Aprovisionamento de QoS por Rótulos Programáveis para Redes Definidas por Resíduos

Wallas F. Oliveira[1], **Lucas S. Santos**[1], **Magnos Martinello**[2], **Leobino N. Sampaio**[1]

[1]Programa de Pós-graduação em Ciência da Computação (PGCOMP)
Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Salvador – BA – Brasil

[2]Universidade Federal do Espírito Santo (UFES)
Vitória – ES – Brasil

{wallas.froes, lucassouzasantos, leobino}@ufba.br, magnos@inf.ufes.br

Abstract. *Most of the current network solutions to provide quality of service is strongly dependable on MPLS protocol which allows the assignment of QoS to different classes of traffic. Despite the benefits, these solutions present high OPEX and CAPEX costs allowing the vendors crucially lock-in the network architecture. Residues Defined Networks (RDN) has emerged with the purpose of enabling low cost solutions simplifying the packet forwarding operation based simply on the remainder of division. This paper presents a QoS provisioning by Programmable Label model for RDNs. The model has been implemented and validated over the P4 language. Emulation tests were realized evaluating QoS and QoE metrics. In particular, the gain on the programmability of QoS guarantees configured on demand is showed demonstrating the feasibility of the model.*

Resumo. *Grande parte das atuais soluções adotadas no provimento de qualidade de serviço em redes é fortemente baseada no protocolo MPLS, que permite a associação de serviços de QoS para diferentes classes de tráfego. Apesar dos seus benefícios, tais soluções apresentam altos custos de OPEX e CAPEX. Redes Definidas por Resíduos (RDN) surgem na perspectiva de viabilizar soluções de baixo custo para o encaminhamento de pacotes baseado simplesmente em operações de resto da divisão (i.e., resíduo). Este trabalho apresenta um modelo para o aprovisionamento de QoS por rótulos programáveis para RDNs. O modelo foi implementado através da linguagem P4. Testes de emulação foram realizados avaliando métricas de QoS e QoE, demonstrando a viabilidade do modelo, mas principalmente ganho na programabilidade de garantias de QoS ajustadas sob demanda.*

1. Introdução

A arquitetura de redes tem evoluído fortemente para sistemas complexos de gerenciar, extremamente caros e principalmente sujeitos ao travamento (*lock-in*) dos fabricantes. Embora a comunidade de redes tenha dedicado um enorme esforço para avançar a arquitetura em direção à sistemas abertos e programáveis, há um conjunto de problemas ainda não resolvidos no projeto de arquitetura de redes programáveis. Mesmo as arquiteturas que seguem o paradigma das Redes Definidas Por *Software*, (do inglês, *Software-Defined*

Networking – SDN) [Kreutz et al. 2015], apresenta questões em aberto. Um dos pontos fundamentais das SDNs tem sido o desacoplamento entre plano de controle e plano de dados. O plano de controle trata com decisões computacionais mais complexas, enquanto o plano de dados meramente processa pacotes baseado em tabelas projetadas a partir de um modelo de encaminhamento *matching-action*.

Apesar das suas contribuições, no plano de dados de arquiteturas SDN não há uma distinção clara entre as partes funcionais dos elementos de borda e núcleo. Essa tarefa é delegada para o projetista de rede e não está prevista na arquitetura de rede. Isso significa que todo o processamento de pacotes é feito baseado em um único modelo de encaminhamento, independentemente se o elemento de rede estiver posicionado na borda ou no núcleo. Por exemplo, considerando um comutador OpenFlow, isso implica que todas as operações de busca em tabela (*lookup table*) são feitas em centenas de *bits* com ações complexas que podem precisar de múltiplas tabelas.

O conceito de Redes Definidas por Resíduos (do inglês, *Residue Defined Network* – RDN) foi proposto em [Martinello et al. 2017] com objetivo de definir blocos funcionais claros construindo um padrão de projeto (*Design Pattern*) na arquitetura SDN [Jyothi et al. 2015]. O conceito utiliza o Sistema Numérico de Resíduos (do inglês, *Residue Number System* – RNS) [Chang et al. 2015] como fundamento para os elementos de núcleo, simplificando drasticamente o modelo de encaminhamento dos comutadores de núcleo. Essa arquitetura contém 3 elementos de rede, são eles: o controlador, comutadores de núcleo e de borda. O controlador tem as funções de selecionar as rotas entre comutadores de borda, computar os rótulos para essas rotas e repassá-las configurando os comutadores de borda. Os comutadores de borda tem a função de inserir os rótulos no cabeçalho do pacote e os comutadores de núcleo atuam simplesmente na operação de resto da divisão [de Oliveira et al. 2013, Martinello et al. 2014].

Mais especificamente nos comutadores de núcleo, o funcionamento da RDN consiste em aplicar o restante (resíduo) da divisão entre identificadores da rota (rótulo) e identificadores dos comutadores de núcleo (números co-primos). Como resultado dessa operação, tem-se a porta de saída que os pacotes devem ser encaminhados. Apesar do tempo de comutação entre portas atingir $0.6\ \mu s$ (microsegundos), sem *jitter* no núcleo, não existe ainda tratamento distinto entre as filas de saída dos comutadores para oferecer garantias de QoS (do inglês, *Quality of Service*). Este trabalho, portanto, propõe um modelo para aprovisionamento de QoS por rótulos programáveis para RDN (*QoS by Programmable Labels* – QoSProLab) que faz priorização de classes de tráfego durante o encaminhamento de fluxos no núcleo da rede. O mérito da proposta está nas suas características, que oferece: i) agilidade ao associar os rótulos na borda sem precisar atualizar tabelas, em contraste as abordagens anteriores; e ii) flexibilidade no uso dos rótulos, que foram implementados através da linguagem P4 [Bosshart et al. 2014], a qual permite a utilização de um campo de cabeçalho variável para processamento dos pacotes. O modelo foi avaliado a partir de um estudo experimental que permitiu atestar a viabilidade da proposta e os benefícios obtidos a partir da priorização das classes de tráfego por rótulo (CTR), implementada nas filas lógicas dos comutadores de núcleo.

