Lab 3 — QoS Implementation with DPDK

Handout: March 25, 2021
Deadline: April 9 23:00, 2021 (No extension)

Assignment overview:

In this assignment, you will implement two algorithms, i.e. **srTCM** and **WRED**, to provide QoS metering and congestion control using DPDK, and experiment on how to meet service requirements. The first one is used to meter network traffic and mark packets with color representing for different flow consumption, and the second one is used to drop packets in advance to avoid congestion. Your task is divided into two steps: the first one is to implement **a meter and a dropper** with the aforementioned algorithms; the second one is to **deduce the parameters** that the algorithms need to meet different users' traffic requirements. **C** language and some DPDK libraries will be used.

Prerequisite:

DPDK must be installed, either in virtual machine or in your physical machine (Linux needed). Since we are under simulated environment(No real NIC and drivers), there is no result difference either in VM or in physical machine. To install DPDK, refer to the guidance (http://doc.dpdk.org/guides/linux_gsg/).

You are **recommended** to develop under DPDK for convenience, rather than develop everything from scratch.

For more information about DPDK, Please go to http://dpdk.org/

Algorithms introduction:

Note: In order to deduce the parameters, you should learn more about the algorithm details from reference materials at the end of this guide.

- Single Rate Three Color Marker (srTCM):

  As defined by RFC2697, the srTCM meters the stream of incoming packets based on the allowance defined in advance for each traffic flow. As a result, each incoming packet is tagged as green, yellow or red based on the monitored consumption of the flow the packet belongs to. The srTCM meters a traffic stream and marks its packets according to three traffic parameters, Committed Information Rate (CIR), Committed Burst Size (CBS), and Excess Burst Size (EBS). A packet is marked green if it doesn't exceed the CBS, yellow if it does exceed the CBS, but not the EBS, and red otherwise. The srTCM can work in two modes: color blind mode and color aware mode, we use color blind mode in our experiment.

- Weighted Random Early Detection (WRED):

  As defined by RFC2309, WRED is an active queue management algorithm for routers. In contrast to traditional queue management algorithms, which drop packets only when the buffer is full, the RED algorithm drops arriving packets probabilistically. The probability of drop increases as the estimated average queue size grows. Note that RED responds to a time–averaged queue length, not an instantaneous one. Thus, if the queue has been mostly empty in the ”recent past”, RED won’t tend to drop packets (unless the queue overflows, of course!). On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped.

  The RED algorithm itself consists of two main parts: estimation of the average queue size and the decision of whether or not to drop an incoming packet. The RED algorithm uses an Exponential Weighted Moving Average (EWMA) filter to compute average queue. (for EWMA details, see reference 1.) For each enqueue operation, the RED algorithm compares the average queue size with minimum and maximum thresholds. Depending on whether the average queue size is below, above or in between these thresholds, the RED algorithm calculates the probability that an arriving packet should be dropped and makes a random decision based on this probability.

  Further, the WRED allows the scheduler to select different RED configurations for the same packet queue at run–time. In our experiment, we select different RED configurations for same user’s flow based on the packet’s color marked by the meter.

Task one: implement a meter and a dropper

The meter : The meter shall mark packets using srTCM algorithm. You are supposed to implement two functions:

- int qos_meter_init(void);

  This function will be called only once at the beginning of the test. You can initialize your meter here.
- enum qos_color qos_meter_run(uint32_t flow_id, uint32_t pkt_len, uint64_t time);

  This function will be called for every packet in the test, after which the packet is marked by returning the corresponding color. Note, we are under simulated environment here, so that the packets given to your meter is not real (i.e. not polled from NIC, but appear to be the same) since the meter only cares about the flow that the packet belongs to, the length of the packet and the time that the packet is metered. The pkt_len is in bytes, the time is in nanoseconds.

You are recommended to use DPDK’s srTCM implementation for convenience, and you’d better firstly read the codes in DPDK/lib/librte_meter, and refer to the sample in DPDK/examples/qos_meter.

The dropper : The dropper shall pass or drop packets using WRED algorithm. You are supposed to implement two functions:

- int qos_wred_init(void);

This function will be called only once at the beginning of the test. You can initialize you dropper here.

- int qos_wred_run(uint32_t flow_id, enum qos_color color, uint64_t time);

  This function will be called for every tested packet after being marked by the meter, and will make the decision whether to drop the packet by returning the decision (0 pass, 1 drop). Similar to the meter, the environment is simulated here. Note, the key factor that RED algorithm cares is the queue's state, namely how many packets are there in the queue. The queues' state shall be maintained by your dropper explicitly. In our experiment, we assume the queues will be cleared (meaning all packets in the queues will be sent out) at end of the time period (1000 ns), and the time will only be updated in unit of time period.

You are recommended to use DPDK's RED implementation for convenience, and you'd better firstly read the codes in DPDK/lib/librte_sched/rte_red*, and refer to the sample in DPDK/examples/qos_sched.

Task two: deduce parameters

The parameters given to the srTCM and WRED are keys to the behavior of the two algorithms, your task is to deduce a set of appopriate parameters to meet the requirements below.

- Packets will pass-through your meter and dropper sequentially. These packets belong to 4 flows, and packets belonging to different flows shall not influence each other.
- The 4 flows share total bandwidth in proportion of 8:4:2:1, that is flow 0 has highest quality of service and its allocated bandwidth is 8 times of flow 3.
- Flow 0's required bandwidth is about 1.28 Gbps (that is also tester's bandwidth).

Note: there are many ways to meet the requirements, and you can choose any one of them. But there will be a bonus if you try more than one way.

Testing environment:

As mentioned above, we use simulator to test your meter and dropper, the testing routine is illustrated below.

```
qos_meter_init();
qos_wred_init();
/* some details omitted */
for (i = 0; i < 10; i++) {
    /* 1000 packets per period averagely */
    int burst = 500 + rand() % 1000;
    for (j = 0; j < burst; j++) {
        uint32_t flow_id = (uint32_t)(rand() % APP_FLOWS_MAX);
        /* 640 bytes per packet averagely */
        uint32_t pkt_len = (uint32_t)(128 + rand() % 1024);
        /* get color */
```

```
        enum qos_color color = qos_meter_run(flow_id, pkt_len, time);
        /* make decision: weather drop */
        int pass = qos_wred_run(flow_id, color, time);
        cnt_send[flow_id] += pkt_len;
        cnt_pass[flow_id] += pass ? 0 : pkt_len;
    }
    time += 1000000;
}
```

Source code:

You can download the source code from here.

Grading:

- 50% for accurate implementation of meter and dropper, code will be checked manually.
- 50% for appropriate parameters deduction and traffic requirements meeting. Detailed parameters deduction shall be written in report.

Handin procedure:

You should turn in a Report and source code. For source code, only qos.c is needed. For report, two parts are needed: parameter deduction process in detail, and introduction to the DPDK APIs that you use.

Upload your report and source code as a gzipped tar file, named as {Your student ID}.tar.gz to **CANVAS**

Reference:

[1] DPDK Programmer's Guide, Quality of Service, http://dpdk.org/doc/guides/prog_guide/qos_framework.html#traffic-metering
[2] RFC2697, A Single Rate Three Color Marker, https://tools.ietf.org/html/rfc2697
[3] RFC2309, Recommendations on Queue Management and Congestion Avoidance in the Internet, https://tools.ietf.org/html/rfc2309