Information Technology Journal 13 (3): 434-442, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.434.442

Fair and Efficient Variable-length Packet Scheduling and Multilink Transmission Support

Guikai Liu
School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, 411201, China

Abstract: Aiming to guarantee scheduling fairness in variable-length packet networks, this study presents a novel easily implementable scheduling algorithm, called Resilient Quantum Round-Robin (RQRR). The quantum assigned to each of the flows in a round is not fixed and is calculated depending on the transmission situation of all the flows in the previous round. The computing method is: For one flow, p-value (namely quantum) of the next round = p-value of the current round+the average count of bytes sent by other flows in current round-the number of bytes sent by the flow in current round. Dynamic quanta can instantly reflect the behavior of the flows in scheduling process. This study proves that the implementation complexity of RQRR is O (1) with respect to the number of flows; it also analytically prove the fairness properties of RQRR and show that its relative fairness measure has an upper bound of 7 Max-1, where Max is the size of the largest packets. On the other hand, multilink transmission is an efficient way to solve the problem that satisfies the subscriber's demand on bandwidth increment. RQRR can support multilink transmission commendably and it not only allocates the bandwidth resource of multilink fairly to keep load-balance amongst links, but also guarantees accordant packet sequence between sending end and receiving end without increasing additional overhead.

Key words: Packet scheduling, variable-length packet, relative fairness, implementation complexity, multilink transmission, high-speed networks

INTRODUCTION

In high-speed packet networks, all the switching or routing points want to output the arrived packets by utilizing packet scheduling algorithms. These algorithms can be categorized into two main classes: Time-stamp based scheduling and round-robin based scheduling. Time-stamp based algorithms such as WF^{2}Q (Worst-case Fair Weighted Fair Queuing) (Bennett and Zhang, 1996) and SCFQ (Self-clocked fair Queuing) (Golestani, 1994) maintain two time-stamps for every packet to indicate its start-serving time and end-serving time, respectively then sort their time and send out the packet with the least end-serving time. This kind of algorithms is to approximately emulate the most ideal packet scheduling algorithm-GPS (Generalized Processor Sharing) (Parekh and Gallager, 1993) by maintaining virtual clocks and achieves good fairness and low latency. However, their time complexity is at least O (logN) (N is the number of active flows) due to computing and sorting each packet's time. They are not suitable for using in high-speed networks, thus round-robin based packet scheduling algorithms with lower complexity are at a premium. In round-robin based scheduling algorithms such as RR (Round-Robin), WRR (Weighted Round-Robin) (Chaskar and Madhow, 2003) and DRR (Deficit Round-Robin) (Shreedhar and Varghese, 1996) etc., the scheduler simply serves all non-empty queues in a round-robin manner. These algorithms neither maintain a time-stamp for every service flow, nor perform sorting among the packets. Most of them have O (1) complexity with respect to the number of flows. But in a variable-length packet environment, round-robin based scheduling algorithms must consider packet length to guarantee scheduling fairness. All in all, a good scheduling algorithm should have the characteristics of fairness, efficiency and low complexity.

DRR can support variable-length packet scheduling with O (1) time complexity and it gains widespread attention (Sivakumar and Ramprasad, 2012; Ayaz *et al*., 2011; Sleem *et al*., 2011; Mansy *et al*., 2011). DRR assigns a quantum for each flow to indicate the number of bytes that the flow can transmit in a round. At the same time, a Deficit Counter (DC) is also set for each flow to store the residual quantum that the flow does not use up in a service opportunity, and through which the flow can bring the residual quantum into the next service. In this way during a relatively long time, the service that a flow receives is directly proportional to its quantum on the whole. It means that DRR is fair and suited to be used in variable-length packet networks. However, DRR also has some shortages: Firstly, it just provides long time scale fairness and it may induce big output burst from short time scale; furthermore, its fairness is related to the minimum quantum and the minimum quantum must be equal or greater than Max in order to assure DRR has O (1) complexity. Secondly, its characteristic of latency is not good and probably some queues cannot receive service for a long time. In addition, DRR wants to know the length of packets before sending them and overhead is increased. Aiming at the above-mentioned shortages, many literatures put forward improvement to DRR. Kanher and Sethu (2001) proposed nested-DRR to adopt fine grain scheduling policy and minish the upper bound of packet latency. LL_DRR (Xiaodong and Lemin, 2002) presented a new algorithm suitable for variable-length packet scheduling based on DRR and had prominent improvement in latency performance. Jian *et al*. (2005) put forward a new scheduling discipline named Prioritized Nested DRR (PNDRR), which introduced a token bucket with virtual allocated token quantum and delay of packet in latency critical queue is effectively diminished. Fei *et al*. (2011) overcome the limitation that DRR cannot output smoothly by adding traffic shaper on the basis of network calculus in nodes and improved network QoS. I-MDRR (Kumar and Priyameenal, 2011) is proposed for broadband wireless access and it can support QoS requirements of real-time services as well as provide higher throughput. The slicing domain scheduling viewpoint is proposed (Bo *et al*., 2012), which adopts Smoothed Deficit Round-Robin (SDRR) scheduling algorithm in carrying group and Longest Queue First (LQF) scheduling algorithm in scheduling domain; Scheduling complexity and delay are decreased. M-DRR (Shuqing and Jinye, 2012) schedules wireless packets by taking advantage of channel conditions and HARQ technology; it improves the throughput and QoS guarantee in IEEE 802.16e system. Aiming at redundant transmission and handling burst flows in avionic networks, D2DRR (Hua and Liu, 2012) properly distributes traffic between switches and end systems, and unifies the representation of heterogeneous flows; it implements fast deduplication and the network load is reduced. In these mentioned algorithms, improvements to DRR made up its some shortages, but their basis is still DRR and fixed quantum is assigned for data flows. In this study, a new variable-length packet scheduling algorithm different from DRR is presented, which is called Resilient Quantum Round-Robin (RQRR). The distinct characteristic is that the quantum of each queue changes dynamically and the permission given to each of the flows in a given round is not fixed and is computed depending on the behavior of the flows in the previous round. During the course of its execution, flows which receive very little service in a round are given an opportunity to receive proportionately more service in the next round. Thereupon, good scheduling fairness among flows can be guaranteed. Besides, as an important application of RQRR, it can support multilink transmission efficiently.

