به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

پروژه P4

درس شبکه های کامپیوتری

دکتر خونساری

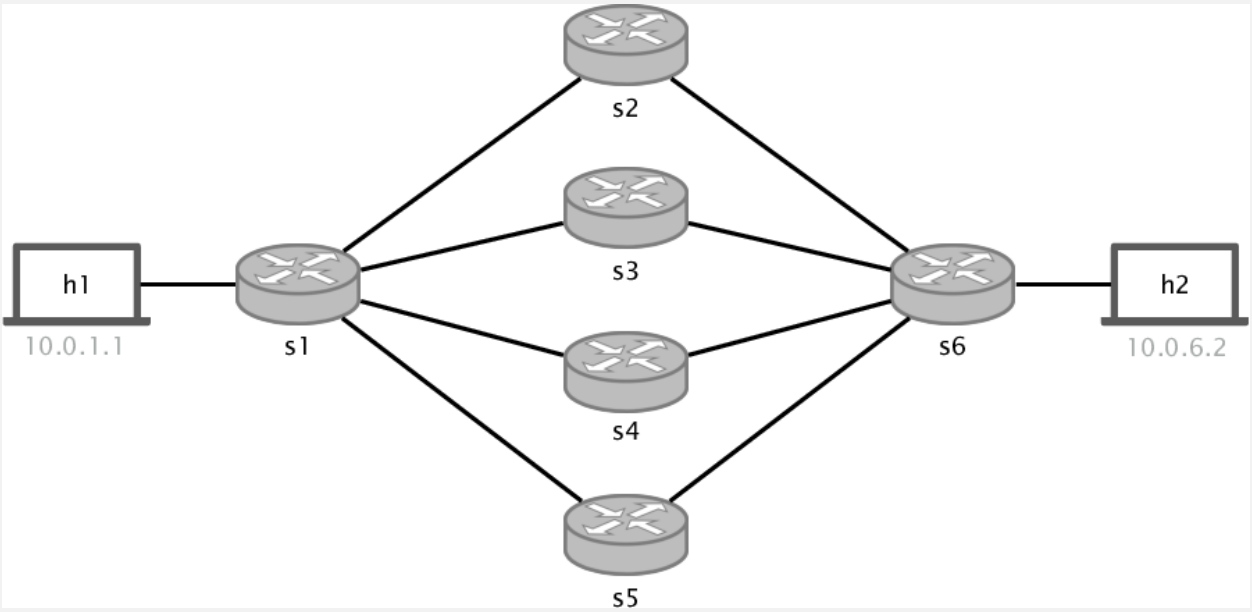
اعضای گروه:

- **علی قنبری** **۸۱۰۱۹۹۴۷۳**
- **اهورا شیری** **۸۱۰۱۹۸۵۵۵**

پاییز ۰۳-۱۴۰۲

ECMP :

مسیریابی چند مسیری با هزینه برابر (ECMP) تکنیکی است که در شبکه های کامپیوتری برای توزیع ترافیک در چند مسیر که هزینه یکسانی دارند، استفاده می شود. اساساً در سناریوهایی استفاده می‌شود که در آن چندین مسیر بعدی برای رسیدن به یک شبکه مقصد وجود دارد و این مسیرها از نظر معیارهایی مانند تعداد پرش (hop) ، پهنای باند معادل اند.



به طور مثال ۴ مسیر بالا به دلیل تقارن ، هزینه های یکسانی دارند (بافرض یکسان بودن سوییچ ها)

درواقع به طور مثال سوییچ های s1 و s6 بر اساس هدر های بسته ها ، یک تابع هش روی بخشی از هدر ها اجرا و تعین میکنند بسته از چه مسیری برود . این کار باعث بهبود بهره وری و کارایی و توزیع بار بین لینک ها خواهد شد.(بسته ها را از پورت های مختلفی میفرستیم) . این کار باعث جلوگیری از ایجاد سربار مضاعف روی یک خط شبکه خواهد شد.

