Lab 3 QoS Implementation with DPDK

Parameter Deduction

方法一： 通过`srTCM` + `cbs`限制流量

RED：绿包和黄包全部接收，红包全部丢弃。对于四个`flow`设置相同

color	min_th	max_th	wq_log2	maxp_inv
GREEN	1022	1023	RTE_RED_WQ_LOG2_MAX	10
YELLOW	1022	1023	RTE_RED_WQ_LOG2_MAX	10
RED	0	1	RTE_RED_WQ_LOG2_MAX	10

srTCM: 因为在`color blind`模式下

- 对于单速单筒（EBS=0）
  - 如果报文长度不超过 C桶中的令牌数 Tc，则报文被标记为绿色，且 Tc=Tc-B
  - 如果报文长度超过C 桶中的令牌数 Tc，报文被标记为红色，Tc 值不变
- 对于单速双桶（EBS≠0）
  - 如果报文长度不超过 C桶中的令牌数 Tc，则报文被标记为绿色，且 Tc=Tc-B
  - 如果报文长度超过C 桶中的令牌数 Tc 但不超过E 桶中的令牌数 Te，则报文被 标记为黄色，且 Te=Te-B
  - 如果报文长度超过E 桶中的令牌数 Te，报文被标记为红色，但 Tc 和 Te 不变

结合`RED`策略红色包全部丢弃的法则，可以使用**单速双筒**，通过**cbs**限制绿色包的数量，使四个`flow`大致成`8:4:2:1`。为了更好的适应突发性包，使用双筒，设置少量`ebs`

（其实设置`ebs=0`单速单筒 也能实现）

参数计算:

$$1.28Gbps = 160Mbyte/s = 160,000,000byte/s$$

所以对于一个`flow`的带宽可以设置为`160000000`

查看发包代码可以发现，一个时间段内(1ms), 平均发给一个`flow`的字节数为

$$1000 * 640 \div 4 = 160000$$

所以可以设置参数为：`160000：80000：40000：20000`

但是因为发现`flow 0`的包总有损失，所以稍稍调大`flow 0`的`cbs`：

flow_id	cir	cbs	ebs
0	160000000	160000	1000
1	160000000	80000	500
2	160000000	40000	250
3	160000000	20000	125

结果:

```
fid: 0, send: 1734784, pass: 1734784
fid: 1, send: 1750867, pass: 803529
fid: 2, send: 1690936, pass: 416787
fid: 3, send: 1677990, pass: 238058
```

综上，4个`flow`的比列大致为`8:4:2:1`，且`flow 0`的带宽为`1.28Gbps`

方法二：通过`srTCM` + `cir`限制流量

RED：绿包和黄包全部接收，红包全部丢弃。对于四个`flow`设置相同

color	min_th	max_th	wq_log2	maxp_inv
GREEN	1022	1023	RTE_RED_WQ_LOG2_MAX	10
YELLOW	1022	1023	RTE_RED_WQ_LOG2_MAX	10
RED	0	1	RTE_RED_WQ_LOG2_MAX	10

srTCM: 单速单筒 + `cir`限制流量

因为`cir`限制带宽，所以将四个`flow`的`cir`设为`8:4:2:1`即可

将"桶的大小"也就是`cbs`设置为足够大即可

flow_id	cir	cbs	ebs
0	160000000	160000	0
1	80000000	160000	0
2	40000000	160000	0
3	20000000	160000	0

结果:

```
fid: 0, send: 1865344, pass: 1485092
fid: 1, send: 1830926, pass: 882153
fid: 2, send: 1817512, pass: 520559
fid: 3, send: 1820186, pass: 339917
```

综上，4个`flow`的比列大致为`8:4:2:1`，且`flow 0`的带宽为`1.28Gbps`

但是效果没有方法一好

方法三：结合`cir`和`cbs`

RED：绿包和黄包全部接收，红包全部丢弃。对于四个`flow`设置相同

color	min_th	max_th	wq_log2	maxp_inv
GREEN	1022	1023	RTE_RED_WQ_LOG2_MAX	10
YELLOW	1022	1023	RTE_RED_WQ_LOG2_MAX	10
RED	0	1	RTE_RED_WQ_LOG2_MAX	10

srTCM:

flow_id	cir	cbs	ebs
0	160000000	160000	160000
1	80000000	80000	80000
2	40000000	40000	40000
3	20000000	20000	20000

结果:

```
fid: 0, send: 1654935, pass: 1654935
fid: 1, send: 1668301, pass: 882060
fid: 2, send: 1622648, pass: 440530
fid: 3, send: 1597211, pass: 236423
```

DPDK APIs

`srTCM`相关

Functions

- `rte_meter_srtcm_color_blind_check`:使用`color blind`策略计算包应该被标记的颜色

```
static inline enum rte_color
rte_meter_srtcm_color_blind_check(struct rte_meter_srtcm *m, /*Handle to
srTCM instance*/
        struct rte_meter_srtcm_profile *p,  /*srTCM profile*/
        uint64_t time,          /*Current CPU time stamp (measured in CPU
cycles)*/
        uint32_t pkt_len        /*Length of the current IP packet (measured
in bytes)*/
                                )
```

    - 注：该函数需要的`time`是`CPU cycles`，而`qos_color qos_meter_run`函数中提供的`time`是纳秒
- `rte_meter_srtcm_profile_config`：用参数`params`配置`srTCM`配置文件`p`

```
int rte_meter_srtcm_profile_config(struct rte_meter_srtcm_profile *p,
    struct rte_meter_srtcm_params *params);
```

    - @param `p` : Pointer to pre-allocated srTCM profile data structure
    - @param `params` : srTCM profile parameters
    - @return : 0 upon success, error code otherwise
- `rte_meter_srtcm_config`：用`srTCM configuration`配置每一个`metered traffic flow`

```
int rte_meter_srtcm_config  (struct rte_meter_srtcm * m,
    struct rte_meter_srtcm_profile * p
)
```

- `rte_get_tsc_cycles`： Get cycles
- `rte_get_tsc_hz`: Get the measured frequency of the RDTSC counter

Data Structures

- `rte_meter_srtcm_params`：`srTCM`所需要的参数
    - `cir`：`Committed Information Rate`，单位：`bytes per second`
    - `cbs`：`Committed Burst Size`，单位：`bytes`
    - `ebs`：`Excess Burst Size`，单位：`bytes`
- `rte_meter_srtcm_profile`：`srTCM`配置
- `rte_meter_srtcm`：存储`srTCM`运行时的状态
    - `time`： Time of latest update of C and E token buckets
    - `tc` : Number of *bytes* currently available in the committed (C) token bucket
    - `te` : Number of *bytes* currently available in the excess (E) token bucket

RED相关

Functions

- `rte_red_config_init`：初始化RED参数的数据结构
- `rte_red_rt_data_init`： Initialises run-time data
- `rte_red_mark_queue_empty`: Callback to records time that queue became empty
- `rte_red_enqueue` : Decides if new packet should be enqeued or dropped. Updates run time data based on new queue size value. Based on new queue average and RED configuration parameters gives verdict whether to enqueue or drop the packet.

Data Structures

- `rte_red_config` : RED configuration parameters
    - `min_th`: min_th scaled in fixed-point format
    - `max_th`: max_th scaled in fixed-point format

    - `pa_const`: Precomputed constant value used for pa calculation (scaled in fixed-point format)
    - `maxp_inv`: maxp_inv
    - `wq_log2`: wq_log2
- `rte_red` : RED run-time data