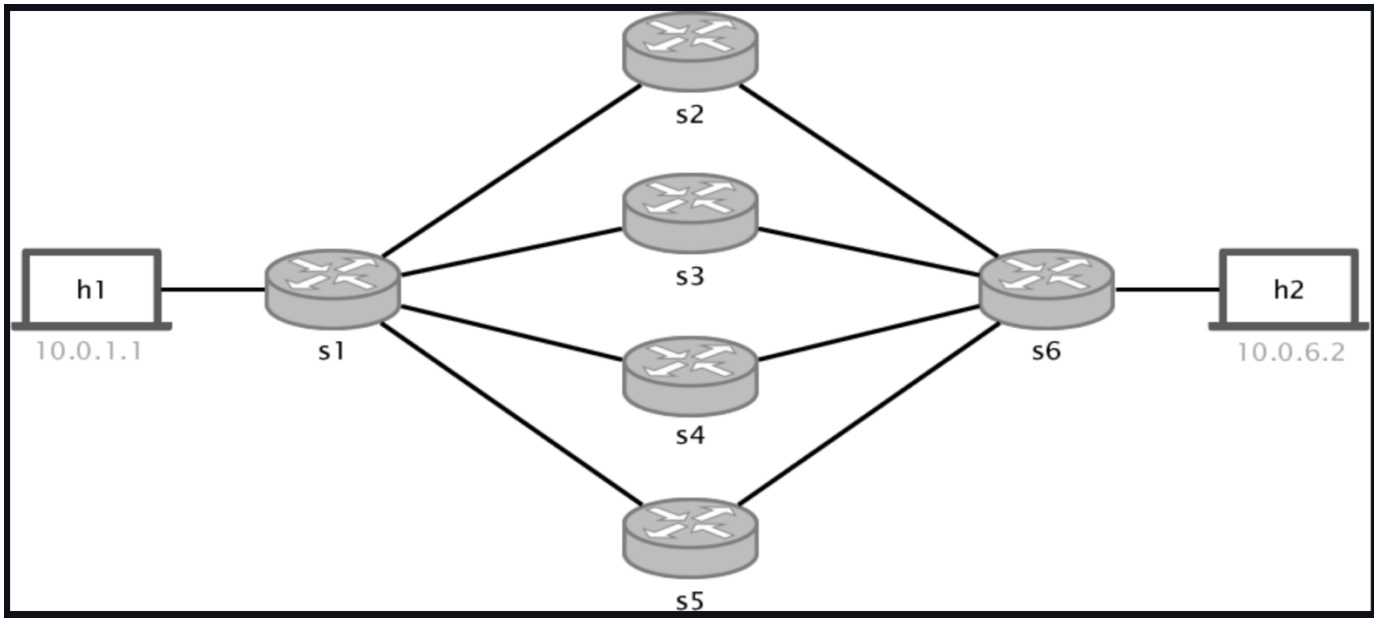
گزارش کار پروژه P4

محمدحسین عقیلی 810199576 سید علی امام زاده 810199377

بخش ۱: ECMP

توپولوژی:



در این تمرین مسیریابی ECMP را برای بارگیری ترافیک در چندین پورت پیاده سازی شده. هنگامی که بسته ای با چندین مسیر نامزد وارد می شود، سوئیچ ما باید با هش کردن برخی از فیلدها از هدر و محاسبه این مقدار هش، تعداد مسیرهای مساوی ممکن را به پرش بعدی اختصاص دهد. به عنوان مثال، در توپولوژی بالا، زمانی که s1 نیاز به ارسال بسته به h2 دارد، سوئیچ باید پورت خروجی را با محاسبه تعیین کند. همه بسته‌ها با آدرس‌های IP مبدا و مقصد یکسان و پورت‌های مبدأ و مقصد یکسان همیشه به همان هاپ بعدی هش می‌شوند.

توضیح کد:

در این بخش ابتدا لایبری های p4 اینکلود میشوند .

سپس یک placeholder برای چک سام تعریف میشود.

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

//My includes
#include "include/headers.p4"
#include "include/parsers.p4"

/*************************************************************************
************   C H E C K S U M    V E R I F I C A T I O N   *************
*************************************************************************/

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}
$ns color 2 Red
```

بلوک کنترلی به نام MyIngress را برای مدیریت بسته های ورودی تعریف می کند.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
```

سپس اکشن ها را معرفی میکند.