فایل های مورد استفاده و API ها :

p4app.json

این فایل ساختار شبکه را تعین میکند (توپولوژی) و همچنین دستورات CLI در سوییچ ها

network.py

توپولوژی را با استفاده از API هایی که به پایتون هست تعریف میکند . برای راه اندازی شبکه می توان از network.py یا p4app.json استفاده کرد.(فرقی ندارد.)

send.py

این اسکریپت تعداد مشخصی از بسته های تصادفی TCP را تولید می کند و آنها را با استفاده از کتابخانه scapy به یک مقصد مشخص می فرستد. معمولاً برای آزمایش رفتار شبکه یا سیستم‌های تشخیص نفوذ شبکه استفاده می‌شود.

به طور مثال اگر در CLI هاست h1 باشیم ، از یکی از پورت های h1 با دستور زیر ۱۰ بسته به آیپی زیر میفرستد :

python send.py 1.1.1.1 10

ecmp.p4

ساختار و معماری کلی سوییچ ها را تعین میکند . درواقع اسکلت برنامه p4 برای استفاده به عنوان نقطه شروع می باشد.

CHECKSUM:

بلوک تایید checksum برای تایید checksum پکت های دریافتی بکار می رود که در اینجا خالی است و کاری انجام نمی دهد.

```
/*************************************************************************
************   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 {  }
}
```

INGRESS PROCESSING :

بلوک کنترلی MyIngress ، مسئول پردازش بسته های رسیده شده و دریافتی و تصمیم گیری برای ارسال آن ها بر اساس هدر آنها است.

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

داده های metadata و standard_metadata :

هر دو برای حمل اطلاعات اضافی در مورد بسته ها استفاده می شوند، اما اهداف متفاوتی دارند:

metadata :

این **metadata** تعریف شده توسط کاربر است .شما می توانید فیلدهای خود را در ساختار **metadata** تعریف کنید تا هر گونه اطلاعات اضافی را که برای مورد استفاده خاص خود نیاز دارید، حمل کنید .به عنوان مثال، ممکن است یک فیلد برای حمل نتیجه یک محاسبات پیچیده تعریف کنید که باید در چندین مکان استفاده شود.

standard_metadata :

این یک ساختار از پیش تعریف شده در P4 است که دارای فیلدهای متادیتا استاندارد است که در بسیاری از موارد استفاده مشترک است .این فیلدها شامل مواردی مانند پورت های ورودی و خروجی، نوع نمونه (عادی، شبیه سازی شده، و غیره) و موارد دیگر است .ساختار standard_metadata توسط معماری هدف (مانند v1model یا PSA) تعریف می‌شود.

mark_to_drop () :

تابع mark_to_drop با standard_metadata به عنوان آرگومان فراخوانی می شود. این تابع یک تابع داخلی در P4 است که یک بسته را با تنظیم یک فیلد خاص در standard_metadata علامت گذاری می کند که باید حذف شود.

ecmp_group () :

این action، عمل گروهی ECMP (هزینه برابر چند مسیر) را تعریف می کند. یک ۵ تایی (IP مبدا، IP مقصد، پورت مبدأ، پورت مقصد، پروتکل) را به تعداد hop های خروجی (پورت‌های خروجی) پیمانه میزدند و آن را هش می‌کند و آن را در متادیتا ذخیره می‌کند. شناسه گروه ECMP نیز در فراداده ذخیره می شود.

```
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;
}
```

set_nhop() :

```
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;
}
```

خط اول آدرس MAC مبدا بسته را با آدرس MAC مقصد قبلی تنظیم می کند .این معمولاً به این دلیل انجام می شود که بسته در حال ارسال است، بنابراین سوئیچ/روتر فعلی به منبع جدید تبدیل می شود. در واقع بسته جدیدی که به یک سوییچ رسیده چون قراره از اون سوییچ ارسال بشه ، hop بعدی میشه مقصد و این سوییچ میشه مبدا (از لحاظ آدرس های MAC) .

