Network Measurement with P4 and C on Netronome NFP

Xiaoban Wu, Yan Luo

Dept. of Electrical and Computer Engineering

University of Massachusetts Lowell

Introduction

- P4 is a high-level language for programming protocol-independent packet processors.
- NFP SDK provides C Sandbox.
- We want to use both P4 and C to develop some protocol-independent measurement functions.

Measurement Functions

- Network measurement has been playing a crucial role in network operations, since it can not only facilitate traffic engineering, but also detect the anomalies.
- For example, counting the heavy hitter and the number of unique flows can be used to detect DoS attacks and port scans.
- However, it is difficult to count in high speed links. Hence, we usually resort to sketch which requires small processing cycle on each packet and maintains good approximation based upon probability.

Measurement Functions

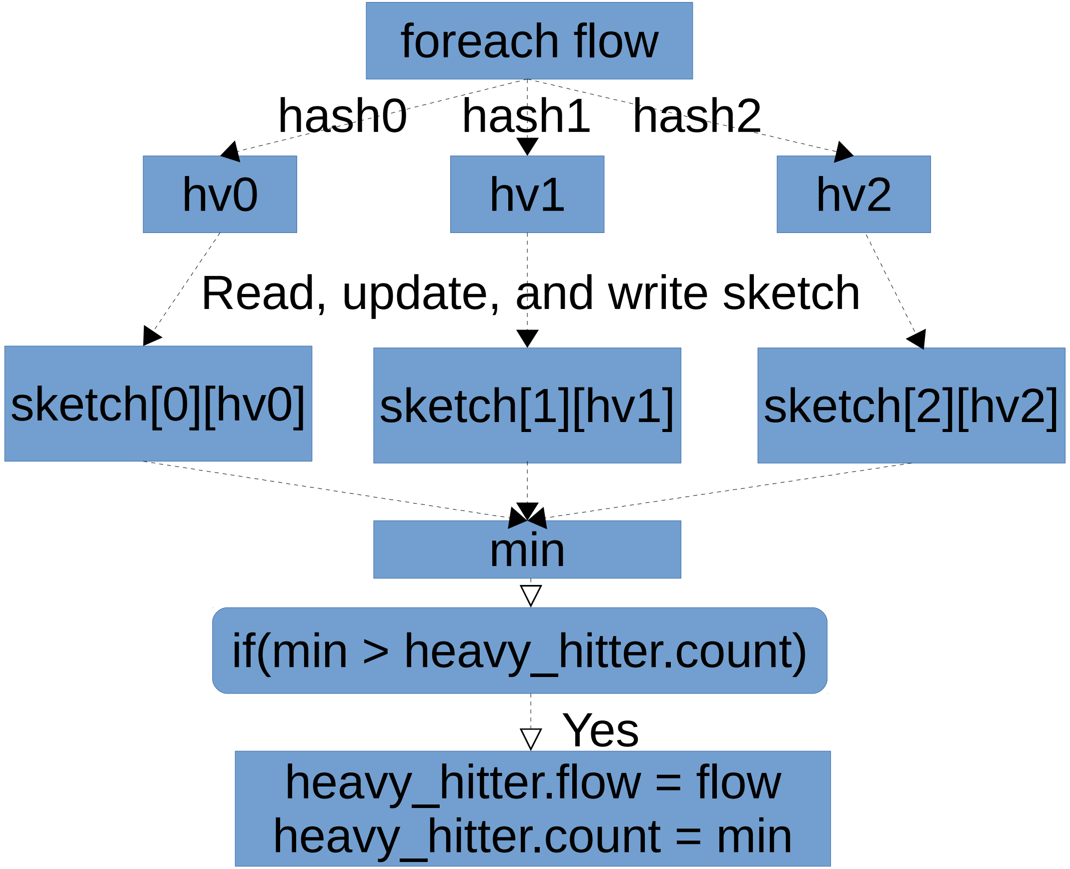
- Heavy Hitter
  - Count-Min
- Number of Unique Flows
  - Bitmap

Algorithm of Count-Min

```
struct Flow;
struct Heavy_Hitter {
    struct Flow flow;
    uint32_t count;
};

//Gloabal
struct Heavy_Hitter heavy_hitter;
uint32_t sketch[3][N];
uint32_t hash0(struct Flow flow);
uint32_t hash1(struct Flow flow);
uint32_t hash2(struct Flow flow);

//Note: max-heap can be used to maintain
multiple global heavy hitters
```



Algorithm of Count-Min

Before flowA, heavy_hitter.flow = flowB and heavy_hitter.count = 2.

