Isolamento de recursos de rede em ambientes virtuais baseados em *Openflow/SDN*

Verônica Saliba Gomes[1], Airton Ishimori[1], Fernando N. N. Farias[1], Eduardo C. Cerqueira[1], Antônio J. G. Abelém[1]

[1]Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA)
Grupo de Pesquisa em Redes de Computadores e Comunicação Multimídia (GERCOM)

{nicasaliba, airton, fernnf, cerqueira, abelem}@ufpa.br

Abstract. *On the context of software-defined networks, the OpenFlow framework has emerged as an important tool for new Future Internet implementations, the creation of a virtualization layer in OpenFlow environments enables execution of multiple experiments on a single network infrastructure through the use of the FlowVisor. However, this solution architecture still has some limitations, among them the most important is the need for new methods to ensure the real resources isolation between the different network slices. Therefore, this article aims to present a solution to improve the FlowVisor isolation functions, through the implementation of new modules that allow the enforcement of traffic control mechanisms and the management of resources between virtual networks. The results show that the solution isolate slices not allowing interferences between them, as currently occurs with the FlowVisor.*

Resumo. *No contexto das redes definidas por software, o framework OpenFlow se apresenta como uma importante solução para o desenvolvimento e validação de novas propostas para a Internet do Futuro. A criação de uma camada de virtualização em ambientes OpenFlow possibilita a execução de múltiplos experimentos em paralelo sobre uma única infraestrutura de rede, através da utilização da ferramenta FlowVisor. Porém, esta solução ainda apresenta algumas limitações em sua arquitetura, dentre as quais a mais importante é a necessidade de novos métodos para garantir o isolamento real de recursos entre as diferentes fatias de rede. Este artigo tem como objetivo apresentar uma solução para o aperfeiçoamento das funções de isolamento do FlowVisor, por meio da implementação de novos módulos que permitam a aplicação de mecanismos de controle de tráfego e gerenciamento de recursos entre as redes virtuais. Os resultados demonstram que a solução é capaz de isolar as fatias não permitindo interferências entre elas, como ocorre atualmente no FlowVisor.*

1. Introdução

A expansão da Internet, aliada ao crescimento na quantidade de usuários e às novas necessidades apresentadas pelas aplicações, vem ocorrendo através do incremento sucessivo de soluções e protocolos na arquitetura original da rede, porém, sem grandes mudanças em sua estrutura básica. A aplicação de contínuas adaptações, cada vez mais substanciais, acabou determinando a ocorrência de novos problemas. Com isto, é

possível verificar a incompatibilidade das novas e crescentes demandas com o atual projeto da Internet. Deste modo, pesquisas são realizadas com o intuito de desenvolver arquiteturas alternativas para a Internet do Futuro (IF), criando-se soluções que proporcionem um progresso ordenado e eficiente da rede [Farias *et al*. 2011].

Neste contexto, o *OpenFlow* [McKeown *et al*. 2008] se destaca como uma solução para garantir a realização de experimentos diretamente sobre equipamentos de produção (ex. redes metropolitanas ou de ensino e pesquisa), sem provocar interferência no tráfego original da rede. Apresentado no âmbito das redes programáveis por *software* (SDN - *Software-Defined Networks*), o protocolo *OpenFlow* propõe um mecanismo que é executado nos comutadores de forma a possibilitar o controle das tabelas de encaminhamento localizadas no plano de dados (*datapath*), através de um elemento central e remoto denominado Controlador.

Desde sua definição, diversas pesquisas, nas mais variadas áreas como virtualização, protocolos ou arquiteturas, são realizadas com o objetivo de contribuir para a evolução e criação de novos elementos que possibilitem a extensão das funcionalidades das redes *OpenFlow*. Neste sentido, o *FlowVisor* [Sherwood *et al*. 2009] destaca-se como um destes elementos, podendo ser integrado à rede com o intuito de aprimorar a alocação dos recursos em um ambiente baseado em *OpenFlow*.

O *FlowVisor* é um controlador específico que atua como um *proxy* entre os dispositivos de rede e controladores *OpenFlow,* criando uma camada de virtualização sobreposta à rede de produção e permitindo o compartilhamento de recursos disponíveis através da criação de várias fatias (ou *slices*), cada uma delas definindo uma rede virtual distinta. Assim, além da separação entre tráfego de pesquisa e produção, é possível a execução simultânea de diversos experimentos em paralelo.

Embora o *FlowVisor* seja capaz de lidar com vários controladores, atribuindo uma fatia para cada um deles, atualmente a definição de uma solução que forneça um controle eficiente para alocação de recursos nas redes virtuais, como provisionamento de largura de banda [Takahashi 2012], ainda é um problema pouco explorado. Além da necessidade de mecanismos para suportar o compartilhamento, as configurações que podem ser realizadas no *datapath* são dependentes de ferramentas externas, como as realizadas estaticamente pelo comando `dpctl` do *framework* OpenFlow ou por um outro protocolo de configuração, dedicado para este fim. Em ambientes virtualizados, considerando a utilização mútua de uma quantidade limitada de recursos, a ausência de mecanismos que garantam o isolamento pode resultar em interferências entre os diferentes ambientes virtuais, de forma que uma única fatia pode causar perturbação em toda infraestrutura.

Desta forma, este artigo apresenta uma solução para o aperfeiçoamento do *FlowVisor*, buscando a separação efetiva de recursos compartilhados entre as fatias de rede. Novos módulos foram implementados agregando funções específicas de controle de tráfego ao controlador. A abordagem de desenvolvimento escolhida inclui a integração do arcabouço denominado *QoSFlow* [Ishimori *et al*. 2012a] à arquitetura atual do *FlowVisor*.

