Universidade Federal do Rio Grande do Sul
Instituto de Informática



INF01002
Protocolos de Comunicação

# Trabalho Prático

## Crash Course sobre In-Band Network Telemetry

Luca Pasquetti Comelli Fritscher (00343044)
Miguel Dutra Fontes Guerra (00342573)

19 de junho de 2025

# Sumário


1.1 Introdução . . . . . 2
2.1 Fundamentos Técnicos . . . . . 2
2.1.1 In-Band Network Telemetry (INT) . . . . . 2
2.1.2 Linguagem P4 . . . . . 2
2.1.3 Controle de MTU (Bônus) . . . . . 3
2.1.4 Ambiente de Testes e Scripts Auxiliares . . . . . 3
3.1 Implementação . . . . . 3
3.1.1 Estrutura Geral da Solução . . . . . 3
3.1.2 Implementação no Arquivo `basic.p4` . . . . . 3
3.1.3 Envio de Pacotes (`send.py`) . . . . . 6
3.1.4 Recepção de Dados (`receive.py`) . . . . . 6
3.1.5 Considerações sobre Generalização . . . . . 6
4.1 Resultados . . . . . 7
4.1.1 Metodologia dos Testes . . . . . 7
4.1.2 Experimentos Realizados . . . . . 7
4.1.3 Controle de Estouro de MTU (Bônus) . . . . . 9
4.1.4 Resumo dos Resultados . . . . . 9
Bibliografia . . . . . 10

## 1.1 Introdução

O crescimento do tráfego de dados em redes modernas impõe desafios crescentes à gestão do desempenho e à identificação de gargalos. Nesse contexto, surge o conceito de *In-Band Network Telemetry* (INT), um mecanismo capaz de embutir dados de monitoramento diretamente nos pacotes em trânsito na rede, permitindo observá-los em tempo real e com maior detalhe.

Em paralelo, temos o avanço das redes programáveis, especialmente com o surgimento da linguagem P4, que possibilita o controle direto dos switches. Através de definições personalizadas de cabeçalhos, parsers e tabelas de encaminhamento, é possível implementar funcionalidades como o INT de forma flexível, eficiente e extensível.

O trabalho trabalho prático se concentra em explorar o uso da linguagem P4 para implementar um mecanismo completo de *In-Band Network Telemetry*, capaz de registrar e transmitir métricas de rede relevantes em cada salto percorrido pelos pacotes. Pra isso, pacotes IP enviados entre hosts da rede são interceptados por switches programáveis, que adicionam cabeçalhos INT contendo informações como ID do switch, timestamp de chegada, tamanho do pacote, entre outros.

#### Objetivo do Trabalho

O objetivo principal é projetar, implementar e validar um sistema de monitoramento baseado em INT, que permite:

- Inserção automática de cabeçalhos de telemetria nos pacotes;
- Coleta de dados relevantes em cada salto da rede;
- Interpretação e separação dos dados de telemetria no destino final;
- Detecção de situações de estouro do MTU como requisito bônus.

A solução opera em topologias genéricas e utiliza recursos oferecidos pelo ambiente de simulação baseado em *Mininet* e *BMv2*.

## 2.1 Fundamentos Técnicos

### 2.1.1 In-Band Network Telemetry (INT)

*In-Band Network Telemetry* é uma técnica de redes programáveis que permite que as informações de desempenho e estado da rede sejam embarcadas nos próprios pacotes de dados. Ao contrário de mecanismos de monitoramento tradicionais, o INT garante que cada pacote colete dados em tempo real sobre os switchs que eles passam, permitindo identificar problemas de desempenho e gargalos com precisão, além da análise de tráfego em tempo real. O funcionamento básico desse mecanismo pode ser descrito da seguinte forma:

1. Quando entram na rede, cada pacote recebe um cabeçalho de telemetria caso ainda não tenha.
2. A cada salto, o switch checa os cabeçalhos, atualiza a contagem e insere um novo cabeçalho filho com dados já exemplificados anteriormente.
3. No final do percurso, os dados de telemetria acumulados podem ser entregues diretamente ao destino para análise ou, extraídos e enviados a um mecanismo externo de monitoramento.