Read flowA,

hash0(flowA)			7	
hash1(flowA)		4		
hash2(flowA)				2

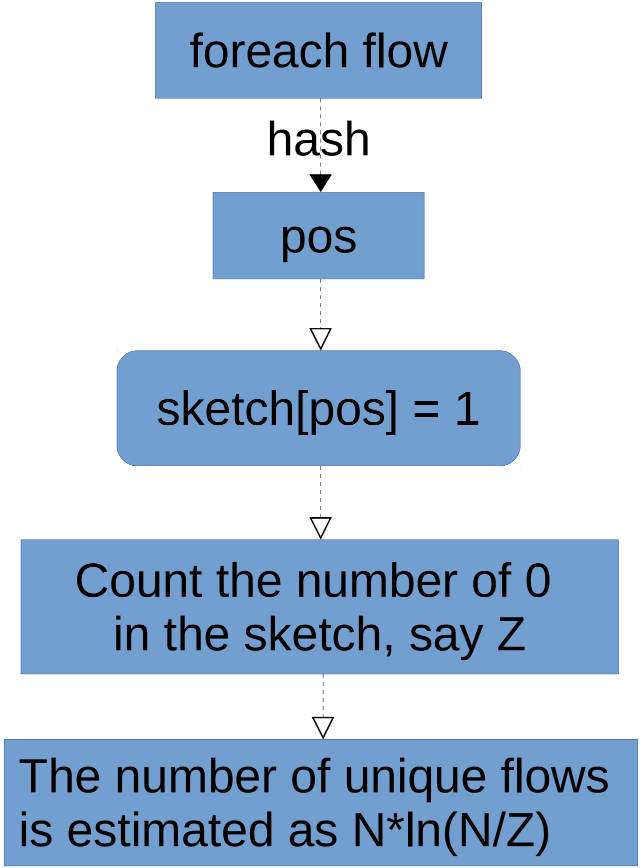
Write flowA,

hash0(flowA)			8	
hash1(flowA)		5		
hash2(flowA)				3

Now, the min{8,5,3} is 3. Since 3 > 2, heavy_hitter.flow = flowA and heavy_hitter.count = 3

Algorithm of Bitmap

```
struct Flow;
uint32_t sketch[N];
unit32_t hash(struct Flow flow);
```



Algorithm of Bitmap

Initial:	0	0	0	0
FlowA:	1	0	0	0
FlowB:	1	0	0	1
FlowC:	1	0	0	1

The number of zero bits is Z = 2

The number of unique flows is estimated as $4*\ln(4/2) = 2.7$

Restrictions of P4-14

- P4-14 has some essential restrictions.
  - If-else statement can only be used in the control block.
  - It does not support for-loop.
  - It has only a limited set of primitive actions.

Restrictions of P4-14 (1)

- If-else statement can only be used in the control block. This implies we can not use if-else statement in the action body of P4.
- Suppose we have 3 variables A, B, C in P4 program, how do we determine the minimum?

Restrictions of P4-14 (1)

```
action do_find_min1{
    modify_field(D, A);
}
table find_min1 {
    actions {
        do_find_min1;
    }
}
```

```
action do_find_min2{
    modify_field(D, B);
}
table find_min2 {
    actions {
        do_find_min2;
    }
}
```

```
action do_find_min3{
    modify_field(D, C);
}
table find_min3 {
    actions {
        do_find_min3;
    }
}
```

Restrictions of P4-14 (1)

```
control ingress {
    apply(find_min1);
    if(D > B) {
        apply(find_min2);
    }
    if(D > C) {
        apply(find_min3);
    }
}
```

AND

Populate the table entries to indicate the default action for each table!!!

