Plano de Controlo

Trabalho Prático 3 - Redes Definidas por Software

15 de Junho de 2024

Guilherme Sampaio	**Miguel Gomes**	**Rodrigo Pereira**
PG53851	PG54153	PG54198

1. Introdução

Este relatório dá continuidade ao projeto desenvolvido nas Etapa 1 — `Implementing a L3, L4 Stateful Firewall with P4` e Etapa 2 — `Manipulação de pacotes com P4`, e propõe a implementação de um controlador no plano de controlo, de modo a gerir e injetar regras nos *routers* `P4` utilizados no projeto.

Neste relatório, a Seção 2, detalha a implementação e análise experimental do requisito relativo ao controlador e a Seção 3 enumera e descreve as dificuldades encontradas durante a implementação e como estas foram resolvidas. Finalmente, a Seção 5 descreve como configurar, compilar e executar o projeto, seguido da Seção 6 com a conclusão e trabalho futuro.

Relativamente à topologia utilizada nesta fase, assume-se o mesmo cenário, mesma topologia e mesmas regras definidas nas duas etapas anteriores.

2. Controlador P4Runtime

Até à data, o método de injeção de regras de *firewall* e *routing* era todo feito manualmente. Na verdade, nas etapas anteriores o *script* da topologia injetava todos os requisitos necessários automaticamente, porém não recorria a um controlador.

Nesta etapa desenvolvemos um controlador `P4Runtime`, escrito em `Python` com base nos ficheiros disponibilizados pela equipa docente, previamente à conceção do projeto.

O controlador desenvolvido é responsável pela gestão da topologia da rede e pela injeção de regras nos *routers* `P4SwitchRuntime`. Tal como na fase anterior, a geração da topologia é feita dinamicamente, permitindo uma configuração flexível e uma geração de regras *"On Demand"*

O controlador desenvolvido é responsável por gerir a topologia da rede e injetar regras nos routers *P4SwitchRuntime*. Como na fase anterior, a geração da topologia é feita dinamicamente, permitindo uma configuração flexível e uma geração de regras *"On Demand"*.

2.1. Implementação

O comportamento *core* do controlador, como já mencionado previamente, é totalmente definido pelo exemplo fornecido pelo corpo docente[1].

`P4Runtime` é uma `API` que utiliza `gRPC` para controlar dispositivos de rede programáveis por `P4`. Permite que controladores configurem tabelas de encaminhamento, gerenciem registos e atualizem a lógica de processamento nos *switches* eficientemente e em tempo real. A comunicação gRPC proporciona uma *interface* bidirecional para operações como adicionar, modificar e excluir entradas de encaminhamento, garantindo flexibilidade e escalabilidade em redes definidas por *software*.

Para a geração de regras aplicadas nos diferentes dispositivos, o controlador dispõe de uma classe `Python` responsável pela conversão do ficheiro de configuração `YAML` do projeto em objetos com os

[1] Código Open Source disponibilizado pelo corpo docente no Github.

diferentes atributos necessários para cada entrada nas tabelas de *routing*. Este módulo foi o mesmo responsável pela criação das regras na Etapa 2. Portanto, tal como é possível a inicialização completa da topologia com apenas o controlador e o ficheiro de configuração.

```
{
  table_id: 48500340
  match {
    field_id: 1
    lpm {
      value: "\n\004\000\001"
      prefix_len: 32
    }
  }
  action {
    action {
      action_id: 31865707
    }
  }
  priority: 1
}
```

Listing 1: `Table Entry` exemplo para uma entrada `ICMP`.

No Listing 1, verificamos uma entrada das tabelas utilizada para definir o comportamento relativo a mensagens `ICMP` em cada *router*. Estes valores são obtidos pelo ficheiro de configuração da rede, conhecido pelo controlador. Os `ids` associados a alguns campos na entrada estão armazenados no ficheiro de *P4Runtime* gerados conforme o explicado na Seção 5. O campo `valor` no campo `lpm` representa um `IP` com o formato `10.4.0.1` convertido para *bytes* sobre um formato de *string*.

3. Dificuldades de Implementação

Durante a realização desta etapa foram encontrados diversos obstáculos, sendo alguns deles a falta de informação sobre uma eventual falha na injeção de regras nos *routers* e a falta de documentação.

Figura 1: `ICMP` com a injeção forçada.

Figura 2: `ICMP` sem a injeção forçada.

O maior obstáculo encontrado foi a falha na injeção das regras `ICMP` pelo controlador, que, mesmo após várias reestruturações, continuavam a não ser injetadas pelo controlador, apesar do correto funciona-

mento das restantes regras. De modo a resolver este problema foi implementado um *"modo de execução"* que, quando ativo, força a injeção das regras nos *routers* com o uso da ferramenta `simple_switch_CLI`. O seu funcionamento pode ser observado na Figura 4. Devido à natureza "disruptiva" deste modo, o utilizador é questionado se pretende mesmo forçar a injeção das regras.