### 2.1.2 Linguagem P4

A linguagem **P4 (Programming Protocol-independent Packet Processors)** é utilizada para programar o comportamento dos switches no plano de dados, permitindo definir a estrutura de pacotes, parsers, ações e tabelas de encaminhamento personalizados. A arquitetura do switch usada possui os seguintes blocos principais:

- **Parser**: responsável por extrair cabeçalhos dos pacotes.
- **Ingress e Egress**: blocos onde são aplicadas as ações de encaminhamento e inserção dos cabeçalhos INT.
- **Deparser**: reconstrói o pacote a ser reenviado.

### 2.1.3 Controle de MTU (Bônus)

A especificação propõe como requisito adicional a verificação do tamanho total do pacote durante a inserção dos cabeçalhos INT. Se o tamanho final ultrapassar os 1500 bytes (valor do MTU), a inserção de novos cabeçalhos filhos é interrompida, e uma flag de `mtu_overflow` é marcada no cabeçalho pai.

Essa lógica foi implementada utilizando o campo `packet_length` disponível nos metadados padrão do P4, garantindo que os pacotes não sejam fragmentados ou descartados em redes reais.

### 2.1.4 Ambiente de Testes e Scripts Auxiliares

Para realizar os testes, foi utilizada uma topologia em **Mininet**, com switches programáveis executando o código em P4, que foi previamente fornecido pelo professor, já com o compilador P4 e a infraestrutura de rede. Dois scripts em Python com a biblioteca **Scapy** foram usados:

- `send.py`: gera pacotes dos tipos Ethernet, IP, TCP com payload arbitrário, simulando uma aplicação.
- `receive.py`: escuta os pacotes recebidos, extrai os cabeçalhos INT e apresenta os dados com o payload final.

Os cabeçalhos customizados também foram definidos em Python no arquivo `INT_headers.py`, usado pra interpretar os campos de telemetria.

```
================================================================
Welcome to the BMV2 Mininet CLI!
================================================================
Your P4 program is installed into the BMV2 software switch
and your initial runtime configuration is loaded. You can interact
with the network using the mininet CLI below.

To inspect or change the switch configuration, connect to
its CLI from your host operating system using this command:
  simple_switch_CLI --thrift-port <switch thrift port>

To view a switch log, run this command from your host OS:
  tail -f /home/p4/Crash_Course_sobre_INT/skeleton/TP-skel/logs/<switchname>.log

To view the switch output pcap, check the pcap files in /home/p4/Crash_Course_sobre_INT/skeleton/TP-skel/pcaps:
 for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

To view the P4Runtime requests sent to the switch, check the
corresponding txt file in /home/p4/Crash_Course_sobre_INT/skeleton/TP-skel/logs:
 for example run:  cat /home/p4/Crash_Course_sobre_INT/skeleton/TP-skel/logs/s1-p4runtime-requests.txt

mininet>
```

Figura 1: Ambiente de Teste

## 3.1 Implementação

### 3.1.1 Estrutura Geral da Solução

O sistema desenvolvido é composto por três elementos principais `basic.p4`, que implementa a lógica de inserção e propagação dos cabeçalhos INT pai e filhos, `send.py` e `receive.py` que foram explicados no capítulo anterior.

### 3.1.2 Implementação no Arquivo `basic.p4`

#### Estrutura de Cabeçalhos

Dois novos cabeçalhos foram definidos para a telemetria,sendo adicionados diretamente após o cabeçalho Ethernet, precedendo o IPv4 original. São eles:

- `int_parent_t`: Contém informações como o número de filhos (saltos), o comprimento dos cabeçalhos, e um campo de flag pro estouro de MTU.
- `int_child_t`: Armazena as métricas dos saltos dados, como o ID do switch, timestamp e comprimento do pacote.

