Comutador P4 com Suporte a Roteamento Multicaminhos

Lucas B. Fernandes, Pedro H. Ribeiro, Leonardo Martins
Rafael Pasquini, Luis F. Faina, Lasaro Camargos

[1] Faculdade de Ciência da Computação – Universidade Federal de Uberlândia (UFU)

Abstract. *In this paper we describe a network switch with multi-path routing capabilities. The switch is implemented using P4, easily extensible and available as a free software project. The switch supports different routing policies, which adhere to different levels of QoS, making the network use very flexible.*

Resumo. *Neste trabalho descrevemos um comutador de rede com capacidade de roteamento multicaminhos implementado em P4. O comutador é facilmente extensível e está disponível como projeto de código livre. O mesmo suporta diferentes políticas de roteamento que aderem a diferentes níveis de QoS e conferem flexibilidade no uso da rede.*

1. Introdução

Comutadores de Rede de "prateleira" são aderentes a uma vasta gama de protocolos, mas embora configuráveis não são reprogramáveis. Assim, as redes são de certa forma ossificadas, tornando inviável ou muito difícil a execução e avaliação de novos protocolos [McKeown et al. 2008]. Comutadores de Redes Definidas por Software (SDN, *Software Defined Network*) reduzem esta barreira por serem programáveis [Batista et al. 2015]. Em SDN, a separação do plano de dados (comutadores) e do plano de controle (controlador) é um ponto de destaque. O OpenFlow [Open Networking Foundation 2012], por exemplo, é um padrão aberto que especifica como comutadores manipulam e encaminham pacotes baseado em regras definidas por controladores programáveis. Já P4 é uma linguagem para programar o plano de dados de dispositivos em software ou baseados em hardware reconfigurável, como ASICs e FPGAs. O sistema de execução do P4 também permite a programação do uso de blocos com funções pré-definidas em comutadores compatíveis, controlando o *pipeline* de processamento de pacotes via, p.ex., definição de tabelas em P4 e sua manipulação em tempo de execução por um controlador.

Neste trabalho usamos P4 para programar o comutador de referência da arquitetura, o BMv2[1], para permitir o uso de caminhos redundantes em um fluxo. O uso do multicaminhos permite tanto ganhos de tolerância a falhas quanto de desempenho na comunicação. Neste trabalho, implementamos e validamos três políticas de roteamento no comutador proposto: i) probabilística, ii) round-robin e iii) com pesos. Não obstante, novas políticas podem ser implementadas, uma vez que o projeto é de código aberto.[2]

2. Trabalhos Relacionados

A programabilidade do P4 foi demonstrada, p.e., na implementação de funções de criptografia nos comutadores para que possam (de)criptografar dados em fluxos Ethernet,

[1] `https://github.com/p4lang/behavioral-model`
[2] `https://github.com/pluxos/p4-dev`

IPv4 e TCP [Antunes 2016], na instanciação de agentes do algoritmo de consenso Paxos [Lamport 1998] no comutador [Dang et al. 2015], ou mesmo realizando processamento paralelo nos mesmos[Chen et al. 2018].

Relativo a roteamento multicaminhos, o próprio projeto P4 implementa a abordagem ECMP (*Equal-Cost Multi-Path*) [Hopps 2000], que distribui a carga uniformemente entre os caminhos. Embora tenhamos usado tal implementação como base para nosso trabalho, na nossa versão o número de pacotes egressos em cada interface depende da política de roteamento usada e de pesos associados a cada interface. Assim, nossa implementação é mais flexível, podendo inclusive emular ECMP, embora o contrário não seja possível.

[Rezende et al. 2017] desenvolveu módulos de monitoração e multicaminhos com OpenFlow. A partir da monitoramento da infraestrutura, o controlador calcula os multicaminhos e informa ao plano de dados que, estendido com um escalonador multicaminhos, implementa o roteamento. Uma vez criado um fluxo, o controlador continua monitorando-o e ajustando o comportamento dos comutadores para evitar que o fluxo extrapole a taxa de transmissão acordada. Usando uma abordagem baseada em P4, é possível que, uma vez definido o fluxo multicaminhos, o comutador seja informado também da taxa máxima a ser usada e que o próprio comutador faça a limitação do uso de banda, reduzindo o tempo de reação da rede. De fato, implementamos tal mecanismo em nosso projeto que pode assim ser visto como uma implementação de [Rezende et al. 2017], mas usando P4 em vez de OpenFlow. Contudo, não discutimos o mecanismo de limitação de banda aqui, por restrições de espaço.