Na Figura 1 é possível observar que a tabela `self_icmp` se encontra vazia, uma vez que a injeção de regras `ICMP` não foi forçada. Já na Figura 1 a tabela encontra-se povoada com uma entrada, sendo esta `{LPM:{10.2.0.1/32}, Action: reply_to_icmp}`.

4. Análise Experimental

Nesta secção descrevemos os testes realizados de modo a verificar a correta injeção de regras nos diversos *routers* da topologia. Como mencionado anteriormente, consideramos dois métodos de injeção de regras, a injeção normal, feita através da conexão `GRPC`, e a injeção forçada, utilizada no caso de faltas bizantinas, que utiliza o `simple_switch_CLI`.

4.1. Injeção de Regras

A injeção de regras nos diferentes *routers* pode ser verificada na Figura 3, onde cada linha com a cor verde representa uma regra injetada com sucesso no `router` e, cada linha vermelha uma regra que não foi injetada corretamente.

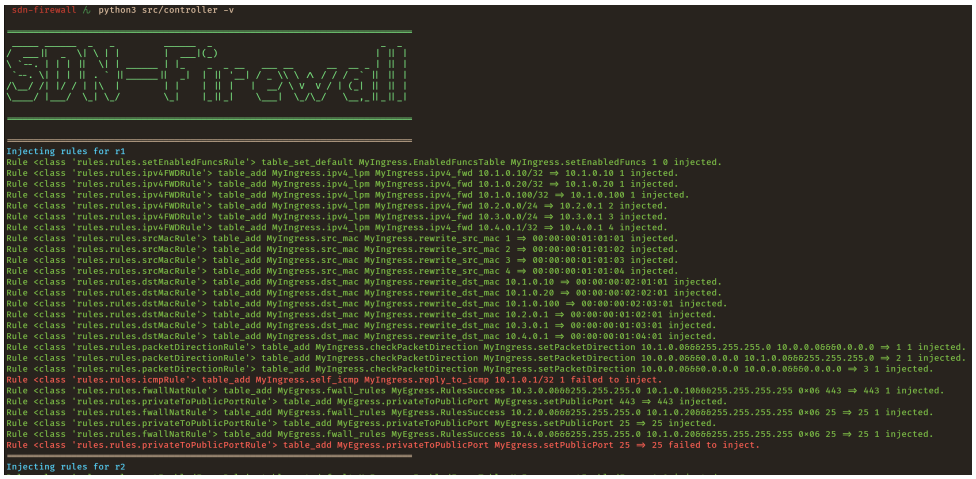

Figura 3: Injeção de regras com o controlador

Como podemos ver, existem duas regras cuja injeção não foi feita com sucesso. No teste seguinte descrevemos como podemos resolver este problema.

4.2. Injeção Forçada

A injeção de regras pode não ser concluída com sucesso, conforme o observado na Figura 3. Nesses casos é possível realizar uma injeção de regras **forçada** recorrendo à ferramenta `simple_switch_CLI`, porém este método causa alguns problemas de sincronização entre o controlador e os *targets*, *routers* da topologia geridos pelo controlador.

Note que, o facto das tabelas de regras estarem num estado *out of sync* não altera o comportamento geral da aplicação. Apenas compromete as leituras, aspeto que não é relevante no contexto do trabalho.

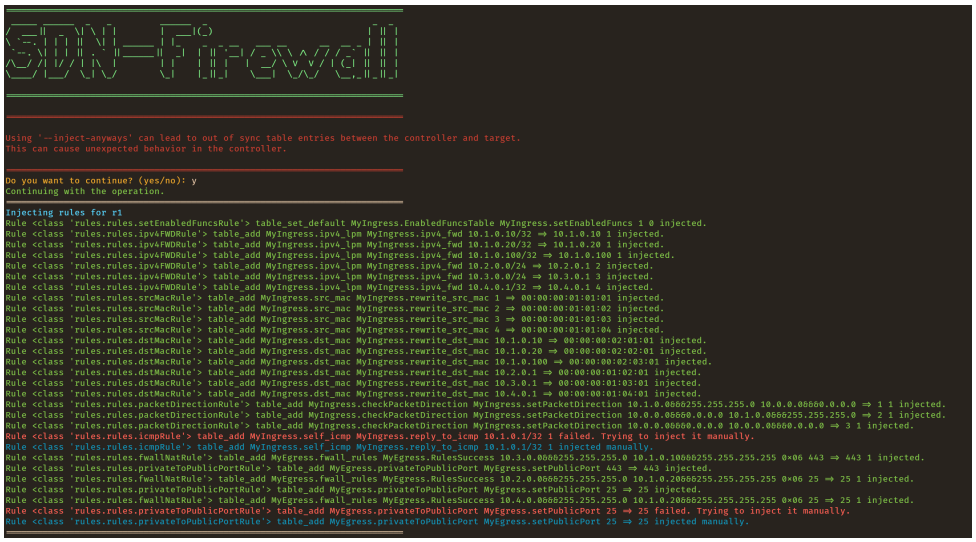

Figura 4: Injeção de regras com o controlador

Na Figura 4 é possível observar a injeção forçada de regras nos *routers*.