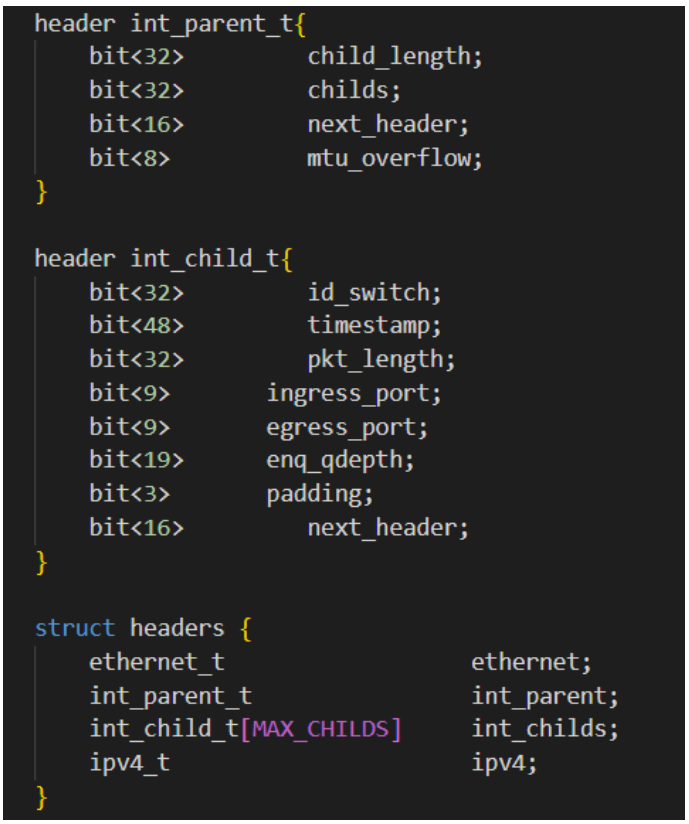



Figura 2: Estrutura Cabeçalhos

### Parser

O `parser` foi estendido para reconhecer os novos cabeçalhos INT com base nos campos `etherType` e `next_header`, permitindo leitura encadeada de múltiplos `int_child_t`.

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_INT_PARENT: parse_int_parent;
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    // State to process the INT Parent header
    state parse_int_parent {
        packet.extract(hdr.int_parent);
        transition select(hdr.int_parent.next_header) {
            TYPE_INT_CHILD: parse_int_child;
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    // State to process the INT Child header
    state parse_int_child {
        packet.extract(hdr.int_childs.next);
        transition select(hdr.int_childs.last.next_header) {
            TYPE_INT_CHILD: parse_int_child;
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }
```

Figura 3: Estrutura `Parser`

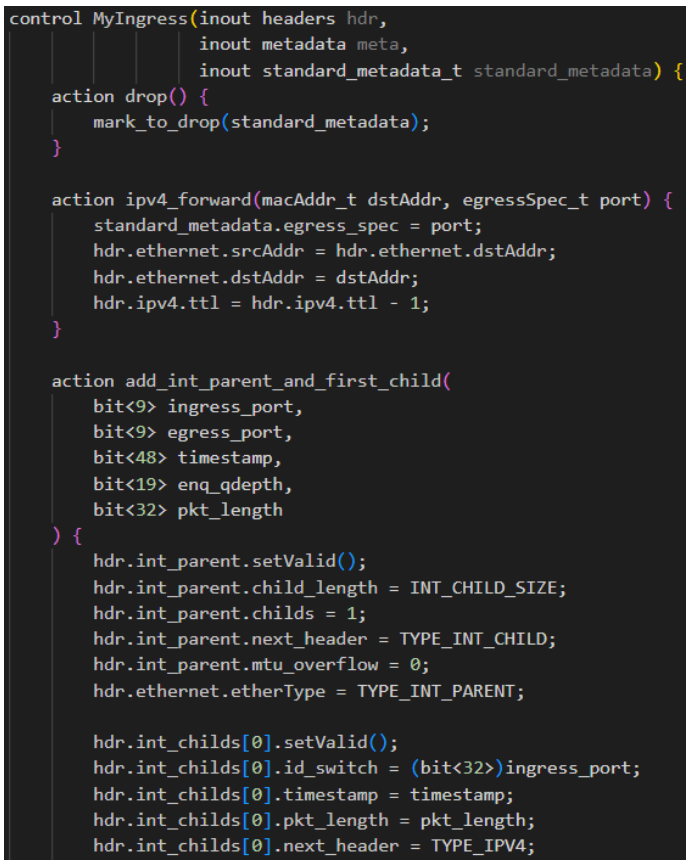
### Lógica de Ingresso

A lógica de inserção dos cabeçalhos INT foi implementada no bloco `MyIngress`:

- Se o pacote **não possui cabeçalho INT**, irão ser inseridos um `int_parent_t` e o primeiro `int_child_t` com os dados do salto atual.
- Se o cabeçalho pai já existe, um novo filho é adicionado com os dados do salto atual.

Também foi implementado o bônus de controle de **MTU**. Caso a adição de um novo cabeçalho filho exceda os 1500 bytes:

- O campo `mtu_overflow` do cabeçalho `int_parent_t` é marcado como 1;
- O pacote segue sem adicionar novos filhos, prevenindo fragmentação.



