Onix: Uma plataforma de desenvolvimento para controle distribuído

Welington Manoel da Silva, *welingtonms@hotmail.com*, UFSCar – *Campus* Sorocaba.

Resumo - *A necessidade de uma plataforma de desenvolvimento que abstraísse os detalhes de infraestrutura física da rede e permitisse a criação de planos de controle mais flexíveis motivou o surgimento da Onix, uma plataforma de controle que disponibiliza as funções mais básicas de controle de redes e permite aos desenvolvedores manipular o estado da rede e seus elementos de acordo com as especificidades de cada aplicação. Quanto às questões de escalabilidade, confiabilidade, consistência e distribuição, uma gama de possibilidades é oferecida, de modo que seja possível a aplicação da plataforma em vários contextos, o que pode ser verificado nas aplicações em desenvolvimento que mostram sua efetividade.*

Palavras-chaves: sistemas distribuídos, plataforma de controle distribuído, gerenciamento distribuído.

1 INTRODUÇÃO

O conceito e a aplicação de redes de computadores e sistemas distribuídos têm sido bastante discutidos nos últimos anos com o advento da *cloud computing*, que surgiu com o intuito de tornar-se a próxima etapa no desenvolvimento, na evolução da internet (1). Com isso torna-se imprescindível a existência de um paradigma geral de controle de redes sob a qual a provisão de serviços seja mais focada na lógica de negócio ao invés de preocupar-se com detalhes de gerenciamento.

Para que se alcance esse estado de desenvolvimento de softwares para ambientes distribuídos, necessita-se de uma plataforma genérica capaz de prover mecanismos de controle em nível de rede que abstraia a distribuição de estado dos elementos que a compõe, funções básicas como descoberta de novos elementos, mecanismos de recuperação de falhas, etc. e que forneça uma interface, através da qual desenvolvedores possam construir aplicações para o gerenciamento de rede sem se preocupar com a visão de baixo nível de controle de rede e que se adeque às condições específicas de cada domínio. Essa necessidade se agrava se levarmos em consideração requisitos de controle inerentes à *cloud computing* tais como maior escalabilidade, segurança, migração de máquinas virtuais, etc.

Como resposta a essa necessidade, tanto academia quanto indústria (especificamente a parcela que opera grandes redes) têm se aplicado na definição de um padrão no qual o plano de controle é desacoplado do plano de encaminhamento e é construído como um sistema distribuído. Neste modelo, denominado *Rede Definida por Software* (RDS) (2), a plataforma de controle roda sobre um ou mais servidores e monitora um conjunto de *switches* gerenciando a distribuição de estado – coletando, distribuindo e coordenando os estados entre os diversos servidores – e provê uma interface sobre a qual desenvolvedores podem construir uma diversidade de aplicações de gerenciamento de rede.

O modelo tradicional de controle engessa a criação de novas funções de controle de rede exigindo que se crie um protocolo próprio, que se tenha desenvoltura com o aspecto baixo-nível envolvido no problema que se pretende resolver, além da dificuldade de implantação da solução no *switch*. A proposta da RDS é fazer com que qualquer nova função seja implementada sobre a API de uma plataforma de controle de modo que as dificuldades associadas sejam responsabilidades da plataforma. Isso é possível por que as primitivas de distribuição de estado são implementadas na própria plataforma de controle, ao invés de estarem em tarefas de controle separadas, e utilizam técnicas extraídas da literatura bem conhecidas e de propósito geral ao invés de soluções mais específicas.

A plataforma de controle é o grande facilitador do paradigma RDS e para que esta seja efetiva, desafios como generalidade, escalabilidade, confiabilidade, simplicidade e desempenho de plano de controle devem ser enfrentados.

Nas seções que se seguem iremos tratar um pouco mais sobre a arquitetura da plataforma Onix (Seção 2), questões que tangem requisitos de escalabilidade e confiabilidade (Seção 3), os mecanismos de distribuição utilizados (Seção 4), algumas informações sobre a implementação da plataforma (Seção 5) e, finalmente, aplicações em contextos reais (Seção 6).

2 ARQUITETURA

Para discussão da arquitetura da Onix dois aspectos são relevantes: O contexto que a engloba enquanto plataforma de controle e a API disponibilizada.

