máquinas virtuais
debian05	estação com suporte a virtualização
debian07	estação com suporte a virtualização

A Figura 3.4 apresenta a topologia da rede de testes. Como dito anteriormente, a máquina `lido` é a única com acesso direto à Internet e serve como gateway para as outras estações.

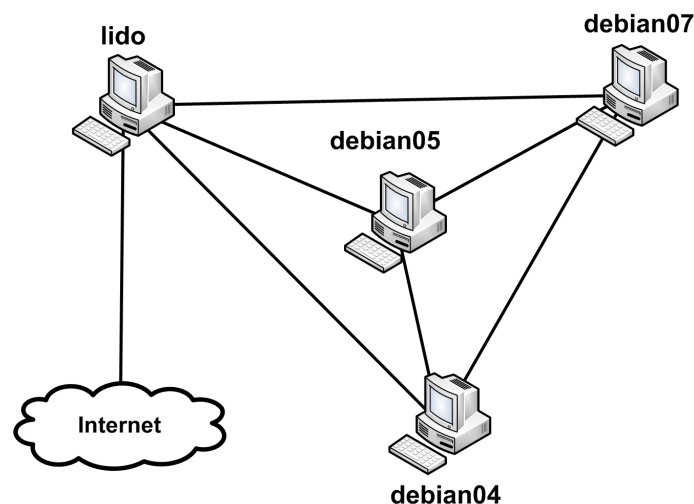


Figura 3.4: Topologia do *testbed*.

A máquina `lido` é a única que possui endereço IP roteável e é assim a única que pode ser acessada externamente. Dessa forma, a página JSP que serve como cliente para o servidor de máquinas virtuais foi instalada nessa máquina para que possa ser acessada externamente. Além disso, essa máquina acumula a função de servidor DHCP para as outras estações do protótipo.

A máquina `debian04` hospeda o servidor de máquinas virtuais. As estações `debian05` e `debian07` são máquinas que executam Xen como software de suporte à virtualização e são as máquinas operadas pelo servidor de máquinas virtuais.

Todas as estações executam o sistema operacional Linux Debian. O kernel utilizado para as máquinas Xen foi o 2.6.26-2-xen-686. Para a estação lido o kernel utilizado foi o 2.6.30-2-amd64. Finalmente na máquina `debian04`, que abriga o servidor de máquinas virtuais, foi utilizado o kernel 2.6.30-2-686.

Com o objetivo de facilitar a autenticação do servidor de máquinas virtuais, um mecanismo de SSH (*Secure Shell*) sem senha foi configurado entre o servidor de máquinas virtuais e as máquinas. Mais especificamente, a autenticação é realizada através da utilização de chaves públicas configuradas previamente [60].

Os resultados referentes aos testes realizados no protótipo são encontrados no Capítulo 4.

Capítulo 4

Resultados e Observações

Com o objetivo de testar o servidor de máquinas virtuais um cliente foi criado. Esse cliente foi desenvolvido como uma interface Web utilizando JSP (*JavaServer Pages*) [61]. Note que essa página Web assume o papel que seria do sistema de pilotagem, atuando na rede.

Select Operation

Virtual Machine Server Address:

Migrate VM
VM: Source: Destination: live

Get VM Status
Server: VM:

Get Server Status
Server:

Sanity Test
Sample String:

Register Node
Node Name: Node PK: Node IP:

Get Registered Nodes

Topology Discover

Create VM
New VM Name: Physical Host: IP Address: RAM size:

Create Virtual Network

VM 1	vm1	Host 1	physerver	IP Address 1	10.0.0.40	RAM size 1	65536
VM 2	vm2	Host 2	physerver	IP Address 2	10.0.0.40	RAM size 2	65536
VM 3	vm3	Host 3	physerver	IP Address 3	10.0.0.40	RAM size 3	65536
VM 4	vm4	Host 4	physerver	IP Address 4	10.0.0.40	RAM size 4	65536

Figura 4.1: Tela inicial do cliente.