RESILIENT QUANTUM ROUND-ROBIN

A pseudo-code implementation of the RQRR scheduling algorithm is shown in Fig. 1, consisting of Initialization, Enqueuing module and Dequeuing module. The Initialization is to initialize packet scheduling module when network node has demand on scheduling packets. Enqueuing module is called

```
Initialization//
        ActiveFlowList=NULL;
        VisitFlowCount=0;

Enqueuing Module: on arrival of a packet
   i = Queueln WhichPacketArrives;
   if (ExistInActiveFlowList(i)==FALSE) then
      AddToAcitveFlowList (i);
      Increment SizeOfActiveFlowList;
      $P_i$=0;
      $Send_i$=0;
end if;

Dequeuing module:
while (TURE) do
   If (ActiveFlowListEmpty==FALSE) then
      VisitFlowCount=SizeOfActiveFlowList;
   else
      waiting for arrival of a packet;
end if;
while (VisitFlowCount>0) do
   i = HeadOf ActiveFlowList;
   Remove Head of Active Flow List
      do
      TransmitPacketFromQueue (i);
      $Send_i$= $Send_i$+LengthOf TransmittedPacket
      while (QueueIsEmpty==FALSR)&&($P_i$-$Send_i$>0);
      if; (QueueIsEmpty==FALSE) then
      Add QueueToActiveFlowList (i);
else
   Decrement SizeOfActiveFlowList;
   end if;
  VisitFlowCount= VisitFlowCount-1;
end while;
for (i=1; i<sizaOfActiveFlowList+1; i=i+1)
   $P_i$=$P_i$+$Ac_i$=$Send_i$;
for (i=1; i<SizeOfActiveFlowList+1; i=i+1)
   $Send_i$=0;

end while;
```

Fig. 1: Pseudo-code for RQRR (Resilient Quantum Round-Robin) scheduling

whenever a new packet arrives at a flow. The packet will enter into the correlative queue and wait for being sent. Dequeuing module is the heart of the algorithm which schedules packets from the queues corresponding to different flows.

A data flow is defined as active if its queue is not empty or its packets are being scheduling. All the active flows are put into a list and this list is called "ActiveFlowList". When a flow changes from active to inactive, it will be removed from ActiveFlowList. When a flow changes from inactive to active, it will be appended at the end of ActiveFlowList.

A round is defined as the process during which the data flows, included in ActiveFlowList at a time instant T1 (T1>0), are accessed by packet scheduling module. The newcome flows or those become active once again can be appended at the end of ActiveFlowList, but they will be accessed in the next round. For example, consider the instant of time, T1, when the scheduler is first initialized. Suppose round 1 is one round-robin iteration starting at time T1 and consisting of visits to all the flows that were in the ActiveFlowList at time T1. Assume that flow-1, flow-2 and flow-3 are the only flows active at the beginning of round 1. The visits of the scheduler to the flow-1, flow-2 and flow-3, comprise round 1. Let flow-4 become active after the time instant T1, but before the completion of round 1. Let the time instant T2 mark the completion of round 1. The scheduler does not visit flow-4 in round 1 since flow-4 was not in the ActiveFlowList at the start of round 1. Round 2 will visit all of the flows that are in the ActiveFlowList at time T2. Assuming that flows-1, flow-2 and flow-3 are still active at timeT2 and round 2 consists of visits to flow-1, flow-2, flow-3 and flow-4. Therefore, the sequence number of rounds is relative to one flow, i.e., round 2 is the second round for flow-1, flow-2, flow-3, but the first round for flow-4.

In order to express the number of data flows wanted to be accessed in a round, RQRR introduces a counter, which is called "VisitFlowCount", to record the number of flows. At the beginning of a round, VisitFlowCount equals the number of flows in ActiveFlowList. When packet scheduling module finishes accessing a flow, the value of VisitFlowCount will be minus 1. At last, when VisitFlowCount equals 0, it means that the current round is over.

