UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDREI POCHMANN KOENICH

TURMA U

TRABALHO PRÁTICO: EXPLORANDO P4 PARA ANÁLISE DE DESEMPENHO

Disciplina: Protocolos de Comunicação

Professor: Alberto Egon Schaeffer Filho

Porto Alegre, agosto de 2024.

1 DESCRIÇÃO LÓGICA DO FUNCIONAMENTO DO MECANISMO

O mecanismo consiste na utilização da linguagem P4 para implementação de um sistema de *In-Band Network Telemetry* (INT). Esse sistema permite a coleta de informações de telemetria nos *switches* que compõem a topologia da rede de computadores a ser analisada. Essas informações de telemetria são inseridas no pacote em circulação na rede, em cabeçalhos específicos relacionados com essa finalidade.

Uma vez que um pacote é inserido na rede, tal pacote é incrementado com a presença de um novo cabeçalho denominado `int_pai`, além de outro cabeçalho `int_filho`, ambos contendo informações relacionadas com a telemetria a ser realizada. Em seguida, a cada novo salto realizado na topologia da rede, um novo cabeçalho `int_filho` é adicionado, de tal forma que esse novo cabeçalho seja colocado antes de todos os cabeçalhos `int_filho` anteriores. Todos os cabeçalhos `int_filho` são colocados depois do primeiro (e único) cabeçalho `int_pai`. Os cabeçalhos de telemetria são inseridos entre o cabeçalho TCP e o *payload* existente no pacote em circulação.

Em cada cabeçalho `int_filho`, são coletadas as métricas de *timestamp* de entrada do pacote no *switch*, porta de entrada do *switch*, porta de saída do *switch* e do número de identificação (ID) do *switch*. Assim, o cabeçalho `int_filho` possui um total de quatro campos, todos utilizados na operação de telemetria.

O cabeçalho `int_pai`, por sua vez, contém três campos, sendo um utilizado para indicar a quantidade de *bytes* de telemetria (adicionados, até então, no pacote em circulação na rede analisada), e outro para indicar a quantidade de cabeçalhos `int_filho` adicionados até o momento. O código em P4 é configurado de modo a não permitir que o *Maximum Transmission Unit* (MTU) da rede (considerada como igual a 1500 *bytes* na implementação) seja ultrapassado, em função da inserção de uma certa quantidade de cabeçalhos de telemetria. Para isso, utiliza-se um terceiro campo no cabeçalho `int_pai` para indicar se, em determinado momento da circulação do pacote na rede, foi necessário reter a inserção de novos cabeçalhos de telemetria, para que o MTU da rede não fosse ultrapassado. Em suma, um cabeçalho de telemetria é inserido somente se o tamanho do pacote resultante não ultrapassar o MTU considerado.

No arquivo `.zip` submetido, a emissão de um determinado pacote por um *host* remetente é simulada com a utilização da aplicação em *Python* `send.py`, enquanto a recepção do pacote por um *host* destinatário é simulada com a utilização da aplicação em *Python* `receive.py`. Ambas as aplicações utilizam a biblioteca *Scapy* para manipulação de pacotes. O diretório `config-topo` contém os arquivos de extensão `.json` responsáveis por simular os planos de dados da topologia de rede utilizada nos experimentos conduzidos, demonstrados na Seção 2. O arquivo `basic.p4` contém todas as ações a serem realizadas por cada *switch* da topologia, quando do recebimento de um pacote qualquer.

2 RESULTADOS DE EXPERIMENTOS CONDUZIDOS

Na Figura 1, observa-se a topologia da rede (configurada por meio dos arquivos `.json` no diretório `config-topo`) que foi utilizada para os testes.

