A Scalable Per-flow Priority Scheduling Scheme for High-Speed Network

Conference Paper · April 2010

DOI: 10.1109/ICCSN.2010.17

li Guodong
2 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Zhen Chen
Tsinghua University
138 PUBLICATIONS **727** CITATIONS
SEE PROFILE

Yibo Xue
Tsinghua University
104 PUBLICATIONS **695** CITATIONS
SEE PROFILE

Jun Li
Tsinghua University
172 PUBLICATIONS **736** CITATIONS
SEE PROFILE

A Scalable Per-flow Priority Scheduling Scheme for High-Speed Network

Guodong Li[1, 2], Zhen chen[1, 2], Anan Luo[1], Yibo Xue[2, 3], Jun Li[2,3] and Chuang Lin[1]
[1]*Dept. Computer Science and Technology, Tsinghua University, Beijing, China*
[2]*Research Institute of Information Technology, Tsinghua University, Beijing, China*
[3]*Tsinghua National Lab for Information Science and Technology, Beijing, China*
guodongli07@gmail.com, laa@mails.tsinghua.edu.cn
{zhenchen, yiboxue, junl,chlin}@tsinghua.edu.cn

Abstract—**In order to provide the service differentiation for various network applications, and guarantee delay and bandwidth requirement, packet scheduling is considered as a hot research topic and a crucial module in network device. In high speed network, it is hard to maintain and schedule a great number of queues for millions of in-progress flows in memory in line speed. In this paper, we propose a scalable per-flow scheduling scheme using a small fast memory to achieve fine-grained service guarantee. A limited number of queues are dynamically shared among concurrent flows based on the interesting fact that the number of simultaneous active flows is only in hundreds whatever the link speed is. The scheduling scheme is in a scalable hierarchical manner, in which the first layer supplies service differentiation and the second guarantees bandwidth and delay. We also implement an instance based on this scheme called DQS-SPQ-DRR (Dynamic Queue Sharing-Strict Priority Queue-Deficit Round Robin). Experiments based on real and synthetic traces are conducted to evaluate the DQS-SPQ-DRR. The results demonstrate that DQS-SPQ-DRR is well held in small memory and supplies per-flow service guarantee.**

Keywords: scheduling,scalable,per-flow

I. INTRODUCTION

Packet scheduling is the key technique to guarantee the service for critical applications. To design a good scheduling scheme, several issues should be addressed: 1) how to distribute bandwidth to each flow on demand; 2) how to guarantee the quality of service for critical applications; 3) how to deploy the scheduling schemes easily.

Using a dedicated queue for each flow and a constant-time scheduler (such as DRR) [1] can provide bandwidth and delay guarantees for service. Unfortunately the number of in-progress flows can be extremely large. With traffic evolution, the number of flows is also growing (in millions scale) and possibly exceeds router's capacity. Caching states for millions of flows is a big challenge in high speed network. If the states are stored in SRAM, the memory cost is too expensive; if they are kept in DRAM, the state lookup and update are too slow. Therefore, how to organize the queues in SRAM and schedule them in different priorities is a significant but unsettled issue.

A flow is a stream of packets that are identifiable using fields in a packet header (such as TCP/IP's five-tuple). Packets of each flow have the same route from the source to the destination and require the same grade of service.

Many researchers [2-4] focus on dealing flows at a time scale of seconds or minutes. However, a packet is buffered in high speed device only for several microseconds mostly, so active flows (flows are not empty currently) should be handled at a microsecond time scale. The new discovery is that the number of concurrent flows is only in hundreds in 10 microsecond scale [5]. Based on this observation, it is possible to use hundreds of queues and share them among active flows [6]. It is very easy to store such a number of queues in SRAM. Therefore a data structure called active flow list (AFL) is designed to store active flows. When the first packet of a flow arrives, an empty queue is allocated and a new entry is inserted to AFL; when the queue becomes empty, the relative entry in AFL is deleted and the queue is freed and can be reassigned to another newly incoming flow.

In this paper, a novel scalable per-flow scheduling scheme is proposed. AFL is used to store active flows in a small memory. The scheduler is hierarchically organized for enqueue and dequeue operation. The first layer provides service differentiation by distributing flows into different priority groups. The second provides delay and bandwidths guarantee for each flow which can prevent one flow from occupying too much resource. It is a scalable scheme and each layer can use existing scheduling algorithms to achieve service differentiation and guarantee. Our results show that all memory required by this scheme is small and well held in fast memory; service of critical flows is better guaranteed than original solution.