In each round, the scheduling algorithm determines the number of bytes that a flow is permitted to send and calls this quantity the p-value for the flow during that round. Suppose there are n (n is a natural number and n>0) flows sharing one output link. The p-value assigned to flow i (i is a natural number, $1\le i\le n$) during round r (r is a natural number, r>0) is denoted by P_i (r). p-value is not fixed and is recomputed in the following rounds except its initial value is 0. The computing method is: For one flow, p-value of the next round = p-value of the current round+the Average Count (AC) of bytes sent by other flows in current round-the number of bytes sent by the flow in current round.

Another counter is used to record the number of bytes, called "ByteNumberCount". Let Send i (r) be the number of bytes that are transmitted from the queue of flow i in round r. Its initial value is 0 before a round starts and the number of bytes of sent packets will be accumulated to it once a packet is sent.

Average count (AC): For flow-i, AC expresses that, starting from the r-th ($r\ge 2$) round, the average number of bytes sent by all the other flows in the previous round, denoted by AC_i (r-1). If there are n flows in the round r-1, then:

$$AC_i\ (r-1) = \frac{Send_1\ (r-1) +...+ Send_{i-1}\ (r-1)+Send_{i+1}\ (r-1)+.....+Send_n\ (r-1)}{(n-1)} \tag{1}$$

If it is not divided exactly, the result equals the integer part of quotient+1.

Whereupon, computing the p-value of a flow as: when:

$$r = 1,\ P_i\ (1) = 0$$

when,

$$r>1,\ \ P_i\ (r) = P_i\ (r-1)+AC_i\ (r-1)-Send_i\ (r-1) \tag{2}$$

Hereinto i is a natural number and $1\le i\le n$. The meaning of p-value is: The quantity of data that a flow is permitted to send out in a round not only relates to its own p-value, but also depends on the quantity sent by other active flows.

Figure 2 illustrates the first four rounds in an execution of the RQRR algorithm. In this figure, there are

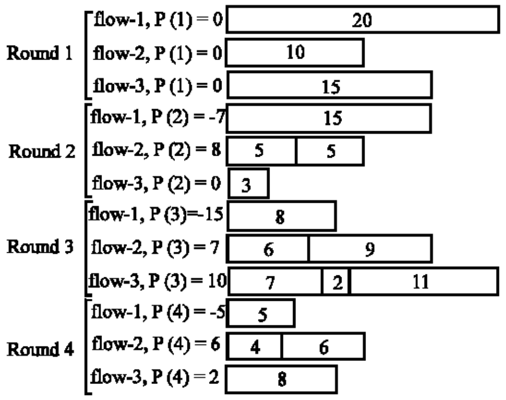

Fig. 2: An illustration of 4 rounds in a RQRR (Resilient Quantum Round-Robin) execution

three flows: Flow-1, flow-2 and flow-3. The sizes of the packets actually sent by the flow during this round are shown by the horizontal bars and the new p-values for the next round are again computed using (1) and (2). One packet is sent at least when p-value = 0 in a round. Although only four rounds are listed, there is no limitation in the quantity of flows and the number of rounds for RQRR, which will continue scheduling packets once data flows become active. Also, it is observed from Fig. 2 that, in general, flows which receive very little service in one round will be given more opportunity to receive proportionate service in the next round. For instance, flow-3 only sent a three-byte packet in round 2, whereas, it sent three packets in round 3 and the total length comes to twenty bytes.

ANALYTICAL RESULTS

In this section, analyzing the implementation complexity of RQRR, and also the fairness using a measure based on a popular metric proposed (Golestani, 1994).

Implementation complexity: Consider an execution of the RQRR scheduling algorithm over n flows. Define the implementation complexity of the RQRR scheduler as the order of the time complexity, with respect to n, of enqueuing and then dequeuing a packet for transmission:

Theorem 1: The implementation complexity of a RRQR scheduler is O (1).

Proof: prove the theorem by demonstrating that enqueuing and dequeuing a packet are each of O (1) time complexity.

The time complexity of enqueuing a packet is the same as the time complexity of the Enqueuing module in Fig. 1, which is executed whenever a new packet arrives at a flow. Determining the flow at which the packet arrives is an O (1) operation. The flow at which the new packet arrives is added to the ActiveFlowList, if it is not already in the list. This addition of an item to the tail of a linked list data structure is also an O (1) operation.

Next, considering the time complexity of dequeuing a packet. During each service opportunity, the RQRR scheduler transmits at least one packet. Thus, the time complexity of dequeuing a packet is equal to or less than the time complexity of all the operations performed during each service opportunity. Each execution of the set of operations inside the while loop of the Dequeuing module in Fig. 1, represents all operations performed during each service opportunity given to a flow. These operations include determining the next flow to be served, removing this flow from the head of the ActiveFlowList and possibly adding it back at the tail. All of these operations on a linked list data structure can be executed in O (1) time. Additionally, each service opportunity includes updating the values of VisitFlowCount, Send_i and p-value. All of these can be done in constant time, as represented by the constant number of operations in the Dequeuing module in Fig. 1. The statement of the theorem is proved.