2.1 *Componentes*

2.1.1 Infraestrutura física

A infraestrutura física inclui além de *switches* e roteadores, quaisquer outros elementos de rede que permitam, através de uma interface, ter seu estado controlado (através de leitura e escrita) habilitando a Onix a manipular seu comportamento. Não é necessária a adição de nenhum software para que este controle seja implementado.

2.1.2 *Infraestrutura de conectividade*

Esta infraestrutura envolve a rede em si e sua conexão com a Onix, que estabelece um *canal de controle*. Esse canal de controle pode ser tanto *in-band* quanto *out-of-band*. No canal de controle *in-band* todo o tráfego de controle compartilha o mesmo canal de tráfego de dados na rede. No *out-of-band* uma rede separada é utilizada exclusivamente para este fim.

A infraestrutura de conectividade deve suportar comunicação bidirecional entre as instâncias da Onix e os *switches*. A criação e a manutenção da estrutura de encaminhamento podem ser feitas utilizando-se protocolos padrão de roteamento, como o OSPF (*Open Shortest Path First)*.

2.1.3 *Onix*

É um sistema distribuído que roda sobre um *cluster* de um ou mais servidores físicos, sendo que cada um destes podem rodar uma ou mais instâncias da Onix.

Enquanto plataforma de controle a Onix é a responsável por habilitar a lógica de controle das aplicações a operar sobre o estado dos elementos de rede (através da leitura e escrita) e por disseminar o estado da rede para outras instâncias dentro de um *cluster*.

2.1.4 *Lógica de controle*

A lógica de controle de rede é codificada sobre a API da Onix e determina o comportamento da rede através do uso das primitivas necessárias para manipulação de estado dos elementos que compõe a rede.

2.2 *A API*

O intuito da Onix é "*definir uma API útil e genérica para controle da rede que permita o desenvolvimento de aplicações escaláveis"* (2). Através da visibilidade da rede física, ela permite às aplicações ler e escrever estado de quaisquer elementos, provendo métodos para manter a consistência tanto entre estes quanto na própria aplicação de controle rodando entre múltiplas instâncias.

Em uma visão mais minimalista, a API Onix consiste de um modelo de dados que representa a infraestrutura da rede, sendo que a lógica utilizada em uma aplicação para controle distribuído pode ler o estado associado a um determinado objeto (que representa um elemento de rede), alterar o estado da rede através da manipulação de objetos e registrar-se para notificação em caso de mudança no estado dos objetos.

A cópia do estado atual da rede é mantida em uma estrutura de dados chamada *Network Information Base* (NIB) que armazena um grafo de todas as entidades na topologia; é através da replicação e distribuição da NIB entre as múltiplas instâncias que a Onix consegue prover escalabilidade. A detecção e a resolução de conflito durante a replicação e a distribuição são dependentes da lógica de cada aplicação em específico. Ainda a garantia de consistência pode ser customizada a fim de se adequar a diferentes cenários.

2.3 *NIB*

A NIB mantém o conjunto de entidades de rede, cada qual com um conjunto de pares chave-valor e identificada por um identificador global de 128-bits.

A Onix suporta tipagem forte através de entidades que representam os elementos da rede ou sua subpartes. Entidades tipadas contém um conjunto de atributos (os pares chave-valor) e métodos que executam operações sobre tais atributos.

A Figura 1 representa o conjunto de entidades tipadas *default* da Onix. Além desses tipos básicos, permite-se que as aplicações estendam o modelo de dados da Onix através da criação de subclasses.

Como as entidades são referenciadas na NIB por seu identificador único e global, é possível executar uma consulta direta por uma entidade específica; adicionalmente, uma aplicação pode registrar-se para receber notificações mediante alguma mudança de estado, adição ou remoção de entidades. Com isso, a lógica de controle de uma aplicação pode atuar a seu modo e criar seus próprios mecanismos de manipulação do estado da rede, modificando os atributos das entidades envolvidas.