Para validação da solução, foram realizados testes em ambientes onde fluxos em fatias distintas competiam por largura de banda em um *link* compartilhado. A aplicação dos mecanismos propostos para limitação do uso da largura de banda disponível

demonstrou a capacidade da solução em isolar as redes virtuais, resultando na mitigação de interferências entre os fluxos concorrentes.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados ao tema, descrevendo as motivações para o desenvolvimento da solução. A Seção 3 fornece um breve resumo sobre o funcionamento do *FlowVisor e QoSFlow*. A Seção 4 abrange a arquitetura da solução com ênfase nas funcionalidades que foram acrescentadas ao *FlowVisor*. Na Seção 5 são apresentados a análise da proposta e resultados de alguns experimentos executados para sua validação, e por fim, na Seção 6 apresentam-se as considerações finais e as perspectivas de trabalhos futuros.

2. Trabalhos relacionados

Em Sherwood (2009) são apresentados os mecanismos aplicados no *FlowVisor* para o isolamento de recursos entre os *slices,* dentre eles, os utilizados para isolamento de largura de banda. Conforme destaca o autor, apesar da versão do protocolo *Openflow* utilizada no desenvolvimento da ferramenta não implementar mecanismos de gerenciamento de *QoS*, o *FlowVisor,* ainda assim, conseguia alavancar características de isolamento de largura de banda nos *switchs*, utilizando para tanto a atribuição do campo *VLAN PCP* (*Priority Code Point*) nos pacotes. A definição de cada classe de tráfego deveria ser configurada diretamente em cada *datapath* por linha de comando.

Considerando melhorias no *FlowVisor*, Min *et al*. (2012) implementam esquemas para o controle de admissão e garantia de largura de banda mínima. A alocação da banda requerida por um fluxo específico ainda é realizada através da marcação dos bits de prioridade de *VLAN* nos pacotes, porém de forma dinâmica, incluindo novos módulos e interfaces de configuração. Conforme destaca Sherwood (2009), a utilização de bits de *VLAN PCP* é apenas uma solução de curto prazo, demandando em novas versões um controle mais direto e preciso de *QoS*.

Entretanto, versões superiores do *FlowVisor* empregam o protocolo *OpenFlow* v1.0, ao qual apesar de encontrar definida a ação "*ENQUEUE*", utilizada para encaminhar pacotes através de uma fila configurada em uma porta do *datapath*, o *FlowVisor* não previa mecanismos para implementar este controle. Desta forma, apenas a ação de "*OUTPUT*" encontrava-se definida, portanto, mesmo que os controladores fornecessem suporte a *QoS*, ao receber mensagens do tipo "*ENQUEUE*" estas eram descartadas pelo *FlowVisor*.

A versão 0.10 do *FlowVisor* foi liberada ao final de 2012 [Al-Shabibi 2012], incluindo algumas melhorias em relação ao tratamento das mensagens "*ENQUEUE*", além da possibilidade de atribuição de filas aos *flowspaces*, através da definição de novos parâmetros para as entradas de fluxos. Ações de "*OUTPUT*" também podem ser reescritas para ações de "*ENQUEUE*", conforme as regras definidas. A principal limitação da solução ainda reside no fato de serem necessárias que as definições de fila no *datapath* sejam realizadas por ferramentas externas e fortemente dependente de atuação manual, o que restringe o gerenciamento completo das classes de serviço pelo *FlowVisor*.

Em outro trabalho, Salvadori *et al*. (2011) propõem o aperfeiçoamento do *FlowVisor* para permitir uma implementação completa de arquiteturas de redes virtuais, não limitadas apenas à topologia física, criando uma solução denominada *ADVisor*. Dentre as alterações realizadas, estão definidos mecanismos para o isolamento entre as

topologias virtuais. Porém, a solução validada pelos autores ainda inclui a necessidade de configuração manual dos dispositivos de rede e o envio de comandos `dpctl`, não sendo expostas alterações ao protocolo *OpenFlow* que permitam as configurações dos *datapaths* pelo *FlowVisor*, como a alocação de filas, definição de escalonadores e estabelecimento de parâmetros como largura mínima e máxima de banda.

Finalmente, o trabalho de Ishimori *et al*. (2012a), com o objetivo de automatizar e suprir as deficiências de gerenciamento de *QoS* (*Quality of Service*) apresentadas pelo protocolo *OpenFlow*, propõe o *QosSFlow*. Trata-se de um arcabouço de software, originalmente implementado através de extensões ao software *switch* (*datapath*) do *OpenFlow* e de novos módulos no controlador NOX. O *QoSFlow* inclui funcionalidades para administração dinâmica de recursos de *QoS*, como largura de banda, tamanho da fila, ou atraso. Através do controlador é possível a programação de aspectos de *QoS* diretamente nas portas de dispositivos habilitados com *OpenFlow*.

No entanto, a arquitetura *QosFlow* apresentada não contempla sua aplicação em cenários virtuais utilizando *OpenFlow*, devido ao não suporte destes às novas mensagens *QoSFlow*. Desta forma, a aplicação de funções de gerenciamento *QoSFlow* no *FlowVisor* permitiria sua adoção no contexto de redes *OpenFlow* virtualizadas, fornecendo ao *FlowVisor* mecanismos para configuração de classes e disciplinas de filas aos nós *OpenFlow*.

3. Arquitetura *FlowVisor* e *QoSFlow*

Tendo-se em vista as restrições estabelecidas pelos fabricantes de equipamentos em expor o projeto de suas plataformas, o *framework OpenFlow* surge como um padrão para permitir a programabilidade da rede, criando uma camada de abstração das especificações de *hardware* e possibilitando aos pesquisadores a definição de ambientes de experimentação para validação de novas soluções para Internet do Futuro.

A associação de importantes funcionalidades às redes de experimentação, como virtualização e gerenciamento dinâmico de *QoS*, agrega benefícios fundamentais durante a avaliação das novas aplicações. O primeiro requisito, fornecido pela ferramenta *FlowVisor*, proporciona um melhor aproveitamento da infraestrutura, considerando a capacidade de execução de múltiplos experimentos sobre o mesmo substrato físico. A presença da segunda característica, oportunizada pelo *QoSFlow*, viabiliza a proposição e avaliação de modelos e técnicas para aplicações mais sensíveis ao atraso [Ishimori *et al*. 2012a]. Os subitens seguintes apresentam uma visão geral do funcionamento destas ferramentas.

