International Journal of Database Theory and Application
Vol.8, No.3 (2015), pp.85-100
http://dx.doi.org/10.14257/ijdta.2015.8.3.09

Design and Implementation of the Algorithm of QoS Virtual Queue

Jian Zhang
Zhejiang Business College,Hangzhou,China
Email: szy425145773@163.com

Abstract

QoS is very important in current internet and NIT(next generation internet), with the development of application, the QoS function is also more and more complex. At present, the application of multi-core processor is becoming more and more popular, because the performance of multi-core is stronger, the QoS function is also implemented by software. This paper describes the background of QoS, discusses the traditional algorithm implementation with software, and describes the shortcomings of the traditional implementation, then presents and analyses the ideas of improvement, at last, proposes a new improved algorithm to solve the problems encountered. This algorithm is fully verified in network controller.

Keywords: *VQ, Virtual queue, QoS, WFQ*

1. Introduction

QoS is short for quality of service. For the network business, service quality includes bandwidth, transmission delay and transmission of data packet loss rate. In the network, to improve the quality of service, there are many ways to do, such as guaranteeing the bandwidth which reduces the transmission delay and packet loss rate of data, delaying jitter and so on.

For the QoS researches in recent years, the international organization for standardization institutions proposed and published a series of solutions of QoS protocol , such as RSVP, IntServ, DiffServ etc. However, it is hard to establish IntServ/RSVP on IP protocol. As no connection of IP itself, it is difficult to establish and maintain an oriented connection, to require resources guaranteeing the channel and to deploy hardly. DiffServ uses a completely different idea with IntServ because of its good scalability. Moreover, DiffServ is simple to realize, operate and deploy capabilities, it becomes the mainstream of the current IP QoS architecture.

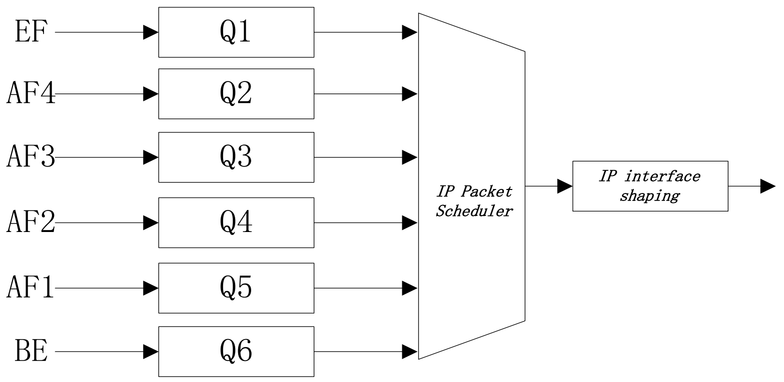

Figure 1. QoS Typical Structure

In figure1, it illustrates a typical DiffServ QoS model that IETF launched Diff-Serv (Differentiated Service) QoS classification standard in December 1998. This model uses 6 bits in TOS byte in IP header in each data service category to distinguish the priority through the coding value. DSCP (Differentiated Services Code Point) uses 6 bit, and its value range is between 0 and 63. Each DSCP code value is mapped to a defined PHB (Per-Hop-Behavior) identification code.

2. Typical Implementation of QoS and Analysis of Exist Problem

2.1Background of Typical QoS

Nowadays, with the high-speed development of network equipment, single port throughput is becoming larger with more users accordingly. Otherwise, new problems encountered. Traditional QoS, a flow management, bases on port bandwidth scheduling. It is attractive to the grade of service instead of the users, in other words, it is for the network side, but not for the business side of access. To solve this problem and provide better QoS service, it is urgent to seek a new solution to not only control the users' rate of flow, but also avoid multiple users simultaneously. Multi-level scheduling QoS can fulfill such requirement, like providing quality assurance for the advanced users and saving the cost of overall network construction. This is done by the strategy of controlling the internal resources in the equipment.

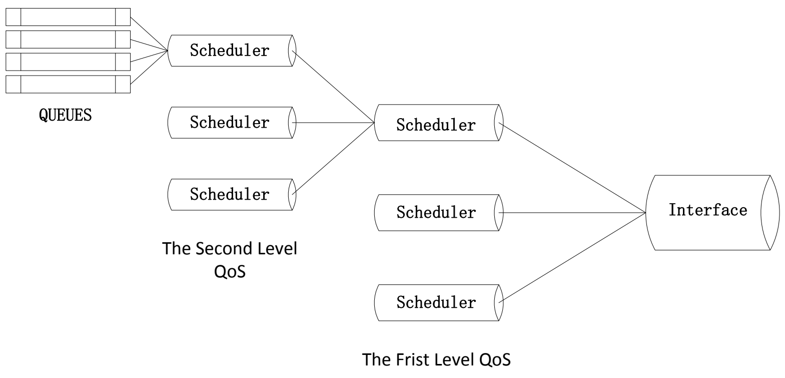

Figure 2. Hierarchical QoS

In Figured 2, it shows a typical application scenario that the end user traffic is shaped in the second layer, multiple users become the first layer scheduler (corridor side), then all the traffic is flowed out via the physical port.

At present, with the simple development under multi-core processor and the swift operation speed, multi-core processor is widely used in many kinds of network equipment. Most of it is able to support the function of QoS in hardware level. Besides, for the equipment manufacturers, multi-core processor achieves the first layer QoS and second layer QoS with the functions of the hardware and with the software respectively.

2.2Typical Implementation of QoS Algorithm

Figure 3 is a typical requirement for QoS function which includes shaping function and weighted fair queuing.