A Figura 4.1 apresenta a tela inicial do cliente do servidor de máquinas virtuais. O primeiro campo a ser preenchido corresponde ao endereço IP do servidor de máquinas virtuais. Note que o endereço é não roteável já que no protótipo em questão o servidor está em uma máquina atendida por um servidor DHCP que

provê endereços sob um NAT. Os outros campos referem-se aos dados necessários para a realização de requisições dos serviços disponíveis. Por exemplo, para o serviço inicialmente selecionado – migração de máquinas virtuais – o cliente espera o nome da máquina virtual, os endereços IP das estações de origem e de destino do processo de migração, além da opção de utilização de migração ao vivo.

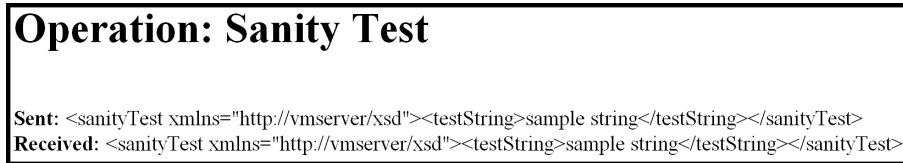


Figura 4.2: Tela de resposta à requisição ao serviço `sanityTest`.

A Figura 4.2 apresenta o resultado da requisição ao serviço que executa um teste de sanidade (`sanityTest`). Note que a string preenchida na tela apresentada na Figura 4.1 é parte da mensagem enviada. Como o servidor está funcionando corretamente, este envia a mensagem inalterada ao cliente sem realizar qualquer tarefa.

A Listagem 4.1 apresenta a implementação do método que cria a mensagem de requisição do serviço de criação de uma nova máquina virtual. Na linha 4, um *namespace* para a mensagem é criado. Em seguida, na linha 6 o elemento `createVirtualMachine` é criado, dessa forma o servidor reconhece o serviço requisitado. Em seguida, para cada um dos parâmetros – endereço IP do servidor, nome da máquina virtual, endereço IP da máquina virtual e tamanho da memória RAM da máquina virtual – um elemento é criado e adicionado como filho do elemento principal. Finalmente na linha 23, o elemento é retornado.

A Listagem 4.2 apresenta a mensagem XML enviada após a utilização do método descrito na Listagem 4.1. Note que o elemento criado pelo código é adicionado como filho do elemento `soapenv:Body`.

A Figura 4.3 apresenta o resultado da requisição do serviço `getPhysicalServerStatus`, antes da criação da máquina virtual. Note que o único domínio virtual ativo é o domínio 0.

A Figura 4.4 apresenta o retorno oferecido pelo servidor de máquinas virtuais após a requisição do serviço `createVM`. Note que na mensagem de requisição,

Listagem 4.1: Implementação de um método da classe `HorizonXenClient`.

```
1 public OMElement createVirtualMachinePayload(String phyServer, String
   vmName, String vmIP, String vmRAM, String vmDiskSize) {
2     OMFactory fac = OMAbstractFactory.getOMFactory();
3     // Set the namespace of the messages
4     OMNamespace omNs = fac.createOMNamespace(URI, PREFIX);
5     // Set the required operation
6     OMElement element = fac.createOMElement("createVirtualMachine", omNs);
7     // Physical Server IP
8     OMElement value = fac.createOMElement("phyServer", omNs);
9     value.addChild(fac.createOMText(value, phyServer));
10    element.addChild(value);
11    // Virtual Machine Name
12    value = fac.createOMElement("vmName", omNs);
13    value.addChild(fac.createOMText(value, vmName));
14    element.addChild(value);
15    // Virtual Machine IP Address
16    value = fac.createOMElement("vmIP", omNs);
17    value.addChild(fac.createOMText(value, vmIP));
18    element.addChild(value);
19    // Virtual Machine RAM Memory Size
20    value = fac.createOMElement("vmRAM", omNs);
21    value.addChild(fac.createOMText(value, vmRAM));
22    element.addChild(value);
23    return element;
24 }
```