The rest of the paper is organized as follows. Section II introduces the related work. Section III presents the scalable hierarchical scheduling scheme. Section IV gives a named DQS-SPQ-DRR implementation. Section V discusses the experimental results. Section VI concludes the paper and future works.

II. RELATED WORK

Packet scheduling has been studied extensively and many scheduling algorithms and architectures are given.

IntServ[7] is the pioneer of scheduling architecture. It reserves resource for all flows in-progress flows. It can achieve well per-flow service guarantees. However, it has to maintain all state information for all flows in its route. Its sophisticated implementation is not feasible to the huge number of flows. So it is not widely used in Internet.

A.Nikologiannis et al. [8] and Aggelos Ioannou et al.[9] introduce special hardware to implement thousands of queues for per-flow queuing for providing advanced service guarantee respectively. It always adopts ASIC for queues organization and scheduling. However, in these methods, it not only takes high cost and long-term developing cycle, but also does not scale up with the increasing of the network.

S.Floyd et al. [10] presents CBQ (Class-Based Queueing), which has been implemented in many Linux distributions [11]. Hierarchical link-sharing is introduced to allow multiple agencies, protocol families, or traffic types to share the bandwidth on a link in a controlled fashion. Link sharing is organized in tree and each node represents one share such as agencies or policy. On the basis of CBQ, H-PFQ (Hierarchical Packet Fair Queueing) [12] and HFSC (Hierarchical Fair Service Curve) [13] are introduced to achieve fair queueing. However, the architectures above only deal with the flow aggregation. To get fine-grained service guarantee, they have to deepen the hierarchical tree, which will increase the scheduling cost a lot.

A.Kortebi et al. [5] firstly discovers the interesting fact that the number of flows in progress is in millions and increases with the link speed, but the number of active flows is only in hundreds even though there may be tens of thousands of flows in progress currently. Based on this discovery, it is a good opportunity to design a new scheduling scheme. However this paper only presents a simple fair queue schedule scheme which can't provide sufficient service guarantee.

III. THE DESIGN OF PACKET SCHEDULING SCHEME

A. *Design principles*

The objective of our scheme is as follows: 1) to provide per-flow queueing in SRAM; 2) to provide delay and bandwidth guarantee for critical flows; 3) to be scalable to existing scheduling algorithms. To achieve these, we combine AFL, priority group and bandwidth guarantee algorithm together.

In order to providing per-flow queueing, queue is organized in AFL. With AFL storing the active flows in a linked-list or hash table, only active flows are dynamically allocated queues. When a packet with priority from configured policy arrives, a look-up action in AFL is triggered. If the look-up returns unsuccessful result, the queue manager applies a new empty queue from the free queue stack. Then the flow and scheduling information of this packet is composed and a new entry is inserted to AFL; if successful, queue manager updates the scheduling information and inserts the packet to the corresponding queue. With this method, active flows are dynamically mapped to finite physical queues, so only a small number of physical queues are needed.

Another novelty of our scheduling scheme is that it is organized in hierarchical manner. In order to provide service differentiation, priority group is the first layer. The priority of flows is configured by policy. Users assign different priority to flows with different characteristic. When a packet arrives, it will look up the policy and get its priority. How to achieve this is beyond the scope of this paper. Usually, higher priority group would get more chance to send packets. Various current scheduling algorithms that provide service differentiation can be employed here. For example, with strict priority queue scheduling algorithm, only when the higher priority group becomes empty, the lower one gets an opportunity to dequeue packets. To prevent low priority groups from starving, it can also implement weighted DRR (WDRR) scheduling among different groups. With WDRR, each group is offered services proportional to its assigned weight. It also maintains current credits for scheduling. Each scheduling round, the current credits is incremented by its quantum. Then this group is served as long as current credits are greater than zero. In this way, each group can send packets no more than its current credits, so lower priority group can get a fraction of services.

The candidate scheduling algorithms should treat flows fairly and prevent a small number flows from occupying too many resources in each priority group. DRR is a suitable for this use, because it is easy to implement by hardware or software and widely deployed in commercial devices. It is an O (1) fair scheduling scheme for delay and bandwidths guarantee. Its dequeue processing is in a round robin manner. During each round, packets no more than its current credits are sent to prevent one flow from occupying too many resource.