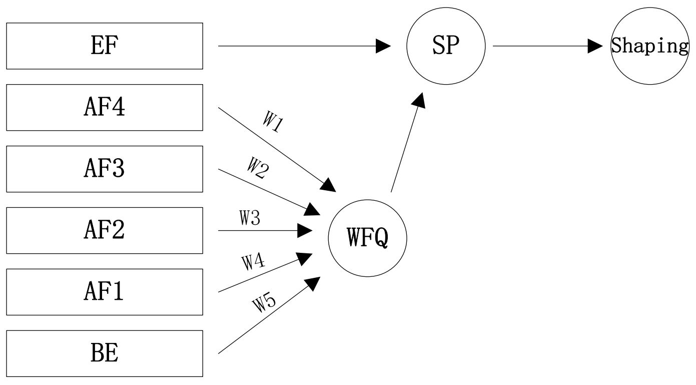

Figure 3. The Functional Module of QoS

Six queues as above show that EF is SP (strict priority) queue, the priority scheduling is needed, AFs'queue scheduling priority is AF4>AF3>AF2>AF1>BE, each queue's (except for the EF queue) rate is calculated with formula as below.

$$QueueAverageRate = AvailableCapacity \times QueueWeight \,/ \sum_{ActiveQueues} QueueWeight$$

As following, it is an implementation pseudo-code of the Queue algorithm to achieve the QoS function. As this paper mainly focuses on the description of the problems, the process of WFQ is not mentioned. If the traffic doesn't exceed the bandwidth, then the packet is just sent out from the scheduler. Conversely, if the traffic exceeds the bandwidth, then the timer is launched to send procedure.

```
typedef struct leaky_fifo_control_t
{
uint16_t      fifo_size;              /* Max. number of records in the fifo */
uint16_t      length;                 /* Number of records in the fifo      */
uint16_t      head;                   /* Index of the first record          */
uint16_t      tail;                   /* Index of the next free record      */
struct mbuf   **queue_ptr;            /* fifo data area                     */
struct mbuf   **pop_queue_ptr;
uint32_t      max_burst_bits;         /* Maximum busrt bits in leaky packet */
uint32_t      speed;                  /* fifo leaky speed kbps */
uint32_t      pko_max_burst_bits;  /* Maximum busrt bits in leaky packet forPKO */
uint64_t      prev_cycle;             /* Last packet sent cycle */
int64_t   borrow_bits;   /* borrowed bits in leaky bucket: should be paid off */
} leaky_fifo_control_t;


static void   qos_read_leaky_buffers_timeout_handler__r(leaky_fifo_control_t
*leaky_bucket_control_ptr )
```

```
{
    struct mbuf *queued_wqe = NULL;
    int64_t b_token = 0;
    uint64_t current_cycle = cvmx_get_cycle_global();
    uint32_t m_length = 0;
    do{
        if ( eitp_fifo_get__r(&queued_wqe, leaky_bucket_control_ptr) == FIFO_EMPTY )
        if(wfq_dequeue(bucket_index,&queued_wqe)!= WFQ_PROCESS_STATUS_OK)
                break;
        current_cycle = cvmx_get_cycle_global();
        b_token = leaky_bucket_control_ptr->borrow_bits;
        b_token-=eitp_qos_increment_token__r(current_cycle,
        leaky_bucket_control_ptr->prev_cycle,
        leaky_bucket_control_ptr->speed);
        b_token += m_len(queued_wqe)* 8;
        m_length = m_len(queued_wqe);
        if (b_token > (int64_t)leaky_bucket_control_ptr->max_burst_bits) {
           qos_start_timer(b_token,bucket_index,leaky_bucket_control_ptr, current_cycle);
          QOS_SEND_PACKET_TO_EXTERNAL(queued_wqe,bucket_index, m_length);
            break;
        }
        leaky_bucket_control_ptr->borrow_bits = b_token;
        leaky_bucket_control_ptr->prev_cycle = current_cycle;
        QOS_SEND_PACKET_TO_EXTERNAL (queued_wqe, bucket_index, m_length);
      }while(1);
}
```

The above diagram algorithm ignores the packet process function, data structure leaky_fifo_control_t defines the control structure of second level scheduler.

■speed---- two level scheduler bandwidth values.

■prev_cycle----the time lunched in the previous

■borrow_bits---- borrowed bits (to deal with burst volume, the ticks number borrowed from token bucket).

When flow overrun (this paper does not pay attention on non-overload situation), it processed in the timeout function. If b_token is greater than max_burst_bits, stop the timer and start a new timer. Every time the amount in the timer would not exceed the burst size.

Above algorithm is the approximate implementation of leaky bucket algorithm, the input flow of the algorithm is continuous flow or burst flow, the expected output flow is continuous flow as well. In real environment, of course, the best expected output are similar with the input flow, it means that if the input flow is continuous, then the output flow is continuous, moreover ,if the input flow is burst, then the output flow is burst.

From the software implementation, the actual input and output relationship in traditional algorithm is illustrated as bellow.

Case1: If the input traffic does not exceed bandwidth, the output flow is similar with the input flow.

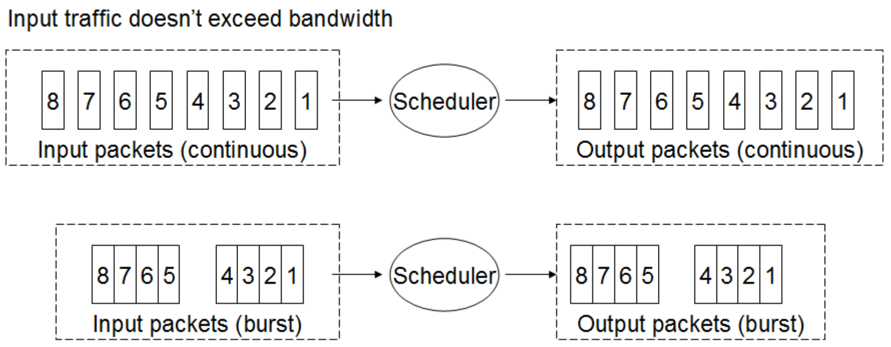

Figure 4. Input Traffic Doesn't Exceed Bandwidth

Case 2: If the input traffic exceeds bandwidth, the output flow is burst flow no matter whether the input flow is burst or continuous.