Operation: Get Server Status

Received: <getPhysicalServerStatusResponse xmlns="http://vmserver/xsd">
<result>Success</result><cores>1</cores><cpus>1</cpus><memory>752640</memory>
<freeMemory>68812800</freeMemory> <hostName>debian07.redes.gta.ufrj.br</hostName>
<domainCount>1</domainCount> <domain>Domain-0</domain>
</getPhysicalServerStatusResponse>

Figura 4.3: Tela de resposta à requisição ao serviço `getPhysicalServerStatus`.

visualizada como *sent* na figura, os parâmetros requeridos, endereço IP da estação física, endereço IP da máquina virtual, nome da máquina virtual e memória RAM

Listagem 4.2: Mensagem de requisição de criação de máquina virtual enviada ao servidor de máquinas virtuais.

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
   envelope/">
3   <soapenv:Body>
4     <createVirtualMachine xmlns="http://vmserver/xsd/">
5       <phyServer>physical server IP</phyServer>
6       <vmName>virtual machine name</vmName>
7       <vmIP>virtual machine IP</vmIP>
8       <vmRAM>virtual machine RAM</vmRAM>
9     </createVirtualMachine>
10  </soapenv:Body>
11 </soapenv:Envelope>
```

Operation: Create VM

```
Sent: <createVirtualMachine xmlns="http://vmserver/xsd">
<phyServer>192.168.1.16</phyServer> <vmName>vmname</vmName>
<vmIP>10.0.0.40</vmIP> <vmRAM>65536</vmRAM> </createVirtualMachine>
Received: <createVirtualMachineResponse xmlns="http://vmserver/xsd">
<result>Success</result> </createVirtualMachineResponse>
```

Figura 4.4: Tela de resposta à requisição ao serviço createVM.

da máquina virtual são apresentados. Solicita-se a criação de uma máquina virtual no servidor físico com o IP 192.168.1.16, com 64 MBytes de memória RAM e com endereço IP 10.0.0.40. A mensagem recebida pelo cliente (parte inferior da Figura 4.4) indica que a operação foi realizada com sucesso, ou seja, a máquina virtual foi corretamente criada.

A Figura 4.5 apresenta a resposta do serviço `getServerStatus` após a criação da nova máquina virtual. Note que agora um domínio adicional aparece entre os domínios virtuais listados pelo serviço.

A Figura 4.6 apresenta a requisição do estado atual do domínio virtual chamado de `migrateVm1.paraty.gta.ufrj.br` que está hospedado na máquina física com endereço IP 146.164.69.215. A resposta à requisição informa que a máquina

Operation: Get Server Status

```
Received: <getPhysicalServerStatusResponse xmlns="http://vmserver/xsd">
<result>Success</result><cores>1</cores><cpus>1</cpus><memory>752640</memory>
<freeMemory>1703936</freeMemory> <hostName>debian07.redes.gta.ufrj.br</hostName>
<domainCount>2</domainCount> <domain>Domain-0</domain>
<domain>vmname</domain> </getPhysicalServerStatusResponse>
```

Figura 4.5: Tela de resposta à requisição ao serviço `getServerStatus` após criação de máquina virtual.

Operation: Get VM Status