اکشنی به نام drop را تعریف می کند که بسته ای را که باید حذف شود را مشخص می کند.

```
  action drop() {
        mark_to_drop(standard_metadata);
    }
```

عملی به نام ecmp_group را برای محاسبه هش گروه ECMP بر اساس فیلدهای خاص در بسته تعریف می کند.

```
action ecmp_group(bit<14> ecmp_group_id, bit<16> num_nhops){
        hash(meta.ecmp_hash,
        HashAlgorithm.crc16,
        (bit<1>)0,
        { hdr.ipv4.srcAddr,
          hdr.ipv4.dstAddr,
          hdr.tcp.srcPort,
          hdr.tcp.dstPort,
          hdr.ipv4.protocol},
        num_nhops);

        meta.ecmp_group_id = ecmp_group_id;
    }
```

اکشنی به نام set_nhop را برای تنظیم اطلاعات hop بعدی، از جمله به‌روزرسانی آدرس‌هایMAC ، پورت خروجی و کاهش TTL تعریف می‌کند.

```
action set_nhop(macAddr_t dstAddr, egressSpec_t port) {

        //set the src mac address as the previous dst, this is not correct right?
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

       //set the destination mac address that we got from the match in the table
        hdr.ethernet.dstAddr = dstAddr;

        //set the output port that we also get from the table
        standard_metadata.egress_spec = port;

        //decrease ttl by 1
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }
```

سپس دو تیبل تعریف میکند.

جدولی به نام ecmp_group_to_nhop را برای نگاشت شناسه های گروه ECMP به اقدامات hop بعدی تعریف می کند.

جدولی به نام ipv4_lpm برای مسیریابی طولانی‌ترین پیشوند مطابقت(lpm) IPv4 تعریف می‌کند. این اقدامات برای set_nhop، ecmp_group و drop را مشخص می کند

```
able ecmp_group_to_nhop {
        key = {
            meta.ecmp_group_id:    exact;
            meta.ecmp_hash: exact;
        }
        actions = {
            drop;
            set_nhop;
        }
        size = 1024;
    }

    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            set_nhop;
            ecmp_group;
            drop;
        }
        size = 1024;
        default_action = drop;
    }
```

بلوک اعمال را برای کنترل MyIngress آغاز می کند. بررسی می کند که آیا هدر IPv4 معتبر است یا خیر و اقداماتی را بر اساس نتیجه جدول ipv4_lpm اعمال می کند.

```
apply {
        if (hdr.ipv4.isValid()){
            switch (ipv4_lpm.apply().action_run){
```

```
                ecmp_group: {
                    ecmp_group_to_nhop.apply();
                }
            }
        }
    }
```

```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {

    }
}
```

محاسبه چک سام:

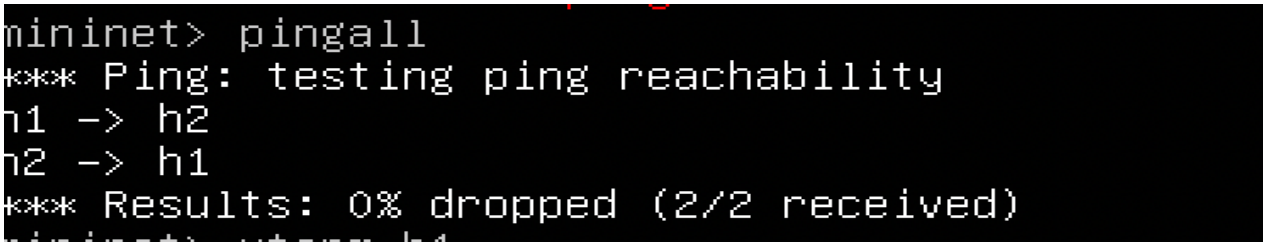
```
control MyComputeChecksum(inout headers hdr, inout metadata meta) {
     apply {
    update_checksum(
        hdr.ipv4.isValid(),
            { hdr.ipv4.version,
              hdr.ipv4.ihl,
              hdr.ipv4.dscp,
              hdr.ipv4.ecn,
              hdr.ipv4.totalLen,
              hdr.ipv4.identification,
              hdr.ipv4.flags,
              hdr.ipv4.fragOffset,
              hdr.ipv4.ttl,
              hdr.ipv4.protocol,
              hdr.ipv4.srcAddr,
              hdr.ipv4.dstAddr },
              hdr.ipv4.hdrChecksum,
              HashAlgorithm.csum16);
    }
}
```