خط بعدی آدرس MAC مقصد بسته را dstAddr تنظیم می کند که به عنوان آرگومان برای عمل ارائه می شود .این معمولاً آدرس MAC هاپ بعدی است. (که از جدول قرض میگیرد.)

خط سوم پورت خروجی بسته را به پورت تنظیم می کند که به عنوان آرگومان عمل نیز ارائه می شود .این پورت فیزیکی روی سوئیچ/روتر را تعیین می کند که بسته از آن به خارج ارسال می شود.

و خط آخر فیلد Time To Live (TTL) بسته را ۱ کاهش می دهد .TTL فیلدی در هدر IP است که از گردش بسته ها در شبکه برای همیشه جلوگیری می کند . .هر بار که یک بسته توسط یک روتر ارسال می شود، 1 TTLکاهش می یابد. اگر TTL به ۰ برسد، بسته drop می شود.

جدول ecmp_group_to_nhop :

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

این جدول با استفاده از شناسه گروه ECMP و هش به عنوان کلید برای جدول (شاخص)، گروه های ECMP را به hop های بعدی نگاشت می کند. می تواند اقدامات drop و set_nhop را انجام دهد. پارامتر size حداکثر تعداد ورودی هایی را که جدول می تواند نگه دارد را مشخص می کند.درواقع یک گروه ECMP و یک مقدار هش را به یک اکشن یا " drop"یا" nhop_set"تطبیق می دهد.همچنین matching از نوع exact هست یعنی دقیقا باید با مدخل ها مطابقت داشته باشد.

جدول ipv4_lpm :

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

این جدول طولانی‌ترین تطبیق پیشوند (LPM) را برای آدرس‌های IPv4 انجام می‌دهد که یک عملیات رایج در مسیریابی IP است. از آدرس IP مقصد به عنوان کلید استفاده می کند و می تواند اقدامات set_nhop، ecmp_group و drop را انجام دهد.

بلاک apply :

```
apply {
    if (hdr.ipv4.isValid()){
        switch (ipv4_lpm.apply().action_run){
            ecmp_group: {
                ecmp_group_to_nhop.apply();
            }
        }
    }
}
```

این بلوک apply تابع کنترل P4 است. عملیاتی را که باید روی هر بسته انجام شود را مشخص می کند. این چیزی است که انجام می دهد:

if (hdr.ipv4.isValid()): این کار بررسی می کند که آیا هدر IPv4 بسته معتبر است یا خیر. عملیات داخل این بلوک «if» تنها در صورتی انجام می شود که هدر IPv4 معتبر باشد.

switch (ipv4_lpm.apply().action_run): جدول ipv4_lpm را به بسته اعمال می کند و یک سوئیچ را روی عملکردی که در نتیجه برنامه جدول اجرا شده است، انجام می دهد.

ecmp_group: { ecmp_group_to_nhop.apply(): اگر اکشنی که توسط جدول «ipv4_lpm» اجرا می‌شود، «ecmp_group» بود، جدول «ecmp_group_to_nhop» روی بسته اعمال می‌شود.تا hop بعدی مشخص کند.

ENGRESS PROCESSING :

```p4
/*************************************************************************
****************  E G R E S S   P R O C E S S I N G  ******************
*************************************************************************/

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

    }
}
```

بلوک کنترل «MyEgress» برای رسیدگی به پردازش بسته ها هنگام خروج از سوییچ (پردازش خروج) یا پورت ها در نظر گرفته شده است .با این حال، بلوک «apply» در «MyEgress» در حال حاضر خالی است، به این معنی که هنوز پردازش خاصی تعریف نشده است.