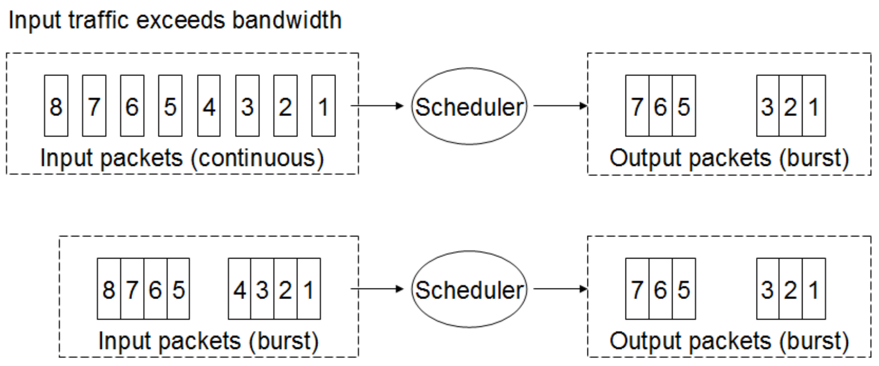

Figure 5. Input Traffic Exceeds Bandwidth

2.3Existed Problem in Typical QoS Implementation

There is no issue in using the above algorithm individually, if in previous case in the paper, the algorithm used in the second level scheduler. In practical environment, if the second level scheduler bandwidth value is less than a first scheduler's, there are the packet drops in first scheduler.

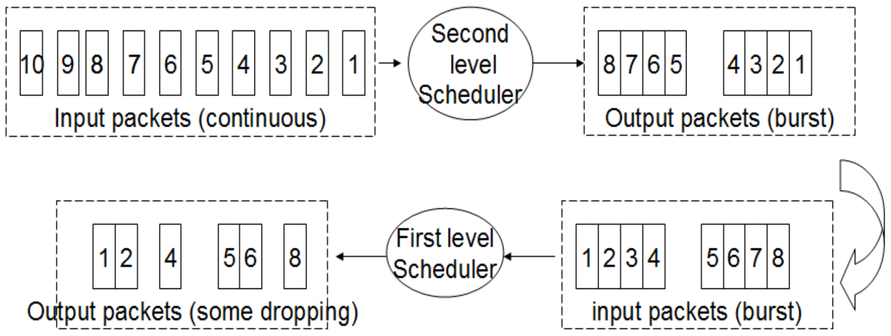

Figure 6. Existed Problem in Typical Algorithm

The problem is that in second level scheduler the algorithm is implemented by software, but the first level scheduler is implemented by hardware. The first level scheduler is even, but the second level scheduler is burst model. And the second level scheduler is implemented by time slice (software timer). Considered the system performance reasons, time slice may not be so small with hardware. When overflow occurs, the second level scheduler will have burst traffic in a time slot. See the above picture, from the output queue in second level scheduler, the packet becomes burst flow. When packets reach first level scheduler, after packet 1 and packet 2 taking the token in bucket, packet 3 may not get token. Therefore it is dropped, although first level scheduler's bandwidth is larger than the second level scheduler's. In continuous mode, the packet 3 can get token in the second level scheduler, but in burst mode, it cannot get token from first level scheduler.

2.4Analysis of Improvement for Typical QoS Algorithm

Algorithm one: Create a barrier for second level scheduler; the barrier value is adjusted by the first level QoS scheduler bandwidth. When the traffic enters the second level QoS scheduler, it adds statistics, and compares the statistics with the barrier value. If the barrier is larger than the first level QoS scheduler, then it stops sending the packets. The key point is that the first level scheduler gives a feedback to the second level scheduler.

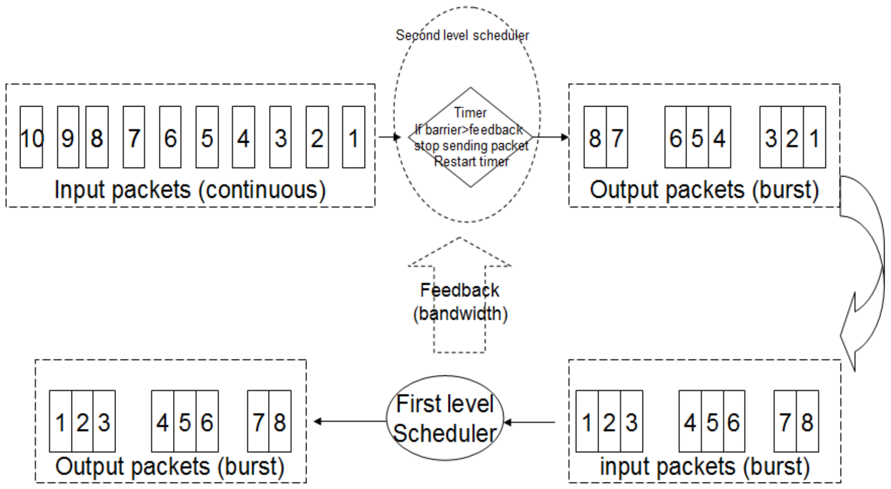

Figure 7. First Improvement Idea

As described in above figure, after second level scheduler sending out the packet, it would get the feedback from first level scheduler; it decides whether to send out the packets or not in real queues. It could fix one scenario that first level scheduler's bandwidth is larger than second scheduler, and then the packet could not be dropped by first level scheduler.

But this method is relatively poor which increases the coupling of first level scheduler and second level scheduler. And the second level scheduler acts based on first level scheduler configuration. If there are several second level schedulers, the behaviors cannot be determined, this is terrible. We need a stable algorithm.

Algorithm two: Reducing the time slice dealers and increasing the execution frequency of the timer.