تعریف معماری سوئیچ P4 با استفاده از قالب V1Switch

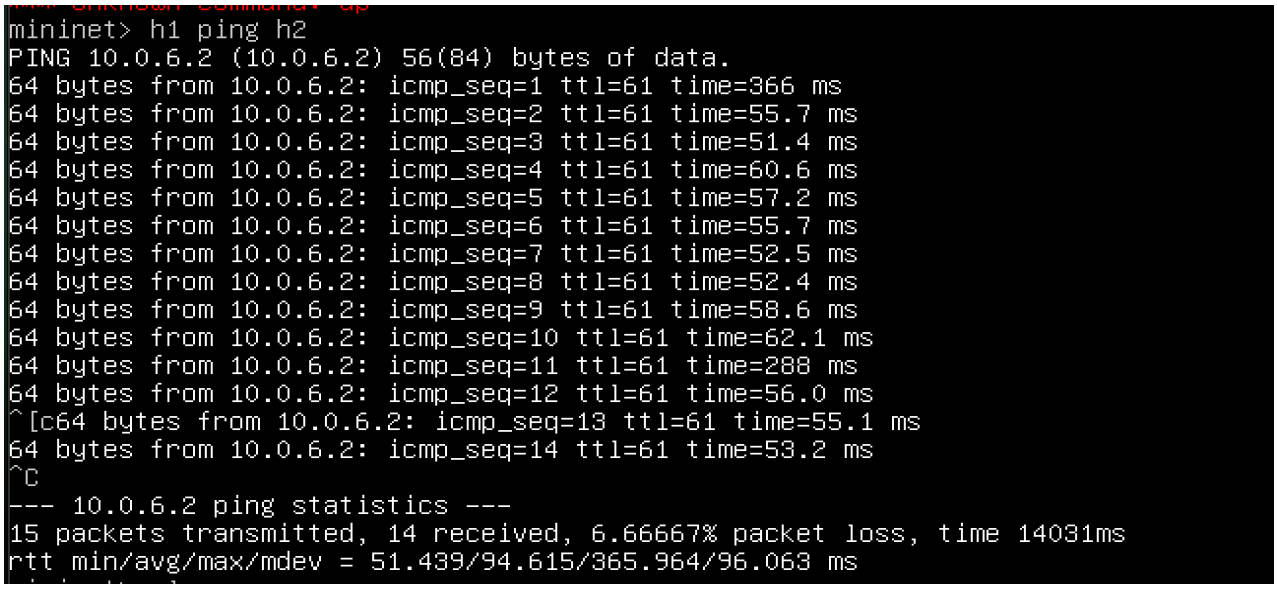
```
//switch architecture
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

نتایج اجرا:

Ping All:



h1 ping h2:



بخش ۲: Heavy Heater

Heavy heater در واقع باعث میشود که اجازه ارسال بیش از مقدار مشخصی از بسته ها را ندهیم و بسته های اضافه را دراپ کنیم.

توضیح کد:

در این بخش کد ثابت ها و هدر، از جمله هدر های اترنت، IPv4، و TCP، و برخی از انواع types و ساختارهای metadata آورده شده است:

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

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;
}

header tcp_t{
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4>  dataOffset;
    bit<4>  res;
    bit<1>  cwr;
    bit<1>  ece;
    bit<1>  urg;
    bit<1>  ack;
    bit<1>  psh;
    bit<1>  rst;
    bit<1>  syn;
    bit<1>  fin;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}

struct metadata {
    bit<32> output_hash_one;
    bit<32> output_hash_two;
    bit<32> counter_one;
    bit<32> counter_two;
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_t        tcp;
}
```

در این بخش MyParser تعریف شده است، که مشخص می کند بسته های ورودی چگونه باید تجزیه شوند.

پارسر استیت های (شروع، ipv4، tcp) و انتقال ها را بر اساس نوع اترنت و پروتکل IPv4 تعریف می کند.