3. Comutador

3.1. Políticas de Roteamento

A alocação de banda para um fluxo permite garantia de SLA (Service Level Agreement) na comunicação entre elementos de borda do fluxo. Neste trabalho, o uso da largura de banda alocada segue políticas pré-definidas de roteamento. Considere o exemplo ilustrado na Figura 1 de alocação um fluxo de 15 Mbps entre os hosts H1 e H3, sendo que o multicaminho proposto usa 10Mbps via switch S2 e 5Mbps via switch S3.

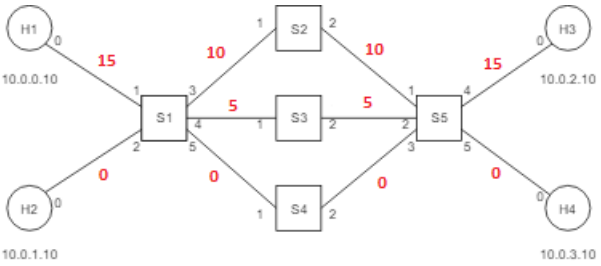

Figura 1. Multicaminhos para Fluxo (15Mbps) H1-H3 em Enlaces de 10Mbps.

Roteamento Probabilístico Uma vez definido o conjunto de caminhos a ser utilizado, p.ex., S1-S2-S5 e S1-S3-S5, como na Figura 1, o controlador define, para cada comutador e cada interface, a probabilidade de um pacote seguir pela interface. Do ponto de vista de S1, $\frac{2}{3} = \frac{10}{15}$ dos pacotes devem ser seguir pela interface 3 (S1-S2), enquanto $\frac{1}{3} = \frac{5}{15}$

pela interface 4 (S1-S3) e 0 pela interface 5 (S1-S4). A abordagem é simples pois não exige manipulação de estado no comutador. Isto é, uma vez criado o fluxo, o estado é apenas lido. Em contrapartida, dado que a decisão é probabilística, existe a possibilidade de algum enlace ficar momentaneamente sobrecarregado e descartar pacotes enquanto outros enlaces são subutilizados.

Roteamento Round-Robin Nesta abordagem, uma vez definidas as interfaces de saída em um comutador para um fluxo, realiza-se um rodízio entre as mesmas para cada pacote do fluxo. Tem a vantagem de ser determinística no número de pacotes colocados em cada enlace, mas tem a desvantagem de ter que manter para cada fluxo a informação sobre qual a próxima interface a ser utilizada. Além disso, considera todas as interfaces como tendo a mesma importância no fluxo, limitação que é removida na abordagem com Pesos.

Roteamento com Pesos O rodízio entre as interfaces ocorre somente após um determinado número de pacotes, proporcional ao peso da interface, ter sido encaminhado pela mesma. Tendo por base a Figura 1, S1 pode alternar entre enviar 10 pacotes via S1-S2 e 5 entre S1-S3, ou 6 pacotes entre S1-S2 e 3 entre S1-S3, ou qualquer outra combinação com mesma razão. A vantagem desta abordagem é a garantia da distribuição precisa dos pacotes, dado um fluxo de duração relevante, mas ao custo de manter um contador de pacotes além de um indicador de qual interface deve ser utilizada, como na Round-Robin. Considerando que o encaminhamento de pacotes pode ser concorrente no comutador, manter o contador implica em controle de concorrência, o que pode diminuir o desempenho do comutador.

3.2. Implementação

Um comutador P4 recebe um programa P4 compilado que descreve e manipula estruturas de dados usadas no encaminhamento de pacotes através das interfaces. Face às limitações do Comutador, o programa pode ter mais ou menos blocos de funcionalidades no plano de dados. Os recursos P4 aqui utilizados, contudo, são simples o suficiente para que a abordagem seja compatível com praticamente qualquer comutador de uso real.

Pipeline P4 Quando um pacote é recebido em um comutador P4, o mesmo passa por um *pipeline* de regras *match-action*. No caso de um *match* do pacote com a especificação de uma regra, a *action* correspondente é executada, p.ex., descarte do pacote, modificação e reinjeção no *pipeline*, ou egresso em determinada porta. Além do pacote em si, as ações podem usar e modificar metadados. O controlador atua no comutador gerenciando as regras do *pipeline*. O comutador, em si, é agnóstico a protocolos, mas o programa P4 carregado e o conjunto de regras no pipeline permitem que protocolos diversos, novos ou consolidados, sejam suportados.