All in all, AFL maintains a small number of mappings between active flows and physical queues, while priority group differentiates services and DRR provides delay and bandwidth guarantee. Existing popular scheduling algorithms can also substitute the methods mentioned here flexibly.

B. *Scheduling algorithms*

To simplify the description, we adopt the strict priority group, i.e., the low priority group is not scheduled until all the packets in high priority ones have been sent out. In each group, we employ DRR algorithms. The scheduling information in AFL includes the flows' packet number PktsNumi, current credit DCi, quantum Qi. Priority group should maintain the AFL entry pointer fi to get DRR scheduling info, current packet number PktsNum, and lastDequeueRound. Fig.1 and Fig. 2 present the pseudo-codes of enqueue and dequeue algorithms.

Enqueue ALG

```
On arrival of packet $p$;
$i$ = ExtractFlow($p$);
$pri$ = GetPriorityFlow($p$);
Search $f_i$->$q_k$ in AFL;
If null
    Assign a new queue $q_k$;
    $i$->PktsNum=1;
    $i$->Quantum = $Q_i$
    $i$->DC = 0;
    InsertAFL($f_i$,$q_k$);
Else
    $i$->PktsNum++;
End If
Insert($p$,$q_k$);
$pri$->PktsNum++;
If($pri$->packet==1)
    Priority[$pri$]=1;
End If
```

Figure 1. Enqueue Module.

Enqueue Operation: On the arrival of a packet p with its priority, it firstly gets the priority in order to find priority group (line 3). If packet p does not belong to any active flow, it apply qk from the free management stack and set PksNum to 1 and initial the DRR quantum and DC, then the new entry inserts to the tail of AFL(lines 5-10). Otherwise it simply adds the PktsNum (lines 12). After searching AFL, insert packet p to relative queue qk in priority group pri(lines 14-15). If it is the first packet in this group, to set the bit in bitmap Priority to 1(lines 16-17).

Dequeue Operation: The dequeue operation firstly gets the highest priority group K that is not empty (line2), and then extracts the packet p at the head of lastDeqRound queue (line 3). Adding a Quantum to the DC of the flow, if DC is larger than the p's length, to send p out and decrease DC by the packet size (line 6-7). The loop allows the flow to emit up to DC byte. If the packet in this flow becomes empty, AFL deletes the entry and breaks the loop (lines 5-15). After the loop, lastDequeRound in group K updates (line 16). At last if the packet in group K becomes zero, the K-th bit in bitmap Priority is set to 0.

```
     Dequeue ALG
1    While(1)
2        Find the first not empty priority group K,;
3        p=Headof(K->lastDeqRound,&fi);
4        fi->DC += fi->Quantum;
5        While(fi->DC > p->length)
6            Dequeue(K->lastDequeueRound);
7            fi->DC -= pac->length;
8            K->PksNum--;
9            fi->pkts--;
10           If(fi->pkts ==0)
11               Delete the fi entry in AFL;
12               break;
13           End If
15       End While
16       K->lastDequeuRound++;
17       if(K->PksNum ==0)
18           Priority[K]=0;
19       End if
20   End While
```

Figure 2. Dequeue Module.

IV. SCHEME IMPLEMENTATION

We have implemented the above scheduling scheme by integrating Dynamic Queue Sharing (DQS) [6], strict Priority Queue and enqueue-time DRR [16], which is called DQS-SPQ-DRR. Please note that, the implemented scheduler works in a per-flow manner, and all the control information can be fit in SRAM.

A. DQS

To speed up the searching and updating on the active flow list, hash is utilized to divide the whole AFL into multi sub-list. In this case, the AFL operation is executed in small sub-list. Flows with the same hash value are directed to the same sub-list. The detail of DQS is in [6].

B. Strict priority queue

Strict priority queue is the easiest scheme to achieve service differentiation. Each flow gets its priority from configured policy, so when it arrives in AFL, the assigned queue for the flow inserts to relative priority group to which it belongs. Since SPQ is from high to low priority, dequeuue scheduling may check for packets for several DRR rounds in each empty group. Avoid polling empty group is a good idea to enhance performance. To solve the problem, one bitmap indicating which group currently has packets is maintained. Instead of polling all groups, it simply read the bitmap and search for bits that are set in this bitmap. Especially many modern processors families (such as IXP, CAVIUM) support the instruction FFS (Find First Set) [14] [15]. The instruction searches one bitmap for the location of the first bit that is set. The result is the location (bit position) of that first set bit. With the solution, the group selection processing only searches a bit vector to determine the current state efficiently.