A NIB não provê nenhum tipo de mecanismo de bloqueio distribuído, mas oferece um mecanismo de requisição de acesso exclusivo e liberação da instância local da NIB. Isso garante que nenhuma *thread* rodando na mesma instância irá executar uma atualização no mesmo instante, mas não garante que a mesma permanecerá intocada por outras instâncias Onix ou mesmo algum elemento de rede. Para cobrir essa deficiência, é necessário utilizar mecanismos implementados externamente, conforme abordado no tópico *Mecanismos de distribuição da NIB.*

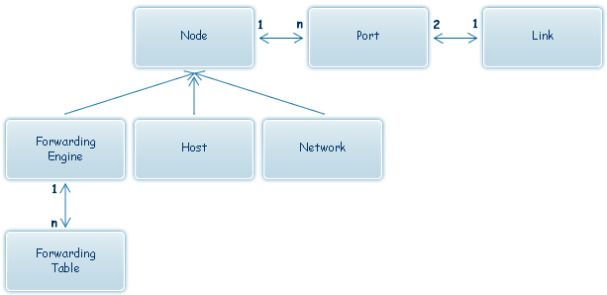

Figura 1 - As classes padrão que representam as entidades de rede fornecidas pela API. Linhas unidirecionais representam herança e linhas bidirecionais representam relações de mapeamento quantitativo. Nós, portas e *links* constituem a topologia de rede.

Todas as operações realizadas na NIB são assíncronas. Isso quer dizer que uma operação de atualização realizada sobre uma entidade da rede, eventualmente, será enviada a um elemento de rede e/ou outra instância Onix sem garantias de ordenação ou latência. Por um lado isso torna operações de múltiplas atualizações mais eficientes, por outro lado, na maioria das vezes é necessário saber se a operação disparada foi completada com sucesso. Para isso, a API fornece uma primitiva de sincronização na qual, uma vez que a requisição de atualização for concluída, a lógica de controle recebe uma chamada de retorno e pode inspecionar o conteúdo da NIB a fim de verificar se o estado corrente era o esperado para o procedimento realizado. Quaisquer falhas

identificadas nas operações realizadas poderão ser notificadas à aplicação pela NIB.
A Tabela 1 lista as categorias de operações disponíveis na NIB bem como a finalidade de cada uma dela.

Categoria	Finalidade
Consulta	Buscar por entidades.
Criar, excluir	Criar e excluir entidades.
Acesso a atributos	Inspecionar e modificar entidades.
Notificações	Receber atualizações sobre mudanças.
Sincronização	Esperar que atualizações sejam aplicadas no conteúdo dos elementos da rede e dos controladores.
Configuração	Configurar como os estados são importados e exportados da NIB.
Pull	Solicitar que entidades sejam importadas sob demanda.

Tabela 1 - Operações disponíveis para a NIB listadas de acordo sua categoria e finalidade.

3 ESCALABILIDADE E CONFIABILIDADE

Sendo a NIB a base para o estado do sistema e seus eventos, sua utilização determina as propriedade de escalabilidade e confiabilidade. Ora, se o número de elementos presentes na rede aumenta, uma NIB não distribuída pode exaurir a memória do sistema; se o número de eventos na rede (gerados pela NIB) aumenta o trabalho requerido para gerenciá-los, pode saturar a CPU de uma única instância da Onix.

3.1 *Escalabilidade*

A Onix suporta três estratégias que podem ser usadas para melhorar a escalabilidade dos sistemas de controle: particionamento, agregação e consistência e durabilidade.

3.1.1 Particionamento

A lógica de controle pode configurar a Onix de modo que uma instância do controlador mantenha apenas um subconjunto da NIB na memória e tal conjunto estará atualizado. Com isso, uma instância da Onix pode ter conexões com um subconjunto de elementos e, subsequentemente, terá um menor número de eventos gerados para processar.

3.1.2 Agregação

Em uma configuração multi-Onix, uma instância pode expor um subconjunto de elementos em sua NIB como um elemento agregado para outras instâncias. Por exemplo, na rede do campus de uma Universidade, cada prédio pode ser gerenciado por um controlador Onix que expõe todos os elementos de deste prédio como um único nó agregado para uma instância global da Onix que controla a engenharia de tráfico do campus inteiro.