Este artigo está organizado da seguinte forma: a Seção 2 descreve o modelo e seus principais elementos responsáveis no encaminhamento dos fluxos; a Seção 3 descreve a implementação do modelo, que envolveu o desenvolvimento de comutadores de borda e

núcleo através da linguagem P4 para processar pacotes de nível 2; a Seção 4 descreve o estudo experimental conduzido em laboratório para atestar a viabilidade do modelo, apresenta e discute os seus resultados; A Seção 5 discute os trabalhos relacionados e, por fim, e a Seção 6 expõe as conclusões do trabalho e aponta direções para trabalhos futuros.

2. Rótulos Programáveis para Redes Definidas por Resíduos

Esta seção descreve o modelo QoSProlab. O modelo provê o aprovisionamento de QoS utilizando classes de tráfego por rótulos programáveis em RDNs. As próximas subseções descrevem o modelo de forma geral, detalha o protocolo flexível proposto, o papel do controlador e os comutadores da RDN.

2.1. Descrição geral do modelo

O QoSProLab (*QoS by Programmable Labels*) consiste num modelo de encaminhamento que faz uso de rótulos programáveis para definir classes de tráfego utilizadas no aprovisionamento flexível e escalável de QoS para aplicações de rede. A Figura 1 apresenta o cenário geral no qual o modelo QoSProLab é adotado. O cenário é composto por 4 comutadores S de núcleo, que aqui são identificados pelas chaves (*ID-switch*) 13, 5, 7 e 11 (números co-primos atribuídos pelo controlador). Como pode ser observado, através do controlador RDN, o plano de controle utiliza os rótulos programáveis para gerar não apenas um caminho, mas também prover prioridade aos fluxos no encaminhamento de núcleo, através de filas lógicas.

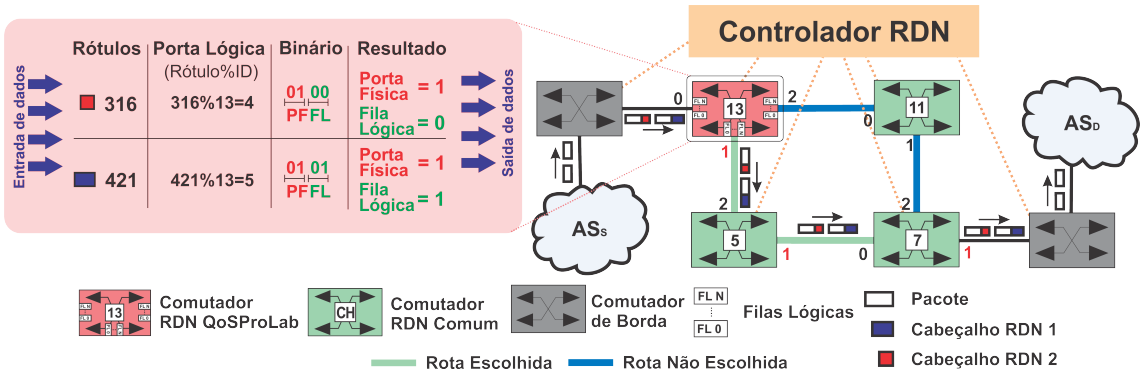


Figura 1. Modelo QoSProLab.

Uma rota em RDN é definida como um conjunto de operação de resíduos, obtida por meio do rótulo embutido no pacote pelo comutador de borda e identificadores (*ID-switch*) atribuídos aos comutadores de núcleo. Esses rótulos permitem aos elementos de núcleo guiar os pacotes pela rede. Assim, para estabelecer a comunicação entre os sistemas autônomos *AS* da Figura 1, o controlador precisa calcular o número (*Rótulo*) para o qual os resultados das operações de módulo direcionarão os pacotes até o destino.

O funcionamento do QoSProLab, portanto, está diretamente relacionado ao uso de rótulos programáveis baseado em resíduos. O resto da divisão informa qual porta de saída o pacote deve ser encaminhado, e uma fração de rótulos é destinada para atender as classes de tráfego por porta física (*PF*), associando à fila lógica (FL) de saída. No exemplo, o controlador definiu a rota através dos comutadores S_{13}, S_5 e S_7. Esse conjunto de comutadores de núcleo compõem a rota. Para ilustrar o conceito, assume-se aqui

quatro classes de prioridade no comutador S_{13}, que possui 3 portas físicas: *PF0*, *PF1* e *PF2*. Para o encaminhamento do pacote, o controlador calcula os rótulos correspondentes a porta física desejada com a respectiva fila lógica. No exemplo da Figura 1, os rótulos calculados são $R_4 = 316$, $R_5 = 421$, $R_6 = 71$ e $R_7 = 177$ para enviar para a *PF1*. O controlador envia tais rótulos com a suas respectivas escolhas de prioridade para os comutadores de borda (em cinza) que, por sua vez, instalam as regras nas tabelas de fluxo. Posteriormente, o comutador de ingresso fica responsável por incorporar os rótulos R em cada pacote.

Uma vez que o pacote entrou no núcleo, em cada comutador é calculado o restante da divisão (denotado como $< a >_b \equiv a \; modulo \; b$) entre o rótulo R localizado no pacote e o respectivo *ID-switch*, definindo a fila lógica e a porta de saída apropriada para encaminhá-lo. No exemplo da Figura 1, utiliza-se o rótulo 316 para identificação da porta física *PF1* e fila lógica *FL0*. Esse cálculo é feito a partir do resto da divisão ($316\%13 = 4$), cujo a representação binária é 0100. Os dois bits mais significativos (01) representam a porta física *PF1* enquanto os dois menos significativos (00) a fila lógica *FL0*. Conforme ilustrado, o comutador S_{13} encaminha para a porta física *PF1* e fila lógica *FL0*, o pacote recebido com rótulo $R_4 = 316$. A definição da referida porta física e fila lógica é feita a partir da operação da operação $< 316 >_{13} = 4$; já, o S_5 ao receber o pacote encaminha para a porta $< 316 >_5 = 1$; depois, S_7 o encaminha para a porta $< 316 >_7 = 1$, chegando até o comutador de borda de saída que removerá o *rótulo* do pacote, entregando para o AS_D.