: CHECKSUM COMPUTATION

```p4
/*************************************************************************
*************   C H E C K S U M    C O M P U T A T I O N   **************
*************************************************************************/

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);
    }
}
```

بلوک کنترل MyComputeChecksum با استفاده از الگوریتم Checksum مکمل ۱۶ بیتی، جمع کنترلی هدر IPv4 بسته های ورودی را محاسبه می کند. این بلوک کد بررسی می‌کند که آیا بسته دارای سرآیند IPv4 معتبر است یا خیر، و اگر دارای سرآیند IPv4 باشد، Checksum هدر IPv4 را محاسبه می‌کند و فیلد Checksum را در هدر به‌روزرسانی می‌کند .این یک عملیات در سوئیچ ها و روترهای شبکه برای اطمینان از یکپارچگی هدرهای بسته است.

SWITCH :

بلوک سوییچ مراحل پارسر، کنترل چک سام، پردازش ورودی و خروجی، محاسبه چکسام و دیپارسر را کنترل و می کند.

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

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

این نقطه ورودی اصلی P4 است که سوئیچ شبکه را تعریف می کند.

V1Switch یک معماری از پیش تعریف شده در P4 است. این نشان دهنده یک مدل خاص از یک سوئیچ شبکه با مجموعه خاصی از قابلیت ها و یک روش خاص برای پردازش بسته ها است.

main در پایان نشان می دهد که این بلوک اصلی (یا Top level) برنامه P4 است .هر یک از این مؤلفهMyParser و MyVerifyChecksum، MyIngress، MyEgress،

MyComputeChecksum، MyDeparser در بالا تعریف شدند .آنها به طور جمعی نحوه پردازش بسته ها توسط سوئیچ را تعریف می کنند.

headers.p4

این برنامه P4 ساختارهای داده‌ای را تعریف می‌کند که نشان‌دهنده هدر بسته‌هایی است که توسط سوئیچ پردازش می‌شوند، و همچنین برخی ابرداده‌هایی که در پردازش استفاده می‌شوند .منطق پردازش خاص در جای دیگری در برنامه P4 تعریف می شود.

هدر های متادیتا شامل ecmgroup_id ها هستند که در فایل ecmp.p4 ازشون استفاده و برای تعین لینک برای انتقال بسته به کار می رود. همچنین هدر های لایه ۲ و ۳ و ۴ تعریف شدند (پروتکل های متناظر Ethernet ، Ipv4 و Tcp) که هرکدام دارای فیلد های خاص خود هستند.

در نهایت ساختار داده headers این هدر های لایه ها را تجمیع و در یک بسته قرار میدهد:

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

parsers.p4

PARSER :

```
/*************************************************************************
*********************** P A R S E R  ***********************************
*************************************************************************/

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;
        }
    }
```

پارسر با چندین state تعریف می شود که هر کدام یک مرحله از فرآیند تجزیه بسته را نشان می دهد. state اولیه start است.

MyParser وظیفه استخراج هدرهای اترنت، IPv4و TCP از بسته های ورودی را بر عهده دارد . بین state تجزیه بر اساس مقادیر فیلدهای هدر خاص جابه‌جا می‌شود و در نهایت هدر های لازم را برای پردازش بیشتر بسته استخراج می‌کند.

تابع extract همچین شکلی دارد و با جابه جا کردن cursor اطلاعات و هدر های هر لایه را استخراج میکند به طور مثال برای پکت های با طول ثابت :

```
void packet_in.extract<T>(out T headerLValue) {
   bitsToExtract = sizeofInBits(headerLValue);
   lastBitNeeded = this.nextBitIndex + bitsToExtract;
   ParserModel.verify(this.lengthInBits >= lastBitNeeded, error.PacketTooShort);
   headerLValue = this.data.extractBits(this.nextBitIndex, bitsToExtract);
   headerLValue.valid$ = true;
   if headerLValue.isNext$ {
     verify(headerLValue.nextIndex$ < headerLValue.size, error.StackOutOfBounds);
     headerLValue.nextIndex$ = headerLValue.nextIndex$ + 1;
   }
   this.nextBitIndex += bitsToExtract;
}
```

11.8.2 Variable-width extraction

در واقع با استخراج هدر ها و پاس دادن به متغیر hdr ، hdr در مراحل بعدی مورد استفاده قرار میگیرد.