```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }

    action add_int_parent_and_first_child(
        bit<9> ingress_port,
        bit<9> egress_port,
        bit<48> timestamp,
        bit<19> enq_qdepth,
        bit<32> pkt_length
    ) {
        hdr.int_parent.setValid();
        hdr.int_parent.child_length = INT_CHILD_SIZE;
        hdr.int_parent.childs = 1;
        hdr.int_parent.next_header = TYPE_INT_CHILD;
        hdr.int_parent.mtu_overflow = 0;
        hdr.ethernet.etherType = TYPE_INT_PARENT;

        hdr.int_childs[0].setValid();
        hdr.int_childs[0].id_switch = (bit<32>)ingress_port;
        hdr.int_childs[0].timestamp = timestamp;
        hdr.int_childs[0].pkt_length = pkt_length;
        hdr.int_childs[0].next_header = TYPE_IPV4;
```

Figura 4: Estrutura `MyIngress`

**Deparser**

O deparser foi configurado para emitir os cabeçalhos na ordem: `Ethernet` → `INT Pai` → `INT Filhos` → `IPv4`.

### 3.1.3 Envio de Pacotes (`send.py`)

O script `send.py` permite o envio de pacotes TCP com uma mensagem textual arbitrária como *payload*. Usa a biblioteca **Scapy** para construir pacotes da forma:

- Cabeçalho Ethernet com broadcast;
- Cabeçalho IP com destino passado por argumento;
- Cabeçalho TCP com porta de destino 1234;
- Mensagem como carga útil.

O código é utilizado para testar a propagação dos pacotes e a inserção de cabeçalhos INT nos switches.

### 3.1.4 Recepção de Dados (`receive.py`)

O script `receive.py` escuta pacotes na interface `eth0` e utiliza o Scapy para interpretar os dados. Quando detecta pacotes com os cabeçalhos `INTParent` e `INTChild` mostra os campos do cabeçalho pai, incluindo a flag de `MTU overflow`, itera sobre os filhos, exibindo os dados de cada salto, e imprime o *payload* original separadamente.

