SOFTWARE DEFINED NETWORKS

LAB 6

AKHILA NAIR
STUDENT ID – 01743358

HAND IN AND DUE DATE: 30th APRIL 2018

ESTIMATED HOURS OF WORKING: 15

1. Purpose:

- To combine DPDK into a real application and operating it with real time problems.
- Integrate DPDK with Count-Min Sketch to solve the Heavy Hitter problem.
- To extract information like Source IP Address, Destination IP Address, Source Port, Destination Port and Protocol.
- Given the data stream of length m and a parameter k, find all elements that occur at least m/k times.
- To use a collection of independent hash functions to each element in the stream and keep tracking the number of times each bucket is hashed to.

2. Lab Setup:

- The setup remains the same as in other labs.
- We have to integrate the lab 2 RX source code to implement the count-min sketch and calculate the hash every time.

3. Software Design:

- Adding the Header Files for Extracting IPV4 Addresses.

```
#include <arpa/inet.h>
```
It serves as the definition for internet operation.

```
#include <netinet/in.h>
```
It functions as internet address family.

```
#include <netinet/ip.h>
```
Defines Internet Version 4

- Creating structure for the UDP

  ```
  struct UDP
  {
      uint16_t sourcePort;
      uint16_t destPort;
      uint32_t dontCare;
  };
  ```

  Here we are assigning the length (from and to) what do we want our information from. Therefore, in UDP after the 16 bits of Source port and 16 bits of destination port, rest all is not required. Therefore, we define that as don’t care conditions.

- Creating the structure for IP

  ```
  struct IP_Header
  {
      u_char ip_vhl;
      u_char ip_tos;
      short  ip_len;
      u_short    ip_id;
      short  ip_off;
  #define    IP_DF 0x4000
  #define    IP_MF 0x2000
      u_char ip_ttl;
      u_char ip_p;
      u_short    ip_sum;
      struct in_addr ip_src;
      struct in_addr ip_dst;
  };
  ```

  In the IP header we have to assign length to each parameters. U_char assigns 8 bits that 1 byte, short assigns 2 bytes, u_short that is unsigned short assigns 2 bytes of address space.

- Creating the structure for flow

```
struct Flow
{
    struct in_addr ip_src;
    struct in_addr ip_dst;
    uint16_t sourcePort;
    uint16_t destPort;
    u_char protocol;
};
```

Here are the parameters we want at the end. Since we do not want anything else than these above information we create a structure and define these parameters.
The parameters are source address, Destination Address, Source port, Destination port and protocol.
They are assigned their respective length.

- Creating the structure for Heavy Hitter and Count min Sketch

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

This will initiate the Heavy hitter whenever this particular structure is called for.

```
struct Heavy_Hitter heavy_hitter;
uint32_t sketch[3][6];
uint32_t hash0(struct Flow flow);
uint32_t hash1(struct Flow flow);
uint32_t hash2(struct Flow flow);
```

The count min sketch is a data structure that serves as a frequency of table of events in to a stream of data. It uses hash functions to map events to frequencies using sublinear space. The goal of the basic version of the count–min sketch is to consume a stream of events, one at a time, and count the frequency of the different types of events in the stream. At any time, the sketch can be queried for the frequency of a particular event type i ($0 \le i \le n$ for some n), and will return an estimate of this frequency that is within a certain distance of the true frequency, with a certain probability.

The actual sketch data structure is a two-dimensional array of w columns and d rows. The parameters w and d are fixed when the sketch is created, and determine the time and space needs and the probability of error when the sketch is queried for a frequency or inner product.

So here we fix the Sketch [3][6] as fix. It will change later in the program as the hash functions are updated.

- Calculating the header information

```
struct Header *data=(struct Header*) (packet+14);
```

Here in struct header, we have defined both IP header as well as UDP header for convenience. This will later point out to the data as shown.

- Using the Flow structure to store the data

```
flow.ip_src=data->ipHeader.ip_src;
flow.ip_dst=data->ipHeader.ip_dst;
flow.sourcePort=data->udpHeader.sourcePort;
flow.destPort=data->udpHeader.destPort;
flow.protocol=data->ipHeader.ip_p
```

Here, we are taking the 5 tuple information and giving it to data.

- Calculating the Hash and returning it

```
    uint32_t hash0(struct Flow flow)
    {
    uint32_t hash;
    hash=((flow.ip_src.s_addr)+(flow.ip_dst.s_ad
dr)*(flow.sourcePort)-(flow.destPort)-
(flow.protocol));
    hash=hash%6;
    return hash;
    }
```

While calculating the hash function, basically what done is, you can add, multiply, subtract the 5 tuple information. Store it and then perform modulus of that hash with number taken as d in the count min sketch. After this is complete, return the hash value.
This is done to calculate the hash 2 and hash 3 value in the same way.

- Storing the hash

```
hv[0] = hash0(flow);
hv[1] = hash1(flow);
hv[2] = hash2(flow);
```

Here the hash is stored in hash value and the flow structure initiated.

