This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

Two-stage Fair Queuing Using Budget Round-Robin

Dong Lin and Mounir Hamdi
Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
ldcse@ust.hk, hamdi@cse.ust.hk

***Abstract*—In current high bandwidth-delay-product networks, traditional end-to-end network protocols cannot guarantee the fair allocation of network resources (i.e., a rogue source that sends at an uncontrolled rate can seize a large fraction of the buffers at an intermediate router which results in dropped packets for other connections). Fair-queuing (FQ) algorithms were proposed to overcome this drawback. However, most of these FQ algorithms either suffer from high time-complexity or greatly rely on the multiple queuing structures which are extremely difficult to implement in large scale due to the access delay of DRAM. Based on the analysis on real-life traces, we are able to determine the short-term stability of number of connections in a trunk. Taking this characteristic into consideration, a new FQ algorithm called Budget Round-Robin (BRR) is proposed in this paper. Both theoretical analysis and experimental results demonstrate that BRR and its corresponding memory hierarchy are much superior to the other FQ algorithms when we have a high bandwidth link with large number of active connections (e.g., high-speed Internet).**

Keywords-Fair-queuing; memory hierachy.

I. INTRODUCTION

A plethora of end-to-end protocols involving both scheduling and congestion avoidance algorithms have been proposed to achieve quality of service in the Internet. However, most of these protocols are either not practical for real implementation or perform poorly especially in current high bandwidth-delay-product networks. For example, when using conventional TCP protocols, a rogue source that sends at an uncontrolled rate can seize a large fraction of the buffers at an intermediate router which results in dropped packets for other connections. To solve the problems with these traditional end-to-end protocols, some researchers developed the so-called fair queuing (FQ) algorithms [1-8] which ensure that the resources are allocated or scheduled fairly by involving queue management into this process.

Unfortunately, these FQ algorithms suffer from at least one of the following drawbacks: high time-complexity, flow aggregation and lack of scalability. Demers *et al.* devised an algorithm called bit-by-bit round-robin (BR) so that each connection sends one bit at a time in a round-robin fashion, which in turn provides a perfect fairness [1]. For ease of implementation, they suggested a method for approximately simulating BR which still requires $O(\log(n))$ time for each packet, where n is the number of connections[2]. In order to further reduce this time-complexity, Stochastic Fair Queuing (SFQ) was introduced in [3]. In SFQ, all the incoming connections will be assigned to their corresponding buckets according to an $O(1)$ time hash computation. All the connections that happen to hash into the same bucket are treated equivalently, and all the buckets are served in round-robin. Thus, when the hash index is not sufficiently large, i.e. less than five or ten times the number of active connections, hash collisions happen and the system exhibits unfair behavior [3]. By taking packet length into further consideration, Shreedhar *et al.* proposed an algorithm called Deficit Round-Robin (DRR) which is based on quantum round-robin [7]. Its only difference from the traditional round-robin is that if a queue was not able to send a packet in the previous round because its packet size was too large, the remainder from the previous quantum is added to the quantum for the next round. Thus, deficits are kept track off; the unfairness resulted from variable packet length is compensated.

Instead of attempting to achieve a better fair scheduling, some other algorithms such as Flow Random Early Drop (FRED) [9] adopted the per-flow dropping mechanisms which have been proven to be quite effective when the congestion avoidance schemes are involved, but extremely vulnerable to the other unresponsive protocols like UDP. In such a case, extra penalty schemes must be developed [10].

As the extensions of above, some researchers took more design factors into account. For example, the method of favoring delay-sensitive services, the unfairness between long-term and short-term TCP flows caused by the retransmission timeout and etc. In [11], Rai *et al.* proposed a size-based scheduling which improved the performance of short TCP flows with limited penalizing long flows.

Despite tangible advances, all the proposed FQ algorithms expose their limitation especially when high bandwidth capacity buffer design is concerned. The majority of FQ algorithms, like SFQ, DRR and FRED, are based on a buffering architecture with a large storage capacity, dynamical queuing, and huge number of queues which is crucial to the reduction of hash collision. Such requirements are even more complicated for FRED where packets are allowed to be dropped randomly. This is highly difficult in practice. The Core-Stateless Fair Queuing proposed in [12] has reduced the cost of state maintenance but still inherits most of the drawbacks of FRED.