3.1 Virtualização sobre *OpenFlow*

Através da definição de uma camada de virtualização para o *OpenFlow,* o *FlowVisor* [Sherwood *et al* 2009] é apresentado como uma abordagem para permitir que o mesmo *hardware* do plano de dados possa ser compartilhado entre múltiplas redes virtuais, cada uma com lógicas de encaminhamento distintas. Sendo assim, diferentes redes podem utilizar os recursos do equipamento de forma independente e aplicando seu próprio controlador.

O *FlowVisor* é um controlador específico que opera de forma transparente entre os *datapaths OpenFlow* e múltiplos controladores. As mensagens *OpenFlow* enviadas de e para os controladores são interceptadas pelo *FlowVisor* e encaminhadas de acordo

com as redes virtuais estabelecidas, através de um conjunto de fluxos definidos para cada *slice*, também chamados de "espaço de fluxo" (ou *flowspace*).

A Figura 1 ilustra, de maneira geral, os procedimentos implementados pelo *FlowVisor* para gerenciamento das fatias de rede. Além do controlador de produção, outros dois controladores são apresentados, identificados por "Alice" e "Bob", o que determina, consequentemente, a presença de três *slices*. A dinâmica de processamento das mensagens interceptadas pelo *FlowVisor* está representada pelos círculos numerados. Inicialmente, mensagens *OpenFlow* originadas de um controlador são interceptadas **(1)**. Aplicando a política do espaço de fluxos definida para a fatia **(2),** as mensagens são reescritas de forma transparente, a fim de garantir o controle de determinada fatia da rede que compõe o plano de dados **(3)**. Da mesma forma, as mensagens provenientes dos comutadores para o plano de controle **(4)** são novamente interceptadas, reescritas e encaminhadas ao controlador correspondente, de acordo com a política de definição da rede virtual.

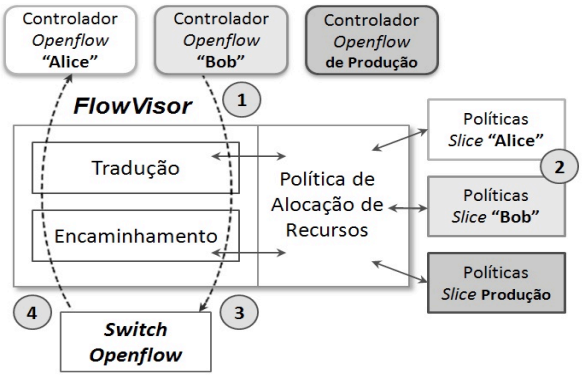

Figura 1. Visão geral de funcionamento do *FlowVisor*. Adaptada de [Sherwood *et al* 2009]

Ressalta-se que, no contexto das redes virtuais, a correta separação de alguns recursos da rede é considerada indispensável para garantir uma completa virtualização. Dentre eles Sherwood *et al*. (2010b) destaca: topologia, largura de banda, capacidade de processamento e tabelas de encaminhamento. Especificamente em relação à largura de banda, os autores descrevem que cada fatia deve possuir sua própria fração, de tal forma que, falhas no isolamento deste recurso acarretariam em interferências entre os *slices*, podendo comprometer o rendimento de cada um.

Neste mesmo sentido, Kanada *et al*. (2012) evidenciam como um requisito chave para alcançar a virtualização de rede o isolamento de recursos entre fatias, ou seja, permitir que uma rede virtual utilize os recursos necessários para fornecer o desempenho esperado, mesmo quando um congestionamento ocorre na rede física ou em outras fatias. De acordo com os mesmos autores, duas funções de controle de tráfego, geralmente utilizadas para garantia de *QoS*, isto é, modelagem e policiamento de tráfego, devem ser aplicadas para o isolamento de recursos.

3.2 *QoSFlow*: aplicando QoS dinâmico sobre *Openflow*

Conforme descrito por Ishimori *et al*. (2012b), a arquitetura modular do *QoSFlow*, representada na Figura 2, é composta por dois elementos principais: o controlador e o *datapath QosFlow*. O módulo controlador foi desenvolvido como uma API (*Application Programming Interface*) sobre o NOX, onde novas mensagens de controle foram implementadas para permitir o gerenciamento dos recursos de *QoS* no plano de dados.

As alterações no *datapath* incluem subcomponentes para o recebimento e processamento de mensagens de *QoS* adicionadas ao protocolo *OpenFlow*.

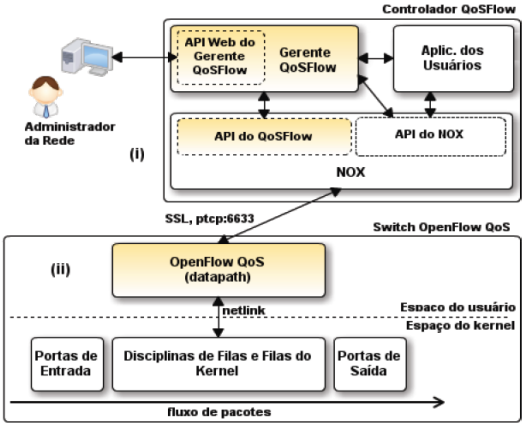

Figura 2. Visão geral da arquitetura *QoSFlow*. Fonte: [Ishimori et al. 2012a]

Considerando a necessidade de aplicação de ferramentas para controle de tráfego pelo *FlowVisor* e a capacidade de gerenciamento de *QoS* proporcionada pelo *QoSFlow* [Ishimori *et al*. 2012a] em domínios *OpenFlow*, optou-se pelo desenvolvimento de um módulo controlador *QosFlow* sobre o *FlowVisor*. Assim sendo, torna-se possível a programação dinâmica de aspectos de *QoS* nas portas dos dispositivos de rede habilitados com *OpenFlow*, proporcionando a definição de classes e disciplinas de fila e, consequentemente, o controle de tráfego nos *datapaths*, a partir do controlador.

