# EA080: Laboratório 06

**Vinícius Esperança Mantovani,**
**247395.**

## Introdução)

Neste experimento, foi estudada a linguagem *P4* e alguns dos conceitos que ela abrange, de modo a se conhecer a "forma P4" de se programar *switches* em *software*. Para tanto, foi utilizada uma topologia para todos os exercícios. Tal topologia é apresentada abaixo.

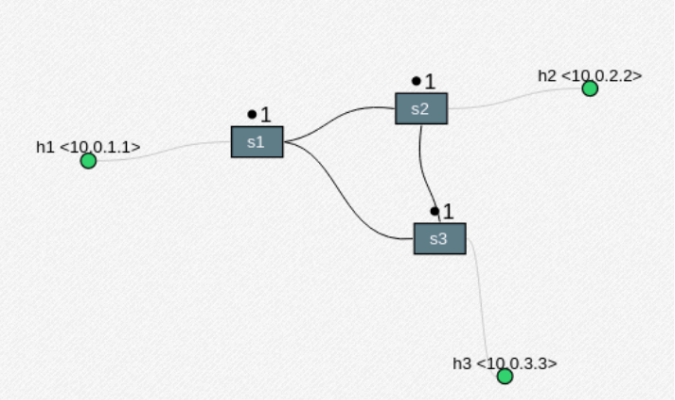


*Figura 1: Topologia comum do experimento*

Vale destacar, também, que neste experimento, o termo *switch* se refere a equipamentos genéricos de rede, não estando restritos a equipamentos de camada 2.

## Exercício 1)

Neste exercício, buscou-se analisar o código localizado em "P4-lab-2023/exercises/ex1/basic.p4", de modo a comentar a função de cada um de seus blocos no geral e no contexto de sua tarefa específica.

No contexto apresentado, tem-se o código abaixo com comentários localizando as regiões requeridas (headers, Parser dos cabeçalhos, Processamento de ingresso, Processamento de egresso, Deparser dos cabeçalhos e Cálculo do checksum):

```
C/C++
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_IPV4 = 0x800;

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
```

```
// DEFINICAO DE CABECALHOS
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
// DEFINICAO DE CABECALHOS
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

struct metadata {
    /* empty */
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}

// PARCER DOS CABECALHOS
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
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_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}
// PARCER DOS CABECALHOS
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑


control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}

// PROCESSAMENTO DE INGRESSO
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
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;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_lpm.apply();
```

```
        }
    }
}
// PROCESSAMENTO DE INGRESSO
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// PROCESSAMENTO DE EGRESSO
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}
// PROCESSAMENTO DE EGRESSO
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// CALCULO DO CHECKSUM
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
       update_checksum(
           hdr.ipv4.isValid(),
             { hdr.ipv4.version,
              hdr.ipv4.ihl,
               hdr.ipv4.diffserv,
               hdr.ipv4.totalLen,
               hdr.ipv4.identification,
               hdr.ipv4.flags,
               hdr.ipv4.fragOffset,
               hdr.ipv4.ttl,
               hdr.ipv4.protocol,
               hdr.ipv4.srcAddr,
               hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}
// CALCULO DO CHECKSUM
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// DEPARSER DOS CABECALHOS
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

```
// DEPARSER DOS CABECALHOS
↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

/************************************************************************
**********************  S W I T C H  ******************************
************************************************************************/