O método de cálculo dos resíduos pode ser resumido na Equação 1, que descreve o funcionamento do Teorema Chinês do Resto[1] , utilizado pelo QoSProLab para o processamento dos rótulos.

$$R = (\sum_{i=1}^{n} \overline{p} * \overline{m} * \overline{s}) \mod C \tag{1}$$

Onde:

- R = Rótulo
- $\overline{p}$ = vetor dos números primos
- $\overline{m}$ = vetor dos múltiplos dos números primos
- $\overline{s}$ = vetor das portas de saída
- C = produto das chaves locais
- n = número de comutadores no caminho

A complexidade do algoritmo em termos de tempo para computação do rótulo é $\mathcal{O}(len(C)^2)$ [Cormen et al. 2001].

2.2. Escalabilidade do modelo QoSProLab

Com relação a escalabilidade da proposta, o rótulo R cresce com o valor dos números primos associados aos comutadores de núcleo (S_i) e,com o numero de saltos que uma rota deve percorrer n. Por exemplo, em uma rede com 60 nós, selecionando o pior caso que corresponde aos maiores números primos na rota, caminhos de comprimento 11 (saltos)

[1]Códigos fontes para calcular o Teorema Chinês do Resto. $https : //rosettacode.org/wiki/Chinese\_remainder\_theorem$

podem ser implementados usando 96 bits do cabeçalho Ethernet [Martinello et al. 2014]. A Equação 2 permite calcular o número de bits. Como foram definidas 4 classes de tráfego no QoSProLab (2 bits a mais por comutador), temos a seguinte equação:

$$R = (\log_2 \prod_{i=1}^{n} S_i * 4) \quad (2)$$

, onde n = número de comutadores no caminho.

O sistema de resíduos, a depender da estrutura topológica, escala bem em função da característica do grafo. Por exemplo, na topologia *2-clos tier*, há múltiplos caminhos curtos (até 4 saltos) com alta conectividade, tipicamente interessante para redes de data center de borda. Considerando reusar apenas os 48 bits do endereço de destino do cabeçalho Ethernet, [Martinello et al. 2017] mostrou que é possível conectar 2304 servidores físicos com proteção completa do caminho. Isso é possível graças a um sistema de congruência utilizado no controlador que permite não somente calcular os rótulos para uma rota, como também utilizar os rótulos para diferentes propósitos (e.g. filas de prioridade).

2.3. Protocolo RDN para Processamento de Rótulos Programáveis

A implementação do modelo QoSProLab exigiu o desenvolvimento de um protocolo de camada 2 flexível para processamento de rótulos, com o propósito de dar maior suporte ao processamento dos rótulos programáveis numa RDN de grande escala, assim como permitir a integração do modelo com protocolos legados. O protocolo prevê um campo flexível que suporta rótulos RDN com maiores dimensões, tornando desnecessário o uso de protocolos legados no núcleo da rede.

A Figura 2 descreve o pacote RDN processado através do protocolo proposto. Observa-se que, o cabeçalho é formado por dois campos. O campo *length* utilizado para informar o tamanho do próximo campo e o campo *label* utilizado para armazenar a informação do rótulo. Também é exposto a forma de encapsulamento do RDN. No exemplo, é descrito um pacote de uma arquitetura Internet padrão TCP/IP. Ou seja, o comutador de borda do modelo proposto encapsula o pacote completo sob o cabeçalho RDN.

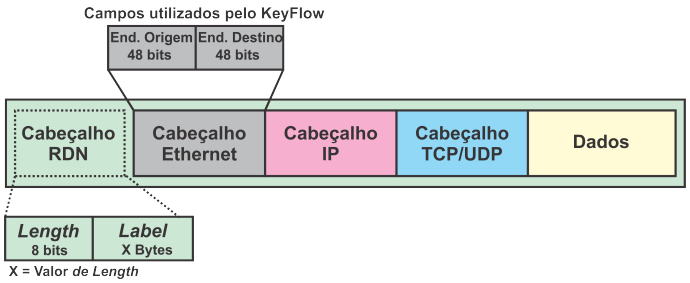


Figura 2. Estrutura e aplicação do cabeçalho RDN.

O cabeçalho RDN descrito na Figura 2 funciona da seguinte forma: o campo *length* possui 8 bits e define o tamanho em Bytes do *label* a ser extraído. Assim o cabeçalho RDN terá um tamanho de (1+ *length*) Bytes podendo variar até um máximo de 256 Bytes. Consequentemente, o cabeçalho será expandido ou reduzido conforme o tamanho

dos rótulos. Através desta funcionalidade o protocolo oferece maior escalabilidade em relação à crescente expansão do tamanho do rótulo, que é proporcional ao número de portas de saída e valores destinados as chaves locais.

Não faz parte do escopo deste trabalho, a análise da sobrecarga resultante da flexibilidade obtida. Certamente, este será objeto de investigação futura. Como será discutido na Seção 3, a utilização da linguagem P4 e seu arcabouço de implementação foram essenciais na criação deste modelo e implementação do protocolo flexível aqui descrito. A lógica de processamento dos pacotes nos comutadores de núcleo e borda foram desenvolvidas nessa linguagem, tornando a solução independente de protocolo legado.

3. Implementação do QoSProLab

O modelo foi implementado através de emulação baseado no Mininet. Para isso, foi utilizado o *simple_switch* [2] e *hosts* padrão do próprio Mininet. Este ambiente foi montado sobre uma máquina virtual Ubuntu 16.04 com 4GB de Memória RAM e 60GB de disco rígido operando na plataforma do VirtualBox 5.1. Para a implementação do protocolo RDN foi utilizada a linguagem P4 que possibilitou o processamento dos rótulos na RDN.