This work was supported in part by a grant from Research Grants Council (RGC) under contract HKUST610307.

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

Since the last decade, high bandwidth and large storage capacity DRAM has become cheaper and more popular, but the memory access time remains almost the same. Considering a buffer where packets arrive and depart at rate R, the memory must has a total bandwidth of $2R$ to accommodate continuous reads and writes. But DRAM has a random access time of T_{RC}, which is the maximum time to write to, or read from any memory location. In practice, this random access time of DRAMs is much higher than that required, i.e, $T_{RC} >> (1/2R)$. Therefore, packets are written to DRAM in batches of size $b=2RT_{RC}$ every T_{RC}, in order to achieve a bandwidth of $2R$. In [13], the authors proposed a memory hierarchy called Nemo to tackle this difficulty. By setting up both the tail and head caches for each logical queue, a scheduling algorithm changes the scattered DRAM accesses into bulk operations which in turn increases the efficiency. Assuming a buffer for FQ which supports Q queues, the Nemo system requires at least $Q(2b-1)$ of SRAM for both tail and head caches provided that the buffers are dynamically allocated and $Q(b-1)+1$ cycles' delay is tolerable. To achieve a shorter delay, the buffers must be statically allocated where the cache size is increased to $2Qb(3+\ln Q)$. According to the N^2-hypothesis, the number of connections is approximately proportional to the square root of bit rate [14]. In the scenario of kR-rate parallel DRAM structures, both Q and b tend to be $k^{1/2}$ times and k times respectively as large as the original size. This will increase the SRAM size exponentially, making the caches impractical for on-chip implementation. In [16], the authors proposed a packet buffer based on interleaved DRAMs where random scheduling is executed for both ingress and egress, making it unsuitable for fairness queuing. As further improvement, Feng *et al.* devised a deterministic algorithm [17] which maximized the performance of interleaved DRAMs. But it doubles the cache size comparing with Nemo and suffers high time complexity O(Q) for each packet in the process of finding the maximal matching.

Given parallel DRAMs, we strive to minimize the overhead of entire system. There are some major concerns in our design:

1) An $O(1)$ FQ algorithm is needed here to guarantee uniform distribution of network resources among numerous connections.

2) A new memory hierarchy fitted the proposed FQ algorithm is required for ease of implementation, especially in the scenario of large number of queues. In addition, the adopted memory management scheme should avoid the shortcomings of dynamic queuing while ensuring high memory utilization.

The subject and contribution of this paper mainly focuses on the above two important and challenging issues. Our proposed algorithm is not only simple for implementation, but also cost-effective. The rest of the paper is organized as follows. In Section II, we propose a FQ algorithm which evenly distributes the queuing resources among millions of connections. In Section III, we present our scheme which dynamically adjusts the quantum among the connections to maximize the memory utilization. Our simulation results are demonstrated as well. Finally, Section IV concludes the paper.

Since the blocks have been organized in circular way, the actual block number can be derived by using MOD. For example, in a system with only 10 blocks, block-number 11 means 1 actually. Unless noted otherwise, we omit this in the following description.

Variables Definition:

AC: Current number of active connections;
//Quantum per unit of block; (*QU*)
QU=((*AC*==0)?*DRAMBlockSize*:(*DRAMBlockSize*/*AC*));
//Maximal accumulated quantum; (*MAQ*)
MAQ= ((*QU*> *MaxAllowedPacketSize*)? *QU*: *MaxAllowedPacketSize*);
INPUTPointer: The first available non-full DRAM block;
OUTPUTPointer: The latest output DRAM block;
For the *i*-th connection: CQ_i: Current quantum; LR_i: Last paid round; TB_i: The DRAM block number where the tail packet is located; TS_i: The total size of data which is being buffered by DRAM;

Function Definition:

GetConnectionID(*p*): Get the connection ID of packet *p*;
GetPacketSize(*p*): Return the size of packet *p*;
Ceil(X):Expression to compute the ceil of X.
UpdataAC(*p*,*i*){ //Update the number of active connections;
  if(enqueuing) **then** TS_i+= GetPacketSize(*p*);
  else if (dequeuing) **then** TS_i-= GetPacketSize(*p*);
  if($TS_i \geq THRESHOLD$) **then** *AC*++;
  else if ($TS_i < THRESHOLD$) **then** *AC*--;}
//Save the packet *p* to the *blocknum*-th DRAM block;
SaveToBlock(*p*, *blocknum*);

Initialization:

AC=0; INPUTPointer=0; OUTPUTPointer=0;
Set (CQ_i, LR_i, TB_i, TS_i) to zeros for all *i*.

Enqueuing:

Define NewArrivalPacket as *p*; *i*= GetConnectionID(*p*);
UpdataAC(*p*,*i*);
CQ_i = ((CQ_i +(TB_i - LR_i)* *QU*)>*MAQ*)? *MAQ*: (CQ_i +(TB_i - LR_i)* *QU*);
LR_i = TB_i;
if($CQ_i \geq$ GetPacketSize(*p*)) **then**
  blocknum= (TB_i> INPUTPointer)? TB_i: INPUTPointer;
else *blocknum*= TB_i+Ceil((GetPacketSize(*p*)-CQ_i)/*QU*);
SaveToBlock(*p*, *blocknum*); TB_i= *blocknum*;
if the INPUTPointer-th block is full, **then**
  update INPUTPointer to the next non-full block in ascending order;

Dequeuing:

Fetch one packet *p* from the OUTPUTPointer-*th* block; *i*= GetConnectionID(*p*); UpdataAC(*p*,*i*);
if (OUTPUTPointer== (INPUTPointer-1)) **then**
  update INPUTPointer to the next non-full block in ascending order;
if (the OUTPUTPointer-*th* block is empty) **then** OUTPUTPointer++;

Dequeuing Reshape:

Exert SFQ upon the output packets in block-level.

Figure 1. Pseudo-code of BRR

II. BLOCK-BASED FQ (BRR)

In order to overcome the drawbacks mentioned above, we propose a two-stage FQ algorithm called "Budget Round Robin" (BRR) in this paper. Our scheme works in two steps. First, a high-bandwidth, high-storage buffer is designed using multiple DRAMs. Instead of interleaving, we adopt parallelism which minimizes the pin numbers and greatly reduces the packing cost. We further divide the entire storage space into separated blocks, each block is of Megabytes in size and these blocks are organized as circular linked list. Second, newly arrived packets are pushed into these blocks under the control of BRR while the outputs are popped from DRAMs continuously. For the convenience of clear description, we discuss the processes of enqueuing and dequeuing separately. Fig. 1 is the pseudo-code of BRR.

In the first setup, BRR tries to store the incoming packets in their output sequences. Since we cannot afford an $O(log(n))$ sorting, BRR just simulates this process. By keeping track of the buffered packets, the BRR records the number of active connections and then estimates the storage quota for each active connection. When a new packet arrives, the target block number can be calculated accordingly by assuming that the number of active connections remains stable for a while. Thus, when the storage quota for the current block is depleted, the available space of the next block can be allocated in advance. In order to maintain the accuracy of such estimation, BRR omits those trivial connections by setting up a threshold. Only those connections with buffered data size larger than *THRESHOLD* are counted as active. Furthermore, BRR also

prevents any connection from gathering a superfluous quantum. Maximal accumulated quantum has been strictly limited, so that the algorithm can correct the inaccuracy as soon as the active connections are updated. This updating happens at intervals of every block dispatching when *QU* is more than *MaxAllowedPacketSize*. When *QU* is so small that each connection has to wait several blocks for the serving opportunity, the updating still occurs every *MaxAllowedPacketSize* at least. The fairness may still be compromised when the number of active connections fluctuates dramatically.