```
Sent: <getVirtualMachineStatus xmlns="http://vmserver/xsd">
<phyServer>146.164.69.215</phyServer>
<vmName>migrateVm1.paraty.gta.ufrj.br</vmName> </getVirtualMachineStatus>
Received: <getVirtualMachineStatusResponse xmlns="http://vmserver/xsd">
<result>Success</result> <getName>migrateVm1.paraty.gta.ufrj.br</getName>
<memory>782592</memory> <getMaxMemory>786432</getMaxMemory>
<vcpus>1</vcpus> <getMaxVcpus>1</getMaxVcpus>
<cputime>100770237863</cputime> <state>VIR_DOMAIN_BLOCKED</state>
</getVirtualMachineStatusResponse>
```

Figura 4.6: Tela de resposta à requisição ao serviço `getVirtualMachineStatus`.

possui atualmente 782592 KBytes de memória alocados, que a alocação máxima de memória disponível para este domínio virtual é de 786432 KBytes, que o número de CPUs virtuais utilizadas é 1, que o número máximo de CPUs que podem ser alocadas para esse domínio também é 1. Além disso, o serviço informa que a máquina já utilizou 100770237863 ns de tempo de CPU e que, no momento da requisição, o domínio estava bloqueado.

A Figura 4.7 apresenta a mensagem de requisição de registro de um nó e sua resposta. Na requisição pode-se observar o nome do nó a ser registrado (`paraty`), a chave pública deste nó (`pk_paraty`), além da lista de endereços IP desse nó. Nesse caso específico o nó possui apenas uma interface de rede física configurada com o endereço 146.164.69.215. Na figura, observa-se também a mensagem de resposta enviada pelo servidor de máquinas virtuais indicando que o serviço foi realizado com sucesso, ou seja, as informações enviadas foram armazenadas localmente pelo

Operation: Register Node