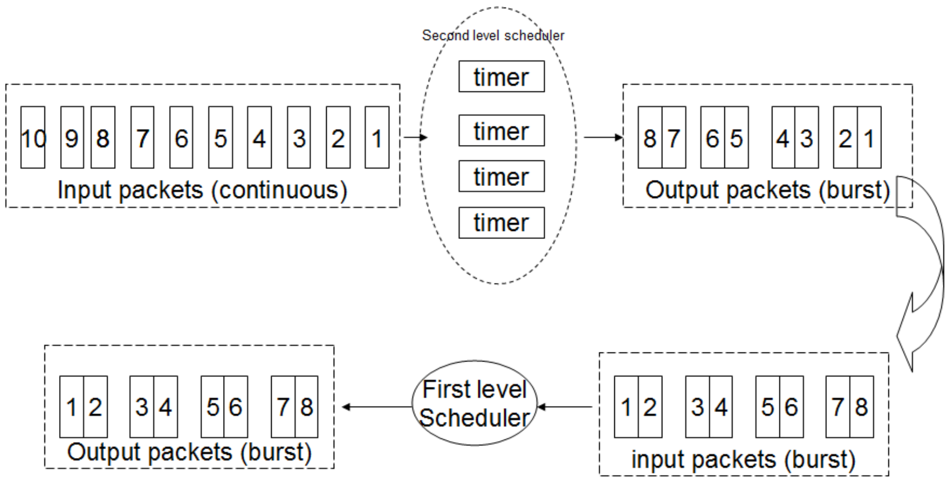

Figure 8. Second Improvement Idea

In above figure(As above figure showed), there are more timers to make the scheduler work, then the packet count in every timer is low. The burst traffic becomes gentle, as the figure is a diagram of the premise, in the event, the timer seems increasing a little. However there are more timer in the real system, maybe 10 times interrupt will come when the timer increases 10 times, which means a significant overload occurred in the system. In the test, it is found that the performance of the system weakened obviously.

Summarily, all these improvements modify the original traffic's mode, so the traffic mode would be changed after the scheduler's operation. As a result, it is hard to fix the issue encountered described in the above.

3. Propose New Improved Algorithm

The root cause in above problems is that there is burst traffic in the second level QoS. The traffic is buffered in the second level QoS. Author tried to use the virtual queue algorithm to solve this problem, but there is not caching the packets in queue.

3.1Basic Ideas

The virtual queue is used in every second level scheduler, which includes control structure for shaping and WFQ. By the usage of the time slice in a public function, each packet can decide whether to discard with virtual queue control structure or not. If it does not drop, the packet will be sent forward directly through the scheduler. In addition, packet process function and the time slice processing function will update the virtual queue structure.

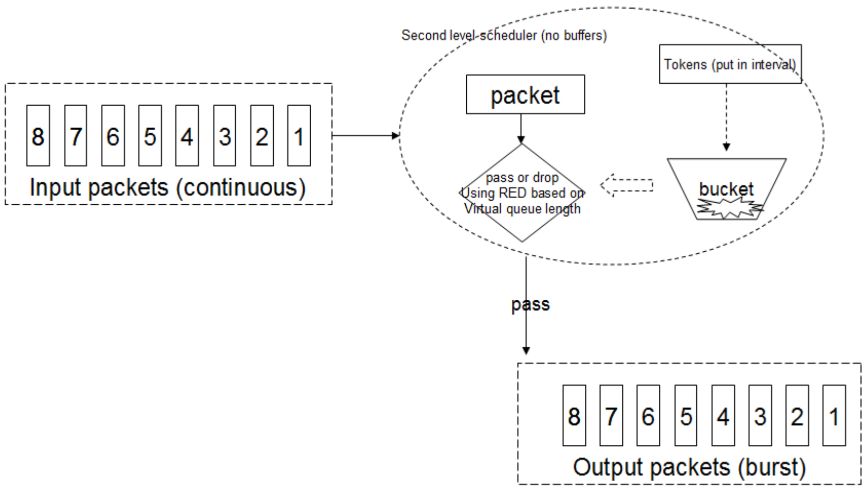

Figure 9. New Improvement Algorithm

Let us use the above figure to describe it in details. There are 6 queues in second level scheduler (BE, AF1~4, EF). Token are put into bucket in every time slice, and then virtual queue length is calculated based on the queue's weight. Every packet arrives at the scheduler, it calculates the virtual queue length, if it does not overflow the bandwidth, then it is sent out, moreover, if there are large packets arriving, no enough tokens in bucket, then the virtual queue length increases quickly and drops to make sure that the packet is not larger than the bandwidth. In one special case, all EF queue packets have strict priority, if the packet consumes all token, then other queue's packets are dropped. It fulfills the real requirement.

3.2Detailed Implementation

Let's watch the detailed implementation for the algorithm.

3.2.1Structure Definition

```
typedef struct vq_t
{
    uint16_t    weight;
    uint32_t    rate_kbps;
    uint64_t    counter;   /*number of bytes that have been sent in current round*/
    uint64_t    q_len;     /*number of bytes that left from previous round*/
    uint32_t    min_th;        /* Min avg length threshold: A scaled */
    uint32_t    max_th;       /* Max avg length threshold: A scaled */
    uint32_t    random_mask; /* Cached random mask*/
    int    count;                    /* Number of packets since last random number
generation */
    uint32_t    random_num;   /* Cached random number */
}vq_t;


typedef struct vq_sched_cb {
    uint32_t      timer_interval;
      timer_t      vq_timer;
    uint32_t      speed_kbps;
    uint64_t      last_sched_time;
    uint64_t      ef_bytes;
    uint64_t      excess_capacity;
    uint32_t      total_weight;
    vq_t          vq[ACTUAL_VQ_NMB];
} vq_sched_cb_t;
```

Data structure definition part:

■vq_sched_cb_t----virtul queue control structure.