A Figura 3 resume os recursos utilizados na implementação do modelo. O detalhamento deste ambiente é apresentado nas próximas subseções, que descrevem o controlador e os comutadores de núcleo e de borda.

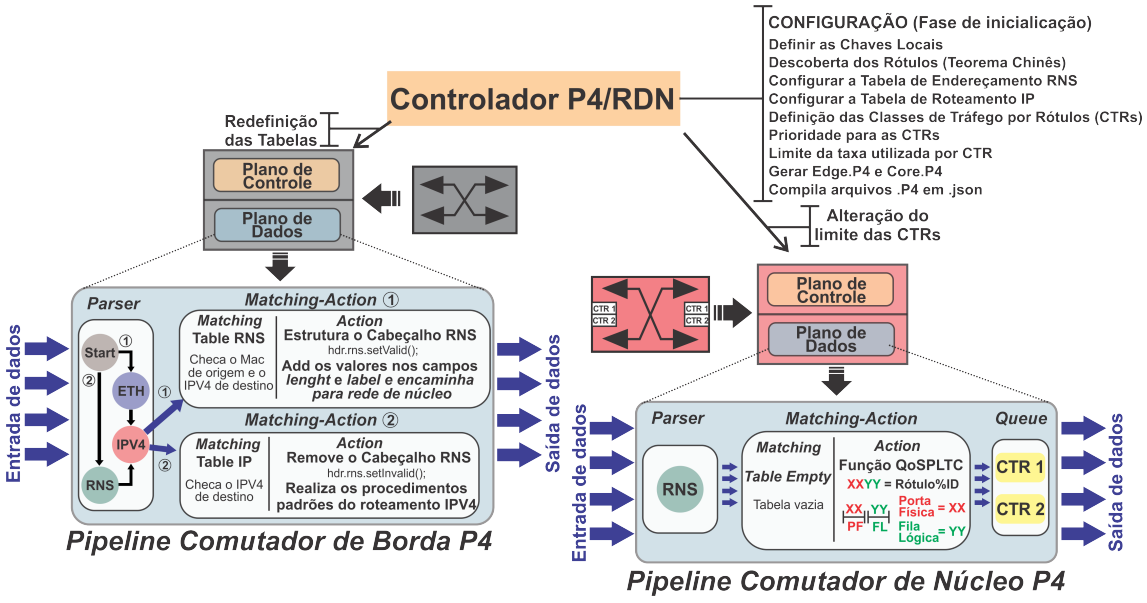


Figura 3. Recursos utilizados na implementação do QoSProLab.

3.1. Controlador

O controlador têm a visão global da rede e realiza o gerenciamento dos fluxos que passam nos comutadores. Na implementação, os comutadores funcionam no modo pró-ativo. Desta forma, ao iniciar a topologia, os mesmos recebem, do controlador, os parâmetros iniciais de configuração, que compreendem as chaves locais, tabela de roteamento IP e tabela de conversão IP para RDN (IP/RDN), e priorização das classes de tráfego para o oferecimento da qualidade de serviço.

[2]https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch

A Tabela de conversão IP/RDN é necessária para auxiliar a construção do cabeçalho RDN nos comutadores de borda. A definição do rótulo também passa pela definição da fila de prioridade, conforme mencionado anteriormente. Deste modo, o controlador define rótulos que não somente gere um resíduo empregado a uma porta de saída, mas que também informa a fila lógica responsável pela priorização da CTR. Uma Tabela de Roteamento IP padrão baseada em regras de camada 3 é encaminhada também aos comutadores de borda para informar qual a rota que o pacote seguirá ao sair da RDN.

3.2. Comutador de núcleo

O comutador de núcleo é responsável pelo gerenciamento de filas lógicas que possibilitam a política de priorização de tráfego. Este tipo de comutador é responsável por duas funções. A primeira consiste na utilização do rótulo, informado no campo do cabeçalho RDN, para o cálculo do resíduo, e a segunda é utilizar esse resíduo como identificador da porta de saída e fila lógica do pacote. É preciso destacar que, diferentemente dos comutadores legados, este não necessita de tabelas de roteamento, o que permite um processamento mais rápido. Além disso, Tabelas de QoS também são desnecessárias, já que o próprio rótulo é utilizado para saber qual a prioridade (fila lógica) do pacote. Por fim é preciso destacar que cada comutador possui um conjunto de filas lógicas para as interfaces de saída[3]. A taxa de atendimento é repassada do controlador para os comutadores de núcleo e informa qual o número de pacotes máximo, em relação a porcentagem da banda, que é permitido a uma fila lógica repassar a porta de saída. Tais filas são utilizadas na implementação da priorização do tráfego das aplicações.

O comutador de núcleo, portanto, possui uma programação P4 específica que determina como um pacote deve ser processado. A Figura 3 exibe o *pipeline* de funcionamento no processamento do pacote conforme se segue: i) Inicialmente é realizado o *parsing* para coletar as informações contidas nos campos do cabeçalho RDN; ii) Apesar de não necessitar de tabelas a estrutura P4 necessita de uma tabela para que aconteça o encaminhamento para as ações, visto isso, uma tabela vazia (*empty*) foi implementada com o objetivo de encaminhar a informação do rótulo para a função módulo na qual será encontrado o resto da divisão a partir da chave local. O valor obtido indicará a porta de saída e fila lógica do pacote. O esquema de processamento das filas está estruturado em um sistema de "rodízio", através do qual as filas são atendidas de forma que todos os pacotes sejam transmitidos sem que haja *starvation*. Desta forma, todos os pacotes serão transmitidos de forma intercalada e, taxas de atendimento enviadas pelo controlador são aplicadas.