هنگام انتقال بین استیت ها، هدرهای مربوطه را استخراج می کند:

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

    state start {

        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){

            TYPE_IPV4: ipv4;
            default: accept;
        }
    }

    state ipv4 {
        packet.extract(hdr.ipv4);

        transition select(hdr.ipv4.protocol){
            TYPE_TCP: tcp;
            default: accept;
        }
    }

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

}

/*************************************************************************
************   C H E C K S U M    V E R I F I C A T I O N   ************
*************************************************************************/

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

```
}

/*************************************************************************
*************  I N G R E S S   P R O C E S S I N G   *******************
*************************************************************************/

control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {


    register<bit<BLOOM_FILTER_BIT_WIDTH>>(BLOOM_FILTER_ENTRIES) bloom_filter;

    action drop() {
        mark_to_drop(standard_metadata);
    }

    action update_bloom_filter(){
       //Get register position
       hash(meta.output_hash_one, HashAlgorithm.crc16, (bit<16>)0, {hdr.ipv4.srcAddr,
                                                            hdr.ipv4.dstAddr,
                                                            hdr.tcp.srcPort,
                                                            hdr.tcp.dstPort,
                                                            hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);

       hash(meta.output_hash_two, HashAlgorithm.crc32, (bit<16>)0, {hdr.ipv4.srcAddr,
                                                            hdr.ipv4.dstAddr,
                                                            hdr.tcp.srcPort,
                                                            hdr.tcp.dstPort,
                                                            hdr.ipv4.protocol},

(bit<32>)BLOOM_FILTER_ENTRIES);

        //Read counters
        bloom_filter.read(meta.counter_one, meta.output_hash_one);
        bloom_filter.read(meta.counter_two, meta.output_hash_two);

        meta.counter_one = meta.counter_one + 1;
        meta.counter_two = meta.counter_two + 1;

        //write counters

        bloom_filter.write(meta.output_hash_one, meta.counter_one);
        bloom_filter.write(meta.output_hash_two, meta.counter_two);
    }
```

```
    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {

        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

       //set the destination mac address that we got from the match in the table
        hdr.ethernet.dstAddr = dstAddr;

        //set the output port that we also get from the table
        standard_metadata.egress_spec = port;

        //decrease ttl by 1
        hdr.ipv4.ttl = hdr.ipv4.ttl -1;

    }

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

    apply {
        if (hdr.ipv4.isValid()){
            if (hdr.tcp.isValid()){
                update_bloom_filter();
                //only if IPV4 the rule is applied. Therefore other packets will not
be forwarded.
                if ( (meta.counter_one > PACKET_THRESHOLD && meta.counter_two >
PACKET_THRESHOLD) ){
                    drop();
                    return;
                }
            }
            ipv4_lpm.apply();
        }
    }
}
```

محاسبه چک سام:

```
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);
    }
}
```

بلوک کنترل MyDeparser نحوه ساخت هدر بسته ها را قبل از انتقال مشخص می کند:

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

بلوک V1Switch معماری کلی سوئیچ را مشخص می کند، از جمله مؤلفه هایی مانند parser، checksum verification، ingress processing، egress processing، checksum computation، and deparser.

```
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
```

```
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

نتایج:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

```
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=62 time=255 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=62 time=26.8 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=62 time=27.8 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=62 time=26.3 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=62 time=28.6 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=62 time=28.7 ms
64 bytes from 10.0.2.2: icmp_seq=7 ttl=62 time=25.5 ms
64 bytes from 10.0.2.2: icmp_seq=8 ttl=62 time=25.1 ms
64 bytes from 10.0.2.2: icmp_seq=9 ttl=62 time=28.1 ms
64 bytes from 10.0.2.2: icmp_seq=10 ttl=62 time=27.2 ms
64 bytes from 10.0.2.2: icmp_seq=11 ttl=62 time=28.8 ms
64 bytes from 10.0.2.2: icmp_seq=12 ttl=62 time=25.7 ms
64 bytes from 10.0.2.2: icmp_seq=13 ttl=62 time=26.4 ms
^C
--- 10.0.2.2 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12029ms
rtt min/avg/max/mdev = 25.113/44.643/255.253/60.809 ms
```