Fairness: GPS (General Processor Sharing) scheduling algorithm has good fairness. Absolute fairness bound of a scheduler, a method of fairness measurement, is defined as the upper bound on the difference between services received by a flow under the scheduler and that under GPS scheduling over all possible time intervals. Since this upper bound is difficult to obtain analytically, Relative fairness bound, proposed by Golestani (1994), is more commonly employed. This quantity is defined as the maximum difference in the services received by any two flows over all possible time intervals:

Definition 1: Let Send_i (t_1, t_2) be the number of bytes transmitted by flow-i during the time interval between t_1 and t_2. Given a time interval (t_1, t_2), the Fairness Measure, FM (t_1, t_2) for the interval is defined as the maximum value of the difference between the number of bytes transmitted by any two flows, flow-i and flow-j. Namely:

$$\text{FM}(t_1, t_2) = \max(|\text{Send}_i(t_1, t_2) - \text{Send}_j(t_1, t_2)|), 1 \le i, j \le n$$

Define FM as the maximum of FM (t_1, t_2) for all possible time intervals (t_1, t_2). A scheduler is more perfectly fair if FM approaches zero more closely. GPS (General Processor Sharing) is proven to have FM = 0, but this condition cannot be met by any packet-by-packet algorithm since packets must be served exclusively. Therewithal, a scheduling algorithm can be considered close to fair if its FM is bounded by a constant. Especially, FM (t_1, t_2) should be independent of the length of time interval.

Definition 2: Define Max as the size in bytes of the largest packet, namely the largest length of packets:

Lemma 1: In the case of no rounding in computing AC_i $(1 \le i \le n)$, for any round r $(r \ge 1)$, the sum of p-value of n flows is zero.

Proof: When r = 1, since $P_i(1) = 0$, so the sum of p-value of n flows is zero. When, $r \ge 2$, since:

$$P_i(r) = P_i(r-1) + \text{AC}_i(r-1) - \text{Send}_i(r-1)$$

so the sum is:

$$\sum_{i=1}^{n} P_i(r) = \sum_{i=1}^{n}[P_i(r-1) + AC_i(r-1) - Send_i(r-1)]$$
$$= \sum_{i=1}^{n} P_i(r-1) + \sum_{i=1}^{n} AC_i(r-1) - \sum_{i=1}^{n} Send_i(r-1)$$

Since:

$$\sum_{i=1}^{n} AC_i(r-1) = \frac{1}{n-1} \times (n-1)\sum_{i=1}^{n} Send_i(r-1) = \sum_{i=1}^{n} Send_i(r-1)$$

then:

$$\sum_{i=1}^{n} P_i(r) = \sum_{i=1}^{n} P_i(r-1) + 0 = \sum_{i=1}^{n} P_i(r-1)$$

Since, P_i (1) = 0, therefore:

$$\sum_{i=1}^{n} P_i(r) = 0$$

The statement of the lemma is proved.

Lemma 2: For any flow-i ($1 \leq i \leq n$) and round r ($r \geq 1$), P_i (r) <2Max.

Proof: When r = 1, P_i (1) = 0. The statement is correct. When:

$$r \geq 2, P_i(r) = P_i(r-1) + AC_i(r-1) - Send_i(r-1)$$

Since:

$$Send_i(r-1) \geq P_i(r-1), \text{ have } P_i(r) \leq AC_i(r-1)$$

According to Lemma 1, if the p-values of flows are not all zero, then they cannot all be positive number. The sum of positive p-value and the sum of negative p-value have the equal numerical value. When the p-value of a flow is negative, this flow is permitted to send one packet and its largest length is Max. Without loss of generality, assume that the p-value of flow-1 is negative at round r-1, the largest length of data that flow-1 can send at this round is Max. Again, assume that flow-2 is the flow with the largest p-value which is more than Max and attains to 2Max-1. The total length of packets that flow-1 and flow-2 can send at the most is: Max+2Max-2+Max = 4Max-2; the average value is 2Max-1 and less than 2Max. If need AC_i (r-1)≥2Max, then the average value of P_j (r-1) of the other flows (j≠i) must be more than Max, that is to say, AC_j (r-2)>Max. By parity of reasoning, it requires AC_j (1)>Max. However, the maximum value of AC_j (1) is Max, and then AC_i (r-1)<2Max; therefore, P_i (r)<2Max. The statement of the lemma is proved.

Theorem 2: Given m consecutive rounds starting from round k ($k \geq 1$) during which flow-i ($1 \leq i \leq n$) is active, the total number of bytes transmitted by flow-i is denoted as Sum-i. For any round r ($k \leq r \leq k+m-1$), if P_i (r)≥0, then:

$$-2Max + \sum_{r=k}^{k+m-1} AC_i(r) < Sum-i < \sum_{r=k}^{k+m-1} AC_i(r) + 2Max$$

Proof: When r≥2, P_i (r) = P_i (r-1)+AC_i (r-1)-$Send_i$ (r-1), therefore, $Send_i$ (r-1) = P_i (r-1)-P_i (r)+AC_i (r-1), then:

$$Sum-i = \sum_{r=k}^{k+m-1} Send_i(r) = \sum_{r=k}^{k+m-1}[P_i(r) - P_i(r+1) + AC_i(r)] = \sum_{r=k}^{k+m-1}[P_i(r) - P_i(r+1)] + \sum_{r=k}^{k+m-1} AC_i(r) = P_i(k) - P_i(k+m) + \sum_{r=k}^{k+m-1} AC_i(r)$$

Since, P_i (r)<0 and P_i (r) = 0 allow flow-i to send one packet, when P_i (r)<0, take P_i (r) = 0, then have P_i (r)≥0. Also, P_i (r)<2Max from Lemma 2, therefore, -2Max<P_i (k)-P_i (k+m) <2Max. The statement of the theorem is proved.