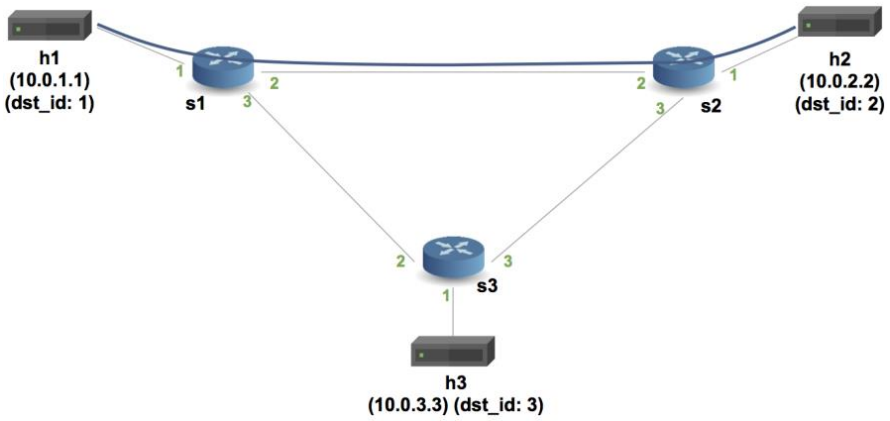

Figura 1 – Topologia da Rede Considerada nos Testes

Na Figura 2, observamos um caso de teste no qual um pacote foi emitido pelo *host* `h1`, tendo o *host* `h2` como destinatário. Assim, esse pacote, após deixar o *host* `h1`, adentra o *switch* `s1` por meio da porta 1, em seguida deixando o *switch* `s1` pela porta 2 e sendo encaminhado para a porta 2 do switch `s2`. Por fim, o pacote deixa o *switch* `s2` por meio da porta 1, sendo finalmente entregue para o destinatário (*host* `h2`). Analisando os terminais (que mostram o conteúdo do pacote no momento do envio e no momento do recebimento) dos dois *hosts* envolvidos na Figura 2, percebe-se que todos os campos dos cabeçalhos `int_filho` foram preenchidos conforme a situação descrita, assim como os campos do cabeçalho `int_pai`, que indicam a presença de dois cabeçalhos `int_filho` no pacote, além de um tamanho total cabeçalhos filhos igual a 26 *bytes*. O *payload* do pacote enviado contém a mensagem "*Hello*".

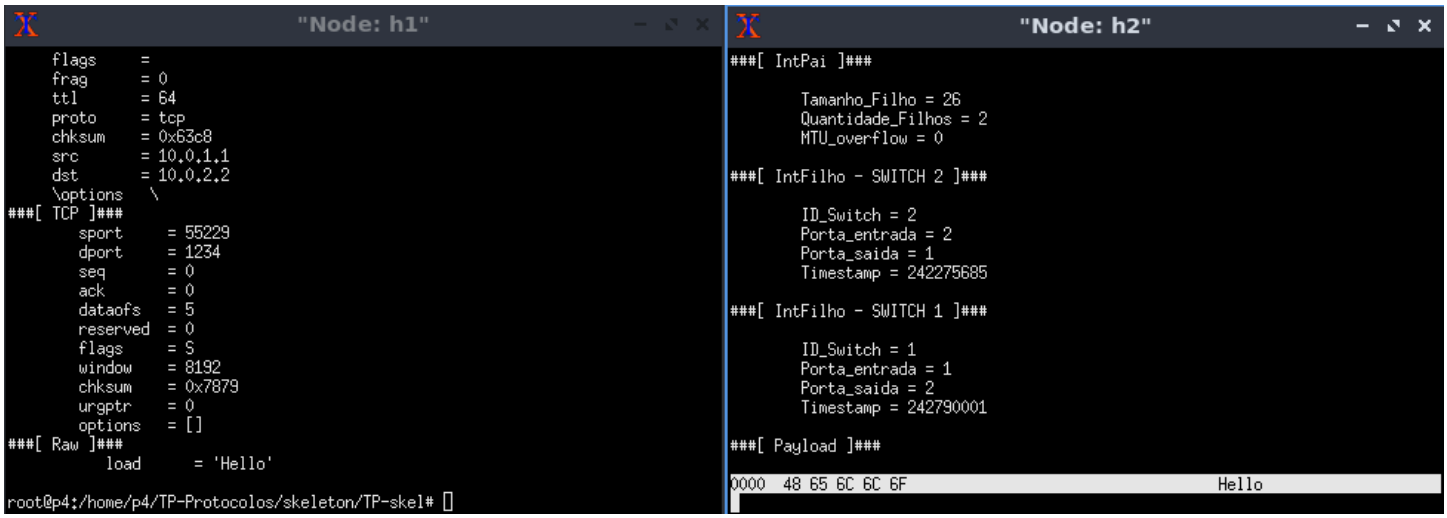

Figura 2 – Informações do Conteúdo do Pacote Exibidas no Terminal do Emissor e do Receptor (Teste 1)

A Figura 3 mostra o resultado de um teste semelhante ao anterior, com a diferença de que o *host* `h3` é o emissor do pacote, e o *host* `h1` é o receptor. Novamente, é possível perceber que os campos dos cabeçalhos `int_filho` são preenchidos de acordo com a topologia e os números de porta indicados na Figura 1.