Assim, o host de destino pode interpretar corretamente os dados de telemetria e a mensagem enviada.

### 3.1.5 Considerações sobre Generalização

A implementação foi feita pra funcionar com topologias arbitrárias, usando de arrays fixos para os filhos no P4 é possível fazer o registro de múltiplos saltos sem restrições de topologia.

Além disso, o controle de MTU garante que os pacotes não sejam descartados por crescimento excessivo de cabeçalhos.

## 4.1 Resultados

Foram feitos experimentos usando o *Mininet*, com pacotes trafegando entre dois hosts e com switches intermediários, pra verificar a inserção dos cabeçalhos INT, a coleta de métricas em cada salto e o controle de estouro do MTU.

### 4.1.1 Metodologia dos Testes

Os testes foram realizados com base na topologia do esqueleto do trabalho, contendo ao menos dois switches entre os hosts. O processo de teste seguiu os seguintes passos:

1. Inicialização da rede via comando `make` dentro do diretório do projeto.
2. Abertura de terminais para os hosts `h1` e `h2` com `xterm h1 h2`.
3. Execução do `receive.py` em `h1` para escutar os pacotes recebidos.
4. Envio de mensagens a partir de `h2` usando o `send.py <destino> <mensagem>`.
5. Análise da saída do `receive.py` para inspecionar os cabeçalhos INT e o payload.

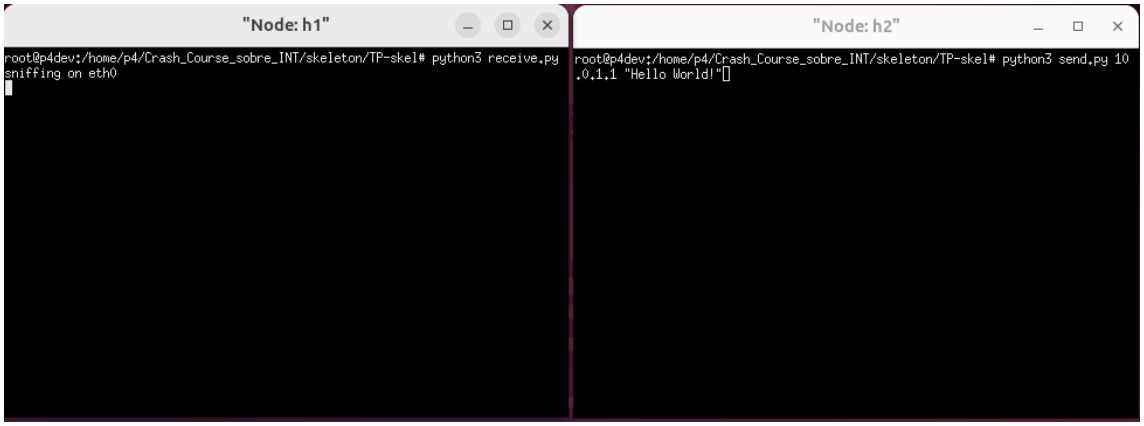



Figura 5: Teste Base

### 4.1.2 Experimentos Realizados

**Teste com Pacote sem Cabeçalho INT**

Ao enviar um pacote sem cabeçalho INT inicial, foi verificado que o primeiro switch da rede inseriu corretamente o cabeçalho pai e o primeiro cabeçalho filho. Os switches seguintes adicionaram seus respectivos cabeçalhos filhos, conforme esperado.

- **Resultado**: Todos os campos dos cabeçalhos (o `id_switch`, o `timestamp`, o `pkt_length` e o `next_header`) foram exibidos corretamente no terminal do receptor.

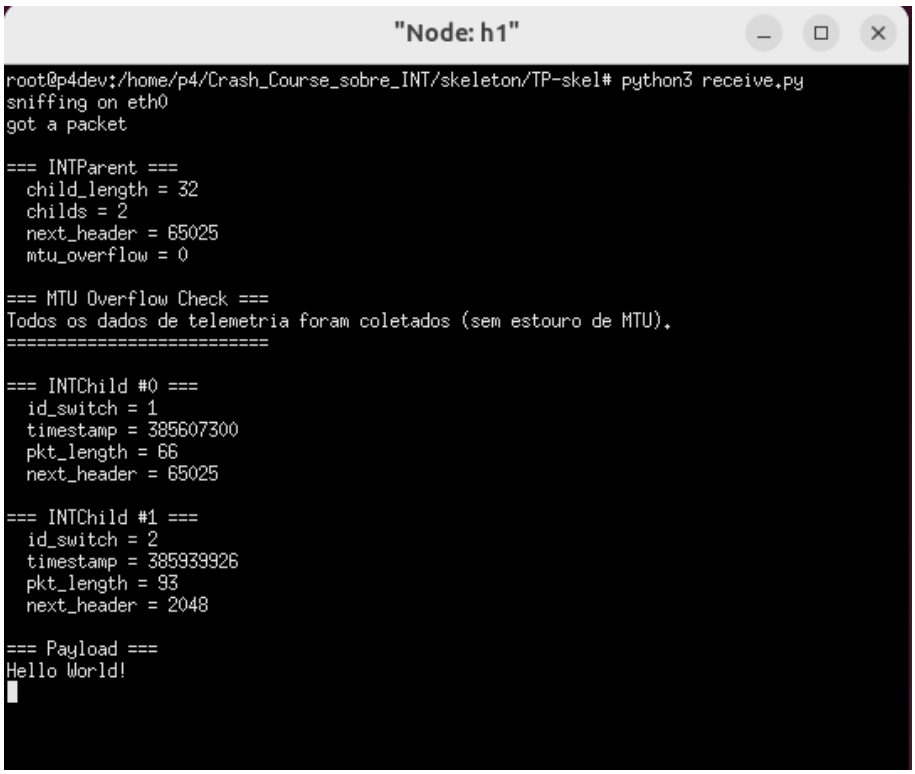