C. Enqueue-time DRR

In current design, network processor is usually multi-core architecture, so it is good at processing packets in parallel. For example, there can be several processes to send packets to different queues. DRR dequeue processing, however, does not have the same characteristics. Each round of DRR needs to run sequentially. It does not allow for one queue to start the second round until all other queues have finished the first round. So dequeue processing has to be always in a single thread or process.

In this condition, with more than one core performing enqueue and only one core doing dequeue, the unbalance results in the need to simplify dequeue process. We introduce the enqueue-time DRR which sorts the packets into DRR rounds at enqueue-time. So it moves the calculating scheduling info expense from deque-time to enqueue-time. The detail of enqueue-time DRR is in [16].

D. Complexity analysis

The memory required for DQS-SPQ-DRR is quite small. Let's denote each entry as flowing:

DQS: it only stores the mapping of queue and active flows. Here, the number of hash slots is W; the number of physical of queue is N; the number of active flows is L; the number of bits to identify a flow is B; the number of bits for DC and Quantum is Q, for PktsNum and BytesNum are both C. Then the memory requirement for the mapping scheme when sub-tables are organized in double linked-list.

$$M_{DQS} = W + L\,(2\log L + \log N + B + 2Q + 2C) \qquad (1)$$

SPQ: It only needs maintain the priority bitmap and some scheduling information. The layer of priority group is P; the number of bits for PktsNum is C too; the max DequeRound is D; the number of bits for CurEntry is I.

$$M_{PQ} = P + P(C + \log D + I) \qquad (2)$$

Enqueue-time DRR: For each round, it should maintain the head and tail of pointer for each dequeue round. As denoted above, the memory required is:

$$M_{DRR} = 2PDI \qquad (3)$$

So the total memory need is

$$M = M_{DQS} + M_{PQ} + M_{DRR} \quad (4)$$

For example given M=512 slots, N=512 queues, active flow L= 512, B=32, Q =32, C=32.P=64, D = 64, I=32. The memory required only about 363Kbit and can be implemented in SRAM easily.

The time cost of this scheme includes enqueue time and dequeue time. Search, insert and delete AFL accounts a big portion of enqueue time. However, it is also a piece of cake. From thesis [6], we can get:

$$T_{search} = 1 + (L-1)/2M \quad (5)$$

$$T_{insert} = L/M \quad (6)$$

$$T_{delete} = 1 \quad (7)$$

The enqueue time cost of priority queue and enqueue-time DRR are both O (1).The total enqueue time cost is this, i.e.

$$T_{enqueue} = MAX\ (T_{search}, T_{insert}) + T_{PQ} + T_{DRR} \quad (8)$$

Dequeue procedure is easy. The scheduler goes through per-round linked-list sequentially and sends out packets from each round linked-list. If the queue becomes empty, the relative entry in AFL is deleted. The time cost is O (1), i.e.

$$T_{dequeue} = T_{delete} + T_{PQ} + T_{DRR} \quad (9)$$

Here we also set M=512 slots and active flow L= 512 as memory calculation above. The enqueue time is 2.5 and dequeue time is 1.

V. PERFORMANCE EVALUATION

We evaluate the performance of DQS-SPQ-DRR by means of synthetic traffic and Internet trace data.

A. *Original trace statistics*

We use three real traces from [17]:

a) NLANR1: It was collected on November 24st, 2002 at the University of Leipzig Internet OC3 access link.

b) NLANR2: It was collected on August 14st, 2002 at the OC48 link from IPLS Abilene router towards CLEV.

c) NLANR3: It was collected on June 1st, 2004 at the OC192 link from IPLS Abilene router node towards KSCY.

Here we set the statistic time scale to be 10 milliseconds. Output bandwidth is shared among all the active flow and the output capacities are set to make the traffic load of each trace to be 0.95, as shown in Table 1. The load is defined as the ratio of the average rate and the output bandwidth. From the table, we get that average rate of each trace is far less than its original output capacity.