4. Solução de Integração de *QoS* sobre virtualização *OpenFlow*

Um projeto para fornecimento de controle de recursos, através do suporte à ação "ENQUEUE" pelo *FlowVisor*, é apresentado por Takahashi (2012). São descritos alguns dos passos necessários para resolução do problema, incluindo: atualização do pacote *OpenFlowJ* [OpenFlowJ 2012] – implementação em Java do protocolo *OpenFlow* – para suportar mensagens de obtenção das configuração de filas a partir do *datapath*; implementação de API para atribuição de fila aos *slices*; e substituição de ações "OUTPUT" por "ENQUEUE". Na proposta não são considerados mecanismos para configuração de filas nos *datapaths*, assumindo o autor que as filas já foram apropriadamente configuradas por ferramentas externas.

Desta forma, considerando o projeto descrito anteriormente e suas limitações quanto à configuração apropriada dos *datapaths*, os procedimentos selecionados para o desenvolvimento da solução consistem basicamente em três etapas: (a) atualização do pacote *OpenFlowJ*, para que o *FlowVisor* interprete e envie mensagens *QoSFlow*; (b) desenvolvimento de uma API de gerenciamento para interagir com o *datapath QoSFlow*, transmitindo mensagens de configuração de parâmetros de *QoS*, bem como possibilitar a atribuição ou modificação de filas aos *slices*; (c) implementação de um conversor para substituir ações nas mensagens *Openflow* recebidas a partir dos controladores. As etapas estão ilustradas na Figura 3.

As alterações no pacote *OpenFlowJ* incluem a definição de mensagens *OpenFlow* do tipo "*QOS_DISCIPLINE*", que são interpretadas pelo *datapath QosFlow*, permitindo a configuração de classes, disciplinas de fila e seus parâmetros, tais como: HTB (*Hierarchical Token Buckets*); PFIFO e BFIFO (*Packet e Byte First In, First Out*); SFQ (*Stochastic Fairness Queueing*) e RED (*Random Early Detection*).

Optou-se pela inclusão de uma nova API, identificada como "*fvctlQoS*", destinada apenas ao tratamento de aspectos de *QoS* e atribuição de filas aos *slices*, mantendo-se inalterada a API original do *FlowVisor* "*fvctl*". As informações das classes configuradas e a identificação das filas atribuídas a cada *slice* são armazenadas no arquivo de configuração do *FlowVisor*.

O papel do conversor de mensagens é tornar transparente aos controladores a utilização de filas. Ou seja, o contrador envia mensagens com ações de "OUTPUT" e o *FlowVisor*, a fim de garantir o controle na utilização dos recursos altera as ações para "ENQUEUE".

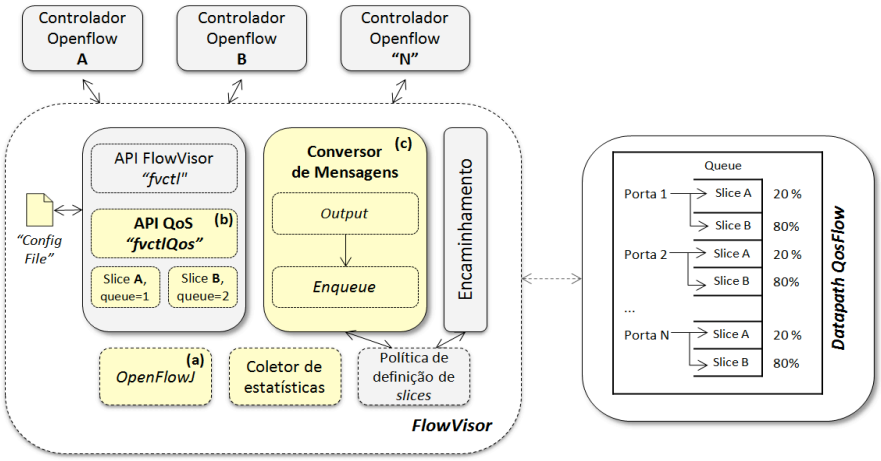

Figura 3. Arquitetura da proposta

A Figura 3 representa a arquitetura proposta para a solução. Seus componentes são descritos nos subitens seguintes:

- **API *"fvctlQoS"***: Permite a interação com o *datapath* para transmissão de mensagens de *QoS*. Desta forma, é possível a implementação de aspectos de *QoS* nos dispositivos de rede *QosFlow*, incluindo, por exemplo, a configuração de disciplinas de fila FIFO ou HTB, bem como das classes de fluxos. Além disso, a API permite configurar ou modificar a atribuição de fila para um *slice*.
- **Conversor de Mensagens**: Quando mensagens *FlowMod* ou *PacketOut*, com uma ação "OUTPUT", são recebidas pelo *FlowVisor*, a partir de um controlador, caso o *slice* de destino esteja atribuído à uma fila, o *FlowVisor* substitui a ação "OUTPUT" por uma "ENQUEUE". O campo "*queue_id*" da ação "ENQUEUE" é preenchido com base na definição realizada para o *slice* através da API *"fvctlQoS"*.
- **Política de definição dos *slices***: Os conjuntos de fluxos são então mapeados para suas respectivas filas de acordo com as especificações de *FlowSpace* definidas para cada *slice*, ou seja, conforme as políticas de definição dos *slices* estabelecidas.
- ***Datapath QosFlow***: O *datapath OpenFlow* habilitado com *QosFlow* interpreta as mensagens de configuração recebidas a partir do *FlowVisor*, aplicando automaticamente as definições de *QoS* estabelecidas.

- **Coletor de estatísticas**: Possibilita a obtenção, a partir de um *datapath*, de estatísticas de *QoS*, como o número total de pacotes e *bytes* transmitidos pelas portas de saída.