Our analysis on real-life data provided by [15] has dispelled this concern. At first sight of this per-second statistical data illustrated in Fig.2, it seems to confirm the dramatic fluctuation of active connections. But when the statistic interval is reduced to ms-level (the typical rate of block dispatching, i.e. *MB*/10*Gbps*), the curve has to be more smooth. The reason is obvious. Thinking about how many new connections can be generated within just 1 *ms* and also the huge number of active connections backlogged in the router, one can easily figure out that active connections in real-life is relatively stable. Hence, the unfairness of BRR is negligible. For example, a sudden appearance of thousands of connections in a core router which maintains millions of active connections leads to only 0.1% perturbation.

As the second key benefit of our design, BRR is an $O(1)$ complexity algorithm. All the defined variables are updated only once for each enqueuing or dequeuing a packet. On the other hand, in order to speed up the time-consuming block list operations, such as finding the next non-full block, we keep an auxiliary list called *NonfullBlockList* that is a linked list of non-full DRAM blocks. Whenever a block is evacuated by the output, it is added to the *NonfullBlockList*. Whenever a block becomes full, it is removed from the *NonfullBlockList*. In this fashion, examining of full DRAM blocks is avoided. Since there is only limited number of blocks, a full-index can be easily built to ensure the $O(1)$ complexity of such operation. For example, for a 4*GB* buffer consisting of 1*MB*-size blocks, a full-index costs only 6*KB*.

Since all the packets have been organized in output order block by block, BRR only introduces one additional full duplex SFQ module to obtain a fine grained output. Whenever the SFQ module contains data less than two DRAM blocks, it will be replenished by the packets stored in DRAMs. Thus, the SFQ module maintains only small size of data, leading to an even small number of connections. Therefore, a tiny scale of dynamic queuing is sufficient. For example, for a 1 *MB* DRAM block and 1*KB* packet sizes, a 2*K* dynamical queues are more than enough.

We now introduce our new memory hierarchy as shown in Fig.3. Compared with Nemo, the major differences are as follows: 1) the per-queue based head cache is replaced by a block-based head cache which doesn't rely on the number of connections; 2) the tail cache is abolished. Instead, a SFQ module is introduced and costs only two blocks' storage; and 3) parallel DRAMs adopt static-allocating strategy which minimizes the overhead. Dynamic allocation can be certainly used here for more flexible implementation, such as adjustable

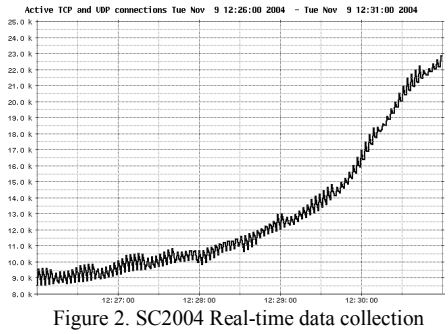


Figure 2. SC2004 Real-time data collection

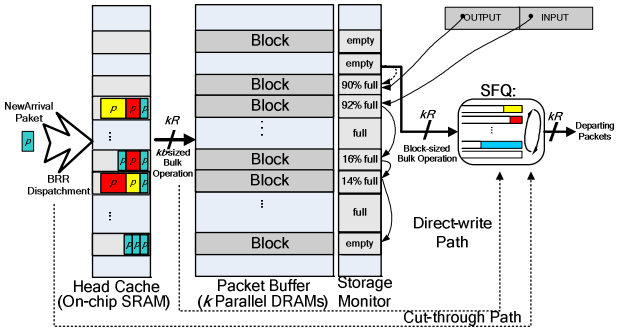


FIGURE 3. BRR IMPLEMENTATION

TABLE I. COSTS COMPARISONS

	BRR	**DRR+Nemo**
Buffer Allocation	Static in DRAM, Dynamic in SRAM	Dynamic
Head cache	*mkb*	$Q(kb\text{-}1)$
Tail cache	$2s+kb$	Qkb

* Q denotes the maximum of coexist queues; k is the number of DRAM chips; kb indicates the equivalent size of data stream measured in DRAM access delay; m equals to the total block number of DRAMs; s is the block size of DRAM.

block size. In the next section, we shall explore the pros and cons of these two schemes as well as their improvements. Besides the differences mentioned above, similar to Nemo, we maintain both direct-write and cut-through paths to achieve a shorter delay in case that system is lightly loaded.