3.1.3 Consistência e durabilidade

A lógica de controle determina os requisitos de consistência para a rede que ela gerencia. Isto é feito através da implementação de um mecanismo de bloqueio distribuído e algoritmos de consistência para estados que requeiram uma consistência forte, e provendo detecção e resolução de conflitos para casos de exceção.
Por padrão, a Onix fornece dois tipos de armazenamento de dados que uma aplicação pode utilizar para lidar com diferentes preferências de durabilidade e consistência. Para estados que exijam alta durabilidade e consistência, oferece-se um banco de dados transacional replicado e para estados voláteis – mais tolerantes a inconsistências – uma DHT (*Distributed Hash Table*) armazenada em memória. Na DHT, itens de dados são associados a uma chave aleatória de um espaço de identificadores de *n*-bits (usualmente, 128 ou 160-bits dependendo da função *hash* utilizada). Da mesma forma, os nós (arquivo, processo, elemento de rede, etc.) são associados a números aleatórios do mesmo espaço de identificadores. O intuito é estabelecer um esquema eficiente e determinístico que mapeia unicamente a chave de um dado ao identificador de um nó. Quando se busca por um dado, o endereço de rede do nó responsável por este dado é retornado e a busca é concluída com a requisição do dado ao nó responsável (3).

3.2 *Confiabilidade*

Para assegurar confiabilidade, as aplicações baseadas na Onix têm de gerenciar quatro tipos de falhas na rede: Falha em elementos de encaminhamento, falha de *link*, falha em instâncias Onix e falha de conectividade entre elementos de rede e instâncias Onix e entre as próprias instâncias Onix.

3.2.1 Falhas de elemento de rede e de link

Se um elemento de rede falha, a função da aplicação de controle é desviar o tráfego de dados dos elementos problemáticos. O tempo mínimo de reação mediante a tais falhas é determinado pelo tempo de disseminação da falha pela rede juntamente com o tempo de recálculo das tabelas de encaminhamento. Com o intuito de tornar esse tempo mínimo o suficiente para a recuperação, pode ser válido a utilização de caminhos *backups* em associação com algum outro mecanismo de recuperação.

3.2.2 Falha de uma instância Onix

Para gerenciar esse tipo de falha a lógica de controle da aplicação possui duas opções: alguma das instâncias ainda em funcionamento pode detectar o nó defeituoso e assumir suas responsabilidades, ou ainda pode se utilizar o gerenciamento redundante de elementos de rede. Caso a segunda opção seja a escolhida, a lógica de controle deve lidar com a perda de atualização em casos de atuação concorrente. Para isso, a Onix fornece um meio para que as aplicações possam determinar se alterações conflitantes feitas por outras instâncias podem ou não ser sobrescritas.

3.2.3 Falha de conectividade

O mecanismo de distribuição de estados da Onix é separado da topologia subjacente. Portanto, em casos de recuperação de falha, a conectividade é requerida tanto entre os elementos de rede e as instancias Onix, quanto entre as instâncias Onix.
Não são incomuns redes que possuem uma rede física ou virtual dedicada para gerenciamento. Nestes ambientes, a Onix pode utilizar essa rede de gerenciamento para controle

de tráfego mantendo-a isolada de possíveis rupturas no plano de encaminhamento. Essa rede de controle funciona da mesma forma que qualquer outra rede, sendo assim, suas falhas são tratadas através dos protocolos tradicionais (por exemplo, OSPF ou *Spanning Tree* (4)).

Mesmo que não haja uma rede de controle separada, a topologia da rede física é conhecida pela Onix. Logo, a lógica de controle pode popular os elementos de rede com um estado de encaminhamento estático para estabelecer a conectividade entre a Onix e os *switches*. Para assegurar a conectividade na presença de falhas, o roteamento padrão pode ser combinado com o *multi-pathing* (também implementado pela Onix) (5): o roteamento de pacotes por múltiplos caminhos pode garantir conectividade confiável para os elementos da rede gerenciada, bem como para as instâncias Onix.

4 MECANISMOS DE DISTRIBUIÇÃO DA NIB

O mecanismo de distribuição de estado da Onix foi guiado por duas observações feitas sobre aplicações de gerenciamento de rede. Primeiro, aplicações possuem diferentes requisitos de escalabilidade, frequência de atualização em espaço compartilhado e durabilidade; segundo, aplicações possuem requisitos distintos de consistência do estado de rede que gerenciam.