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Analisando o código apresentado, nota-se que os *headers* definidos são *ethernet_t* e *ipv4_t*, cada um determinando uma estrutura de dados para capturar partes do cabeçalho dos pacotes referentes a, respectivamente, camada de enlace (mac address de destino e origem e, tipo) e de rede (ipv4 address de destino e origem, protocolo, checksum…). Seguindo, tem-se o campo de *parsing*, que define a extração dos dados dos cabeçalhos na forma como definidos nos *headers* e, posteriormente, na estrutura de cabeçalhos criada, em que se tem *ethernet_t = ethernet* e *ipv4_t = ipv4*. Nesse processo de *parsing*, tem-se de começo a extração dos dados do cabeçalho relacionados a *ethernet* e, em sequência, a extração dos dados relacionados ao *ipv4*, caso o campo *etherType* do *header ethernet* seja *TYPE_IPV4*. Já em *MyIngress* são definidas as ações possíveis a serem tomadas para cada pacote (*drop* ou repasse para alguma porta de saída) e, em seguida, são definidos os moldes da tabela de roteamento com tamanho igual a 1024 (provavelmente linhas) e ações possíveis baseadas no endereço *ipv4* de destino do pacote que entra. Finalmente, ainda em *MyIngress*, é determinada a aplicação da tabela para pacotes que tenham cabeçalho *ipv4* válido.

Seguindo com a explicação do código, vê-se que o controle de egresso não contém determinação alguma, ou seja, o pacote sai diretamente por uma de suas portas, sem alterações no *egresso*. Já quanto ao cômputo do *checksum*, tem-se que, caso o cabeçalho *ipv4* seja válido, o *checksum* é calculado com base nos dados desse cabeçalho e atualizado no pacote. Por fim, no bloco *MyDeparser*, é determinada a alteração dos cabeçalhos dos pacotes antes de sua saída para que assumam os novos valores determinados durante sua passagem pelo *switch*.

**Exercício 2)**

Neste exercício, visou-se a estudar a execução de um código *p4*, cujas instâncias foram usadas para definir as tabelas de roteamento de cada *switch* (roteadores) da rede de topologia apresentada na *Figura 1*. Para tanto, utilizaram-se os códigos da pasta “P4-lab-2023/exercises/ex2/”, com um *make file* responsável por rodar todo o conjunto de códigos.

Após a execução do conjunto, foram abertos os terminais de *h2* e *h1* e, no primeiro, foi executado o código “./receive.py”, enquanto que, no segundo, executou-se “./send.py 10.0.2.2 "abacate"”. Desse modo, o *host h2* se tornou um receptor TCP pela interface *eth0* na porta “1234” e *h1* realizou uma conexão *TCP* com *h2*, transmitindo a mensagem entre aspas do comando executado em *x1*. As respostas dos dois terminais se apresentam abaixo, nas *Figuras 2* e *3*.