■vq_t----every queue in one end user

3.2.2Time Slice Timeout Processing Function

```
void vq_sched_timeout_handler(vq_sched_cb_t *vq_sched_cb_ptr)
{
      uint64_t   current_time;
      uint64_t   free_capacity = 0;
      uint64_t   total_capacity;
      uint64_t   queue_capacity[EITP_ACTUAL_VQ_NMB] = {0};
      uint32_t   active_weights = 0;
      current_time = cvmx_get_cycle_global_il( );
      total_capacity = (current_time - vq_sched_cb_ptr->last_sched_time)*
                              vq_sched_cb_ptr->speed_kbps/g_cpu_clock_khz;
      total_capacity >>= 3; //bits to bytes
      vq_sched_cb_ptr->last_sched_time = current_time;
      if ( vq_sched_cb_ptr->ef_bytes < total_capacity){
            free_capacity = total_capacity - vq_sched_cb_ptr->ef_bytes;
      }
      vq_sched_cb_ptr->ef_bytes = 0;
        for(int i=0; i<EITP_ACTUAL_VQ_NMB; i++){
            vq_sched_cb_ptr->vq[i].q_len += vq_sched_cb_ptr->vq[i].counter;
            vq_sched_cb_ptr->vq[i].counter = 0;
            if ( vq_sched_cb_ptr->vq[i].q_len > 0 ){
                active_weights += vq_sched_cb_ptr->vq[i].weight;
        }
    while( (active_weights > 0) && (free_capacity >= QOS_MIN_FREE_CAPACITY)){
            for(i=0; i<EITP_ACTUAL_VQ_NMB; i++){
                if ( vq_sched_cb_ptr->vq[i].q_len > 0 ){

queue_capacity[i]=free_capacity*vq_sched_cb_ptr->vq[i].weight/active_weights;
                }
            }
            for(i=0; i<EITP_ACTUAL_VQ_NMB; i++){
                if (vq_sched_cb_ptr->vq[i].q_len <= 0 ){
                    continue;
                }
                if (vq_sched_cb_ptr->vq[i].q_len <= queue_capacity[i]){
                    free_capacity -= vq_sched_cb_ptr->vq[i].q_len;
                    active_weights -= vq_sched_cb_ptr->vq[i].weight;
                    vq_sched_cb_ptr->vq[i].q_len = 0;
                }
                else{
                    vq_sched_cb_ptr->vq[i].q_len -= queue_capacity[i];
                    free_capacity -= queue_capacity[i];
                }
            }
        }
vq_start_timer(&g_qos_sched[sched_index].vq_timer,
vq_sched_cb_ptr->timer_interval,vq_sched_timeout_handler,(void*)(unsigned
long)sched_index);
    }
```

Referring above Pseudo code, the total_capacity is the unit of time slice which the traffic forwards, and then calculates the weight of each queue. By adjusting the queue length (q_len), the value q_len will be used for packet loss in the RED function.

3.2.3RED Function

```
    int vq_red_handler(uint8_t q_id, uint16_t q_len, uint16_t pkt_size)    {
        uint64_t que;
        uint32_t pb, pa;
        switch (red_cmp_thresh(p, q_len)) {
```

```
    case RED_BELOW_MIN_THRESH:
        break;
    case RED_BETWEEN_TRESH:
        ++p->count;
        que = (q_len - p->min_th)*RED_REF_RATE/rate;
        pb = red_mark_probability (que/RED_DFT_AVG_SIZE);
        pa = pb*pkt_size/RED_DFT_AVG_SIZE;
        if(pa > RED_MULTI_VALUE) {
            pa = RED_MULTI_VALUE;
        }
          if(p->count > 0 && ((uint32_t)(p->count) >=
            (RED_MULTI_VALUE/2+p->random_num)/pa)){
            p->count = 0;
            p->random_num = red_random(p);
            return RED_PROB_MARK;
        }
        if(0 ==p->count) {
            p->random_num = red_random(p);
        }
        break;
        case RED_ABOVE_MAX_TRESH:
        p->count = -1;
        return RED_HARD_MARK;
    }
    return RED_DONT_MARK;
}
```

The new algorithm fulfills calculation method of classic drop probability:

$$d(avg) = \begin{cases} \dfrac{avg - \min_{th}}{\max_{th} - \min_{th}} \max_p = p_b & if \min_{th} < avg < \max_{th} \\ 0 & if avg < \min_{th} \\ 1 & if avg < \max_{th} \end{cases}$$

Through the equivalent transform as follows, calculation method can reach pb.

$$P_b = \frac{\max_p \times avg}{\max_{th} - \min_{th}} - \frac{\max_p \times \min_p}{\max_{th} - \min_{th}}$$

Packets enter the queue, and the queue length determines whether to drop the packet or not. If the queue length is less than the minimum threshold, packet is directly forwarded. Besides, packets will be discarded when the queue length is greater than the maximum threshold. Moreover, if the queue length is between the minimum and maximum threshold, use the formula to calculate the dropping probability.

3.3 Analysis of the New Algorithm

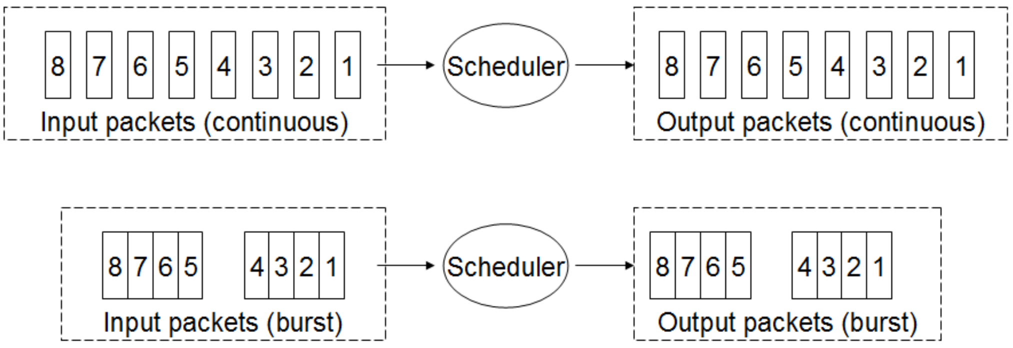

Figure 10. Input and Output of the Algorithm

Referring the above picture, whether the input flow overflows, and the output flow is similar with the input flow, if the input flow is continuous mode, the output flow is continuous mode. Similarly, if the input flow is burst mode, the output flow is burst, we could conclude that the algorithm does not affect the next stage, for example, it would not bring noise to the next stage in hardware design. So we could say that the algorithm can fulfill our requirement.