هنگامی که دستور «packet.extract(hdr.ethernet);» دستور اجرا می شود، تجزیه کننده مجموعه بیت های بعدی را از داده های بسته خام می گیرد (اندازه مجموعه بیت ها با اندازه هدر اترنت همانطور که در ساختار "ethernet_t" تعریف شده است تعیین می شود) و آنها را به عنوان یک هدر اترنت سپس این بیت‌ها از جلوی داده‌های بسته خام حذف می‌شوند و فیلدهای ساختار «hdr.ethernet» با مقادیر مربوطه از سربرگ اترنت استخراج‌شده پر می‌شوند.

DEPARSER :

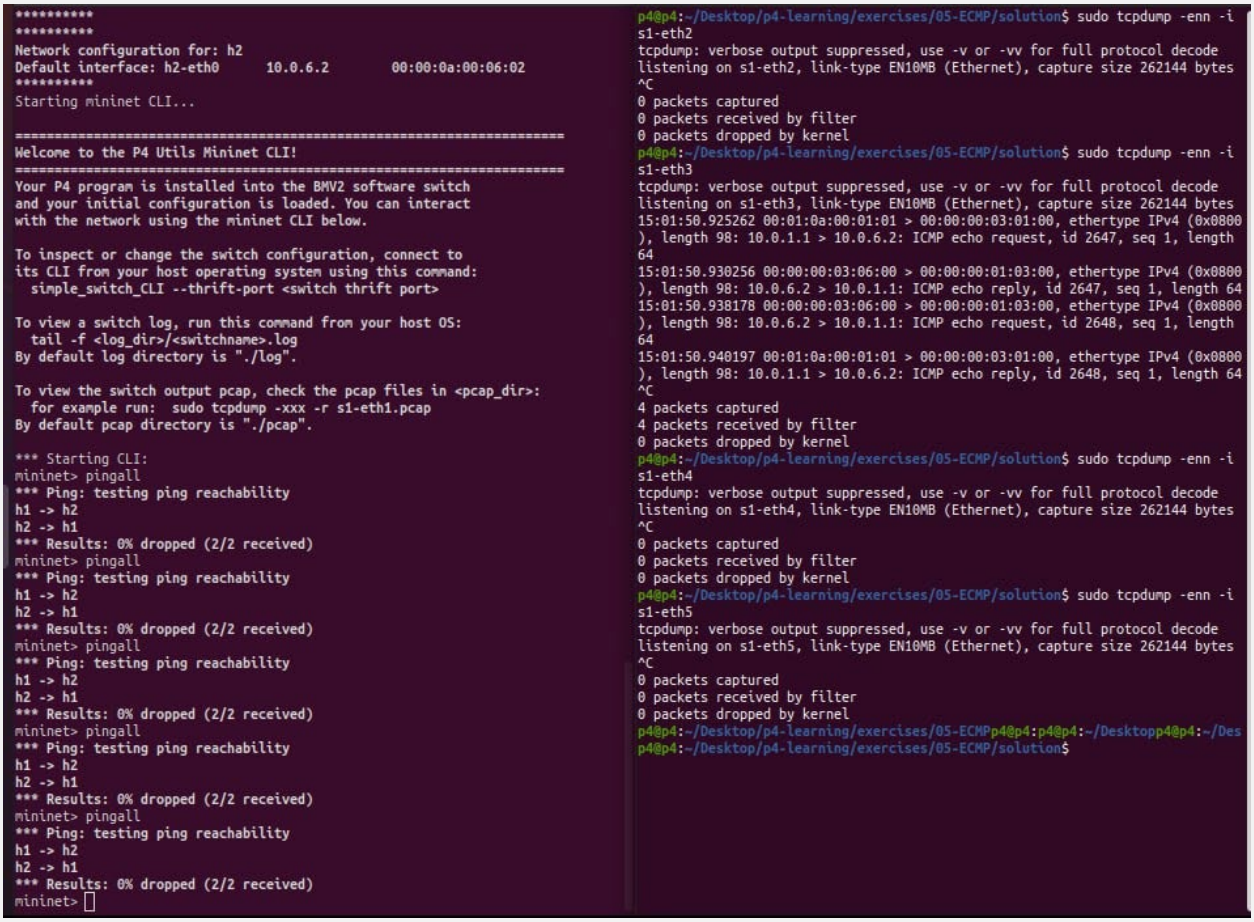
```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {

        //parsed headers have to be added again into the packet.
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);

        //Only emited if valid
        packet.emit(hdr.tcp);
    }
}
```