3.3. Comutador de Borda

O comutador de borda faz a conversão de protocolos entre as redes externas e a rede de núcleo que implementa a arquitetura RDN. Para isso, inicialmente verifica-se por qual interface de entrada o pacote está chegando, a partir disso é determinado se é um pacote RDN (caso seja proveniente da rede de núcleo) ou IP (caso seja proveniente da rede externa). Então, este comutador faz a conversão de protocolo entre as redes legadas e a RDN. Para isso, a implementação P4 do comutador de borda constitui-se em 2 caminhos, como mostrado na Figura3:

[3]A configuração do número de filas e as taxas de atendimento foram definidas com base em trabalhos relacionados da literatura [Le Faucheur and Lai 2003]

1. Quando trata-se de um pacote originário da rede externa, significa que o mesmo provavelmente está seguindo para a RDN, se o percurso for confirmado, o mesmo deverá ser convertido para ser processado por resíduo, seguindo o caminho 1. Deste modo, verifica-se o *MAC* de origem e o IP de destino, se a porta destino proveniente deste *matching* for a rede de núcleo RDN, o comutador criará o cabeçalho RDN com base nas informações de rótulos existentes em sua tabela de conversão IP/RDN. Por fim, encapsulará o pacote todo sob o cabeçalho RDN para encaminhar à porta de saída.
2. Por outro lado, sendo um pacote proveniente do núcleo RDN, assume-se que o mesmo é um pacote RDN, portanto deve ser convertido para o protocolo legado (caminho 2). Assim, remove-se o cabeçalho RDN, e utilizando as informações contidas no cabeçalho IP e na tabela de roteamento, encaminha-se o pacote para a porta de saída destinada.

4. Estudo experimental

O modelo QoSProLab foi avaliado a partir de um estudo experimental conduzido em laboratório por meio de emulações de uma RDN reproduzida num ambiente Mininet/P4 (descrito na Seção 3). As próximas subseções descrevem a metodologia e cenário de análise, as métricas de desempenho avaliadas, fatores e níveis adotados. Ao final, analisa e discute os resultados obtidos.

4.1. Metodologia e cenários avaliados

O estudo experimental foi elaborado de forma a identificar a viabilidade de adoção de um modelo de aprovisionamento de QoS através de uma RDN. O processo consistiu na reprodução de uma RDN com dois clientes e dois servidores de vídeo. Através de emulações, um tráfego sintético foi inserido no ambiente, e medições foram realizadas para posterior análise dos seus resultados baseado nas métricas escolhidas.

Os cenários foram executados em um ambiente com topologia linear, pelo fato de ser um encaminhamento fim a fim independente do caminho escolhido, será uma rota linear. Desenvolvido para múltiplos comutadores virtuais, sendo que 2 comutadores são de borda e os outros de núcleo, conectados a hospedeiros virtuais como apresentado na Figura 4. Alguns parâmetros foram fixados, como a capacidade da banda dos *links* fixada em 4 Mbps, 2 clientes e 2 servidores, e 10 comutadores de núcleo (12 no total da rede borda + núcleo). Foi utilizado um máximo de 2 CTRs (CTR 1 e CTR 2) para todos os experimentos.

No primeiro experimento métricas de QoS são utilizadas para validar o comportamento das CTRs ativadas. Neste experimento o ambiente foi implementado sobre uma RDN convencional e testes foram realizados inicialmente sem nenhuma CTR ativa e posteriormente foi ativada a CTR1 com uma taxa de atendimento em 60% e depois a CTR 2 foi ativada com um taxa de 30%. Um tráfego de fundo foi utilizado de forma que a banda ocupada obtivesse uma taxa excedente (TE) de 5 e 10%. O experimento foi executado 10 vezes para cada variação dos fatores, cada execução foi realizada em um tempo de 60 segundos e o nível de confiança foi de 95%. O tráfego sintético enviado dos clientes para os servidores acontecia de forma paralela.

O segundo experimento consiste na alteração da taxa de atendimento definida pelo controlador as CTRs como forma de avaliar o comportamento das filas lógicas perante

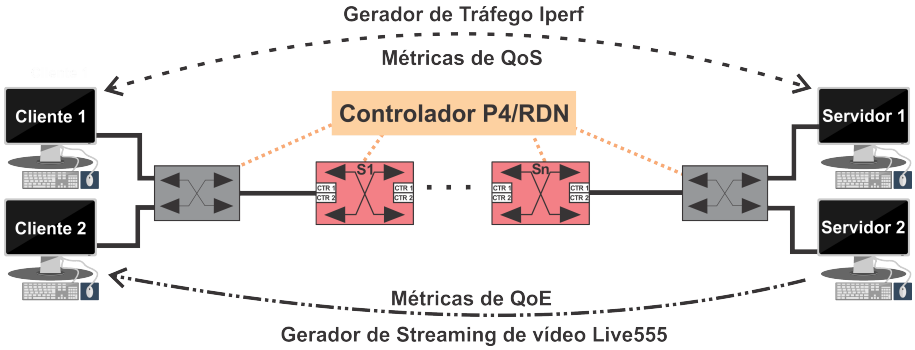


Figura 4. Cenário de experimentação.

novas informações repassadas pelo controlador. Nesse experimento os valores destinados as taxas da CTR 1 e CTR 2 foram alterados durante certos períodos de tempo, onde cada cliente possui um certo nível de prioridade computado no controlador e o mesmo realiza as alterações nas taxas dos comutadores de núcleo conforme as necessidades dos clientes.

Por fim, um terceiro experimento realizado sobre as métricas de QoE. Nesse experimento fatores como o uso de tráfego concorrente CBR(*Constant bit rate*) e a implementação do modelo QoSProLab (como no primeiro experimento) foram alternados para mostrar o quanto um fluxo concorrente pode influenciar na qualidade de uma *streaming* de vídeo. E com isso avaliar como o uso das CTRs podem ajudar a prover uma melhor qualidade ao vídeo resultante do cliente. As taxas de atendimento destinadas às CTRs, utilizadas nesse experimento, são as mesmas do primeiro experimento.