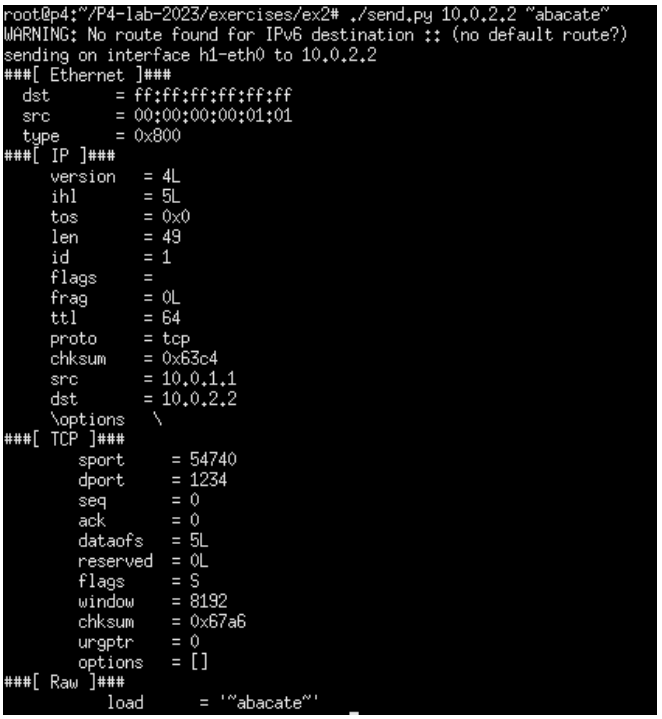


*Figura 2: Resposta do terminal de h1 para o comando send.py*

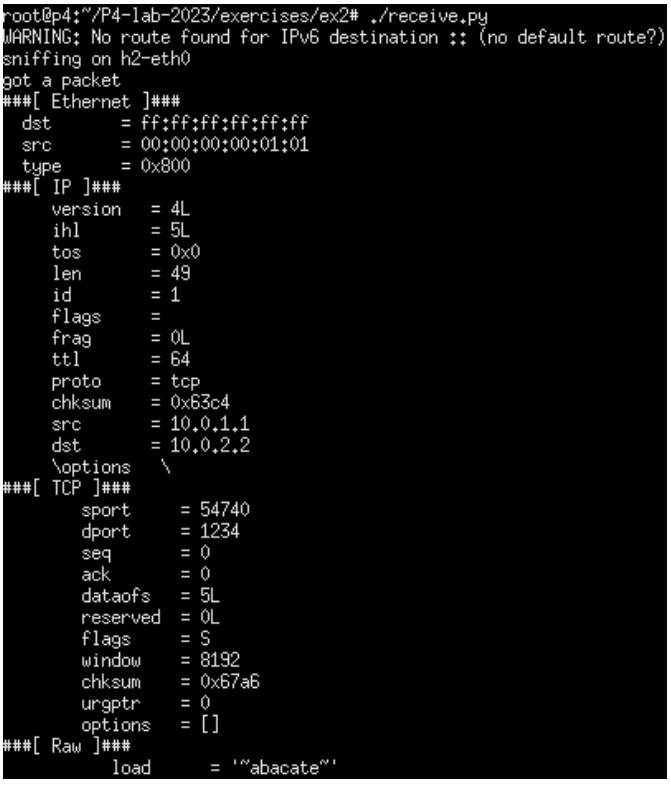


*Figura 3: Recepção da mensagem de h1 em h2 usando comando receive.py*

Observando as figuras acima, nota-se que os valores dos campos de endereço de destino e fonte *ethernet* se mantêm os mesmos no envio e na recepção. Isso se dá de maneira correta ante à programação dos *switches*, no entanto, de maneira incorreta quando analisada a função que os *switches* em questão visam a desempenhar. Isso porque, os *switches* aqui trabalham, em tese, como

roteadores, pois analisam os *IPs* para repasse e, por consequência, deveriam alterar o endereço MAC de fonte de todos os pacotes. Ademais, quanto ao endereço de destino, nota-se que ele também não é alterado, embora seja especificado nos *.json* dos *switches* que isso deveria ocorrer.

Abaixo, apresenta-se uma parte do código "s1_runtime.json", que demonstra o comportamento do *switch* como roteador à medida que se implementa um repasse baseado em endereço *IP*.


*Figura 4: Parte do arquivo s1_runtime.json*

Tratando-se do *TTL* do pacote no receptor e no emissor, nota-se que são iguais (64), de forma incorreta, uma vez que seu valor deveria ser decrescido a cada roteador pelo qual o pacote passa. Isso porque, o *TTL* visa a evitar que os pacotes circulem indefinidamente pela rede, de modo a matá-los no caso de darem muitos saltos antes de chegarem ao destino.

Seguindo, para corrigir os problemas apresentados acima, foram feitas modificações no código *P4*, conforme as mostradas na *Figura 5*. Para tanto, foi alterada a definição de *ipv4_foward*, responsável pelo encaminhamento de pacotes no caso de correspondência do *IP* com algum da tabela, de modo a adicionar a alteração do *TTL* e dos endereços de destino e fonte *MAC* nos pacotes. Vale ressaltar que, após as modificações, foi encerrada a emulação *mininet* e usado o comando "*make clean*" antes da posterior re-execução.

```
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;
}
```