The following theorem establishes the fact that the fairness measure of RQRR is bounded by a constant, which is independent of the length of time interval.

Theorem 3: In any execution of RQRR algorithm, when P_i (r)<0, take P_i (r) = 0 ($1 \leq i \leq n$), Then, for any time interval (t_1, t_2), FM (t_1, t_2)<7Max-1.

Proof: Consider two active flows i and j in time interval (t_1, t_2). Since, RQRR is a round-robin algorithm, flow-j must have an opportunity between any two opportunities given to flow-i. Let m_i be the number of opportunities given to flow-i in time interval (t_1, t_2) and let m_j be the number of opportunities given to flow-j in the same interval, then $|m_i - m_j| \leq 1$.

Without loss of generality, assume that in the interval (t_1, t_2), flow-i starts receiving service before flow-j. Thus, $m_j = m_i$ or $m_j = m_i-1$. From Theorem 2, have:

$$Send_i(t_1, t_2) < \sum_{r=k}^{k+mi-1} AC_i(r) + 2Max \quad (3)$$

$$Send_j(t_1, t_2) > -2Max + \sum_{r=k}^{k+mj-1} AC_j(r) \quad (4)$$

Thus:

$$Send_i(t_1, t_2) - Send_j(t_1, t_2) < \sum_{r=k}^{k+mi-1} AC_i(r) + 2Max - [-2Max + \sum_{r=k}^{k+mj-1} AC_j(r)]$$
$$= 4Max + \sum_{r=k}^{k+mi-1} AC_i(r) - \sum_{r=k}^{k+mj-1} AC_j(r) \quad (5)$$

When $m_j = m_i$:

$$\sum_{r=k}^{k+mi-1} AC_i(r) - \sum_{r=k}^{k+mj-1} AC_j(r) = \sum_{r=k}^{k+mi-1} AC_i(r) - \sum_{r=k}^{k+mi-1} AC_j(r)$$
$$= \sum_{r=k}^{k+mi-1}[AC_i(r) - AC_j(r)] = \frac{1}{n-1}\sum_{r=k}^{k+mi-1}[Send_j(r) - Send_i(r)]$$
$$= \frac{1}{n-1}[\sum_{r=k}^{k+mi-1} Send_j(r) - \sum_{r=k}^{k+mi-1} Send_i(r)]$$
$$= \frac{1}{n-1}[Send_j(t_1, t_2) - Send_i(t_1, t_2)]$$

So, from Eq. 5 have:

$$\text{Send}_i\ (t_1,t_2)\text{-Send}_j\ (t_1,t_2)<4\text{Max}+\frac{1}{n-1}[\text{Send}_j\ (t_1,t_2)\text{-Send}_i\,(t_1,t_2)]$$

namely:

$$(1+\frac{1}{n-1})\,[\text{Send}_i\,(t_1,t_2)\text{-Send}_j\ (t_1,t_2)]<4\text{Max}$$

then, get:

$$\text{Send}_i\ (t_1,t_2)\text{-Send}_j\ (t_1,t_2)<\frac{n-1}{n}\times 4\text{Max}<4\text{Max} \tag{6}$$

When $m_j = m_i\text{-}1$:

$$\begin{aligned}
&\sum_{r=k}^{k+mi-1} AC_i(r)-\sum_{r=k}^{k+mj-1} AC_j(r)=\sum_{r=k}^{k+mi-1} AC_i(r)-\sum_{r=k}^{k+mi-2} AC_j(r)\\
&=\sum_{r=k}^{k+mi-1} AC_i(r)-\sum_{r=k}^{k+mi-1} AC_j(r)+AC_j\ (k+m_i\text{-}1)\\
&=\sum_{r=k}^{k+mi-1} [AC_i(r)-AC_j(r)]+AC_j\ (k+m_i\text{-}1)\\
&=\frac{1}{n-1}\sum_{r=k}^{k+mi-1} [\text{Send}_j(r)-\text{Send}_i(r)]+AC_j(k+m_i\text{-}1)\\
&=\frac{1}{n-1}[\sum_{r=k}^{k+mi-1} \text{Send}_j(r)-\sum_{r=k}^{k+mi-1} \text{Send}_i(r)]+AC_j(k+m_i\text{-}1)\\
&=\frac{1}{n-1}[\sum_{r=k}^{k+mj-1} \text{Send}_j(r)+\text{Send}_j(k+m_j)-\sum_{r=k}^{k+mi-1} \text{Send}_i(r)]+AC_j(k+m_i\text{-}1)\\
&=\frac{1}{n-1}[\text{Send}_j(t_1,t_2)\text{-Send}_i(t_1,t_2)]+\frac{1}{n-1}\text{Send}_j(k+m_j)+AC_j(k+m_j)\\
&=\frac{1}{n-1}[\text{Send}_j\ (t_1,t_2)\text{-Send}_i(t_1,t_2)]+\frac{1}{n-1}\sum_{u=1}^{n}\text{Send}_u(k+m_j)
\end{aligned}$$