- Final step

```
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;
}
```

The sketch value is stored in min after iterating it through a for loop. Only if the min value is greater than the heavy hitter count value will the heavy hitter will be detected and will give us the information that was needed.

4. Troubleshooting:

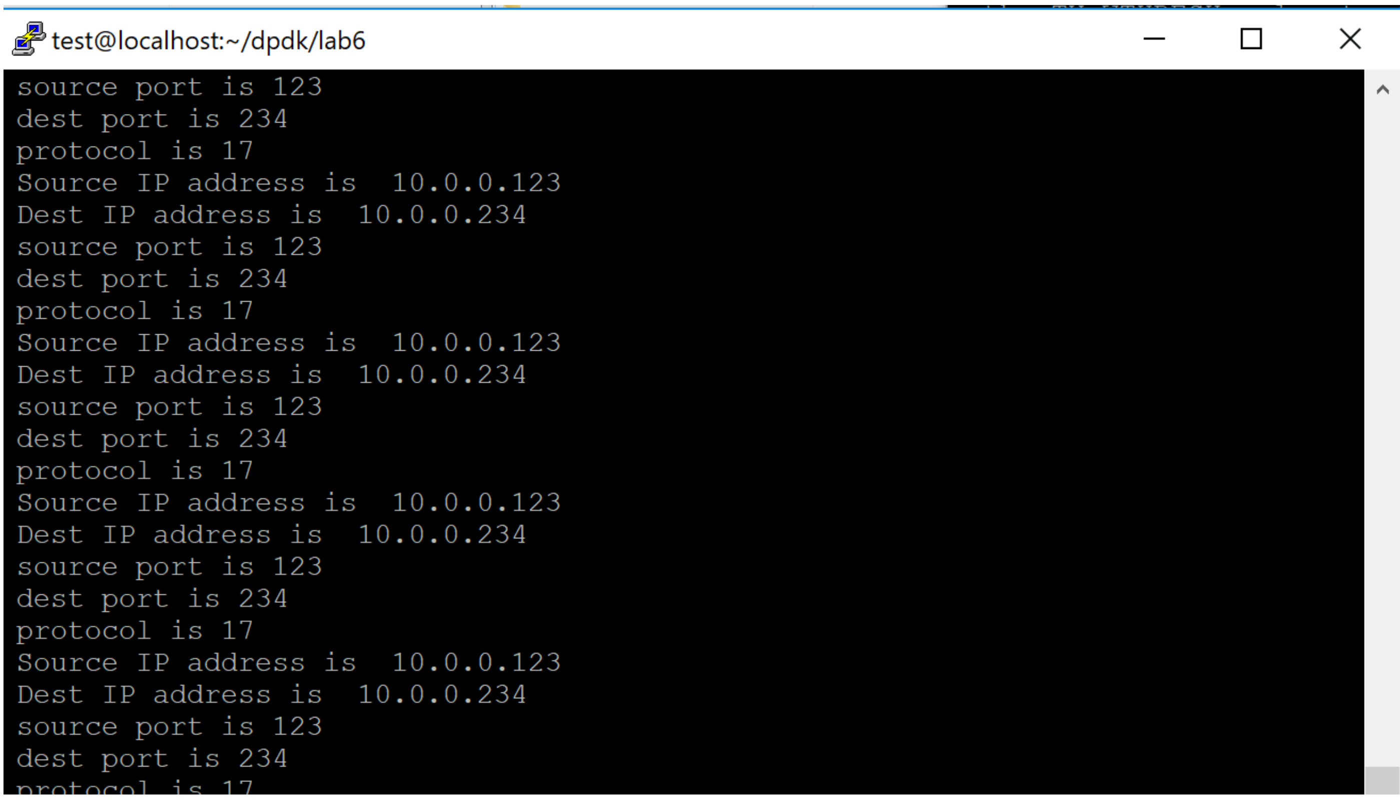
- **Indentation Errors.**
  After careful go through of code this was solved.

- **Garbage Value was getting printed.**
  Since I had used two for loop with same i as the iteration value through it, there was confusion and hence garbage was getting printed.
  I changed the next for loop to j and the problem was solved.

5. RESULTS:

The following image shows the working of RX side and the command used to run the program is

sudo ./build/rx_demo -w 0000:02:00.0 --socket-mem 256 --file-prefix rx -c 0x01



test@localhost:~/dpdk/lab6

```
source port is 123
dest port is 234
protocol is 17
Source IP address is  10.0.0.123
Dest IP address is  10.0.0.234
source port is 123
dest port is 234
protocol is 17
Source IP address is  10.0.0.123
Dest IP address is  10.0.0.234
source port is 123
dest port is 234
protocol is 17
Source IP address is  10.0.0.123
Dest IP address is  10.0.0.234
source port is 123
dest port is 234
protocol is 17
Source IP address is  10.0.0.123
Dest IP address is  10.0.0.234
source port is 123
dest port is 234
protocol is 17
```

6. Conclusion:

Hence the Heavy Hitter detector was implemented with the help of count-min sketch in the DPDK application Box.