Table I presents a simple comparison of implementation cost between BRR and DRR+Nemo. In order to build a 100*Gbps* rate system which supports 1 million queues and 4*GB* storage capacity by using DRAM chips with 50*ns* access time, BRR requires only 9*MB* SRAM when 1*MB* block-size is chosen. In contrast, the Nemo demands more than 2.5*GB* SRAM which is approximately 280 times larger than BRR.

III. SIMULATIONS AND QUANTUM ADJUSTMENT

A. *Default simulation settings*

Our experimental results are presented in this section. Unless otherwise specified, the default for all the experiments is as specified here. We measure the throughput in terms of delivered bits in a simulation interval defined by *Ts*, typically the time required to generate 1*MB* of data stream. In other words, we record the statistical result every 1*MB* (10243*Bytes*). As a result, our experimental environments are not bounded with specific link rate, but always in back-to-back mode. To show how BRR performs with respect to First Come First Served (FCFS), we buffered the incoming

traffic for 1000*Ts* before a full-speed output. After that, the system inputs and outputs simultaneously, so that dynamic behavior can be monitored.

During the experiments, the normal-behaved connections generate the packets at a Poisson average rate of *Rp*, while those ill-behaved connections send packets at 3*Rp*. The packet sizes are randomly selected between 40*Bytes* and 1500*Bytes*. These two kinds of connections occupy the entire bandwidth. Regarding the memory hierarchy, the default DRAM block size is 1*MB* and there are infinite blocks. In other words, the Drop-tail scheme doesn't apply. In contrast, we keep track of the memory utilization for a detailed analysis. We further assume that the scale of dynamical queuing inside SFQ modules is sufficient in guaranteeing negligible hash collisions.

B. Experiment Alpha

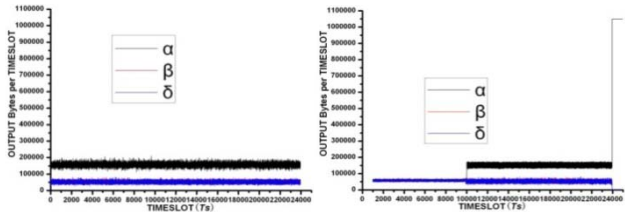
We first experiment with the system behavior under a stable condition. As shown in Fig. 4(a), 18 connections have been generated and last for 24000 *Ts* in total. α is an ill-behaved connection which takes up 15% of the total bandwidth. β and δ are selected as the representatives of the remaining normal-behaved connections. As can be seen from Fig.4(b), through BRR, all the output of these connections have been reshaped. Before 10000 *Ts*, when the backlogged packets of normal-behaved connections aren't drained out, all the connections share the bandwidth equally. After that, the system performs the same as an FCFS, because only α is backlogged. α doesn't occupy the entire bandwidth until the last 1000 *Ts*, when all the inputs have been ceased.

C. Experiment Beta

We have demonstrated that BRR operates in a very good way in the face of stable condition. What's about the performance when the system confronts a highly unstable environment?

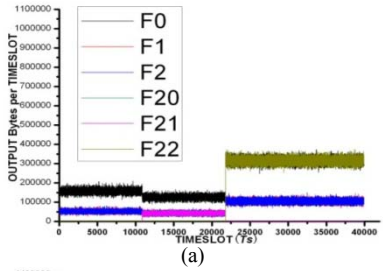
In order to test that, we generate an input as shown in Fig.5(a). The inputs have been divided into three stages. In the first stage, there are 18 connections in total. Only one of them is ill-behaved as indicated by *F0*, while *F1* and *F2* are the representatives of the other 17 normal-behaved connections. In the second stage, five more normal-behaved connections participate, leading to a total number of 23 active connections. *F20* and *F21* are two of these new arrival connections. As for the last stage, there is a tremendous change that all these five new connections plus another 13 normal-behaved connections are gone, leaving only one ill-behaved and four normal-behaved connections. To make this experiment more comprehensive, we further introduce one more ill-behaved connection (*F22*) in this stage. Thus, the final configurations for each stage can be denoted as 1+17, 1+22 and 2+4. ("The number of ill-behaved connections" + "The number of normal-behaved connections").