So, from (5) have:

$$\text{Send}_i\ (t_1,t_2)\text{-Send}_j(t_1,t_2)<4\text{Max}+\frac{1}{n-1}[\text{Send}_j\ (t_1,t_2)\text{-Send}_i\ (t_1,t_2)]+\frac{1}{n-1}\sum_{u=1}^{n}\text{Send}_u(k+m_j)$$

namely:

$$(1+\frac{1}{n-1})[\text{Send}_j(t_1,t_2)\text{-Send}_i(t_1,t_2)]<4\text{Max}+\frac{1}{n-1}\sum_{u=1}^{n}\text{Send}_u(k+m_j)$$

then, get:

$$\text{Send}_j\ (t_1,t_2)\text{-Send}_i\ (t_1,t_2)<\frac{n-1}{n}\times 4\text{Max}+\frac{1}{n}\sum_{u=1}^{n}\text{Send}_u(k+m_j)$$

Since, $\text{Send}_u\ (k+m_j)\le P_u\ (k+m_j)\text{-}1+\text{Max}<2\text{Max-}1+\text{Max} = 3\text{Max-}1$ hence:

$$\text{Send}_j\ (t_1,t_2)\text{-Send}_i\,(t_1,t_2)<\frac{n-1}{n}\times 4\text{Max}+\frac{1}{n}\times n\times(3\text{Max-}1)<7\text{Max-}1 \tag{7}$$

Combining Eq. 6 and 7, get FM $(t_1, t_2)<7\text{Max-}1$. The statement of the theorem is proved.

RQRR SUPPORTS MULTILINK TRANSMISSION

Many medium-small enterprises access Internet through an E1 leased line firstly. When business increments want higher bandwidth than E1, telecommunications operators have no good method to solve this problem because the higher bandwidth standard is E3 line. The bandwidth of E3 is 34 Mbps and this kind of line not only has expensive rent charge, but also subscribers do not need such high bandwidth. In this case, multilink transmission is an efficient way to solve the problem; that is, satisfying the subscriber's demand on bandwidth by increasing links based on the former lines. The PPP Multilink Protocol (MLPPP) (Sklower *et al.*, 1996) is the typical representative among them. The distribution mechanism of multilink will divide a large packet into many small packets and transmit them through multilink simultaneously. However, if just assign packets to multi channels and transmit them simply, the receiver cannot reassemble the packets correctly because of the improper arriving sequence. Therefore, MLPPP appends 4 bytes sorting identifier in a packet header to control the sequence and adopts simple synchronous principle so that it can transmit packets on parallel channels and reassemble them with correct order at the destination. The other problem which multilink transmission needs solve is the fairness of link utilization, namely load-balance between links. In this respect, MLPPP only mentioned two simple round-robin methods and they cannot guarantee fairness as well as adequately take advantage of the bandwidth provided by multilink.

RQRR algorithm can settle datagram reassembly and load-balance commendably in multilink transmission. When there are two or more links between sending end and receiving end, RQRR can realize load-balance at the sending end. At the receiving end, RQRR carries out queue scheduling and the receiving packet sequence is entirely consistent with the sending sequence. Next, specifying the process of multilink transmission supported by RQRR algorithm. Notice that: Some concepts such as ActiveLinkList, VisitLinkCount etc., can be understood easily based on the above algorithm description, so there is no specific explanation for them.

RQRR operating at sending end: There are one input link and multi output links at sending end. RQRR will initialize sending end after all output links are ready, including: initializing "ActiveLinkList" of sending end; "VisitLinkCount" is set initial value 0; the p-value of each link is also set initial value 0. When a packet arrives at sending end, it enters into "DataRequestSendQueue"

Load-balance Operation at Sending End:

(1) When DataRequestSendQueue is empty, namely there is no packet wanted to be sent, load-balance module goes to the status of waiting for packet coming otherwise, counting;
(2) Judging whether VisitLinkCount equal 0 or not, if yes, continue; Otherwise, executing step (5);
(3) ByteNumberCount of each link in ActiveLinkList is set as0;
(4) VisitLinkCount = the number of links in ActiveLinkList,
(5) Choosing the first link in ActiveLinkList picking up a packet from
DataRequestSendQueue and sending it out through the link, its ByteNumberCount is
increased by the length (bytes) of the packet;
(6) Judging whether p-value of the link minus its ByteNumberCount is more than 0 or not, if yes, continue, Otherwise, executing step (9);
(7) Judging whether DataRequestSendQueue is empty or not, if yes, executing step (1); Otherwise, continue;
(8) Picking up a packet from DataRequestSendQueue sequentially and sending it out through the chosen link ByteNumberCount of the link is incresed by the length of the packet; executing step (6);
(9) Moving the chosen link to the tail of ActiveLinkList
(10) VisitLinkCount minus 1;
(11) Judging whether VisitLinkCount is 0 or not, if yes, continue; Otherwise, executing step (14);
(12) Computing the p-value of the next round of each link in ActiveLinkList
(13) Executing step(1);
(14) Judging whether DataRequestSendQueue is empty or not, if yes, executing step (1); Otherwise, executing step (5).