A Figura 4 apresenta um fluxograma contemplando as atividades executadas no *FlowVisor* durante a configuração de classes e atribuição aos *slices*. Algumas etapas, por exemplo, as mensagens de erro de sintaxe para os comandos, foram suprimidas, a fim de fornecer uma visão geral e simplificar o diagrama. Na mesma Figura, se encontram em destaque os passos introduzidos pela solução desenvolvida.

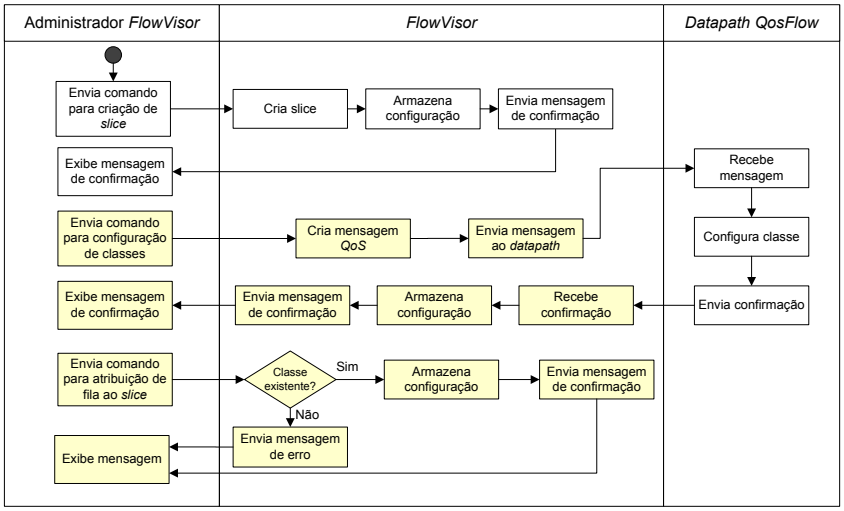

Figura 4. Fluxograma de definição de classes e atribuição aos *slices*

Sendo assim, com a implementação de módulos específicos, considerando os pontos aqui destacados, o *FlowVisor* é habilitado para o gerenciamento e alocação de recursos de *QoS* no *datapath*, além da definição de filas aos *slices*, garantindo que cada um utilizará apenas a fatia a ele atribuída. Testes foram realizados com o intuito de constatar esta capacidade.

5. Testes de validação

Dois cenários foram aplicados durante a realização dos primeiros testes para validação da solução proposta. Conforme representado na Figura 5(a), o primeiro consiste em um cenário semelhante ao apresentado em Sherwood *et al*. (2009) e Sherwood *et al*. (2010b) durante testes de validação de isolamento de largura de banda, incluindo uma topologia *OpenFlow* básica. No segundo, por sua vez, foram incluídos novos dispositivos, considerando-se, ainda, a presença de três controladores – Figura 5(b).

Os testes foram realizados com o intuito de confirmar a capacidade da solução em limitar a largura de banda disponibilizada a diferentes *slices* e, consequentemente, atestar a diminuição de interferências entre os fluxos concorrentes. Os cenários e resultados encontrados estão detalhados nos subitens seguintes.

5.1. Descrição dos cenários

5.1.1. Cenário (a)

Para os experimentos executados no primeiro cenário, uma topologia *OpenFlow* básica foi utilizada. Conforme representado na Figura 5(a), 02 (duas) máquinas foram utilizadas: uma como origem dos fluxos, rodando duas instancias cliente, e outra como

um destino comum executando duas instancias servidor, conectadas através de um *datapath Openflow* com *QoSFlow* implementado. O comutador utilizado em todos os experimentos foi o: TP-Link modelo TL-WR1043ND, com sistema operacional OpenWRT versão 10.3.1 (*backfire*). Apesar de se tratar de um comutador sem fio, foram utilizadas apenas as interfaces *ethernet* cabeadas.

Dois controladores NOX foram aplicados para avaliação e encaminhamento de fluxos em *slices* distintos. O experimento executado consistiu na aplicação de fluxos TCP e UDP, competindo por largura de banda em um *link* compartilhado. Conforme destaca Sherwood *et al*. (2009) estes padrões de tráfego são representativos de um cenário onde um *slice* de produção (TCP) compartilha um *link* com, por exemplo, um *slice* (UDP) executando um experimento DDoS (*Distributed Denial-of-Service*).

Para geração dos tráfegos foi adotada a ferramenta *iperf* [Iperf 2012]: configurada no modo TCP (porta 8010) para produção do fluxo a ser transmitido pelo primeiro *slice* e no modo UDP (porta 8011) para o fluxo do segundo. As configurações de *FlowSpace* foram realizadas para tratamento do tráfego *iperf* com base nas portas de origem e destino, conforme estabelecidas para cada protocolo.

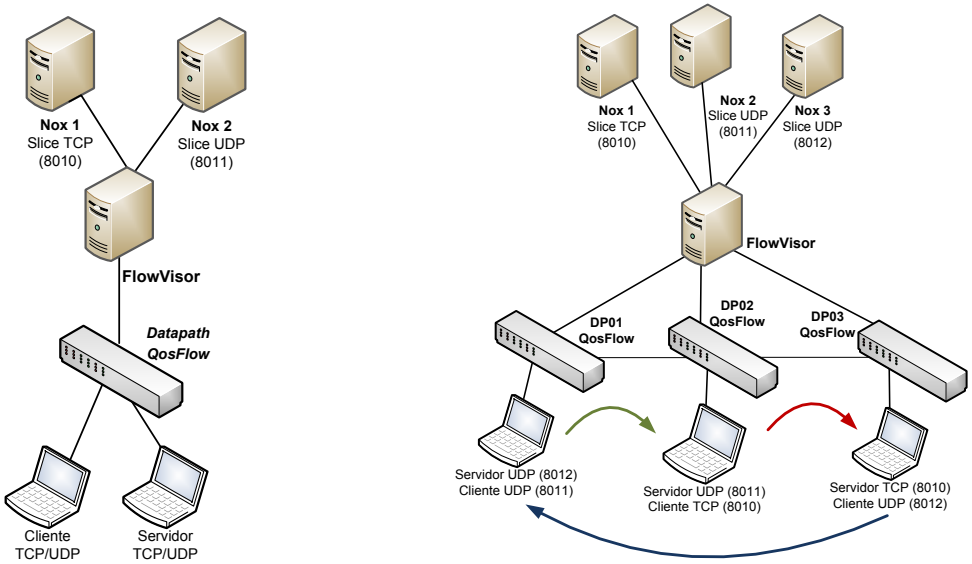

Figura 5. Cenários de testes

5.1.2. Cenário (b)

Neste cenário novos nós foram aplicados à topologia, incluindo três *datapaths* com *QosFlow* habilitado e três controladores NOX conectados ao *FlowVisor*. Para cada um dos três nós, conectados a *datapaths* distintos, foram iniciadas uma instância cliente e uma instância servidor.