*Figura 5: Alteração no script "basic.p4"*

Depois da alteração apresentada, foi executado o conjunto de códigos com "*make*" e repetido o procedimento de execução dos comandos "./send.py 10.0.2.2 "abacate"" e "./receive.py" nos *hosts*. Feito isso, os resultados obtidos foram:

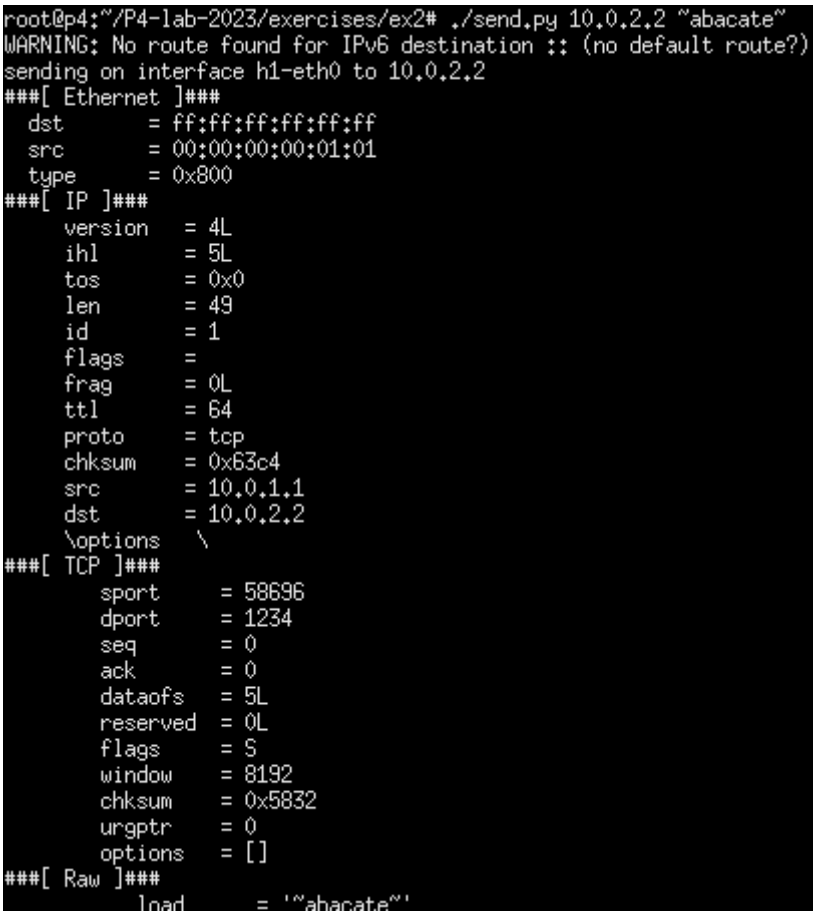


*Figura 6: Resposta do terminal de h1 para o comando send.py, após correções*

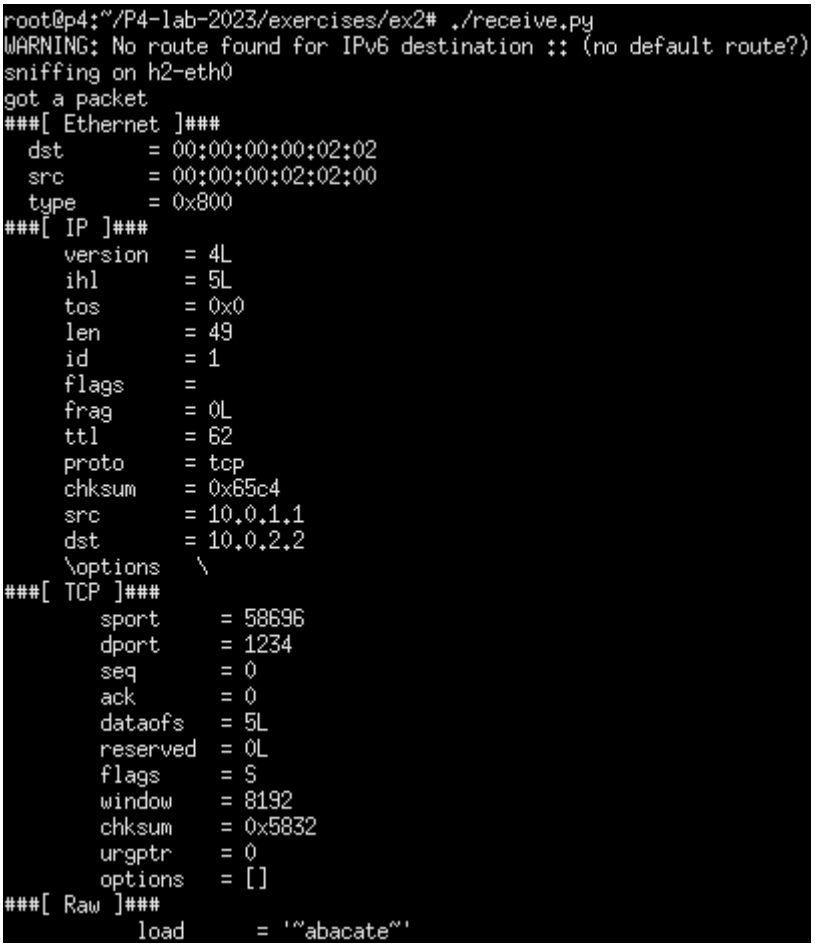


*Figura 7: Resposta do terminal de h1 para o comando receive.py, após correções*

No contexto apresentado, conforme se percebe acima, o valor *MAC* de destino dos pacotes está sendo corretamente alterado, uma vez que, na fonte esse valor é de *broadcast* e, no destino é o *MAC* de *h2*. No entanto, os valores de MAC de *source* ainda não estão sendo corretamente alterados, pois o *MAC* apresentado no destino como sendo da fonte é o *MAC* da interface de entrada em *s2*, não da de saída. Isso se explica pela forma como se altera o valor de *MAC* de fonte nos *switches* (definida no *script P4*), que atribui a esse campo o valor que era do campo de *MAC* de destino anteriormente, sendo, por consequência, o da interface de entrada. Ainda assim, quanto ao *TTL*, observa-se que a correção do código *P4* funcionou, porque