4.2. Métricas de análise

A implementação do modelo QoSProLab foi avaliada a partir da análise de métricas de qualidade de Serviço (QoS), para medir as influências dos elementos de rede, protocolos e outros elementos envolvidos na transferência de fluxos, de forma que o sinal não seja tão degradado em aplicações (e.g., *streaming* de vídeo). Além disso, foram também avaliadas métricas de Qualidade de Experiência (QoE) com o objetivo de verificar os efeitos da priorização de tráfego nas filas dos comutadores de núcleo.

Mais especificamente, as métricas de QoS utilizadas foram:

- *Goodput*: Refere-se ao total de pacotes úteis processados pelo servidor. Essa métrica servirá para exibir o funcionamento do limite da taxa de atendimento em pacotes por segundo (pps) oferecido às filas de prioridade;
- Perda de Pacotes: Consiste no percentual de pacotes perdidos entre o cliente e o servidor. Essa métrica tem como função informar que a implementação da QoSProLab não causará um aumento no descarte de pacotes;
- Latência (atraso fim a fim): Indica o tempo em segundos (s) que um pacote leva de um ponto da rede ao outro. Esta métrica foi utilizada para aferir o desempenho da comutação na RDN.
- *Jitter*: Consiste na variação da latência em segundos (s). Esta métrica foi utilizada para avaliar o desempenho do QoSProLab em relação a qualidade do sinal percebido pelo usuário da aplicação.
- Percentual de Banda Ocupada: Mostra a porcentagem de ocupação da banda total da rede. Métrica empregada com o intuito de mostrar a reconfiguração das taxa de

atendimento direcionadas as CTRs, e disponibilizadas aos clientes à depender do nível de prioridade do mesmo.

Tais métricas foram avaliadas a partir da geração de tráfego sintético produzido através da ferramenta Iperf3[4].

No caso do QoE, as métricas de PSNR (*Peak Signal to Noise Ratio*) e SSIM (*Structural Similarity Index*), foram utilizadas na comparação do vídeo original e o degradado. A PSNR, métrica bastante utilizada pela comunidade científica, estabelece a conexão entre a máxima energia disponível de um sinal e o ruído que influência o sinal representado entre os quadros do vídeo degradado e do original. A PSNR varia seu intervalo de valores de 0 a 100, onde 100 é o melhor resultado possível. A métrica SSIM compara quadro a quadro do segmento de vídeo estimando a similaridade entre o vídeo degradado e original para avaliar a degradação causada. Para avaliar tais métricas, foi utilizada a ferramenta Live555[5] para transmitir uma *streaming* de vídeo[6]. Como resultado da transmissão tem-se um vídeo no console do cliente, que representa o vídeo com se tivesse sido assistido de forma *on-line*. Esse vídeo resultante é comparado ao vídeo original através da ferramenta MSU-VQMT[7] e o resultado é apresentado sobre as métricas de QoE.

4.3. Resultados e Análise

Os resultados obtidos através da experimentação são apresentados nesta subseção. Onde, as discussões são focadas nas medidas de QoS coletadas a partir do tráfego sintético gerado pelo Iperf3 e os resultados das medidas de QoE analisados pelo MSU-VQMT.

4.3.1. Validando o comportamento das CTRs

Como pode ser visto na Figura 5 e 6 tem-se os resultados para os dois clientes sobre as métricas colocadas. Com relação as métricas de *goodput* e perda de pacotes notou-se relativo ganho com relação à melhora na entrega de pacotes válidos por segundo, o que indica que quando as CTRs estão ativadas o cliente consegue ter uma maior quantidade de pacotes atendidos ao fim do caminho, além de ter seus valores finais obtidos estabilizados visto pelo pequeno intervalo de confiança gerado.

Foi separado os resultados do cliente 1 do cliente 2 porque os resultados do cliente 2 mostram melhor o ganho que um cliente pode vir a ter quando parte da rede já está disponibilizada para outra aplicação e o mesmo tem que concorrer por uma fatia menor com o tráfego de fundo como pode ser visto na Figura 6.

Apesar do cliente 1 ter mostrado uma relativa deterioração na Latência, percebe-se que no caso do cliente 2 já acontece uma leve melhora. Mas os ganhos com relação a perda de pacotes e com o nivelamento do *goodput* são nítidos para os dois clientes.

[4]https://iperf.fr/

[5]http://www.live555.com/

[6]O vídeo escolhido foi a animação Big Buck Bunny na resolução de 240p (320x240)

[7]http://www.compression.ru/video/quality_measure/video_measurement_tool.html

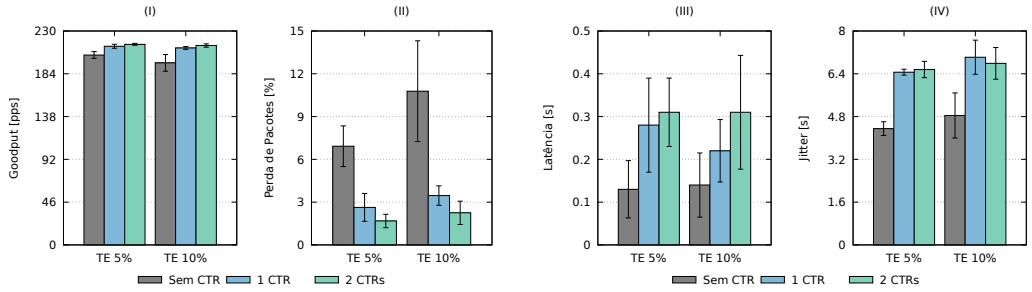


Figura 5. Resultados Cliente 1.