Três *slices* foram configurados: o primeiro para lidar com tráfego TCP na porta 8010 (representado em vermelho na Figura 5(b)), o segundo com fluxo UDP na porta 8011 (em verde), e o terceiro também com fluxo do tipo UDP, porém na porta 8012 (em azul). As setas representam o direcionamento estabelecido para cada fluxo entre as instâncias cliente/servidor. A ferramenta *iperf* também foi utilizada neste caso para geração dos tráfegos em cada *slice*.

Em cada um dos cenários apresentados, os experimentos executados consistiram em duas etapas. Na primeira os fluxos foram gerados em cada *slice* sem os mecanismos de isolamento habilitados. Na segunda etapa os recursos para isolamento de largura de banda foram aplicados, através da configuração de disciplinas de filas e classes no *datapath,* além da atribuição de cada *slice* a uma destas classes. Os resultados encontrados estão detalhados nos subitens seguintes.

5.2. Resultados

5.2.1. Resultados para os experimentos no cenário (a)

Conforme destacado anteriormente, na primeira fase do experimento os fluxos foram gerados em cada *slice* sem os recursos de isolamento habilitados. Através do gráfico da Figura 6 é possível observar que o compartilhamento de banda entre *slices* TCP e UDP faz com que a presença de um afete o desempenho do outro. No tempo t=30seg o tráfego UDP, configurado para utilizar aproximadamente a taxa de recebimento de *bits* completa alcançada (50Mbps), é iniciado. Percebe-se a utilização de considerável fatia da banda disponível, causando queda na taxa de transmissão do fluxo TCP. A duração definida para o fluxo foi de 60 segundos e quando finalizado percebe-se o retorno do TCP ao seu estado inicial.

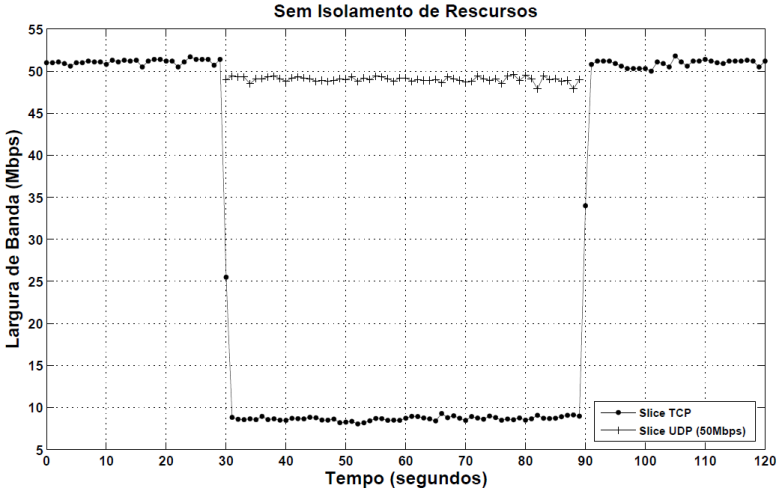

Figura 6. Resultado para o cenário (a) sem isolamento de recursos

As características inerentes a cada um dos protocolos utilizados contribuiu para a ocorrência das interferências entre os fluxos no cenário aplicado, permitindo a avaliação do comportamento da rede, com e sem a aplicação de mecanismos para isolamento de recursos. Quando fluxos TCP e UDP tentam competir por largura de banda, esta não é equitativamente partilhada. O comportamento padrão do protocolo UDP é considerado mais simples, não provendo mecanismos de controle de fluxo ou retransmissões, buscando utilizar largura de banda tanto quanto disponível.

Em contrapartida, as características básicas do protocolo TCP incluem a utilização de janelas deslizantes para controle de fluxo, confirmações e retransmissões. Congestionamentos na rede são detectados pelo protocolo através da perda de pacotes. Quando isso ocorre, o pacote é retransmitido, e ao mesmo tempo, o TCP reduz o tamanho da sua janela de congestionamento, efetivamente reduzindo sua taxa de saída para evitar congestionamentos adicionais. Na ausência de congestionamento, o TCP aumenta o tamanho de sua janela de congestionamento e a taxa de saída.

Portanto, na segunda etapa os recursos para isolamento de largura de banda foram aplicados, através da configuração de disciplinas de filas e classes no *datapath*, além da atribuição de cada *slice* a uma destas classes, conforme descrito a seguir:

- *Slice* TCP (8010): atribuído à "Classe 1", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 15Mbps;
- *Slice* UDP (8011): atribuído à "Classe 2", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 6Mbps;

Da mesma forma que na etapa anterior, iniciou-se pela aplicação do fluxo TCP, seguido pela transmissão do tráfego UDP. Conforme representado da Figura 7, mesmo com o início do fluxo UDP no tempo t=20s (neste caso configurado para tentar utilizar 30Mbps, aproximadamente a taxa de transmissão constatada quando o *QoSFlow* está habilitado no *datapath*), o fluxo TCP consegue manter a taxa de transmissão configurada para sua classe. Da mesma forma, a taxa verificada para o UDP não ultrapassou o limite de 6Mbps estabelecido para a classe.