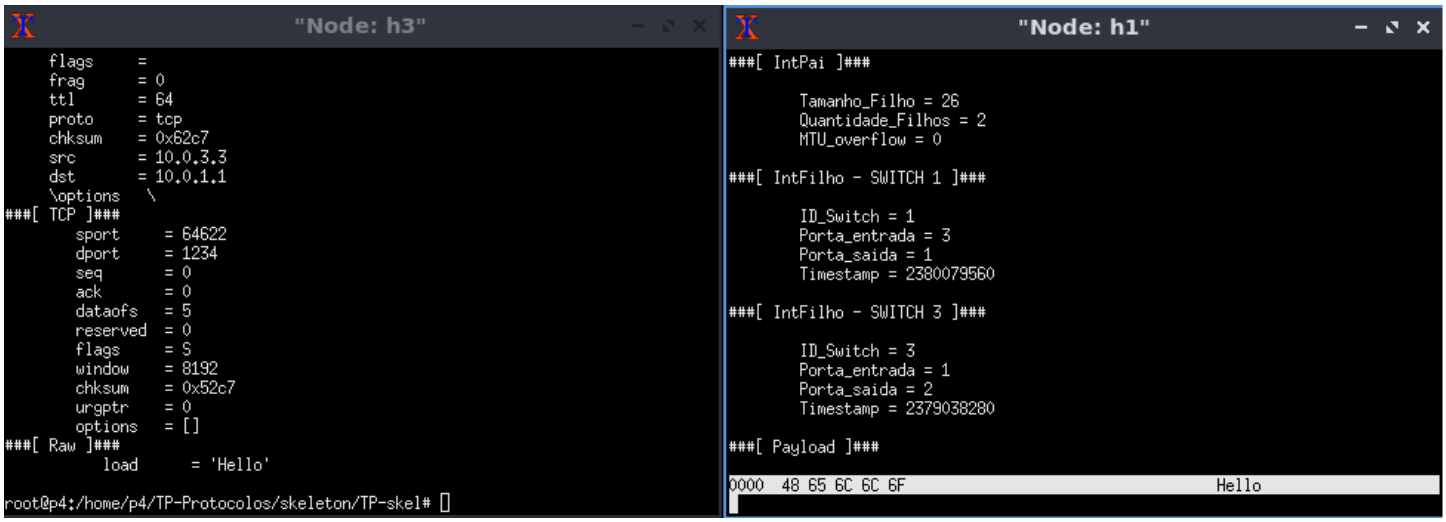

Figura 3 – Informações do Conteúdo do Pacote Exibidas no Terminal do Emissor e do Receptor (Teste 2)

Na Figura 4, foi realizado um teste no qual foi possível adicionar apenas um cabeçalho `int_filho`, uma vez que a adição de um segundo cabeçalho `int_filho` ultrapassaria o MTU. Para a realização desse teste em específico, o valor do MTU considerado pelos *switches* foi propositalmente configurado como sendo igual a 80 *bytes*, a fim de demonstrar o mecanismo de controle da ultrapassagem do MTU implementado no código em P4. Nota-se que, nesse caso, o campo `MTU_overflow` do cabeçalho `int_pai` possui valor igual a um, ao contrário dos outros dois testes anteriores, nos quais esse campo possuía valor igual a zero. No caso de teste do Figura 4, novamente, um pacote foi emitido pelo *host* `h1`, tendo o *host* `h2` como destinatário.