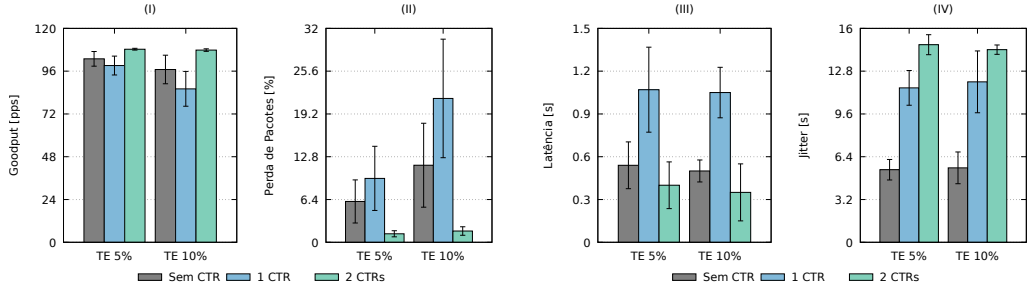


Figura 6. Resultados Cliente 2.

4.3.2. Reconfiguração das Taxas de Atendimento

Essa experimentação teve como objetivo mostrar a reconfiguração das taxas de atendimento nos comutadores de núcleo a medida que o cliente com maior prioridade necessitar de um consumo maior da largura de banda. Nesse experimento o cliente 1 tem maior prioridade na rede do que o cliente 2, mas apesar da maior prioridade o mesmo não pode ultrapassar um máximo de 80% para que não ocorra *starvation* ao restante dos fluxos.

Esse experimento foi dividido em quatro períodos de tempo como visto na Figura 7. No 1º período (T1-T2) o cliente 2, com menor prioridade, solicita uso da rede, como não possui outro tráfego o controlador permite o consumo de 100% da banda. No 2º período (T2-T3) o cliente 1, inicia um fluxo de dados, correspondente a um tráfego que consome 40% da banda. O controlador, ao notar o tráfego vindo do cliente com maior prioridade, reconfigura as taxas de atendimento nos comutadores de núcleo para que o rótulo deste cliente seja priorizado, ao receber as regras os comutadores de núcleo estabilizam o tráfego do cliente 1 na taxa configurada. No 3º período (T3-T4) o cliente 1 amplia o fluxo de dados. Ao perceber esse aumento o controlador verifica a prioridade do cliente 1 e, permite que o mesmo, utilize a banda desejada desde que esse consumo não ultrapasse o teto máximo de 80% que uma CTR pode possuir. Desta forma, a taxa do cliente 2 que já havia sido reconfigurada para 40% passa a ser agora de 20% da banda. No último período (T4-T5) o cliente 1 finaliza o seu fluxo de dados e com isso o controlador realoca 100% da banda ao cliente 2 para que o mesmo possa concluir seu tráfego.

Algumas instabilidades nos fluxos de dados no início de cada período são causados devido ao envio das regras com as taxas de atendimento do controlador aos comutadores de núcleo. Mas, após realizada a comunicação e as regras serem configuradas nos comutadores de núcleo o fluxo de dados de cada cliente estabiliza na porcentagem de ocupação

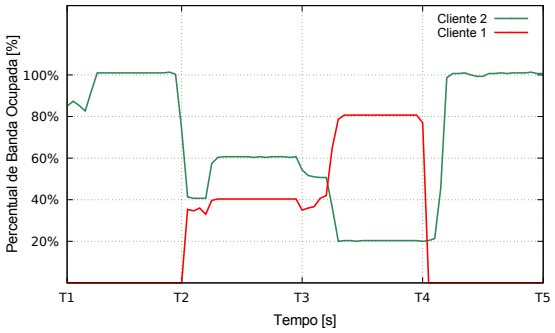
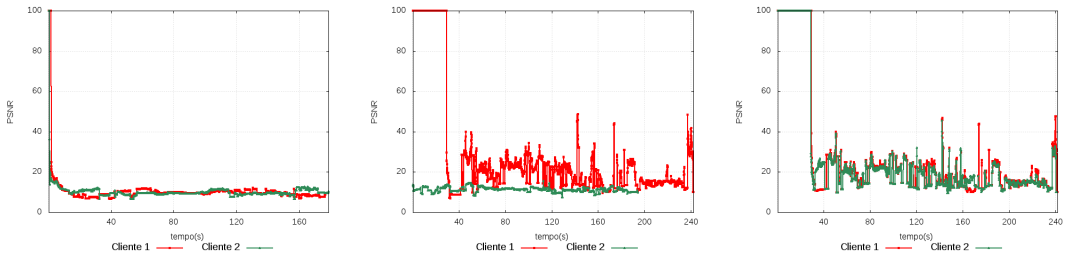


Figura 7. Reconfiguração das Taxas de Atendimento.

da banda determinada nas taxas de atendimento estabelecidas pelo controlador às CTRs.

4.3.3. Análises de QoE

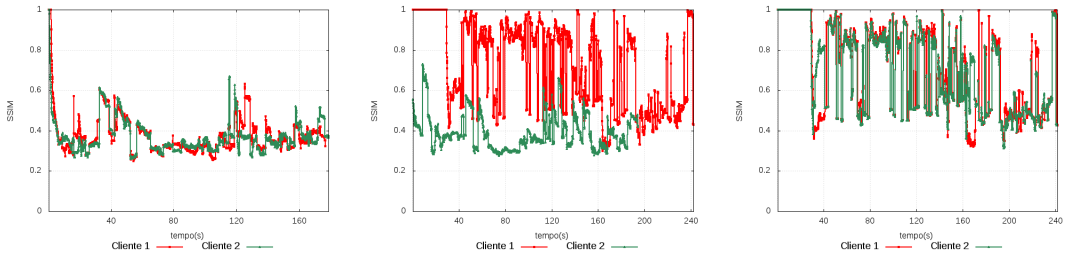
Nesses experimentos foram analisados a influência de um fluxo concorrente sobre uma *streaming* de vídeo. Para isso foi criado um cenário constituído por 2 clientes que tentam assistir um vídeo ao mesmo tempo. Nas Figuras 8 e 9 são apresentados os resultados sobre as métricas PSNR e SSIM respectivamente, obtidos a partir da variação no número de CTRs, em um cenário no qual os clientes disputam banda com um tráfego de fundo concorrente (TF). Analisando os gráficos pode ser visto o ganho que a implementação do QoSProLab proporciona a *streaming* de vídeo resultante no cliente 1 nos casos da Figura 8(b) e 9(b) devido ao fato de se ter uma CTR priorizando especificamente o tráfego do cliente 1. A mesma melhora pode ser notado ao tráfego do cliente 2 ao ser ativado uma CTR específica para ele, onde se pode notar que a QoE resultante do cliente 2 melhora quase ao nível do cliente 1. O que comprova que o uso do QoSProLab proporciona uma QoS a partir da priorização de CTRs.