Para atender a aplicações com esses diferentes requisitos, Onix permite a escolha entre velocidade de atualização e durabilidade fornecendo dois mecanismos separados para distribuição de atualização de estados de rede entre instâncias: um projetado para altas taxas de atualização com disponibilidade assegurada, e outra projetada para dar suporte à durabilidade e consistência. O desenvolvedor de uma aplicação pode então estabelecer um *trade-off* entre performance e escalabilidade e determinar explicitamente o mecanismo que será adotado para os estados da NIB.

A Onix também suporta diferentes sistemas de armazenamento, desde que as aplicações escrevam seus próprios módulos de importação e exportação que transferem dados do sistema de armazenamento para a NIB e da NIB para o sistema de armazenamento, respectivamente.

Sobre as preferências por diferentes requisitos de consistência, a Onix espera que as aplicações utilizem as facilidades de coordenação providas pela plataforma para implementar bloqueio distribuído ou protocolo de consenso, conforme necessário. Também espera-se que seja codificada a lógica para tratamento de inconsistência devido a atualizações caso não se tenha optado por uma consistência mais rígida; a plataforma disponibiliza um *framework* para auxiliar as aplicações nestas implementações.

4.1 *Distribuição de estado entre instâncias Onix*

Diferentes mecanismos são disponibilizados para manter o estado consistente entre as instâncias Onix e entre as instâncias Onix e os elementos de rede, basicamente porque *switches* possuem baixa capacidade de processamento e limitações de memória, sendo necessário que o protocolo adotado seja leve e garanta consistência no estado de encaminhamento.

A Onix implementa um banco de dados transacional replicado para disseminação de todas as atualizações que requerem durabilidade e gerenciamento simplificado de consistência. Um banco de dados replicado vem com severas limitações de desempenho (6) e, portanto, serve apenas como um mecanismo de propagação confiável para estados de rede com baixas taxas de mudança. Se necessário, uma API de consultas baseada em SQL, *triggers*, e modelagem de dados é disponibilizada.

Pra integrar o banco de dados replicado com a NIB, Onix inclui módulos de importação/exportação. Estes componentes leem e armazenam entidades e seus atributos de/para o banco de dados. As modificações realizadas na NIB podem ser agrupadas e enviadas ao banco de dados em uma única transação. Quando um módulo de importação recebe a invocação de uma *trigger* do banco de dados que anuncia mudança de conteúdo, ele aplica estas alterações à NIB.

Para estados de rede que requerem altas taxas de atualização e disponibilidade, a Onix provê uma DHT eventualmente consistente, armazenada em memória, relaxando as garantias de durabilidade e consistência. Atualizações na DHT disparados por múltiplas instâncias Onix podem levar a estados inconsistentes. Neste caso, fica por conta da aplicação ou resolver o conflito ou evitar essas condições usando mecanismos de coordenação distribuídos.

4.2 *Gerenciamento de estado de elemento de rede*

A plataforma não determina um protocolo particular para gerenciamento de estado de encaminhamento de um elemento de rede. Através da interface provida pela NIB, qualquer protocolo suportado pelos elementos na rede pode ser usado desde que as entidades na NIB estejam sincronizadas com o estado atual da rede. Um exemplo de protocolo suportados é o OpenFlow (7); Para permitir a integração, os eventos e operações do OpenFlow são transformados em estados, que por sua vez são armazenados e manipulados como entidades da NIB.

4.3 *Consistência e coordenação*

A NIB é o ponto de integração central para dados vindos de múltiplas fontes (outras instâncias Onix e outros elementos de rede), ou seja, os mecanismos de distribuição de estado não interagem diretamente com cada um deles; ao invés disso, eles importam e exportam estado de/para a NIB. Para suportar aplicações com diferentes requisitos de escalabilidade e confiabilidade é necessário que cada aplicação declare que dados devem ser importados/exportados de determinada fonte através da configuração dos respectivos módulos.