o valor no emissor é de 64, enquanto que no receptor é de 62. Tal resultado é coerente com o esperado, já que o pacote enviado de *h1* para *h2* passa por *s1* e *s2* e, por consequência, tem seu *TTL* decrescido em 1 duas vezes (64 - 1 - 1 = 62).

**Exercício 3)**

Neste exercício, foi adicionado o tratamento de cabeçalhos *UDP* e *TCP* ao código base *P4*. Para tanto, foram adicionados os *headers TCP* e *UDP* ao código, conforme abaixo, sendo o segundo criado a partir da referência (https://en.wikipedia.org/wiki/User_Datagram_Protocol) e contendo os campos de *length*, *source* e *destiny port* e, *checksum*. Além disso, foi atualizado o parser no código para tratar os novos *headers* (*Figura 10*), de forma que o protocolo para *UDP* foi encontrado no site *CCNA Network* e assume o valor 17:
https://ccna.network/pacote-ipv4/#:~:text=O%20cabe%C3%A7alho%20do%20pacote%20IPv4,informa%C3%A7%C3%B5es%20importantes%20sobre%20o%20pacote

```
header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<3>  res;
    bit<3>  ecn;
    bit<6>  ctrl;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}
```

*Figura 8: Header TCP no código P4*

```
header udp_t{
    bit<16> srcPort;
    bit<16> dstPort;
    bit<16> checksum;
    bit<16> length;
}
```