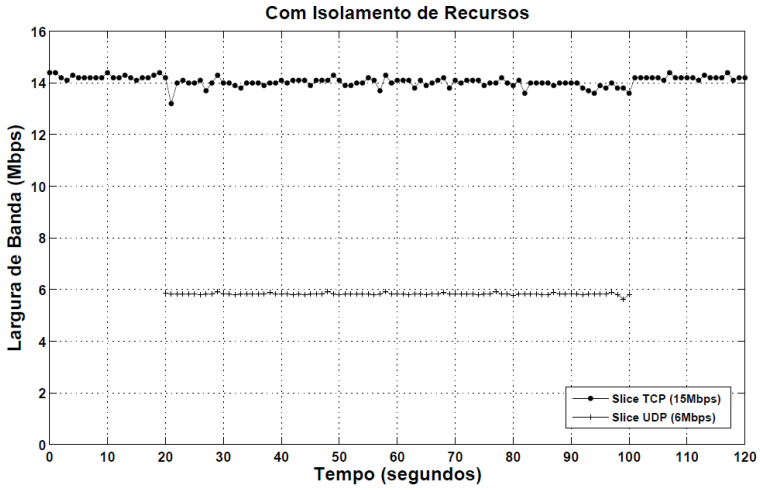

Figura 7. Resultados para o cenário (a) com isolamento de recursos habilitado

5.2.2. Resultados para os experimentos no cenário (b)

Como no cenário anterior, duas etapas foram executadas. Na primeira, fluxos foram gerados em cada *slice*, em tempos distintos, sem a configuração das propriedades de controle de tráfego. O gráfico da Figura 8 apresenta os resultados encontrados. O inicio de fluxos UDP nos tempos t=20seg e t=40seg, cada um com duração de 40 segundos e configurados para transmissão a uma largura de banda de 25Mbps, afetam a taxa de transmissão no *slice* TCP. É perceptível, ainda, uma pequena interferência entre os fluxos UDP, no intervalo de tempo em que são transmitidos em conjunto.

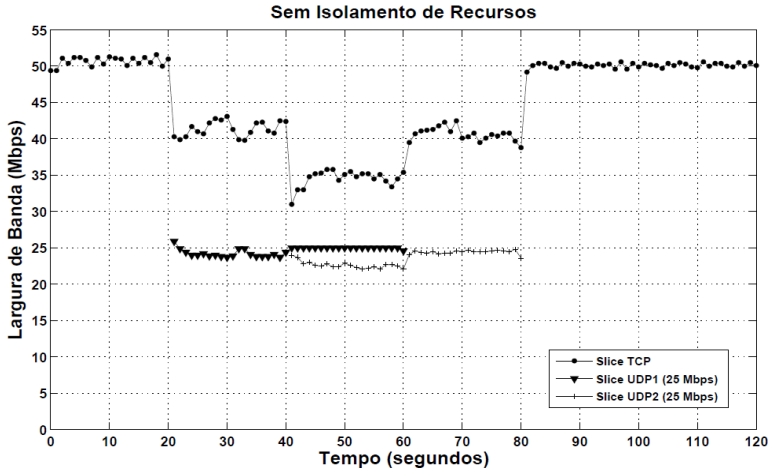

Figura 8. Resultados para o cenário (b) sem isolamento de recursos

Como descrito anteriormente, na segunda etapa os mecanismos para isolamento de largura de banda foram habilitados, através da configuração de disciplinas de filas e classes nos *datapaths*, além da atribuição de cada *slice* a uma destas classes, conforme descrito a seguir:

- *Slice* TCP (8010): atribuído à "Classe 1", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 10Mbps;
- *Slice* UDP (8011): atribuído à "Classe 2", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 5Mbps;
- *Slice* UDP (8012): atribuído à "Classe 3", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 3Mbps;

Da mesma forma que na etapa anterior, iniciou-se pela aplicação do fluxo TCP, seguido pela transmissão de cada um dos tráfegos UDP, nos tempos t=20seg e t=40seg, cada um com duração de 40 segundos. O gráfico da Figura 9 apresenta os resultados encontrados. Mesmo com o aumento no número de nós e o estabelecimento de mais uma classe, foi possível constatar a capacidade de isolamento, onde para cada *slice* verificou-se a manutenção da taxa média de banda garantida, conforme configurado em cada classe.

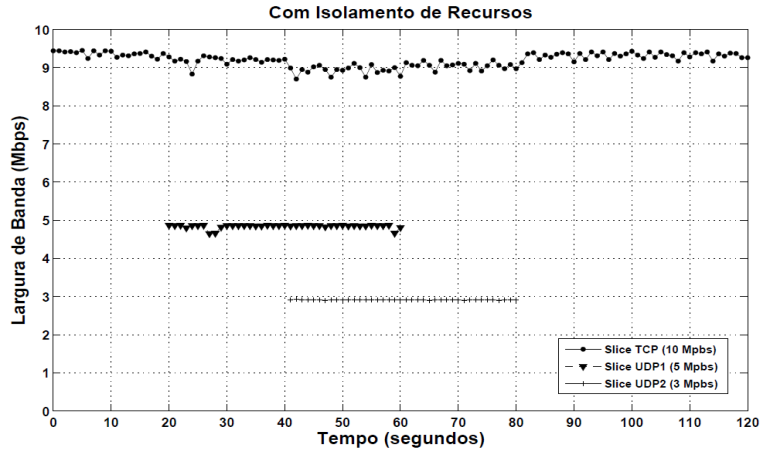

Figura 9. Resultados para o cenário (b) com isolamento de recursos habilitado

6. Considerações Finais e Trabalhos Futuros

A virtualização de redes tem sido considerada por pesquisadores como uma importante ferramenta para o desenvolvimento e validação de novas tecnologias voltadas à Internet do Futuro. Dentre seus benefícios destaca-se a possibilidade de definição de ambientes para a execução de múltiplos experimentos simultâneos e independentes. No âmbito de redes *OpenFlow*, algumas questões ainda estão em aberto para a definição de uma solução completa de virtualização dos recursos, já que o *FlowVisor* apresenta algumas restrições, dentre elas a necessidade de uma solução para o isolamento dos recursos compartilhados entre os diferentes *slices*.