Como a integração de fontes de dados é feita sem exigir consistência, estados inconsistentes são armazenados ou devido à inconsistência de estado dentro da uma fonte específica de dado (DHT) ou devido à inconsistência entre diferentes fontes. Por isso, a Onix espera que as aplicações registrem sua lógica de resolução de inconsistência na plataforma. As aplicações podem fazê-lo de duas formas: a) Na Onix, as entidades são classes C++ que a aplicação pode estender e, portanto, espera-se que a herança seja utilizada para englobar a lógica de detecção de inconsistência referencial nas entidades para que as aplicações não sejam

expostas a estados inconsistentes; caso isso ocorra, as mudanças inconsistentes permanecem pendentes dentro da NIB até que possam ser aplicadas ou as aplicações declarem-nas como inválidas; b) Os *plug-ins* que as aplicações passam para os componentes de importação/exportação implementam a lógica de resolução de conflitos, permitindo ao módulo de importação discernir como resolver situações onde tanto a NIB local quanto a fonte do dado possuem mudanças para o mesmo estado.

5 IMPLEMENTAÇÃO

A Onix consiste de aproximadamente 150.000 linhas de código escritas em C++ e integra certo número de bibliotecas de terceiros. Basicamente ela contém lógica para comunicação com elementos de rede, agregar informações na NIB e prover um *framework* no qual os programadores possam escrever aplicações de gerenciamento.

Uma única instância da Onix pode rodar sobre muitos processos, cada qual codificado usando diferentes linguagens de programação, se necessário. Os processos são interconectados usando *Remote Procedure Call* (RPC) – a mesma interconexão das instâncias Onix – mas ao invés de rodar sobre TCP/IP, roda sobre conexões IPC[1] local. Neste modelo, suportar uma nova linguagem de programação é só uma questão de escrever alguns milhares de linhas de código para integração. Atualmente a Onix suporta as linguagens C++, Python e Java.

Independente da linguagem de programação, todos os módulos da Onix são escritos como componentes com baixo acoplamento, que podem ser substituído por outros sem necessidade de recompilação da plataforma, desde que a interface do componente binário permaneça a mesma. Componentes podem ser carregados e descarregados dinamicamente, e os desenvolvedores podem expressar as dependências entre os componentes para assegurar que eles sejam carregados e descarregados na ordem apropriada.

6 APLICAÇÕES

As sessões seguintes discorrem sobre algumas aplicações construídas sobre a Onix.

6.1 *Distributed Virtual Switch*

Em ambientes corporativos virtualizados, as bordas na topologia da rede consistem de *switches* baseados em *software* dentro de *hypervisors*[2] ao invés de *switches* de rede física. Não é incomum a implantação virtual (especialmente em provedores de hospedagem em *cloud*) consistir de milhares ou dezenas de milhares de *Virtual Machines* (VMs, Máquinas Virtuais) no total. Estes ambientes são altamente dinâmicos, de modo que VMs são adicionadas, excluídas e migradas *on the fly*[3]. Para atender a necessidade de tais ambientes surge o conceito de *Distributed Virtual Switch* (DVS).

Um DVS provê uma abstração lógica de *switch* em cujas portas declaram-se políticas. Estas portas estão ligadas às máquinas virtuais através da integração com o *hypervisors*. Enquanto as máquinas vão e vem em torno da rede, o DVS assegura que as políticas sigam as VMs e, portanto, não têm de ser reconfiguradas manualmente; com essa finalidade, o DVS integra-se à plataforma de virtualização.

Quando operando como parte de uma aplicação DVS, a Onix não é envolvida na estrutura de fluxo do plano de encaminhamento, mas é invocada quando as VMs são criadas, excluídas ou migradas. *Hypervisors* são organizados em *pools*[4] que consistem de um número razoavelmente pequeno de *hypervisors* e VMs tipicamente não migram entre *pools*. Baseando-se nessa organização, a lógica de controle pode facilmente particionar-se de acordo com esses *pools*. Uma única instância Onix lida com todos os *hypervisors* de um único *pool*. Todo o estado de configuração de *switch* é persistido no banco de dados transacional, ao passo que a localização das VMs não e compartilhada entre as instâncias Onix.

Em caso de falha em uma instância da Onix, a rede ainda pode operar, entretanto, VMs dinâmicas não mais serão permitidas (2).