Restrictions of P4-14 (1)

- This scenario is exactly one part of the Count-Min algorithm. Hence, implementation of the Count-Min algorithm with vanilla P4 become tedious labor work.

- How to work around of this?

Restrictions of P4-14 (2)

- P4 does not support for-loop.
- Suppose we have an array ARR of size 1024 with 0 and 1 in it, how to find the number of 0 in this array?

Restrictions of P4-14 (2)

- Shall we try the mentioned approach we used to find the minimum before? If so, we need to implement 1 table and 1024 if-statements.

```
action do_inc_count {
	add_to_field(count, 1);
}
table inc_count {
	actions {
		do_inc_count;
	}
}
```

```
control ingress {
	if (ARR0 == 0) {
		apply(inc_count);
	}
	…
	if (ARR1023 == 0) {
		apply(inc_count);
	}
}
```

- This scenario happens exactly in the Bitmap algorithm.
- How to work around of this?

Restrictions of P4-14 (3)

- P4 has only a limited set of primitive actions.
- Suppose now we need a completely new P4 primitive, so that we could put an elephant into a refrigerator, how can we do this?

Characteristic of NFP SDK

- P4 C Sandbox function
  - We can call into C code from P4 program
  - This fixes every restriction we mentioned before

Characteristic of NFP SDK – P4 Side

```
header_type ipv4_t {
    fields {
        srcAddr : 32;
        dstAddr : 32;
    }
}
header ipv4_t ipv4;
```

```
header_type A_t {
    fields {
        timestamp : 32;
    }
}
metadata A_t A;
```

```
primitive_action my_function();
action work() {
    my_function();
}
```

Characteristic of NFP SDK – C Side

```
int pif_plugin_my_function (EXTRACTED_HEADERS_T *headers,
MATCH_DATA_T *match_data)
{
    PIF_PLUGIN_ipv4_T *ipv4_header = pif_plugin_hdr_get_ipv4(headers);
    uint32_t srcAddr = PIF_HEADER_GET_ipv4___srcAddr(ipv4_header);
    uint32_t dstAddr = PIF_HEADER_GET_ipv4___dstAddr(ipv4_header);

    uint32_t prev = pif_plugin_meta_get__A__timestamp(headers);
    pif_plugin_meta_set__A__timestamp(headers, prev +20);
    return PIF_PLUGIN_RETURN_FORWARD;
}
```

Implementations

- Count-Min
  - Count-Min with Vanilla P4
  - Count-Min with P4 and C Sandbox
  - Count-Min with P4 and C Sandbox with Lock
- Bitmap
  - Bitmap with P4 and C Sandbox
  - We skip the detail of this one
- Source Code is available at https://github.com/open-nfpsw/M-Sketch

Count-Min with Vanilla P4

- Stateful memory: register
- Race condition: “@pragma netro reglocked”
- For safety: “@pragma netro no_lookup_caching”

Count-Min with Vanilla P4

```
#define ELEM_COUNT   4
register r1 { width : 32; instance_count : ELEM_COUNT; }
register r2 { width : 32; instance_count : ELEM_COUNT; }
register r3 { width : 32; instance_count : ELEM_COUNT; }
register hh_r { width : 32; instance_count: 3; }

@pragma netro reglocked r1;
@pragma netro reglocked r2;
@pragma netro reglocked r3;
@pragma netro reglocked hh_r;
```

r1, r2 and r3 forms the sketch[3] [4] in the Count-Min algorithm we introduced before.

hh_r forms the global heavy hitter.

Count-Min with Vanilla P4

```
header_type counter_table_metadata_t{
    fields{
        h_v1 : 16;
        h_v2 : 16;
        h_v3 : 16;
        count1 : 32;
        count2 : 32;
        count3 : 32;
        count_min : 32;
    }
}
metadata counter_table_metadata_t
counter_table_metadata;
```

```
header_type heavy_hitter_t {
    fields{
        srcAddr : 32;
        dstAddr : 32;
        count : 32;
    }
}
metadata heavy_hitter_t heavy_hitter;
```

These are used for transition in/out register, since we can not directly operate on register in P4.