MyDeparser، مسئول بازسازی بسته از هدر های تجزیه‌شده قبل از ارسال به خارج از سوئیچ است .هدرهای اترنت و IPv4 را به بسته خروجی بازمی گرداند و در صورت معتبر بودن، هدر TCP را نیز اضافه می کند. ترتیب اضافه کردن هدرها مهم است زیرا ترتیب هدرها را در بسته نهایی تعیین

می کند .در این حالت، هدر اترنت، بیرونی ترین هدر، به دنبال آن هدر IPv4 و در نهایت هدر TCP (در صورت معتبر بودن) خواهد بود.

خروجی :

برای شروع با ران کردن برنامه تعدادی بسته ارسال میکنیم که همانطور که مشاهده می شود این بسته ها فقط از یک لینک رد می شوند. در واقع به ازای مانیتور هر لینک s1→s2 تا به s6 آن ها را مانتیور کرده و همزمان ping گرفتیم و مشاهده میکنیم فقط از لینک s1→s3 بسته ها عبور میکنند.



بین ۲ هاست پینگ گرفتن :

بین ۲ هاست پینگ گرفتیم و مشاهده میکنیم ارتباط بر قرار هست.

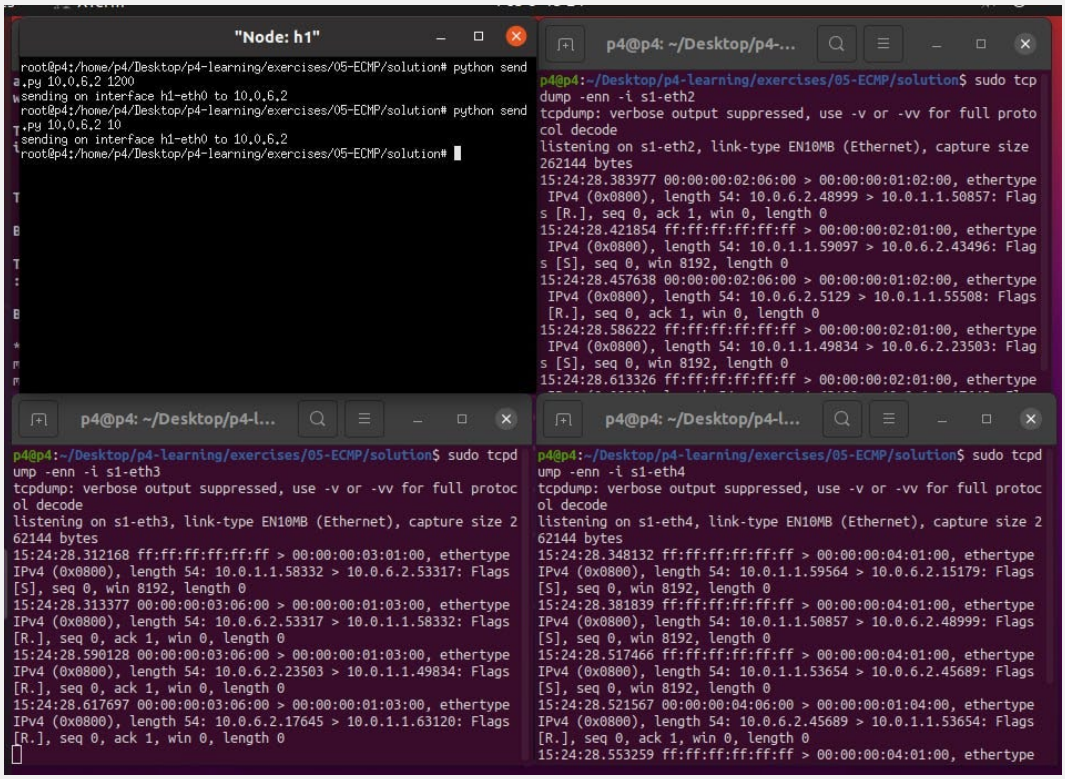
```
mininet> h1 ping h2
PING 10.0.6.2 (10.0.6.2) 56(84) bytes of data.
64 bytes from 10.0.6.2: icmp_seq=1 ttl=61 time=7.46 ms
64 bytes from 10.0.6.2: icmp_seq=2 ttl=61 time=7.60 ms
64 bytes from 10.0.6.2: icmp_seq=3 ttl=61 time=6.53 ms
^C
--- 10.0.6.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 6.529/7.196/7.597/0.474 ms
mininet> h2 ping h1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=61 time=3.65 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=61 time=7.39 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=61 time=7.89 ms
^C
--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 3.653/6.312/7.889/1.891 ms
mininet>
```