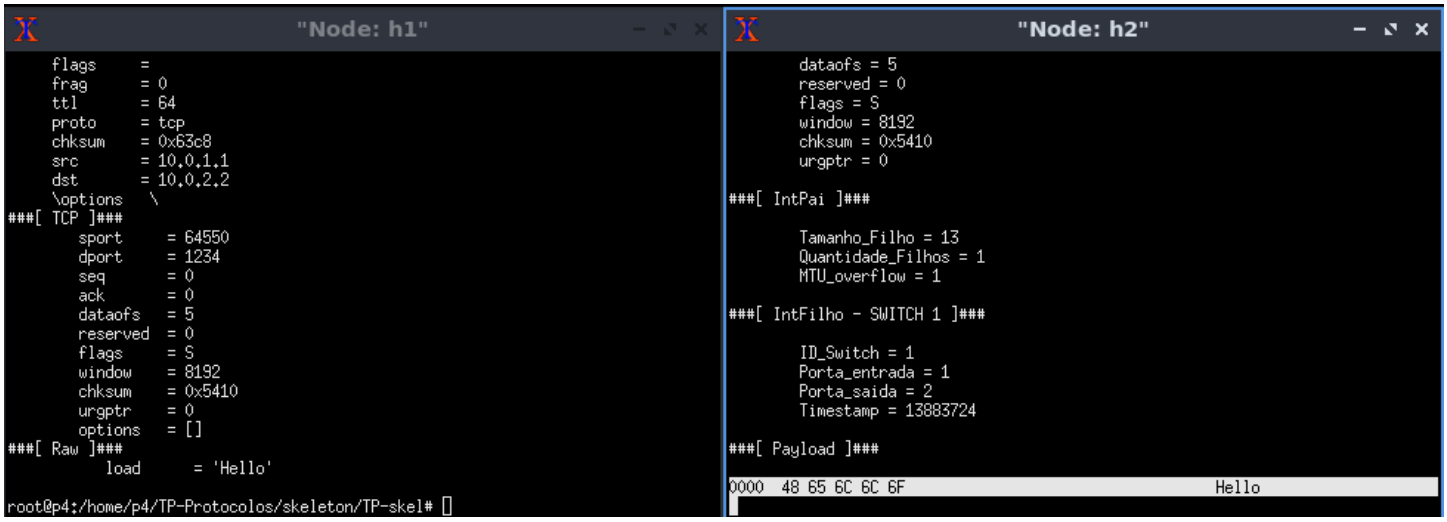

Figura 4 – Informações do Conteúdo do Pacote Exibidas no Terminal do Emissor e do Receptor (Teste 3, com Presença de *MTU Overflow*)

3 DESCRIÇÃO DOS EVENTUAIS PROBLEMAS ENCONTRADOS DURANTE A IMPLEMENTAÇÃO

Na aplicação em *Python* `receive.py`, os métodos de alto nível `show()` e `show2()` da biblioteca *Scapy* não exibiam corretamente os dados dos cabeçalhos de telemetria, quando do recebimento de um pacote. Entretanto, a função `hexdump()` indicava que os pacotes recebidos possuíam, de fato, todos os cabeçalhos preenchidos corretamente. Assim, ao invés de utilizar os métodos `show()` ou `show2()`, foi criada uma lógica de implementação para percorrer o conteúdo do pacote byte a byte, exibindo os campos dos cabeçalhos do pacote conforme o desejado. Essa lógica está presente na função `handle_pkt()`, definida a partir da linha 31, no arquivo `receive.py`.

4 PASSOS TOMADOS PARA IMPLEMENTAÇÃO DOS REQUISITOS FUNCIONAIS

- **Verificação da presença do cabeçalho int_pai e inclusão deste cabeçalho nos pacotes em que ele não esteja presente (linhas 115 e 147 do arquivo `basic.p4`):** o campo `etherType` do cabeçalho *Ethernet* é responsável por indicar se o cabeçalho `int_pai` já foi inserido. Para realizar essa indicação, o valor do campo é alterado para 0x1212, na *action* `ipv4_forward`. A verificação da presença do cabeçalho `int_pai` é realizada no *parser* do *switch*, especificamente no *state* `parse_tcp`.
- **Criação e inserção, a cada *hop*, de um cabeçalho `int_filho` (linhas 188 e 208 do arquivo `basic.p4`):** após determinar se um novo cabeçalho `int_filho` pode ser inserido sem ultrapassagem do MTU, a ação `add_int_filho()` (definida na tabela `new_int_filho`) é utilizada para inserir um novo cabeçalho de telemetria do tipo `int_filho()`.
- **Preenchimento dos campos dos cabeçalhos filhos com as informações dos *switches* (linha 167 do arquivo `basic.p4`):** feito utilizando a ação `add_int_filho()` (definida na tabela `new_int_filho`). Essa ação também realiza a leitura do ID e dos números de porta do *switch* em questão, por meio da análise do seu arquivo `.json` correspondente, para gravar esses valores nos campos do cabeçalho `int_filho`.
- **Ao chegar no *host* destino, o *host* deverá ser capaz de, por *software*, extrair as informações do pacote e separá-las do *payload* original do pacote (linha 79 até a linha 96 do arquivo `receive.py`):** a lógica implementada em *Python* é responsável por analisar cada um dos cabeçalhos de telemetria encadeados, revelando os dados de cada campo referente a cada cabeçalho, assim como o *payload*, separadamente.