Why the algorithm can fix the issue? As every packet enters into the scheduler, the packets would re-calculate the algorithm control parameters, it removes the real queue in the algorithm, and there are no packet buffers in the block, so there is no burst traffic generated by the algorithm, this algorithm does not generate the burst packets to the next stages.

4. The Actual Test Results in New Algorithm

In this project researched in the paper, OCTEON processor is used, there are two level QoS scheduler is system, one is for BTS level (second level), it means that every logical BTS use one second QoS scheduler. The other is interface level (first level), all BTSs traffic are restricted by interface. The test bench is charted as below.

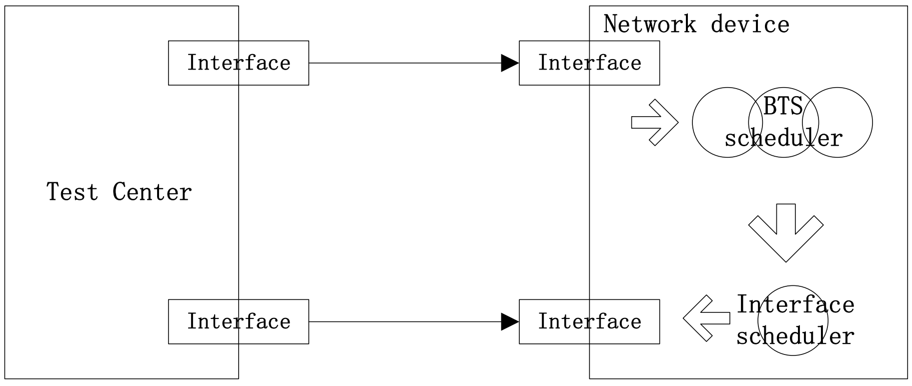

Figure 11.Test Bench

For testing the algorithm, first we set the interface bandwidth to big value to bypass the interface level, just focusing the algorithm performance alone, and then configuring the interface bandwidth to a proper value, watch whether the issue is fixed described in the paper.

4.1Configure one second level scheduler, ingress traffic is 100 times than the shaping bandwidth.

The test condition is that the shaping value is 10mb/s and the ingress traffic is 1Gb/s. As test results showed, a small concussion for shaping in the whole course except 10mb/s in the central area. However, the shaping effect is acceptable.

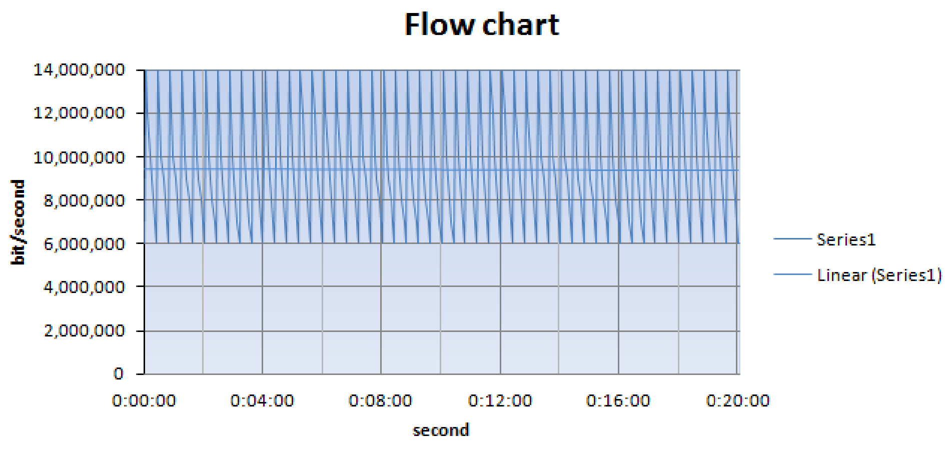

Figure 12. The Test Report with New Algorithm

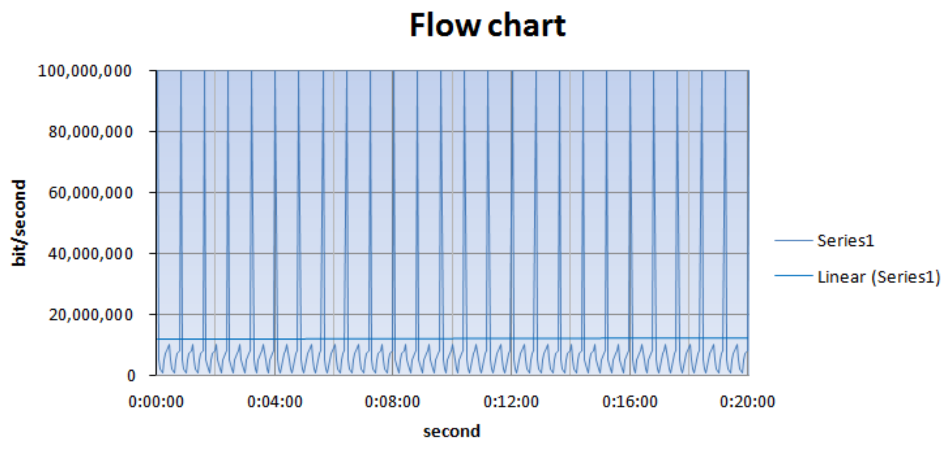

Figure 13. The Test Report with Typical Algorithm

4.2Configure a Second Level Scheduler, Ingress Traffic is 4 Times with the Route Bandwidth

The test condition is that shaping value is 10mb/s and ingress traffic is 40mb/s. From the test results in the following chart, the shaping function is not being a little concussion works already excellently.