تست پهنای باند بین h1 و h2 :

با استفاده از این دستور پهنا باند بین ۲ نقطه مشخص میشود. اولی تست خط h1 به h2 و دومی برعکس می باشد.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['117.3 Mbits/sec', '127.7 Mbits/sec']
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['115.0 Mbits/sec', '121.5 Mbits/sec']
mininet>
```

ارسال ۱۰ بسته از هاست h1 به h2 :



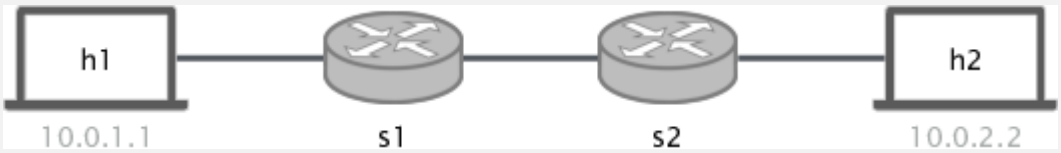
مشاهده میکنیم بسته ها از تمامی لینک ها می گذرند و این یعنی توزیع بار بین لینک ها و ECMP به درستی کار میکند درحالی که در حالت قبلی فقط از اینترفیس یا پورت eth3 میگذشت.

Heavy Hitter Detection:

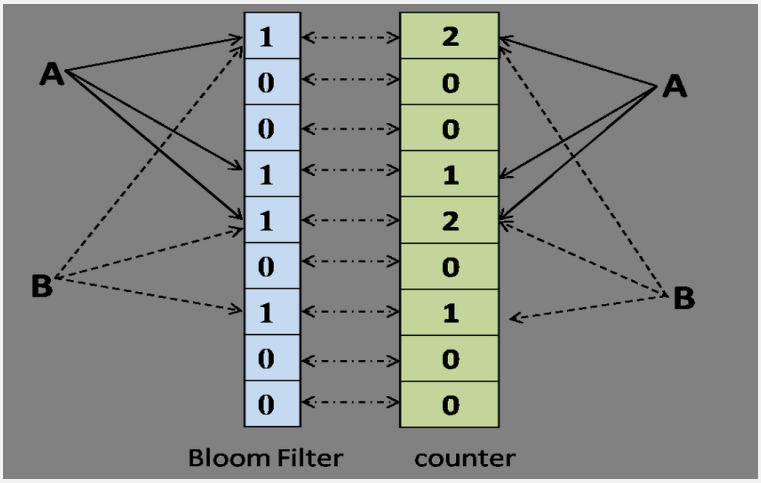
یک تکنیک است برای تشخیص جریان هایی که ترافیک سنگین روی شبکه ایجاد کردند و بسته های متعلق به این جریان ها (احتمالا) drop خواهند شد . (اگر از یه تعداد زیادی عبور کنند).

به همین دلیل از generalized bloom filter یا counting استفاده میکنیم که داخل آرایه ها عدد هستند . در واقع این اعداد اعداد نگاشت شده بسته ها هستند که با هر بسته جدید یکی زیاد میشوند. در صورتی که از یه مقدار آستانه بیشتر شود ، بسته ها drop میشوند.

توپولوژی ما :



نمونه ای از bloom filter تعمیم یافته (یا همان counting) در مقابل bloom filter ساده :



تنها تفاوت ها با بخش قبلی و تمرین قبلی گفته خواهد شد زیرا دارای اشتراکات زیادی هستند .

heavy_hitter.p4