```
Sent: <registerNodes xmlns="http://vmserver/xsd"> <Node>  
<NodeName>paraty</NodeName> <NodePK>pk_paraty</NodePK> <IPs>  
<IP>146.164.69.215</IP> </IPs> </Node> </registerNodes>  
Received: <registerNodesResponse xmlns="http://vmserver/xsd">  
<result>Success</result> </registerNodesResponse>
```

Figura 4.7: Tela de resposta à requisição ao serviço registerNodes.

servidor.

Atualmente, o protótipo de máquinas virtuais está sendo integrado ao protótipo do projeto Horizon no Grupo de Teleinformática e Automação. Alguns pontos devem ser observados. Inicialmente, o projeto previu um servidor cujas funções exclusivas consistiam em criação de máquinas virtuais e de redes virtuais. A partir disso, observou-se que seria interessante a adição de certas capacidades administrativas ao servidor, tornando um controlador de redes com possibilidade de virtualização de seus elementos.

Capítulo 5

Conclusão

Este trabalho desenvolve um servidor de máquinas virtuais adaptado a diferentes pilhas de protocolos. O servidor aqui apresentado pode ser utilizado como uma importante ferramenta no desenvolvimento de novas propostas de protocolos de comunicação para lidar com a complexidade e a multiplicidade de requisitos que ora se apresentam para a Internet.

O servidor foi implementado utilizando Java como linguagem de programação. Um conjunto de serviços está atualmente disponível e pode ser acessado com ajuda da classe de apoio desenvolvida no projeto. Essa classe, de uso opcional, foi desenvolvida também em Java e pode ser utilizada como forma de redução do tempo de desenvolvimento de clientes para o servidor de máquinas virtuais.

Um protótipo do servidor foi configurado para prova de conceito e está atualmente em teste e expansão no laboratório do Grupo de Teleinformática e Automação. Esse servidor é parte importante do protótipo resultante das pesquisas do projeto Horizon e será estendido para um número maior de estações.

De forma mais específica, o servidor está em fase de integração com outras ferramentas desenvolvidas dentro do Laboratório do Grupo de Teleinformática e Automação. Entre as ferramentas vale destacar a existência de ferramentas para o diagnóstico e a medição de sistemas virtualizados Xen e uma interface gráfica que substituirá a interface desenvolvida em JSP provendo aos usuários uma interação mais amigável e intuitiva.

Como parte dos trabalhos futuros, espera-se ampliar o número de serviços oferecidos pelo servidor, tais como o oferecimento de estatísticas mais detalhadas

sobre os estados das máquinas físicas e virtuais da rede, tornando-o, na prática, um controlador de máquinas virtuais. Além disso, o protótipo deve ser ampliado de forma a possibilitar testes com maior riqueza de detalhes, permitindo a exploração de um cenário maior do que o experimentado até o momento.

Referências Bibliográficas

- [1] HE, J., ZHANG-SHEN, R., LI, Y., *et al.*, “DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet”. Em: *ACM CoNEXT*, Dezembro de 2008.
- [2] CLARK, D. D., WROCLAWSKI, J., SOLLINS, K. R., *et al.*, “Tussle in Cyberspace: Defining Tomorrow’s Internet”, *IEEE/ACM Transactions on Networking*, v. 13, n. 3, pp. 462–475, Junho de 2005.
- [3] FELDMANN, A., “Internet clean-slate design: what and why?”, *ACM SIGCOMM Computer Communication Review*, v. 37, n. 3, pp. 59–64, Julho de 2007.
- [4] CLARK, D. D., PARTRIDGE, C., RAMMING, J. C., *et al.*, “A knowledge plane for the Internet”. Em: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 3–10, Agosto de 2003.
- [5] ROBERTS, L., “The Arpanet and computer networks”. Em: *ACM Conference on The History of Personal Workstations*, pp. 51–58, Janeiro de 1986.
- [6] BARAN, P., “On Distributed Communications Networks”, *IEEE Transactions on Communications Systems*, v. 12, n. 1, pp. 1–9, Março de 1964.
- [7] CLARK, D., CHAPIN, L., CERF, V., *et al.*, “Towards the Future Internet Architecture”, RFC 1287, Dezembro de 1991.
- [8] FULLER, V., LI, T., VARADHAN, K., “Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy”, RFC 1519, Setembro de 1993.

