Onix: A Distributed Control Platform for Large-scale Production Networks

Grupo 1
Francisco Caeiro, **47823**
Bruno Andrade, **47829**
António Estriga, **47839**

Qual é o problema que os autores tentam resolver?

Os *routers* são compostos por dois planos: o *control plane* e o *forwarding plane*. O *forwarding plane* está encarregue de encaminhar os pacotes pela rede. O *control plane* é responsável pela configuração do *forwarding plane*, permitindo decidir como tratar os pacotes que recebe.

Este artigo apresenta-nos uma forma de alterar as configurações de *routers* sem ter de o fazer um a um, utilizando um sistema distribuído para uma maior escalabilidade e sem prejudicar a eficiência da rede, recorrendo a uma ideia já existente: *Software-Defined Network*.

Este problema é relevante?

Este problema é extremamente importante. A utilização do *control plane* e do *forwarding plane* na mesma caixa torna a configuração dos *routers* extremamente ineficiente, obrigando a fazer uma configuração individual utilizando demasiada mão de obra e demorando demasiado tempo.

Qual é a sua solução?

Este artigo apresenta uma API simples, responsável pela comunicação com os *routers*, persistência da configuração e mapa da rede. Esta API permite a construção de um *control plane* fora do *router* na forma de um sistema distribuído, permitindo a integração com ferramentas de sincronização como o Zookeeper.

A API baseia-se numa estrutura de dados, a NIB, para guardar os dados necessários da estrutura e estado da rede. Esta estrutura é uma rede de nós com pares chave-valor, em que cada nó é uma entidade da rede. A partir desta estrutura, é possível fazer operações básicas como *create*, *query*, *destroy*, entre outros, e ainda receber notificações de mudanças na rede. Este NIB pode ser guardado numa base de dados relacional baseada em transações, no caso de situações onde é necessária uma maior consistência, ou numa DHT, para um melhor desempenho (ainda que com o custo de consistência "eventual"). Esta mudança é feita de forma fácil a partir do carregamento de módulos.

Que novas técnicas foram usadas?

Para que esta API seja utilizável no mundo real, é necessário que estes sistemas distribuídos sejam escaláveis e fiáveis, para evitar redes lentas e situações de negação de serviço. Algumas soluções são propostas para alcançar estes objetivos.

Para alcançar uma maior escalabilidade, são utilizadas as mesmas técnicas que são utilizadas em redes de computadores. Uma dessas técnicas é a partição da rede, isto é, cada instância do Onix tem apenas um subconjunto de rede atualizado em memória, permitindo a divisão do *overhead* pelos vários elementos. Outra técnica, a agregação, permite que, em vez de serem fornecidos dados de cada elemento do subconjunto, são fornecidos dados médios desse mesmo subconjunto. E por fim, para permitir controlar a consistência e durabilidade, pode ser utilizado uma DHT, com garantias relaxadas, ou uma base de dados relacional, com forte consistência e durabilidade. Estas duas opções permitem à aplicação uma escolha apropriada dos seus requisitos.

Com o intuito de ter um sistema fiável, o Onix integra o Zookeeper, ferramenta de coordenação,

para reagir a falhas de instâncias, realizando alterações necessárias para manter o sistema em funcionamento. Por exemplo: se num sistema de 2 instâncias, onde uma é o master e o outro é o backup, é detetado que o master foi abaixo (*watcher* no *getChildren* do nó com os metadados dos servidores), o *backup* entra em funções. Como o Onix está *decoupled* do resto do sistema, as restantes falhas na rede são tratadas tais como quando não existem instâncias Onix no sistema.

Como é que se destaca de trabalhos anteriores?

Os trabalhos anteriores oferecem uma boa base de estudo na separação do *control plan*e e do *forwarding plane*, *forwarding planes* extensíveis e arquiteturas de *data center* flexíveis.

Consolidando-os, o principal objetivo do Onix é transformar a ideia de *Software-Defined Network* num sistema distribuído. Para concretizar esse objetivo, sem prejudicar o desempenho do sistema, criaram uma API fiável, escalável e adaptável às necessidades de cada sistema.

Quais são os pontos mais fortes deste artigo? E os seus pontos fracos?

Um dos pontos mais fortes deste artigo é que nos fala de um sistema real e não apenas de um protótipo ou um projeto de investigação teórico. É explicado com grande detalhe como é que este sistema distribuído consegue resolver a distribuição do estado da rede, sem haver uma preocupação para os *developers* sobre os mecanismos de baixo nível. Apesar do detalhe ser um ponto positivo, sentimos também que existe demasiado detalhe a explicar os requisitos não funcionais do sistema, quase como que a realizar um pouco de publicidade a um produto.

Sem dúvida, este artigo é um passo para desenvolver redes mais eficientes, onde é necessário menos tempo de configuração e menos recursos humanos.

Como seria uma extensão deste trabalho?

Apesar de já existirem exemplos e referências de *deployment* deste sistema no artigo, uma extensão deste trabalho poderia ser a utilização de um sistema num caso real de grande escala, de modo a que fosse possível comparar o funcionamento prolongado de *Software-Defined Networking* e outros recursos habitualmente usados.