(a) S/ CTR. (b) C/ 1 CTR. (c) C/ 2 CTRs.

Figura 8. Resultados de QoE sobre a métrica PSNR.

Os Resultados de QoE sobre a métrica SSIM pode nos mostrar resultados parecidos com os descritos para a métrica PSNR, mas por utilizar a similaridade dos quadros na comparação seu resultado pode ser melhor visualizado, como por exemplo a Figura 8(b) em comparação com a Figura 9(b). O mais importante nessas experimentações é notar a necessidade de QoS para esses tipos de aplicações, pois, como pode ser notado nos resultados, um fluxo concorrente sempre proporciona uma degradação maior do vídeo, piorando a QoE resultante. E por fim demonstrar o funcionamento do QoSProLab



(a) S/ CTR. (b) C/ 1 CTR. (c) C/ 2 CTRs.

Figura 9. Resultados de QoE sobre a métrica SSIM.

proporcionando uma melhor QoE através da priorização de CTRs.

5. Trabalhos Relacionados

Muitas propostas tem surgido na literatura para realizar decisões de encaminhamento baseados em etiquetas ou rótulos, tais como MPLS [Rosen et al. 2001], VLANs [IEEE 2009]. O MPLS é o protocolo mais conhecido e utilizado para encaminhar pacotes pela escrita e *matching* de rótulos anexados ao pacote. Este protocolo troca o rótulo a cada salto (Label-Switched Path – LSP) a partir de tabelas de encaminhamento. Na mesma linha, o Path Switching [Hari et al. 2015] propõe uma alternativa ao MPLS, tendo a vantagem de codificar a mensagem de encaminhamento em um conjunto fixo de bits no cabeçalho. Entretanto, a proposta é conceitual sem validação experimental, faltando evidência de sua viabilidade. Por fim, o Segment routing (SR) [Filsfils et al. 2016] é uma proposta no qual os pacotes são encaminhados usando uma pilha de rótulos. Nesse caso, cada comutador encaminha o pacote com operações de pop desempilhando os rótulos, mas precisa re-escrever o pacote a cada salto e também precisa seguir a ordem no qual os segmentos foram empilhados.

Em contraste com as abordagens mencionadas acima, que precisam de operações de *swap* (MPLS) e *pop* (SR) por nó, o RDN calcula operações de resíduos baseadas em comutadores sem tabela no núcleo e sem precisar re-escrever o pacote por salto. O QoSProLab estende a abordagem RDN incluindo filas lógicas de prioridade cuja novidade está no projeto dos comutadores com representação binária do sistema de resíduos que originalmente era centrado em números inteiros.

6. Conclusão e Trabalhos Futuros

Este trabalho apresentou uma implementação para o modelo de aprovisionamento de QoS baseado em classe de tráfego por rótulos programáveis (QoSProLab) com o objetivo de prover QoS em RDNs através de rótulos programáveis. Este modelo foi implementado através da linguagem P4 e reproduzido por meio de emulação no ambiente Mininet. Resultados obtidos no estudo experimental demonstrou a viabilidade do modelo proposto. Vislumbra-se como trabalhos futuros a aplicação do modelo em ambientes reais, como as redes experimentais disponíveis através dos projetos FIBRE[8], FUTEBOL[9] e BAMBU[10].

[8] fibre.org.br
[9] http://www.ict-futebol.org.br/
[10] https://www.pop-ba.rnp.br/Bambu

Referências

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Chang, C.-H., Molahosseini, A. S., Zarandi, A. A. E., and Tay, T. F. (2015). Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. *IEEE Circuits and Systems Magazine*, 15(4):26–44.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

de Oliveira, R. E. Z., Vitoi, R., Ribeiro, M. R., and Martinello, M. (2013). Keyflow: Comutaçao por chaves locais de fluxos roteados na borda via identificadores globais. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-SBRC*, 13(9).

Filsfils, E. C., Previdi, E. S., Systems, I. C., Decraene, B., Litkowski, S., Orange, Shakir, R., and Communications, J. (2016). Segment Routing Architecture. Internet-Draft Segment Routing Architecture draft-ietf-spring-segment-routing-09, Network Working Group. Standards Track.

Hari, A., Lakshman, T. V., and Wilfong, G. (2015). Path switching: Reduced-state flow handling in sdn using path information. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 36:1–36:7, New York, NY, USA. ACM.

IEEE (2009). IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pages C1–72.

Jyothi, S. A., Dong, M., and Godfrey, P. (2015). Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 10. ACM.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Le Faucheur, F. and Lai, W. (2003). Requirements for support of differentiated services-aware mpls traffic engineering", rfc3564.

Martinello, M., Liberato, A. B., Beldachi, A. F., Kondepu, K., Gomes, R. L., Villaca, R., Ribeiro, M. R., Yan, Y., Hugues-Salas, E., and Simeonidou, D. (2017). Programmable residues defined networks for edge data centres. In *Network and Service Management (CNSM), 2017 13th International Conference on*, pages 1–9. IEEE.

Martinello, M., Ribeiro, M. R., de Oliveira, R. E. Z., and de Angelis, R. (2014). Keyflow: a prototype for evolving sdn toward core network fabrics. *IEEE Network*, 28(2):12–19.

Rosen, E., Viswanathan, A., and Callon, R. (2001). RFC 3031: Multiprotocol Label Switching Architecture. Technical report, IETF.