4.3. Modo Interativo

Como funcionalidade extra, o controlador suporta um modo de funcionamento *interativo*, onde o utilizador consegue, por meio de uma linha de comandos, executar *queries* aos diferentes dispositivos geridos.

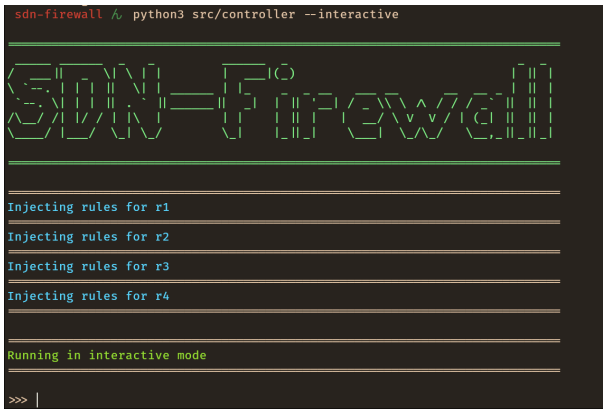

Figura 5: *Prompt* do modo interativo.

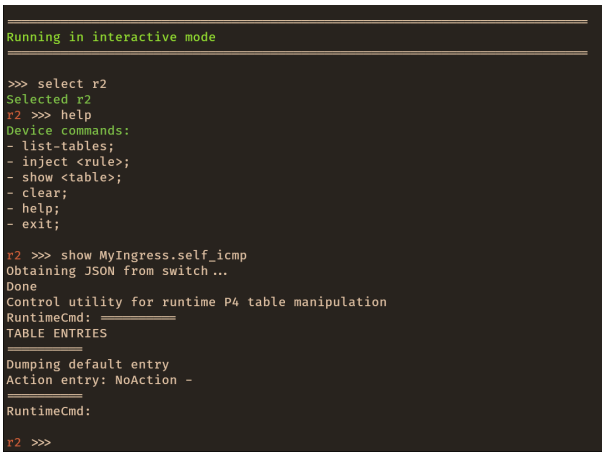

Figura 6: Operações no *router* 2.

Na Figura 6 é possível observar que o dispositivo selecionado no modo interativo é o *R2*, *router* 2 e diversas operações realizadas sobre este, como, listagem de tabelas, injeção de regras e descrição de tabelas.

5. Instruções de Execução

Devido à estrutura complexa do projeto, ficheiros de configuração e diversas dependências, executar o controlador pode ser algo com alguma complexidade.

Os seguintes comandos ilustram um *"how to guide"* de como inicializar o controlador:

1. Instalação de dependências usadas pela linguagem `Python`:

   ```
   pip install grpcio==1.51.3 protobuf==4.21.6 p4runtime==1.3.0
   ```

2. Adicionar os repositórios para `P4Runtime`:

```
> . /etc/os-release
> curl -sS \
"http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${VERSION_ID}/
Release.key" \
| gpg --dearmor | sudo tee /usr/share/keyrings/p4lang.gpg
echo "deb [signed-by=/usr/share/keyrings/p4lang.gpg] \
http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${VERSION_ID}/ /" \
| sudo tee /etc/apt/sources.list.d/p4lang.list
> sudo apt update
```

3. Instalar `p4lang-pi`:

   ```
   sudo apt install p4lang-pi
   ```

4. Copiar os módulos de `P4Runtime` para a diretoria `.local`:

   ```
   sh tools/cp-python-libs.sh
   ```

   Este passo difere de máquina para máquina, mas em princípio deverá ser só executar o *script* dentro da diretoria `tools`

3. Compilar o código `P4` para `JSON` e `p4info`:

   ```
   p4c-bm2-ss --p4v 16 src/etapa2/router.p4 --p4runtime-files json/simple-
   router.p4.p4info.txt -o json/simple-router.json
   ```

4. Iniciar o controlador com os parâmetros específicos:

   ```
   python src/controller --help
   ```

Depois de executar todos os passos, o controlador deve ser executado na janela de acordo com os parâmetros passados.

6. Conclusão

Com isto, damos por concluído o trabalho prático, consideramos uma etapa bastante interessante, uma vez que, finalmente, introduzimos um controlador na aplicação. Apesar do trabalho ter sido, praticamente, apenas a adaptação de código antigo, ainda foram encontradas algumas dificuldades devido à complexidade das regras e respetivas tabelas, assim como a falta de documentação "explícita" relativa ao `P4Runtime`.

Em particular, o facto da maior parte das regras serem injetadas com sucesso e depois existirem duas inválidas, dificultou o processo de desenvolvimento e ficou por resolver.

Relativamente a trabalho futuro, o grupo gostaria de, primeiramente, resolver todos os problemas relativos à injeção de regras, e, no modo interativo, utilizar comunicação nativa (da plataforma `P4Runtime`) em vez da ferramente `simple_switch_CLI`, de modo a evitar a divergência das tabelas.