Action Profiles permite que um grupo de ações seja associado a uma mesma regra. Quando ocorre um *match*, uma função *hash* é aplicada ao pacote para escolher qual das regras do grupo deve executada. Neste trabalho, combina-se metadados e *Action Profiles* para implementar o Roteamento Multicaminhos.

Metadados + Action Profiles Na criação de um fluxo, o controlador informa ao comutador quais interfaces são de egresso de pacotes bem como pesos para cada uma delas. Estas informações são passadas em um vetor de bytes, em que o primeiro byte indica a quantidade de pesos e, os demais, os pesos para cada interface do comutador, p.e., o vetor `(4,0,30,30,40)` informa que as portas 1, 2, 3 e 4 tem, respectivamente e no fluxo em questão, pesos 0, 30, 30, e 40.

Recebido o vetor, o comutador cria uma *Action Profile* com uma ação de egresso para cada porta. Na abordagem probabilística os pesos se traduzem diretamente em probabilidades que são salvas em um metadado associado à *Action Profile*. No exemplo anterior, as probabilidades das interfaces 1, 2, 3 e 4 são então 0, $\frac{30}{100}$, $\frac{30}{100}$ e $\frac{40}{100}$, onde 100 é o somatório dos pesos. No caso de um *match*, a "função *hash*" escolhe uma das portas segundo as probabilidades nos metadados.

Já na abordagem com pesos, os valores se traduzem em número de pacotes a serem enviados em cada porta. Esta informação é salva nos metadados juntamente com a identificação da interface e um contador de pacotes egressos da mesma interface. Após cada pacote, verifica-se se o contador alcançou o número de pacotes especificado para interface e, caso afirmativo, zera-se o contador e passa-se para a próxima interface.

Finalmente, na abordagem Round-Robin, o controlador indica apenas quais portas devem entrar no rodízio. Neste caso, apenas a identificação da porta em uso é adicionada aos metadados, sendo modificada a cada egresso.

3.3. Controlador

Para testar o comutador, implementamos um controlador em linguagem Python v3 que explora as funcionalidade do CLI do Projeto P4, cuja comunicação com o comutador se dá por Thrift[3]. O controlador representa a topologia da rede como um grafo usando a biblioteca NetworkX[4]. Para escolha de caminhos, utiliza-se o algoritmo *capacity scaling* também disponível na biblioteca. A escolha de caminhos por meio do *capacity scaling* resulta em escolha equivalente àquela de [Rezende et al. 2017, Rezende et al. 2016]. Quanto à limitação de banda, desenvolvemos uma abordagem implementada no próprio comutador, sendo de maior a escalabilidade. Contudo, por restrições de espaço, apresentaremos tal mecanismo em comunicação futura.

Políticas de Alocação Cada vértice no grafo corresponde a um elemento de rede e contempla propriedades associadas (p.e., se comutador ou host, número de portas). Arestas correspondem a enlaces e contemplam as seguintes propriedades: capacidade de enlace correspondente ($CapTotal$); capacidade alocada dado os fluxos já criados e suas especificações de demanda ($CapAloc$); capacidade em uso medida pelo plano de dados e informada periodicamente ao controlador ($CapUso$). O controlador pode ser configurado para usar ou $CapTotal - CapAloc$ como capacidade remanescente nas arestas, de forma a respeitar alocação de banda determinada em SLA, ou $CapTotal - CapUso$, que permite super-provisionamento de enlaces para cenários em que o uso de banda é inferior ao alocado.

[3] `http://thrift.apache.org/`
[4] `https://networkx.github.io/`

Para simplificar a apresentação, neste trabalho assumimos que todas as arestas tem a mesma capacidade em ambos os sentidos e que os fluxos demandam a mesma taxa em ambos os sentidos. De fato, controlador e comutadores permitem a criação de fluxos assimétricos.