- [9] CLARK, D., BRADEN, R., SOLLINS, K., *et al.*, *New Arch: Future Generation Internet Architecture*, Relatório técnico, MIT Laboratory for Computer Science and International Computer Science Institute(ICS), Agosto de 2004.
- [10] COSTA, L., FDIDA, S., DUARTE, O., “Incremental Service Deployment Using the Hop-by-Hop Multicast Routing Protocol”, *IEEE/ACM Transactions on Networking*, v. 14, n. 3, pp. 543–556, Junho de 2006.
- [11] PERKINS, C., “IP Mobility Support for IPv4”, RFC 3220, Janeiro de 2002.
- [12] BRADEN, R., CLARK, D., SHENKER, S., “Integrated Services in the Internet Architecture”, RFC 1633, Junho de 1994.
- [13] BLAKE, S., BLACK, D., CARLSON, M., *et al.*, “An Architecture for Differentiated Services”, RFC 2475, Dezembro de 1998.
- [14] MOREIRA, M. D. D., FERNANDES, N. C., COSTA, L. H. M. K., *et al.*, *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC’2009*, capítulo Internet do Futuro: Um Novo Horizonte, Rio de Janeiro, SBC, pp. 1–59, Maio de 2009.
- [15] POTAROO, “IPv4 Address Report”, <http://www.potaroo.net/tools/ipv4>, 2010, (Acessado em 5 de Março de 2010).
- [16] NIEBERT, N., BAUCKE, S., EL-KHAYAT, I., *et al.*, “The Way 4WARD to the Creation of a Future Internet”. Em: *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1–5, Junho de 2008.
- [17] MOREIRA, M. D. D., *Usando Pontos de Verificação para Rastrear Ataques de Negação de Serviço Maciçamente Distribuídos*. Dissertação de Mestrado, COPPE/PEE/UFRJ, Setembro de 2009.
- [18] ANDREOLINI, M., BULGARELLI, A., COLAJANNI, M., *et al.*, “HoneySpam: honeypots fighting spam at the source”. Em: *Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pp. 77–83, Julho de 2005.

- [19] LIU, J., SINGH, S., “ATCP: TCP for Mobile Ad hoc Networks”, *IEEE Journal on Selected Areas of Communications*, v. 9, n. 7, pp. 1300–1315, Julho de 2001.
- [20] CHANDRAN, K., RAGHUNATHAN, S., VENKATESAN, S., *et al.*, “A Feedback Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks”, *IEEE Personal Communications*, v. 8, n. 1, pp. 34–39, Fevereiro de 2001.
- [21] TIAN, Y., XU, K., ANSARI, N., “TCP in Wireless Environments: Problems and Solutions”, *IEEE Communications Magazine*, v. 43, n. 3, pp. S27–S32, Março de 2005.
- [22] FALL, K., “A Delay-Tolerant Network Architecture for Challenged Internets”. Em: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 27–34, Agosto de 2003.
- [23] RUBINSTEIN, M. G., MORAES, I. M., CAMPISTA, M. E. M., *et al.*, “A Survey on Wireless Ad Hoc Networks”, *Mobile and Wireless Communications Networks*, pp. 1–34, Agosto de 2006.
- [24] RUBINSTEIN, M. G., ABDESSLEM, F. B., CAVALCANTI, S. R., *et al.*, “Measuring the Capacity of In-Car to In-Car Vehicular Networks”, *IEEE Communications Magazine*, v. 47, n. 11, pp. 128–136, Novembro de 2009.
- [25] ALVES, R. S. A., CAMPBELL, I. V., COUTO, R. S., *et al.*, *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC’2009*, capítulo Redes Veiculares: Princípios, Aplicações e Desafios, Rio de Janeiro, SBC, pp. 199–254, Maio de 2009.
- [26] “ANA: Autonomic Network Architecture”, <http://www.ana-project.org/>, (Acessado em 25 de Março de 2010).
- [27] “Horizon Project: A New Horizon to The Internet”, <http://www.gta.ufrj.br/horizon>, (Acessado em 25 de Março de 2010).
- [28] “NewArch Project: Future-Generation Internet Architecture”, <http://www.isi.edu/newarch/>, (Acessado em 31 de Março de 2010).

- [29] “GENI - Exploring Networks of the Future”, <http://www.geni.net>, (Acessado em 31 de Março de 2010).
- [30] “FIRE - Future Internet Research & Experimentation”, <http://cordis.europa.eu/fp7/ict/fire>, (Acessado em 31 de Março de 2010).
- [31] TENNENHOUSE, D. L., WETHERALL, D. J., “Towards an Active Network Architecture”, *ACM SIGCOMM Computer Communication Review*, v. 26, n. 2, pp. 5–17, Abril de 1996.
- [32] CROWCROFT, J., HAND, S., MORTIER, R., *et al.*, “Plutarch: an Argument for Network Pluralism”. Em: *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pp. 258–266, Agosto de 2003.
- [33] KELLER, A., HOSSMANN, T., MAY, M., *et al.*, “A System Architecture for Evolving Protocol Stacks (Invited Paper)”. Em: *International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, Agosto de 2008.