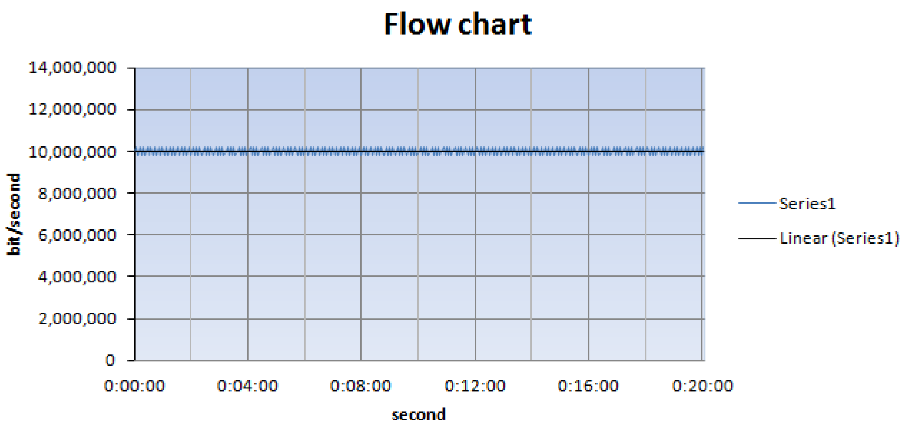

Figure 14. The Test Report with New Algorithm

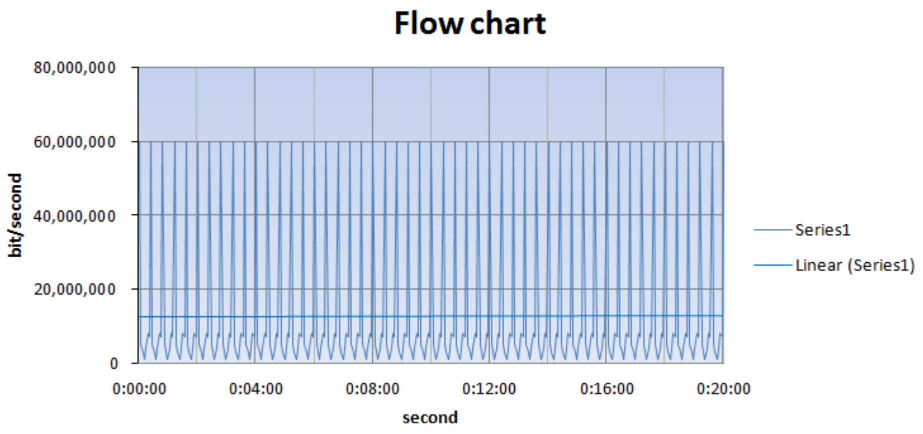

Figure 15. The Test Report with Typical Algorithm

4.3 Configure a Second Level Scheduler; Ingress Traffic is a Little Larger than the Route Bandwidth

The test condition is that shaping value is 10mb/s and ingress traffic is 12mb/s. From the test results in the following chart, the shaping function in virtual queue is as good as typical algorithm (real queue algorithm); they are all very good, it is proved that the virtual queue algorithm is also very stable.

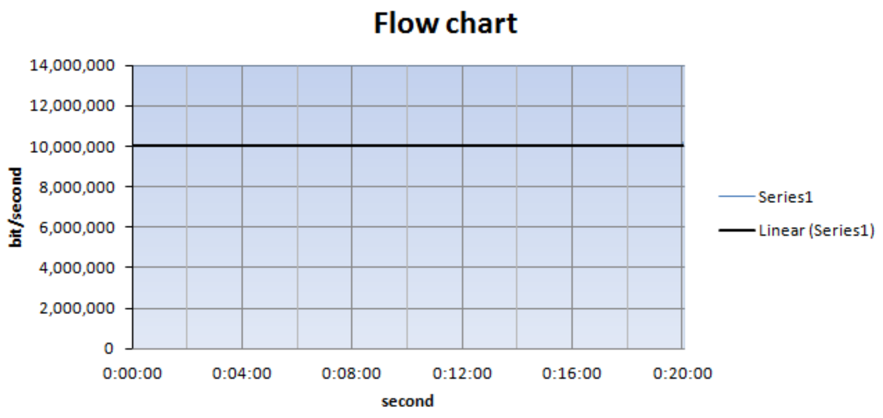

Figure 16. The Test Report with New Algorithm

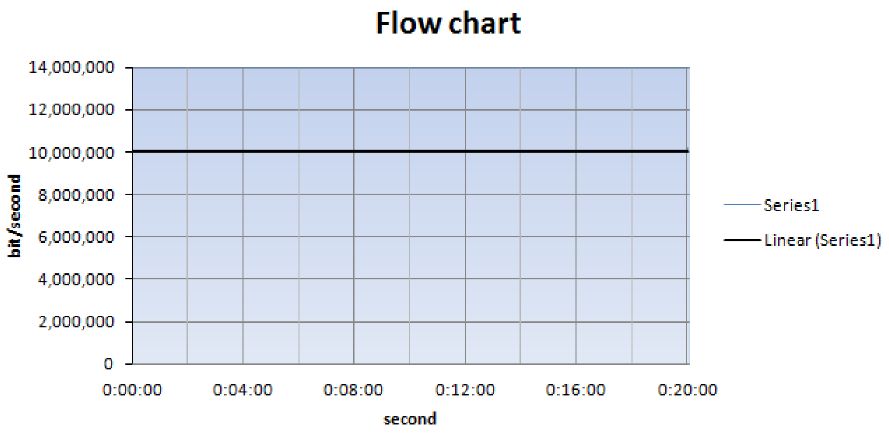

Figure 17. The Test Report with Typical Algorithm

4.4 Configure two second level scheduler

One second level scheduler bandwidth is 10mb/s, another is 6mb/s. the ingress traffic is 40mb/s. In the following figure, it is illustrated that the two flows works well in the second level scheduler and the scenario as well.