4. Validação

As estratégias de roteamento implementadas foram validadas usando o Comutador P4 de referência, BMv2, em rede simulada mininet[5], usando o programa P4 implementando *Action Profiles* tal descrito aqui. A topologia usada nos testes foi a apresentada na Figura 2, com 4 hosts e 4 comutadores e, para cada teste, foram atribuídos pesos P1, P2, P3, P4, P5 e P6 aos enlaces. Foram realizados diversos testes, tendo sido todos os mecanismos bem sucedidos distribuindo pacotes via os multicaminhos e nas proporções especificadas por cada política. Por restrições de espaço, apresentamos aqui apenas um teste simples. Os demais são descritos em detalhes em [Borges 2017].

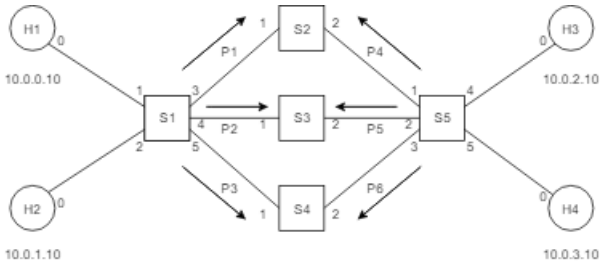

Figura 2. Topologia utilizada para o projeto: multicaminhos utilizando abordagem probabilística.

Para o teste descrito aqui, foi criado um fluxo de H1 para H3 com pesos descritos na Tabela 1. Foram então enviados 10000 pacotes ICMP de H1 para H3 variando-se as políticas. Para cada política, são apresentados o número de pacotes que trafegaram por cada enlace no sentido H1-H3. Os testes foram repetidos 10 vezes com resultados similares; os números apresentados na tabela são de uma destas execuções. Como se pode ver, o número de pacotes corresponde ao esperado para cada estratégia e pesos.

Tabela 1. Pesos, probabilidades e pacotes transitados por enlace, para cada estratégia, tendo sido enviados 10.000 pacotes de H1 a H3.

Enlace	Peso	Prob	Pkt Prob.	Pkt. RR	Pkt. Peso
P1	30	0,3	2983	3334	3000
P2	30	0,3	3017	3333	3000
P3	40	0,4	4000	3333	4000
P4	100	1	2983	3334	3000
P5	100	1	3017	3333	3000
P6	100	1	4000	3333	4000

[5] http://mininet.org/

5. Conclusão

Neste trabalho descreve-se um comutador com capacidade de roteamento multicaminhos implementado em P4. O comutador foi validado experimentalmente, é facilmente extensível e está disponível para modificações.

Algumas extensões possíveis para este trabalho incluem: a) estudo do uso concorrente de diferentes políticas de alocação e roteamento em uma Rede SDN e o impacto desta abordagem no QoS e capacidade de *multi-tenancy*; b) descrição e avaliação de mecanismo de limitação de banda implementado no comutador, c) descoberta de elementos de rede e criação automática de fluxos com largura de banda pré-definida pelo controlador usando *callbacks* providos por gRPC[6]. d) considerar política de roteamento por fluxo, em vez de global

Referências

Antunes, P. A. R. (2016). Soluções de segurança em redes utilizando a linguagem p4.

Batista, D. M., Blair, G., Kon, F., Boutaba, R., Hutchison, D., Jain, R., Ramjee, R., and Rothenberg, C. E. (2015). Perspectives on software-defined networks: interviews with five leading scientists from the networking community. *Journal of Internet Services and Applications*, 6(1):22.

Borges, L. (2017). Roteamento multicaminhos em redes definidas por software usando p4.

Chen, L., Chen, G., Lingys, J., and Chen, K. (2018). Programmable Switch as a Parallel Computing Device. *ArXiv e-prints*.

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *1st ACM SIGCOMM Sym. on Software Defined Networking Research*, SOSR '15. ACM.

Hopps, C. (2000). Analysis of an equal-cost multi-path algorithm.

Lamport, L. (1998). The part-time parliament. *ACM TRANSACTIONS ON COMPUTER SYSTEMS*, 16(2):133–169.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Open Networking Foundation (2012). Software-defined networking: The new norm for networks.

Rezende, P., Faina, L., Camargos, L., and Pasquini, R. (2016). Roteamento multicaminhos em redes definidas por software. In *2016 XXXIV Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*, pages 528–541.

Rezende, P. H. A., Coelho, P. R. S. L., Faina, L. F., Camargos, L. J., and Pasquini, R. (2017). Analysis of monitoring and multipath support on top of openflow specification. *Intl J. of Network Mgmt*, pages e2017–n/a. e2017 nem.2017.

[6] `https://grpc.io/docs/guides/`