```
/* CONSTANTS */
const bit<16> TYPE_IPV4 = 0x800;
const bit<8>  TYPE_TCP  = 6;

#define BLOOM_FILTER_ENTRIES 4096
#define BLOOM_FILTER_BIT_WIDTH 32
#define PACKET_THRESHOLD 1200
```

سایز هر عنصر فیلتر بلوم ، تعداد عناصر فیلتر و مقدار آستانه برای این فیلتر تعین شده است .

سایز استانه پکت ۱۲۰۰ تعین کردیم.

metadata :

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

در متا داده هر بسته ۲ تابع هش و ۲ خروجی شمارنده ای دارد. ۲ فیلد اول برای ذخیره خروجی توابع هش استفاده می شوند .در یک فیلتر Bloom شمارشی، معمولاً از چندین توابع هش برای تعیین شاخص های افزایش در فیلتر استفاده می شود .نتایج این توابع هش را می توان در این فیلدها ذخیره کرد. ۲ فیلد بعدی برای ذخیره مقادیر شمارش جاری خوانده شده از فیلتر Bloom در موقعیت هایی که با مقادیر هش نشان داده شده اند استفاده می شوند .در فیلتر Bloom شمارش، هر موقعیت در فیلتر شمارنده ای است که هر بار که عنصری به آن موقعیت هش می کند، افزایش می یابد .از این فیلدها می توان برای ذخیره مقادیر شمارش فعلی استفاده کرد تا بتوان آنها را افزایش داد و به فیلتر Bloom بازگرداند.

update_bloom_filter:

```
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);
    }
```

بلوک کنترل MyIngress در حال پیاده‌سازی یک فیلتر Bloom شمارشی برای تشخیص ضربه‌گیر سنگین در ترافیک شبکه است. در خط چهارم یک رجیستر به نام bloom_filter را با تعداد ورودی«BLOOM_FILTER_ENTRIES» ، هر کدام با عرض «BLOOM_WIDTH._BIT» اعلام می‌کند .این ثبات به عنوان فیلتر Bloom شمارش عمل می کند.

تابع update_bloom_filter فیلتر Bloom را با اطلاعات هدر های هر بسته جدید به روز می کند.

دو مقدار هش ۵ تایی را محاسبه می کند .هش اول از الگوریتم CRC16 و هش دوم از الگوریتم CRC32استفاده می کند:

```
//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);
```

مقادیر شمارش فعلی را از فیلتر Bloom در موقعیت هایی که با مقادیر هش نشان داده شده است می خواند:

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

مقادیر را یک واحد افزایش می دهد .(آن هایی که با هش map شدند (اندیس ها)):

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

در نهایت مقادیر به روز شده شمارش را در فیلتر Bloom در موقعیت هایی که با مقادیر هش نشان داده شده است، می نویسد:

```
//write counters

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

خروجی ها :

سایز آستانه بسته ۱۲۰۰ تعین کردیم و با ارسال ۱۵۰۰ بسته از h1 به h2 مشاهده میکنیم که ۱۲۰۰ تا h2 دریافت کرده و یعنی فیلتر به درستی عمل کرده است.

ارسال ۱۵۰۰ بسته به گره h2 با ایپی 10.0.2.2 :

```
python send.py 10.0.2.2 1500
```