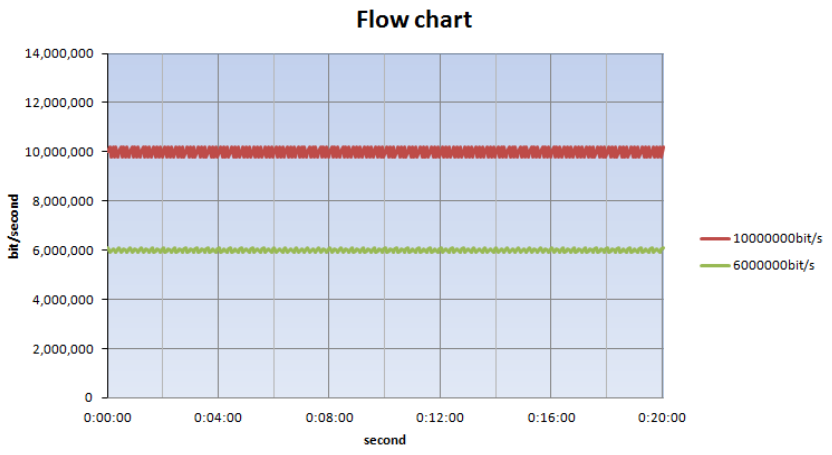

Figure 18. The Test Report with New Algorithm

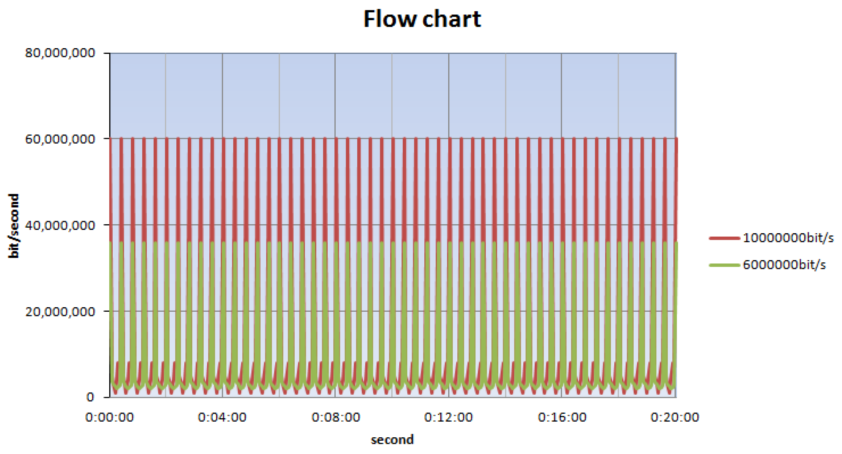

Figure 19 .The Test Report with Typical Algorithm

In the test, it is found that this algorithm can fulfill the requirement of the shaping and priority scheduling. Virtual algorithm, compared with the real queue algorithm, avoids the data cache function and reduces the delay. Additionally, shaping function is improved obviously by refraining from burst traffic produced by the second level scheduler and the impact to the first level scheduler.

5.Conclusion

In conclusion, this new improved algorithm will meet customers' requirement, which solves the problem of traditional algorithm without affecting the system performance. It is believed that this algorithm has very high practical value in engineering. We are looking forward to applying this improved algorithm actually in the future.

References

[1] L. Palopoli and T. Cucinotta, "Quality of service control in soft real-time applications", Proceedings of the IEEE Decision and Control, **(2003)**, pp. 664-669.
[2] L. Abeni, T. Cucin-Otta and G. Lipari, "QoS Management through Adaptive Reservations", Real-Time Systems, **(2005)**, pp. 131-155.
[3] J. Geelan, "Twenty one experts define cloud computing", available: http://cloudcomputing.sys-con. com /read/612375p.htm, **(2008)**.
[4] I. Foster, Y. Zhao and I. Raicu, "Cloud computing and grid computing 360 Degree Compared", Grid Computing Environments Workshop, **(2008)**, pp. 1-10.
[5] T. Andronikos, N. Koziris, "Optimal scheduling for UET-UCT grids into fixed number of processors", The 8th Euromicro Workshop on Parallel and Distributed. Cancun, **(2000)**, pp. 237-243.
[6] D. Zhang, "Training algorithm for neural networks based on distributed parallel calculation", Systems Engineering and Electronics, vol. 32, no. 2, **(2010)**, pp. 386-391.
[7] E. Ekici, I. F. Akyildiz and M. D. Bender, "A distributed routing algorithm for datagram traffic in LEO satellite networks", IEEE/ACM Transaction on Networking, vol . 2, **(2001)**, pp. 137-147.
[8] A. Namatame and N. Ueda, "Pattern classification with Chebyshev neural networks", International Journal of Neural Networks, vol. 3, **(1992)**, pp. 23-31.
[9] L. Fei, X. Naixue and A. V. Vasilakos, "A sustainable heuristic QoS routing algorithm for pervasive multi-layered satellite wireless networks", Wireless Networks, vol. 16, no. 6, **(2010)**, pp. 1657-1673.
[10] P. M. Pardalos, "A genetic algorithm for the weight setting problem in OSPF routing", Journal of Combinational Optimization, vol. 6, **(2002)**, pp. 299-333.
[11] R. Li and H. Wang, "A Novel Approach of Calculating Information Entropy in Information Extraction", International Journal of Database Theory and Application, vol. 6, no. 5, **(2013)**, pp. 45-52.
[12] L. Barolli, H. Sawada and T. Suganuma, "A new QoS routing approach for multimedia applications based on genetic algorithm", IEEE CW, vol. 6, **(2002)**, pp. 289-295.
[13] D. Mou, G. P. Biswas and B. Chandan, "Optimization of multiple objectives and topological design of data network using genetic algorithm", Guangzhou, RAIT, **(2012)**.
[14] S. Tseng, C. Lim and Y. Huang, "Ant colony- based algorithm for constructing broadcasting tree with degree and delay constraints", Expert Systems with Applications, vol. 35, no. 3, **(2008)**, pp. 1473-1481.
[15] L. Wang, W. Liu and H.Shi, "Delay-constrained multicast routing using the noisy chaotic neural networks", IEEE Transactions on Computers, vol. 58, no. 1, **(2009)**, pp. 82-89.

Author

Jian Zhang, he was born in Hangzhou of China in 1978, he obtained a master's degree in Zhejiang University software engineering, he now works at Zhejiang Business College, He has published more than 10 papers, and his main research direction is the application of software engineering in the network.