Count-Min with Vanilla P4

```
action do_update_cm(){
    modify_field_with_hash_based_offset(counter_table_metadata.h_v1, 0, ipv4_hash0, ELEM_COUNT);
    modify_field_with_hash_based_offset(counter_table_metadata.h_v2, 0, ipv4_hash1, ELEM_COUNT);
    modify_field_with_hash_based_offset(counter_table_metadata.h_v3, 0, ipv4_hash2, ELEM_COUNT);
    register_read(counter_table_metadata.count1, r1, counter_table_metadata.h_v1);
    register_read(counter_table_metadata.count2, r2, counter_table_metadata.h_v2);
    register_read(counter_table_metadata.count3, r3, counter_table_metadata.h_v3);
    add_to_field(counter_table_metadata.count1, 0x01);
    add_to_field(counter_table_metadata.count2, 0x01);
    add_to_field(counter_table_metadata.count3, 0x01);
    register_write(r1, counter_table_metadata.h_v1, counter_table_metadata.count1);
    register_write(r2, counter_table_metadata.h_v2, counter_table_metadata.count2);
    register_write(r3, counter_table_metadata.h_v3, counter_table_metadata.count3);
}
@pragma netro no_lookup_caching do_update_cm;
```

ipv4_hash0, ipv4_hash1, ipv4_hash2 are of type field_list_calculation, and they match hash0, hash1 and hash2 functions we mentioned in the Count-Min algorithm

Count-Min with P4 and C Sandbox

- No need to go through that tedious process of finding the minimum.
- Replace “@pragma netro reglocked” by “mem_read_atomic()” and “mem_write_atomic()”

Count-Min with P4 and C Sandbox with Lock

Wait, if we look closely at our previous implementation, there is a loophole.

```
if(min > heavy_hitter.count) {
        heavy_hitter.flow = flow;
        heavy_hitter.count = min;
}
```

The loophole is that such comparison and updating process have to be a critical section, otherwise we could have a scenario where two threads are all thinking they are having the heavy hitter, so that the updating process could go wrong.

Moreover, we can see that it is impossible to implement a lock with pure P4 under this case. We have to use C sandbox.

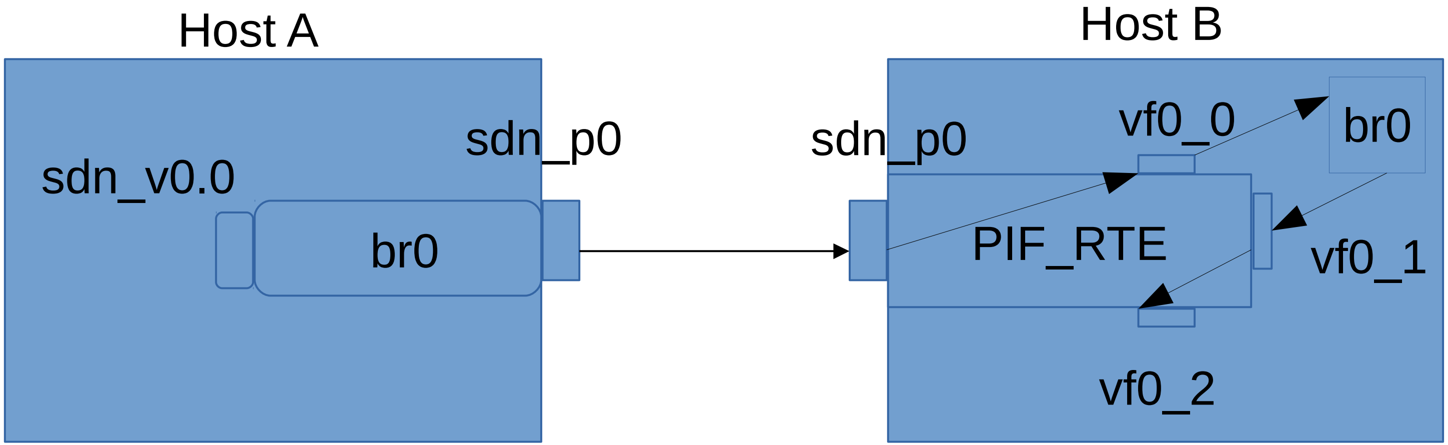
Count-Min with P4 and C Sandbox with Lock