Figura 6: Resultado no h1

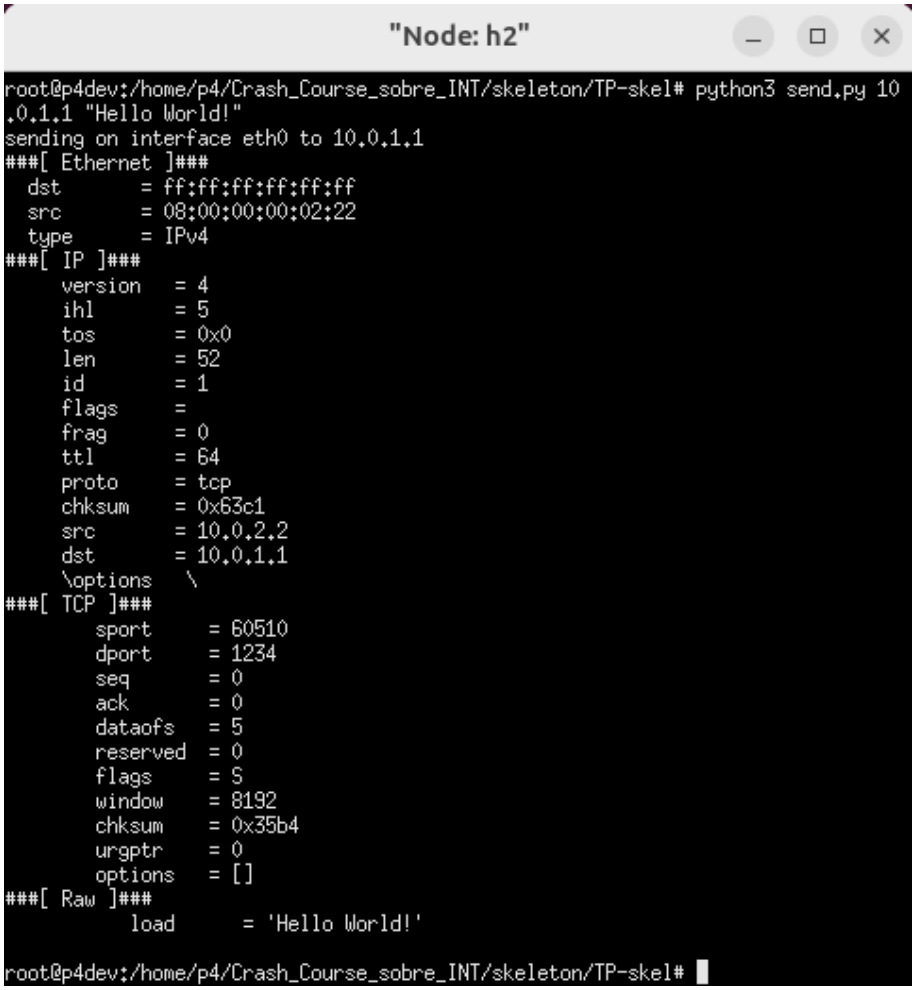



Figura 7: Resultado no h2

**Verificação de Ordem e Encadeamento dos Cabeçalhos**

A análise do `receive.py` confirmou que os cabeçalhos filhos estavam corretamente encadeados em ordem, de acordo com os saltos percorridos, graças ao campo `next_header` presente em cada filho.

**Verificação do Payload**

Após o processamento dos cabeçalhos INT, o script foi capaz de exibir o *payload* original da mensagem enviada, separando-o da estrutura de telemetria.

### 4.1.3 Controle de Estouro de MTU (Bônus)

Foi simulado um cenário onde o pacote se aproximava do limite de 1500 bytes. Nesse caso, o código detectou que a adição de um novo cabeçalho ultrapassaria o MTU e evitou sua inserção, onde a flag `mtu_overflow` do cabeçalho pai foi setada para `1`.


Figura 8: Teste *MUT*

### 4.1.4 Resumo dos Resultados

- Os cabeçalhos INT foram corretamente inseridos, encadeados e lidos;
- Métricas como `id_switch`, `timestamp` e `pkt_length` foram exibidas com sucesso;
- O controle de MTU foi funcional, com marcação adequada da flag de overflow;
- A carga útil (mensagem enviada) foi preservada e recuperada no destino.

# Bibliografia


[1] *BMv2 Simple Switch documentation*. https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md. Acesso em: jun. 2025.

[2] Pat Bosshart et al. *P4: Programming Protocol-Independent Packet Processors*. Acesso em: jun. 2025.

[3] *In-band Network Telemetry (INT), Working Draft*. https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf. Acesso em: jun. 2025.

[4] *P4 Cheat Sheet*. https://github.com/p4lang/tutorials/blob/master/p4-cheat-sheet.pdf. Acesso em: jun. 2025.

[5] *P416 Language Specification v1.2.5*. https://p4.org/wp-content/uploads/2024/10/P4-16-spec-v1.2.5.pdf. Acesso em: jun. 2025.