TABLE I. PACKET TRACE STATISTICS SUMMARY.

Trace	**NLANR1**	**NLANR2**	**NLANR3**
average trace rate	14.4Mbps	372Mbps	557Mbps
original output capacity	155Mbps	2.5Gbps	10Gbps
regulate output capacity	15.2Mbps	392Mbps	586Mbps
packets number	10M	20M	100M
flows in progress	53K	75K	100K
MTU	1500byte	1537bytes	9000bytes

The complementary distribution of AFL size is shown in Fig. 3, which indicates the number of active flows. We observe that it is only in the number of hundred (256 in the worst case), much less than the number of flows in progress.

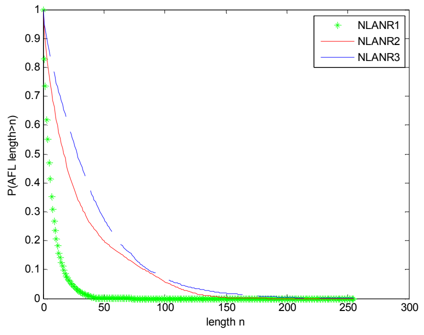
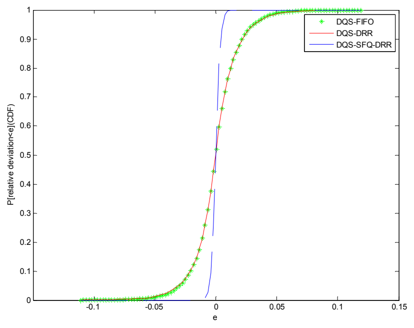
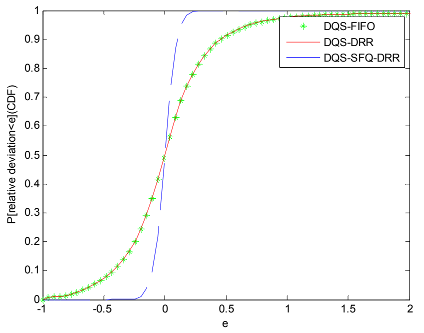


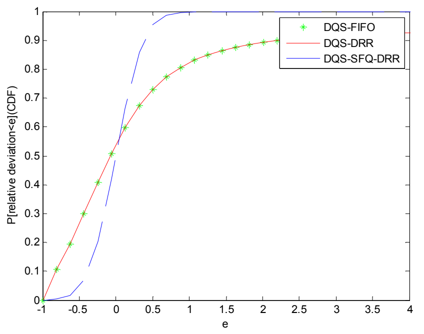
Figure 3. Complementary distribution of AFL size.



(a)relative deviation of 1Mbps Flow



(b)relative deviation of 32Mbps Flow



(c)relative deviation of 128Mbps Flow

Figure 4. CDF of relative deviation for flows of various rate

B. *Performance results*

The following experiment results about the three trace data are similar. In order to simplify the description, NLANR3 with the highest speed is selected to show the performance. We add six constant rate flows to the NLANR3. The rates of the flows are 1Mbps, 8, 16, 32, 64 and 128 Mbps and the packet is 1024 bytes. As a result, to get traffic load as 0.95, the link capacity is regulated to 848Mbps. To guarantee delay and bandwidth of these 6 flows, trace flows are set lower priority than them.

To verify the performance of DQS-SPQ-DRR, we compare it with DQS-FIFO that each packet is scheduled as its enqueue sequence and DQS-DRR that packets are scheduled in round robin manner without service differentiation.

In the following, we focus on bandwidth and delay of these six flows. The expected delay of a flow is defined as the average incoming interval among packets, i.e.

Expected = packet length/flow rate ***(10)***

To compare the result of these three algorithms, we define "relative deviation" to show the deviation between experiment and expected result.

Relative deviation= (experiment – expected)/expected (11)

Here we only present the comparisons of 1Mbps, 32Mbps and 128Mbps flows. The same thing happens for the other three flows.

Fig. 4 shows the cumulative distribution of relative deviation. We can get that DQS-SPQ-DRR performs the best among the three schemes no mater of the protected flow rate. DQS-FIFO and DQS-DRR perform almost the same, because both of them don't treat flows different and flows are not protected finely. Another fact is that smaller rate flows can get better service guarantee for all the algorithms. For example, about 90% of the interval is exact to expected interval with 1Mbps flow in DQS-SFQ-DRR. If the flow rate goes to 128Mbps, the relative deviation is more dramatic. However, more than 95% of the interval falls in the rage [-0.5, 0.5] for DQS-SFQ-DRR.