- Implementation of a lock in C Sandbox

```
__export __mem uint32_t lock = 0; //Global
//local below
__xwrite uint32_t xfer_out = 0;
__xrw uint32_t xfer = 1;
mem_test_set(&xfer, &lock, sizeof(uint32_t));
while(xfer == 1) {
    mem_test_set(&xfer, &lock, sizeof(uint32_t));
}
// Critical Section
mem_write32(&xfer_out, &lock, sizeof(uint32_t));
```

Performance Evaluation

- We use 2 2x40G Agilio cards for our performance evaluation, one on each host.
- We use the intrinsic_metadata.ingress_global_tstamp to collect the time stamp at the ingress port.
- In order to get the latency, we let each packet go through the PIF_RTE twice, where vf0_0 to vf0_1 is done by ovs bridge br0



Performance Evaluation

Vanilla P4	P4 and C Sandbox	P4 and C Sandbox with Lock
7798 ME cycle	7376 ME cycle	7441 ME cycle

- The “Vanilla P4” has the longest latency, 5.7% larger than “P4 and C Sandbox”, 4.8% larger than “P4 and C Sandbox with Lock”. This is probably due to many tables on the pipeline.
- The “P4 and C Sandbox with Lock” has almost the same ME cycle with “P4 and C Sandbox”, but it guarantees the accuracy of the algorithm. Hence, for Count-Min, we would always opt for “P4 and C Sandbox with Lock”.

Reference

- An Improved Data Stream Summary: The Count-Min Sketch and its Applications, Graham Cormode, S. Muthukrishnan.
- Bitmap Algorithms for Counting Active Flows on High Speed Links, Cristian Estan, George Varghese, Mike Fisk.

Thanks!

We especially want to thank David George, Mary Pham, Gerhard de Klerk, Behdad Besharat, Nick Viljoen and Hun Namkung for their invaluable input on the Open-NFP Google group.

Appendix: algorithm of Count-Min

```
struct Flow;
struct Heavy_Hitter {
    struct Flow flow;
    uint32_t count;
};


//Gloabal
struct Heavy_Hitter heavy_hitter;
uint32_t sketch[3][N];
uint32_t hash0(struct Flow flow);
uint32_t hash1(struct Flow flow);
uint32_t hash2(struct Flow flow);

//Note: max-heap can be used to maintain
multiple global heavy_hitters
```

```
foreach flow in flow_set {
    uint32_t hv[3];
    uint32_t hv[0] = hash0(flow);
    uint32_t hv[1] = hash1(flow);
    uint32_t hv[2] = hash2(flow);
    for(i=0; i<3; i++)
        sketch[i][hv[i]] += 1;
    uint32_t min = sketch[0][hv[0]];
    for(i=1; i<3; i++) {
        if (min > sketch[i][hv[i]]) {
            min = sketch[i][hv[i]];
        }
    }
    if(min > heavy_hitter.count) {
        heavy_hitter.flow = flow;
        heavy_hitter.count = min;
    }
}
```

Appendix: algorithm of Bitmap

```
struct Flow;
uint32_t sketch[N];
unit32_t hash(struct Flow flow);
foreach flow in flowset {
    uint32_t pos = hash(flow);
    sketch[pos] = 1;
}

uint32_t Z = 0;
for(i=0; i<N; i++) {
    if(sketch[i] == 0) {
        Z += 1;
    }
}
```

The number of unique flows is estimated as N*ln(N/Z)