*Figura 9: Header UDP no código P4*

```
state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition select(hdr.ipv4.protocol) {
            6: parse_tcp;
            17: parse_udp;
        default: accept;
    }
}

state parse_tcp {
    packet.extract(hdr.tcp);
    meta.tcpLen = hdr.ipv4.totalLen - 20;
    transition accept;
}

state parse_udp {
    packet.extract(hdr.udp);
    transition accept;
}
```

*Figura 10: Tratamento dos headers adicionados no processo de parsing*

Feito isso, foi repetido o processo do exercício anterior, com o uso dos comandos de *send* e *receive* apresentados nele. No entanto, notou-se que o pacote, embora enviado, foi recebido por *h2*, mas não corretamente. Isso porque, sob análise razoavelmente superficial, pode-se perceber que a causa de tal problema é a falta de processo de *deparsing* para os novos *headers*, que ocasiona sua ausência nos pacotes e, consequentemente, a impossibilidade de desencapsulamento do pacote no hospedeiro receptor. Isso é verificado pelas figuras a seguir, em que se tem a captura do pacote em *s1* e em *s2*, respectivamente.

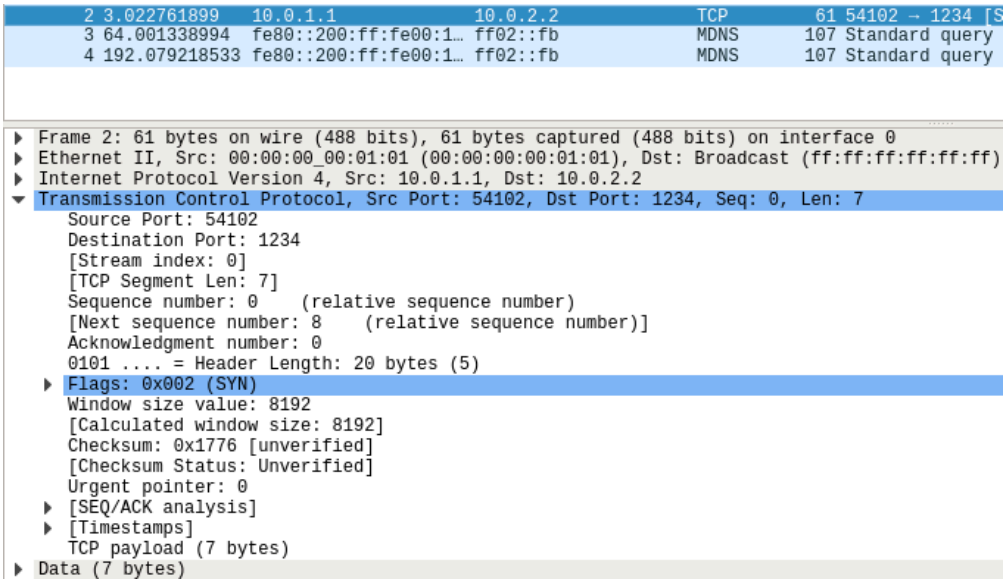


*Figura 11: Captura do pacote TCP enviado de h1 para h2, em s1*

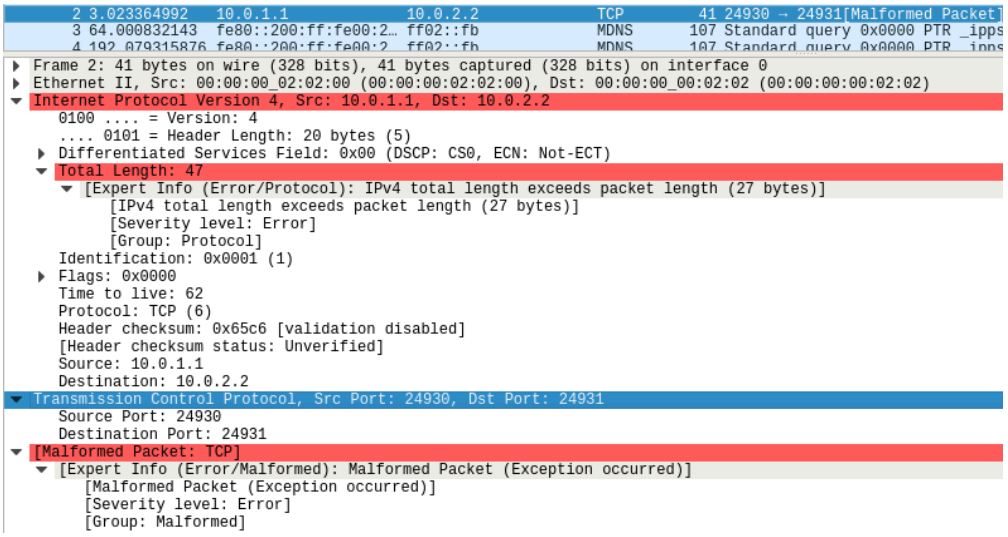


*Figura 12: Captura do pacote TCP enviado de h1 para h2, em s2*

Nas figuras apresentadas acima, pode-se observar que há um problema de má formação do pacote que chega a *s2*, o que gera o drop desse pacote por tal *switch*. Isso se dá, por causa da ausência de *deparsing* do cabeçalho *TCP* nos *switches*, que faz com que o pacote que sai de *s1* e *s2* tenha o cabeçalho *TCP* bastante corrompido, conforme as figuras acima. Assim, esse pacote, embora seja sim recebido por *h2* (*Figura 13*), não é passado para as camadas de cima em tal dispositivo, o que explica a falta de resposta no *prompt* em que se executa "./receive.py".