Fig. 3: RQRR (Resilient Quantum Round-Robin) operating procedure at sending end

Queue Scheduling Operation at Receiving End:

(1) When the queue of the first link of ActiveLinkList is empty, namely there is no packet wanted to be scheduled in the queue of the current link, queue scheduling module goes to the status of waiting for packet coming Otherwise, continue;
(2) Judging whether VisitLinkCount equals 0 or not, if yes, continue; Otherwise Executing step (4);
(3) VisitLinkCount = the number of links in ActiveLinkList
(4) Choosiing the first link in ActiveLinkList sending out the first packet in the queue of the link through the output links; ByteNumberCount of the chosen link is increased by the length (bytes) of the packet;
(5) Judging whether p-value of the link minus its ByteNumberCount is more than 0 ot not, if yes, continue; Otherwise, executing step (8);
(6) Judging whether the queue of the first link in ActiveLinkList is empty or not if yes, executing step (1); Otherwise, contniue;
(7) Sending the next packet of the chosen queue sequentially, ByteNumberCount of the corresponding link is increased by the length of the packet executing step (5);
(8) Moving the chosen link to the tail of AcriveLinkList
(9) VisitLinkCount minus 1;
(10) Judging whether VisitLinkCount is 0 or not, if yes, continue; Otherwise, executing step (13);
(11) Computing the p-value of the next round of each link in ActiveLinkList; ByteNumberCount of each link in ActiveLinkList is set as0;
(12) Executing step (1);
(13) Judging whether the queue of the first link in ActiveLinkList is empty or not if yes, executing step (1); Otherwise, executing step (4).

Fig. 4: RQRR (Resilient Quantum Round-Robin) operating procedure at receiving end

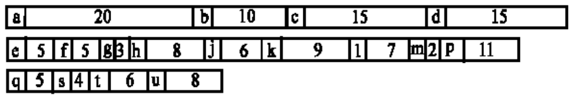

Fig. 5: A flow consisting of seventeen packets arrives at sending end and each packet is denoted with a letter and its length (bytes). Hereinto, packet a arrives firstly and u is the last one

firstly and waits for being sent. Load-balance module executes RQRR algorithm and selects one link from multi output links through which packets in "DataRequestSendQueue" are sent out. The operating procedure is shown as Fig. 3.

RQRR operating at receiving end: There are multi input links and one output link at receiving end. The initializing process involves: Initializing "ActiveLinkList" of receiving end; "VisitLinkCount" is set initial value 0; the p-value of each link is set initial value 0; The "ByteNumberCount" of each link in "ActiveLinkList" is set initial value 0 as well. Notice that, receiving end has some same components as sending end, but their operation is independent. When a packet sent from sending end arrives at receiving end, it will enter into the queue belonging to the relevant link. Queue scheduling module of receiving end executes RQRR algorithm to pick up packets from relative queue and send them out through the output link. The operating procedure is illustrated in Fig. 4.

An example: Assume that a flow, arriving at sending end, consists of seventeen packets expressed successively by a, b, c, d, e, f, g, h, j, k, l, m, p, q, s, t and u shown as in Fig. 5. The numbers behind letters denote the length of packets with unit byte. There are three links between sending end and receiving end, expressed as link-1, link-2 and link-3, respectively.

At sending end, load-balance module executes RQRR algorithm to distribute seventeen packets into three links after four rounds (hereinto SP denotes p-value at sending end) illustrated in Fig. 6. At receiving end, there are four packets a, d, h, g entering into the queue of link-1; seven packets b, e, f, j, k, s, t entering into the queue of link-2; six packets c, g, l, m, p, u entering into the queue of link-3 and the status is revealed in Fig. 7. Moreover, the load-balance situation between links can be seen from Fig. 7.

Queue scheduling process at receiving end is similar to the scheduling process in section 2. After four rounds queue scheduling, obtaining the following packet sequence on the output link at receiving end: a, b, c, d, e, f, g, h, j, k, l, m, p, q, s, t and u. Hereinto, a is the first packet sent out through the output link and u is the last one. Thus, it can be seen, the packet sequence sent out

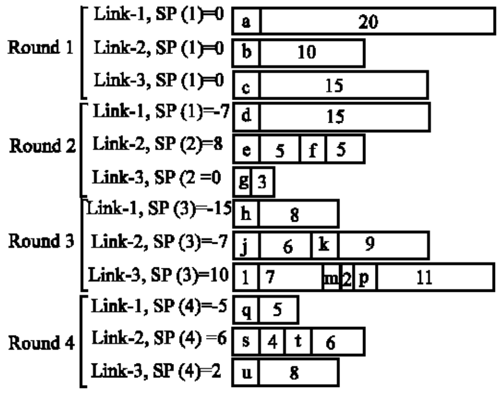

Fig. 6: An illustration of 4 rounds in a RQRR (Resilient Quantum Round-Robin) execution at sending end. There are three links between sending end and receiving end and SP denotes the p-value at sending end