VI. Conclusion

We propose a novel scalable per-flow scheduling scheme which use a small fast memory to achieve fine-grained service guarantee. The queue and scheduler's data structure can be all stored in SRAM. It is a per-flow queuing scheme by only maintaining dynamic queue for active flows in active flow list (AFL). The scheduler is organized as a hierarchical manner, in which the first layer providing service differentiations and the second does the service guarantee.

The advantages of this architecture lies in 1) it only maintains a small number queues to achieve per-flow queuing; 2) it is a scalable hierarchical architecture and compatible to existing scheduler algorithms; 3) it can use SRAM to achieve the best performance for high speed network.

An instance implementation called DQS-SPQ-DRR is presented to evaluate the performance. Trace-driven experiment shows that under DQS-SPQ-DRR, the AFL length is still in the number of hundreds. The guaranteed flow acquires its service quarto no matter of the variation of the other background traffic.

Acknowledgment

This work has been supported by the National High-Tech R&D Program (863 Program) of China under grant No.2007AA01Z468. The authors would like to thanks Yaxuan Qi, Baohua Yang and other colleagues in the Network Security Lab of Tsinghua University(NSLAB) [18] for their help. We also thank Doctor Chengchen Hu for his valuable suggestion

References

[1] M.Shreedhar, G.Varghese. Efficient Fair Queuing using Deficit Round-Robin. IEEE/ACM Trans. Networking, pp. 375-385, 1996.

[2] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell,T. Seely, and S. C. Diot. Packet-level traffic measurements from the sprint IP backbone. IEEE Network, vol. 17, no. 6, pp. 6–16, 2003.

[3] Nan Hua, Bill Lin, Jun Xu, Haiquan Zhao. BRICK: A Novel Exact Active Statistics Counter Architecture. ACM ANCS 2008.

[4] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, A. Kabbani. Counter Braids: A Novel Counter Architecture for Per-Flow Measurement. In Proceeding of SIGMETRICS, 2008.

[5] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing. ACM SIGMETRICS 2005, pp. 217–228, 2005.

[6] Chengchen Hu, Yi Tang, Xuefei Chen, Bin Liu. Per-flow Queuing by Dynamic Queue Sharing. IEEE INFOCOM 2007, May 2007, pp.1613-1621.

[7] R. Braden, D. Clark, S. Shenker. Integrated services in the internet architecture: an overview. RFC1633, 1994.

[8] A. Nikologiannis and M. Katevenis. Efficient per-flow queueing in DRAM at OC-192 line rate using out-of-order execution techniques. IEEE International Conference of Communications (ICC 2001), Helsinki, Finland. pp. 2048-2052, Jun. 2001,

[9] Aggelos Ioannou, Manolis G. H. Katevenis. Pipelined heap (priority queue) management for advanced schedule-ing in high-speed networks. IEEE/ACM Transactions on Networking (TON) Volume 15, Issue 2, pp 450-461, April 2007.

[10] S.Floyd, V. Jacobson. Link-sharing and resource management models for packet networks. ACM/IEEE Transaction Networking, 1995.

[11] Main page of CBQ. www.icir.org/floyd/cbq.html

[12] Jon C R Bennett, Hui Zhang. Hierarchical packet fair queueing algorithms. IEEE/ACM Transactions on Networking (TON) Volume 5, Issue 5, pp: 675 - 689 , October 1997.

[13] Ion Stoica, Hui Zhang, T.S.Eugene Ng, A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. IEEE/ACM Trans on Networking (TON), 2000,8(2),pp 185-199.

[14] Official website of Cavium about OCTEON product family. www.cavium.com/OCTEON_MIPS64.html

[15] Erik J.JJohnson, Aaron R.Kunze. IXP2400/2800 Programming. INTEL PRESS, 2004.

[16] Uday R.Naik, Prashant R.chandra. Designing High-Performance Networking Applications. INTEL PRESS, 2004.

[17] NLANR. Passive measurement and analysis (pma). [Online]. Available: http://pma.nlanr.net

[18] NSLAB. Main page of Network Security Lab.