*Figura 13: Captura do pacote TCP enviado de h1 para h2, em h2*

Seguindo com os procedimentos propostos, foi adicionado o *deparsing* dos cabeçalhos criados, os cálculos de *checksum* para o *TCP* e para o *UDP* e, foi adicionada a ação de alteração da porta de destino para casos em que o pacote é *TCP*, como forma de se verificar o funcionamento das alterações feitas. Vale ressaltar que o cálculo de *checksum* para o *UDP* foi baseado no documento *pdf*

encontrado na internet, https://jvasconcellos.com.br/wp-content/uploads/2013/10/checksum-udp.pdf. Por fim, as alterações citadas são apresentadas a seguir:

```
update_checksum(
    hdr.tcp.isValid(),
        {  hdr.ipv4.srcAddr,
           hdr.ipv4.dstAddr,
           8w0,
           hdr.ipv4.protocol,
           meta.tcpLen,
           hdr.tcp. srcPort,
           hdr.tcp.dstPort,
           hdr.tcp.seqNo,
           hdr.tcp.ackNo,
           hdr.tcp.dataOffset,
           hdr.tcp.res,
           hdr.tcp.ecn,
           hdr.tcp.ctrl,
           hdr.tcp.window,
           hdr.tcp.urgentPtr},
        hdr.tcp.checksum,
        HashAlgorithm.csum16);
```

```
update_checksum(
    hdr.tcp.isValid(),
        {  hdr.ipv4.srcAddr,
           hdr.ipv4.dstAddr,
           8w0,
           hdr.ipv4.protocol,
           hdr.udp.length},
        hdr.udp.checksum,
        HashAlgorithm.csum16);
}
```

*Figura 13 e 14: Adição de cálculo de checksum para TCP e UDP ao código, respectivamente*

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
        packet.emit(hdr.udp);
    }
}
```

*Figura 15: Adição de deparsing para UDP e TCP no código*