- [34] FEAMSTER, N., GAO, L., REXFORD, J., “How to Lease the Internet in Your Spare Time”, *ACM SIGCOMM Computer Communication Review*, v. 37, n. 1, pp. 61–64, Janeiro de 2007.
- [35] CARISSIMI, A., *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC'2008*, capítulo Virtualização: da Teoria a Soluções, Rio de Janeiro, SBC, pp. 173–207, Maio de 2008.
- [36] ANDERSON, T., PETERSON, L., SHENKER, S., *et al.*, “Overcoming the Internet Impasse Through Virtualization”, *ACM Computer*, v. 38, n. 4, pp. 34–41, Abril de 2005.
- [37] EGI, N., GREENHALGH, A., HANDLEY, M., *et al.*, “Towards high performance virtual routers on commodity hardware”. Em: *ACM CoNEXT*, Dezembro de 2008.
- [38] FIUCZYNSKI, M. E., “PlanetLab: Overview, History, and Future Directions”, *SIGOPS Operating Systems Review*, v. 40, n. 1, pp. 6–10, Janeiro de 2006.
- [39] “VMware”, <http://www.vmware.com/>, (Acessado em 7 de Março de 2010).

- [40] BARHAM, P., DRAGOVIC, B., FRASER, K., *et al.*, “Xen and the art of virtualization”. Em: *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 164–177, Outubro de 2003.
- [41] POPEK, G. J., GOLDBERG, R. P., “Formal Requirements for Virtualizable Third Generation Architectures”, *Communications of the ACM*, v. 17, n. 7, pp. 412–421, Julho de 1974.
- [42] UHLIG, R., NEIGER, G., RODGERS, D., *et al.*, “Intel Virtualization Technology”, *ACM Computer*, pp. 48–56, Maio de 2005.
- [43] “Introducing AMD Virtualization”, http://www.amd.com/gb-uk/Processors/ProductInformation/0,,30_118_88_26_14287,00.html, (Acessado em 31 de Março de 2010).
- [44] STAHL, E., THEURER, A. M., *A POWER6 Virtualization Performance Study Using Integrated Virtual Ethernet*, Relatório técnico, IBM, Outubro de 2007.
- [45] BISWAS, K., ISLAM, M. A., “Hardware Virtualization Support in Intel, AMD and IBM Power Processors”, *International Journal of Computer Science and Information Security (IJCSIS)*, v. 4, n. 1 & 2, pp. 72–77, Agosto de 2009.
- [46] MENON, A., COX, A. L., ZWAENEPOEL, W., “Optimizing Network Virtualization in Xen”. Em: *USENIX Annual Technical Conference*, pp. 15–28, Maio de 2006.
- [47] CLARK, C., FRASER, K., HAND, S., *et al.*, “Live Migration of Virtual Machines”. Em: *Symposium on Networked Systems Design & Implementation (NSDI)*, pp. 273–286, Maio de 2005.
- [48] MURRAY, D. G., MILOS, G., HAND, S., “Improving Xen Security through Disaggregation”. Em: *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pp. 151–160, Março de 2008.
- [49] W3C, “Web Services Activity”, <http://www.w3.org/2002/ws/>, (Acessado em 5 de Março de 2010).

- [50] BOX, D., EHNEBUSKE, D., KAKIVAYA, G., *et al.*, “Simple Object Access Protocol (SOAP) 1.1”, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>, 2000, (Acessado em 6 de Março de 2010).
- [51] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., *et al.*, “Web Services Description Language (WSDL) 1.1”, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, 2001, (Acessado em 6 de Março de 2010).
- [52] “CORBA”, <http://www.corba.org/>, (Acessado em 10 de Março de 2010).
- [53] GOKHALE, A., KUMAR, B., SAHUGUET, A., “Reinventing the Wheel? CORBA vs. Web Services”. Em: *WWW2002 Conference*, Maio de 2002.
- [54] “Libvirt: The Virtualization API”, <http://libvirt.org/>, (Acessado em 5 de Março de 2010).
- [55] POTAROO, “Xen Management API Project”, <http://wiki.xensource.com/xenwiki/XenApi>, (Acessado em 31 de Março de 2010).
- [56] “The Source for Java Developers”, <http://java.sun.com/>, (Acessado em 10 de Março de 2010).
- [57] “Apache Axis2/Java - Next Generation Web Services”, <http://ws.apache.org/axis2/>, (Acessado em 5 de Março de 2010).
- [58] “Apache Tomcat”, <http://tomcat.apache.org/>, (Acessado em 25 de Março de 2010).
- [59] “NetBeans”, <http://netbeans.org/>, (Acessado em 25 de Março de 2010).
- [60] SSH Communications Security, *SSH Secure Shell for Servers Version 3.2.9 - Administrator’s Guide*, Setembro de 2003.
- [61] “JavaServer Pages Technology”, <http://java.sun.com/products/jsp/>, (Acessado em 8 de Março de 2010).