As shown in Fig.5(b), the output of BRR is still quite good. In the beginning, all the connections share the bandwidth equally. After 10000 *Ts*, when all the backlogged packets of those normal-behaved connections have been released, an

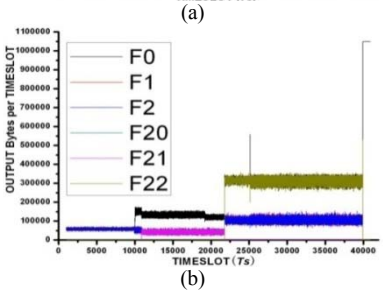


(a) (b)

Figure 4. Exp. Alpha: BRR and FCFS comparison



(a)



(b)

Figure 5. Exp. Beta: BRR and FCFS comparison

FCFS mode takes over. When the second stage starts, all the 22 normal-behaved connections are served fairly. Since BRR reserves storage for *F0* in advance and this operation cannot be undone, the sudden increase of active connections results in a clear stepped output. But as the input becomes stable, the ideal result is achieved after 19000 *Ts*. Meanwhile, the sudden decreasing of active connections in the third stage doesn't cause any fluctuation, only a short burst around 25000*Ts* is detected. Recall the previous analysis, this dramatic change of active connections is high unlikely in real-life situations, especially in current high bandwidth-delay product networks where core routers usually backlog millions of active connections (measured in *RTT*).

D. Memory Utilization and Quantum Adjustment

Since the introduced memory hierarchy adopts the static-allocation, memory utilization has become another major concern, especially for an algorithm like BRR that not only allocates the bandwidth but also reserves the storage for all active connections. Fig.6 shows the memory utilization (denoted by *MU*) of each DRAM blocks for both experiments. As the figure depicts, without sufficient time for the buildup of buffer, the memory utilization may decline greatly when the system backlogs some ill-behaved connections. The more aggressive those ill-behaved connections are, the lower the memory utilization is achieved. Although this phenomenon only leads to the early drop of ill-behaved connections, the vacant storage is not necessarily to be utilized by the other moderate connections. Therefore, it leads to relatively low memory utilization.

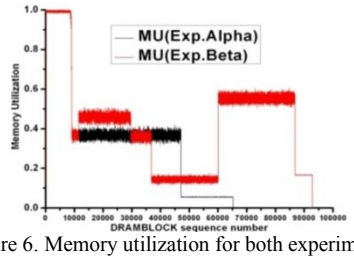
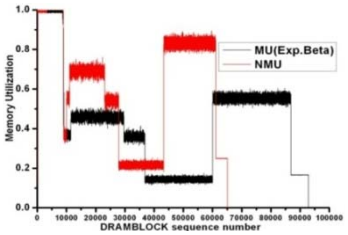
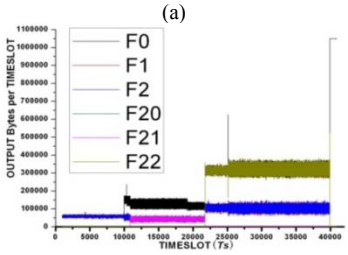


Figure 6. Memory utilization for both experiments



(a)



(b)

Figure 7. Demonstration of quantum adjustment

Here we discuss the corresponding solutions which increase memory utilization. Dynamic buffer allocation can be a straightforward idea. It allows a buffer to be shared among blocks, leading to a higher efficiency. Since we defined the DRAM block by the size of Megabytes, thousands of dynamic queues could support the addressing of Gigabytes' DRAM buffer. The major drawback of such scheme is the extra cost of maintaining the link lists.

We adopt the static buffer allocation and quantum adjustment to remedy this defect. It increases the memory utilization by changing the quantum-estimation scheme of BRR. Theoretically speaking, we can merge two lightly loaded DRAM blocks into one block, so that one block can be spared. Without dynamic allocation, this merging is very difficult in practice. Therefore, we amend the quantum of BRR instead of adjusting the DRAM blocks. If partial quantum is certainly to be squandered due to less input than that expected, we can increase the quantum evenly among active connections. As no connection is allowed to accumulate unused quantum, the quantum wasted in this round is doomed to fade away. As long as all the distributed quantum can be "fulfilled" when they are really needed, the real size of a block is irrelevant. (If a packet cannot be stored to the target block as it is anticipated, due to the insufficient storage of this block, then it's not fulfilled.)

