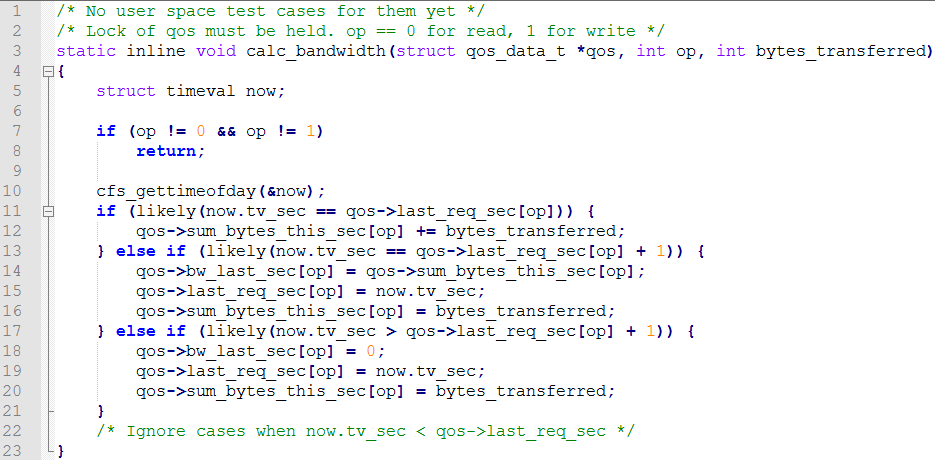
1. metric.h中计算bandwidth



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T | T+1 | T+2 | T+3 | T+4 | T+5 |

过程分析：图示中一个方块代表1单位时间（一般为1s）

long last\_req\_sec[2]; /\* second of last request we received \*/

\_\_u64 bw\_last\_sec[2]; /\* bw of last sec \*/

\_\_u64 sum\_bytes\_this\_sec[2]; /\* cumulative bytes read within this sec \*/

二位数组0是read，1是write：

last\_req\_sec是收到上一个请求到达的时间

bw\_last\_sec是上一秒内的bandwidth

sum\_bytes\_this\_sec是本秒内积累的数据量

技巧：计算bandwidth，如果选取1s作为时间区间，则1s内积累的数据量就是bandwidth；因此在计算bandwidth的时候，程序采取计算1s内数据量的方式来计算bandwidth，那么如何计算呢？

1. 如果到达的request的时间戳和last\_req\_sec相同，说明该request和前面的request在1s区间内，直接将该request的数据量加到sum\_bytes\_this\_sec中，进行累计；
2. 如果当前request的时间戳等于qos->last\_req\_sec[op] + 1，说明该请求刚好进入了“下一个”时间段，此时就可以知道前一秒的bandwidth，也就是sum\_bytes\_this\_sec 中累计的数据量，也即qos->bw\_last\_sec[op] = qos->sum\_bytes\_this\_sec[op];同时进行新一轮的qos->sum\_bytes\_this\_sec[op]记录，而刚好这个到达的request就是qos->sum\_bytes\_this\_sec[op]中累计的第一拨数据；
3. 如果到达的request的时间戳和前面的请求间隔不在1s内，也即now.tv\_sec > qos->last\_req\_sec[op] + 1，说明这个到达的request和前一个request相差几个时间段，前一个时间段的bw\_last\_sec就肯定为零，那么置qos->bw\_last\_sec[op] = 0; 同时更新qos->last\_req\_sec[op] 为当前时间戳，从新开始计时，也更新qos->sum\_bytes\_this\_sec ，从新开始计算数据量：qos->last\_req\_sec[op] = now.tv\_sec;qos->sum\_bytes\_this\_sec[op] = bytes\_transferred；

ASCAR Score的格式

SCORE\_LINE format: ${RULE\_NO},${SCORE},${CANDIDATE\_AVG\_BW},${CANDIDATE\_AVG\_VAR},${CANDIDATE\_TRIED\_TIMES}

(new\_rule\_no， rules\_per\_sec，， m100， b100， tau100， used\_times， ack\_ewma\_avg， send\_ewma\_avg， rtt\_ratio100\_avg)

|  |
| --- |
| lctl set\_param osc.lustrefs-OST0000-osc-ffff8800787f8000.qos\_rules="8,2  0,107193,0,107448,0,411,157,-29,23319,1839,80926,81021,173  0,107193,0,107448,411,2147483647,87,-59,25916,153,83910,84083,2092  0,107193,107448,2147483647,0,411,87,-59,25916,22,106273,109143,180  0,107193,107448,2147483647,411,2147483647,87,-59,25916,2,106526,111891,1957  107193,2147483647,0,107448,0,411,87,-59,25916,27,108285,106752,226  107193,2147483647,0,107448,411,2147483647,87,-59,25916,4,108526,106813,1656  107193,2147483647,107448,2147483647,0,411,94,-59,28012,1960,144271,144560,173  107193,2147483647,107448,2147483647,411,2147483647,87,-59,25916,263,143655,143959,2476" |

new\_rule\_no：指规则的数量

used\_times， ack\_ewma\_avg， send\_ewma\_avg， rtt\_ratio100\_avg初始值为0，是在使用的过程中计算得出

**关于指数加权平均EWMA**

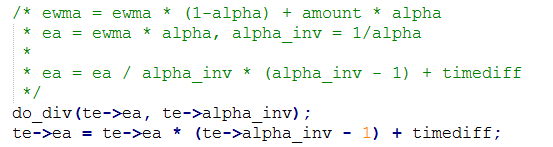
部分CPU不支持浮点数计算，因此内核中一般不直接定义float，但是+

其中，不宜直接定义在内核中，因此取其倒数定义；令

+

+

令则上式



关于规则合并时代码部分数学推导

= =

== =