6.2 *Data centers virtualizados* multi-locatários

Em ambientes multi-locatários, além de ter de gerenciar a dinâmica de hospedeiros finais, a rede têm de forçar o isolamento de endereçamento e de recursos entre locatários. Redes locatárias podem ter, por exemplo, sobreposição de endereços MAC ou IP e podem rodar sobre a mesma infraestrutura física.

Para atender a essa necessidade, foi desenvolvida uma aplicação baseada na Onix que permite a criação de redes específicas com configuração independente para cada rede locatária e provê um modelo de serviço *Ethernet* padrão.

A lógica de controle isola as redes locatárias encapsulando os pacotes nas bordas antes que eles adentrem a rede física e descapsulando–os quando entram em outro *hypervisors* ou são entregues à internet.

A lógica de controle estabelece túneis entre todos os *hypervisors* que rodam as máquinas virtuais anexadas a uma determinada rede locatária virtual e, para evitar que o gerenciamento desses túneis exceda a capacidade de uma instância Onix, ela particiona a rede entre múltiplas instâncias distribuindo as responsabilidades entre elas. Uma única instância gerencia apenas um subconjunto de *hypervisors*, mas publica a informação de seu ponto terminal de túnel na DHT para que qualquer outra instância que necessite estabelecer um túnel envolvendo um de seus *hypervisors* possa configurar o módulo de importação da DHT para ler as informações relevantes para a NIB.

7 CONCLUSÃO

Considerando que ambientes de rede específicos exigem implantação de soluções específicas, e que estas por sua vez requerem protocolos específicos que demandam uma abordagem mais baixo nível e um maior nível de conhecimento, a Onix, dentro do paradigma SND, sem

[1] Inter-Process Communication é qualquer método de passagem de dados entre processos rodando em espaços de endereçamento diferentes.

[2] Responsável por monitorar as máquinas virtuais

[3] Ou seja, as máquinas virtuais são migradas sem que seus processos em execução sejam interrompidos.

[4] Grosseiramente um *pool* é equivalente a um *cluster*, um agrupamento.

dúvida vem para facilitar a vida dos desenvolvedores de aplicações de gerência de rede ao abstrair detalhes de infraestrutura física e permitir o foco na lógica de controle baseado na API. Além disso, proporciona a liberdade de determinação do *trade-off* entre arquitetura potencialmente simplificadas – promovendo consistência e durabilidade – e operações mais eficientes com aumento considerável de complexidade.

Graças as possibilidade e versatilidade de aplicação em diferentes domínios a Onix tem grandes chances de crescimento no ambiente de *cloud computing*, permitindo gerência e customização de mais baixo nível em serviços sob demanda.

8 REFERÊNCIAS

1. **Hurwitz, Judith, Bloor, Robin e Kaufman, Marcia.** *Cloud computing for dummies - HP Special Edition.* Hoboken, NJ : Wiley Publishing, Inc, 2010. 978-0-470-63881-1.

2. **Koponen, Teemu, et al., et al.** Onix: A Distribuited Control Platform for Large-Scale Production Networks.

3. **Tanenbaum, Andrew.** *Distribuited Systems, Principles and Paradigms.* New Jersey : Prentice Hall, 2007. 0-13-239227-5.

4. **Kurose, James F. e Ross, Keith W.** *Computer networking, a top-down approach featuring the Internet.* 3ª. s.l. : Pearson, 2006. 85-88639-18-1.

5. **Akella, Aditya.** Multi-Path Routing: A Different Perspective. s.l. : Carnagie Mellon University.

6. **Elmasri, Ramez e Navathe, Shamkant B.** *Fundamentals of database systems.* s.l. : Pearson, 2004. 0-321-12226-7.

7. **Mckeown, N., et al., et al.** OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM CCR 38.* 2, 2008.

8. **Hunt, P., et al., et al.** ZooKeeper: Wait-free Coordination for Internet-Scale Systems. *Usenix Annual Technical Conference.* Junho de 2010.

9. Link Layer Discovery Protocol (LLDP), a new standart for discovering and Managing converged network device. *Extreme Networks Technical Brief - Extreme Networks, Inc.*