```
if (hdr.tcp.isValid()){
    hdr.tcp.dstPort = 4321;
}
```

*Figura 16: Adição de alteração da porta de destino para pacotes TCP*

Feitas tais alterações citadas, obteve-se o que se apresenta nas *Figuras 17* e *18*, nas quais se nota que os *switches* estão, agora, funcionando corretamente. Nesse sentido, nota-se ainda que os valores da porta de destino no emissor e no receptor são diferentes, conforme se esperava, indicando o funcionamento dos roteadores e, consequentemente, do código, para o repasse de pacotes *TCP*.

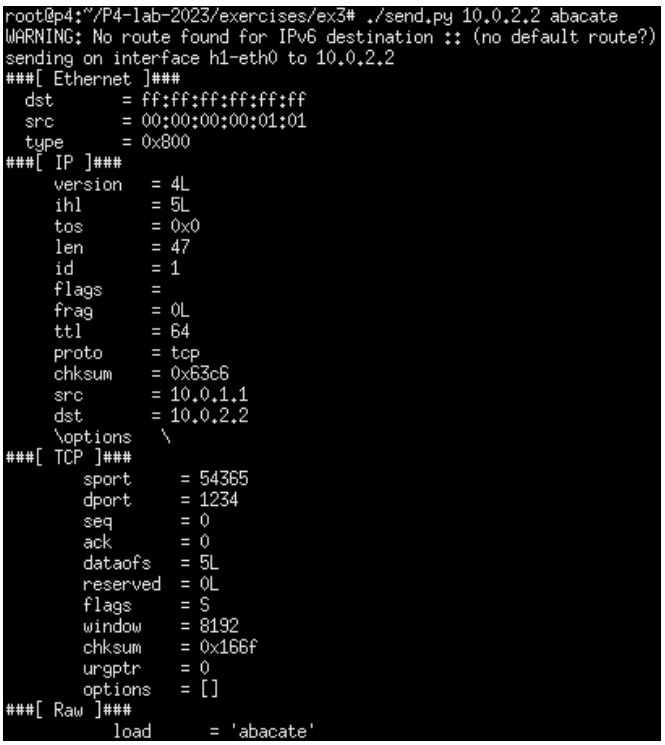


*Figura 17: Resposta do terminal de h1 para o comando send.py, após alterações*

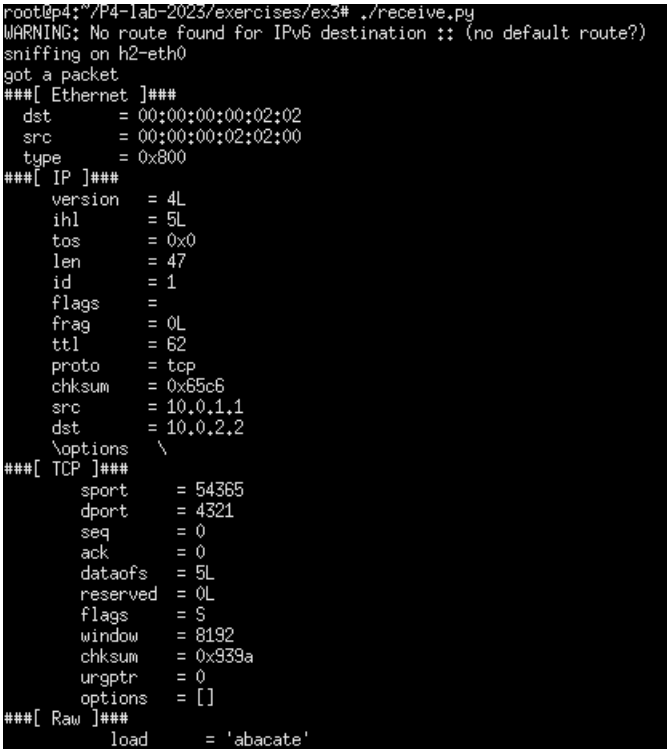


*Figura 18: Resposta do terminal de h2 para o comando receive.py, após alterações*

## Conclusão)

Em suma, pode-se afirmar que, com todo o experimento feito, aprendeu-se a respeito do *P4* mesmo que de maneira limitada a um único relatório. Desse modo, mais um conceito foi devidamente estudado e mais objetivos foram cumpridos..