Fig.7 presents a simple demonstration where quantum adjustment has been introduced. The experimental configurations are exactly the same as experiment beta. After 20000 Ts, the quantum of BRR is increased by 50%. Fig.7(a) presents the new memory utilization data (depicted as NMU). Comparing with MU, the new scheme costs almost 30000 blocks less than the naïve one, while the corresponding output changes little as shown in Fig.7(b).

IV. CONCLUSION

In this paper, we have proposed architecture and the associated algorithms to solve the key drawbacks of conventional fair queuing algorithms that is: high time-complexity, flow aggregation and lack of scalability. Based on the analysis on real-life traces, we are able to determine the short-term stability of active connections. Taking advantage of this characteristic, we propose a new FQ algorithm called BRR and also its corresponding memory hierarchy. They are extremely easy to implement in large scale. In the scenario of high bandwidth and large number of active connections, BRR was shown to be much superior to related algorithms. In addition, the dynamic quantum adjustment we devised for our architecture is of very high practical value that greatly increases the memory utilization.

ACKNOWLEDGEMENT

The authors would like to express their appreciations to the reviewers for their suggestions on this paper.

REFERENCES

[1] A. Demers, S. Keshav, and S.Shenker, "Analysis and simulation of a fair queueing algorithms," in Proc. SIGCOMM'89, vol.19, no. 4, Sept. 1989, pp. 1-12.

[2] A.Greenberg and N. Madras, "How fair is fair queueing?", in Proc. Performance' 90, 1990.

[3] P. McKenney, "Stochastic fairness queueing," in Internetworking: Research and Experience, vol. 2, Jan. 1991, pp. 113-131.

[4] L.Zhang, "Virtual clock: A new traffic control algorithm for packet switched networks," in ACM Trans. Comput. Syst. Vol.0, no.2, pp. 101-125, May 1991.

[5] S. Golestani, "A self clocked fair queueing scheme for broadband applications," in Proc. IEEE INFOCOMM'94, 1994.

[6] S. Floyd and V.Jacobson, "Link-sharing and resource management models for packet networks," in IEEE/ACM Trans. Networking, August 1995.

[7] M.Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in Proc. SIGCOMM'95, Boston, MA, August 1995.

[8] J.C.R. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," in Proc. IEEE INFOCOM'96, pages 120-128, San Francisco, CA, March 1996.

[9] D. Lin and R. Morris, "Dynamics of random early detection", in Proceedings of ACM SIGCOMM '97, pages 127-137,Cannes, France, October 1997.

[10] S. Floyd and K. Fall, "Router mechanisms to support end-to-end congestion control", LBL Technical Report, February 1997.

[11] I. A. Rai, E. W. Biersack, and G. Urvoy-Keller, "Size-based scheduling to improve the performance of short TCP flows," in IEEE Network, January/February 2005.

[12] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair banwidth allocations in high speed nteworks", in ACM SIGCOMM Computer Communication Review, vol. 28, Issue 4, October 1998.

[13] Sundar Iyer, Ramana Kompella, Nick McKeown, "Designing packet buffers for router linecards", in IEEE Transactions on Networking, vol.16, Issue 3, Jun. 2008.

[14] B. St. Arnaud, "Scaling issues on Internet networks," 2001, http://www.canet3.net/library/papers/scaling.pdf

[15] NLANR's PMA Project, 2004, http://pma.nlanr.net/Special/sc04rt.html

[16] Gireesh Shrimali and Nick McKeown, "Building Packet Buffers with Interleaved Memories", HPSR'05, Hong Kong, May 2005.

[17] Feng Wang and Mounir Hamdi, "Scalable Router Memory Architecture Based on Interleaved DRAM", HPSR'06, Poznan, Poland, June 2006.