Este artigo apresentou uma solução para extensão das funcionalidades do *FlowVisor*, incluindo novos módulos que permitem a configuração de parâmetros de controle de tráfego nos dispositivos de rede, a fim de prover funções para garantir que uma rede virtual mantenha seu desempenho, independente de eventos que possam ocorrer nos demais *slices*.

A arquitetura da solução apresentada inclui a utilização de *datapaths* configurados com *QoSFlow* e a implementação de um módulo de gerenciamento *QosFlow* no *FlowVisor*, considerando que o *QosFlow* foi inicialmente desenvolvido como uma API sobre o NOX. Desta forma, são habilitadas funcionalidades para administração e configuração de recursos de *QoS* nos dispositivos, como largura de banda, tamanho da fila, ou atraso.

O enfoque inicial da solução desenvolvida tem por finalidade evitar interferências de fluxos entre diferentes *slices* e não entre fluxos dentro de um mesmo *slice*. Esta última abordagem será considerada em trabalhos futuros, pois demanda, também, alterações na implementação do *datapath QosFlow*.

Experimentos realizados para validação da solução permitiram a comparação do comportamento da rede, com e sem a aplicação dos mecanismos definidos para o isolamento de recursos. Demonstrou-se que, com a definição de classes nos dispositivos e sua atribuição aos *slices*, foi possível o isolamento da largura de banda disponível em um ambiente onde fluxos de *slices* distintos competiam pelo uso da banda em um *link* compartilhado.

Outros trabalhos futuros envolverão a realização de novos experimentos e a evolução da solução, incluindo, por exemplo, mecanismos de controle de admissão e a possibilidade de sua aplicação em ambientes que implementem hierarquia de *FlowVisors*, através da troca de informações de configuração de filas entre estes controladores específicos e a definição de políticas para redistribuição dos recursos disponíveis à fluxos dentro de um mesmo *slice*.

Referências

Al-Shabibi, A. "*FlowVisor 0.10.0 released*". Disponíel em: <https://mailman.stanford.edu/pipermail/openflow-discuss/2012-December/003900.html>. Dezembro, 2012.

Farias, F; Júnior, J.; Salvatti, J.; Silva, S.; Abelém, A.; Salvador, M.; e Stanton, M. "*Pesquisa experimental para a Internet do Futuro: uma proposta utilizando virtualização e o framework OpenFlow*". In: Fabiola Greve; Ronaldo Alves. (Org.). Minicursos. 1 ed. Porto Alegre - RS: Editora da SBC, 2011, v. 1, p. 1-61.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). "*Nox: towards an operating system for networks*". ACM SIGCOMM Computer Communication Review, 38(3):105–110.

Iperf. Disponível em <http://iperf.sourceforge.net/>. Acesso em: 10/12/2012.

Ishimori, A.; Salvatti, J; Farias, F; Gaspary, L; Granville, L; Cerqueira, E; Abelém, A. *"QoSFlow: Gerenciamento Automático da Qualidade de Serviço em Infraestruturas de Experimentação Baseadas em Framework OpenFlow"*. In: III Workshop de Pesquisa Experimental da Internet do Futuro - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012), Ouro Preto, 2012a.

Ishimori, A.; Farias, F.; Carvalho, I.; Cerqueira, E.; Abelém, A. "*Automatic QoS Management on OpenFlow Software-Defined Networks*". 7th API Think-Tank - Software Defined Networking. Junho, 2012b.

Kanada, Y; Shiraishi, K; and Nakao, A. "*Network-resource Isolation for Virtualization Nodes"*. Fourth International Conference on Communication Systems and Networks (COMSNETS). Jan, 2012.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; and Turner, J. "*OpenFlow: Enabling innovation in campus networks*". ACM SIGCOMM Computer Communication Review, 38(2):69-74, 2008.

Min, S.; Kim, S.; Lee, J.; Kim, B.; Hong, W.; and Kong, J. *"Implementation of an OpenFlow Network Virtualization for Multi-Controller Environment"*. 14th International Conference on Advanced Communication Technology (ICACT). Feb, 2012.

OpenFlowJ. OpenFlow Java. Disponível em: <https://openflow.stanford.edu/fisheye /browse/OpenFlowJ> . Acessado em Novembro, 2012.

Salvadori, E.; Corin, D.; Broglio, A.; Gerola, M., "*Generalizing Virtual Network Topologies in OpenFlow-Based Networks*". GLOBECOM 2011 IEEE, p.1,6, 5-9 Dec. 2011.

Sherwood, R.; Gibb, G; Yap , K.; Appenzeller, G; Casado, M.; McKeown, N; and Parulkar, G. "*FlowVisor: A network virtualization layer*". Technical Report Openflow-tr-2009-1, Stanford University, 2009. Disponível em: <http://www.openflow.org/downloads /technicalreports/openflow-tr-2009-1-flowvisor .pdf>.

Sherwood, R.; Chan, M.; Gibb, G.; Handigol, N.; Huang, T.; Kazemian, P.; Kobayashi, M.; Underhill, D.; Yap, K.; Appenzeller, G.; and McKeown, N. (2010a). "*Carving research slices out of your production networks with OpenFlow*". SIGCOMM Computer Communication Review, V. 40, nº. 1, p. 129–130, Jan. 2010.

Sherwood, R.; Gibb, G.; Yap, K; Appenzeller, G.; Casado, M.; McKeown, N.; and Parulkar, G. (2010b). "*Can the Production Network Be the Testbed?*". Operating Systems Design and Implementation - OSDI 2010, Vancouver, BC, Canada, Oct. 2010.

Takahashi, Masahiko. "*Design Documents: ENQUEUE action support*". Disponível em: https://openflow.stanford.edu/display/DOCS/ENQUEUE+action+support. Acesso em: 15/09/2012.