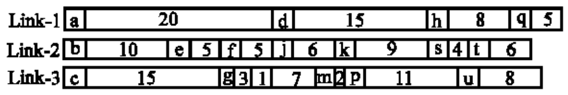
Link-1 a 20 d 15 h 8 q 5
Link-2 b 10 e 5 f 5 j 6 k 9 s 4 t 6
Link-3 c 15 g 3 1 7 m 2 p 11 u 8

Fig. 7: At receiving end, packets a, d, h and q enter into the queue of Link-1, packets b, e, f, j, k, s and t enter into the queue of Link-2, packets c, g, l, m, p and u enter into the queue of Link-3

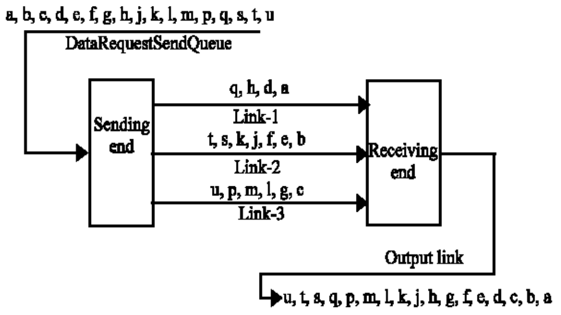

Fig. 8: An illustration of RQRR (Resilient Quantum Round-Robin) supporting multilink transmission. Load-balance operation at sending end distributes seventeen packets to be transmitted through three links. After executing queue scheduling at receiving end, packet sequence through the output link is the same as packets arrived at sending end

through the output link at receiving end is consistent with the packet sequence arriving at sending end. The whole process is presented in Fig. 8.

Therefore, realizing multilink data transmission by RQRR algorithm, it not only allocates the bandwidth resource of multilink fairly to keep load-balance amongst links, but also guarantees accordant packet sequence between sending end and receiving end without increasing additional overhead. So, it has low complexity and hardware implementation is simple as well. In order to realize load-balance and accordant packet sequence, sending end and receiving end must satisfy the following conditions: (1) Sending end and receiving end should be synchronous and have same ActiveLinkList, (2) There is no packet loss during the transmission, or accordant packet sequence cannot be guaranteed.

CONCLUSION

This study presented a novel scheduling discipline called Resilient Quantum Round-Robin (RQRR), which is fair, efficient and supports variable-length packet scheduling. The implementation complexity of RQRR is O (1) and therefore, can be easily implemented in high-speed networks with large numbers of flows. The relative fairness measure of RQRR is independent of the length of time interval and has an affirmatory upper bound of 7Max-1, where Max is the largest size of the packets. In comparison to DRR of similar efficiency, complexity and fairness of RQRR are not related to quantum; each flow sends out one packet at least in a round and RQRR has better characteristic of latency; on the other hand, RQRR does not want to know the length of a packet before scheduling it because it gets hold of the length information from the already sent packets. Finally, in supporting multilink transmission, RQRR can efficiently implement load-balance as well as guarantee the consistent packet sequence between receiving end and sending end.

ACKNOWLEDGMENTS

This study is partially sponsored by the S and T Plan Projects of Hunan Province (No. 2010GK3045) and Scientific Research Fund of Hunan Provincial Education Department (No. 10C0687).

REFERENCES

Ayaz, S., F. Hoffmann, R. German and F. Dressler, 2011. Analysis of deficit round robin scheduling for future aeronautical data link. Proceedings of the IEEE 22nd International Symposium on Personal Indoor and Mobile Radio Communications, September 11-14, 2011, Toronto, ON., Canada, pp: 1809-1814.

Bennett, J.C.R. and H. Zhang, 1996. WF^2Q: Worst-case fair weighted fair queueing. Proc. IEEE INFOCOM, 1: 120-128.

Bo, Z., W. Bin-qiang, W. Shan-shan, W. Hong-quan and L. Hui, 2012. Research on input-queued slicing domain scheduling based on Crossbar in the reconfigurable network. J. Commun., 33: 105-115.

Chaskar, H.M. and U. Madhow, 2003. Fair scheduling with tunable latency: A round-robin approach. IEEE/ACM Trans. Network, 11: 592-601.

Fei, G., Z. Yuan and Y. Baizhan, 2011. A new and better algorithm for combining token bucket filter with DRR. J. Northwestern Polytechn. Univ., 29: 49-53.

Golestani, S.J., 1994. A self-clocked fair queuing scheme for broadband applications. Proceedings of the 13th INFOCOM'94, Networking for Global Communications, June 12-16, 1994, Toronto, Ont., Canada, pp: 636-646.

Hua, Y. and X. Liu, 2012. Scheduling heterogeneous flows with delay-aware deduplication for avionics applications. IEEE Trans. Parallel Distrib. Syst., 23: 1790-1802.

Jian, G.Z., N. Ge and C.X. Feng, 2005. A prioritized nested DRR algorithm. J. Electron. Inf. Technol., 27: 123-126.

Kanher, S.S. and H. Sethu, 2001. Fair efficient and low-latency packet scheduling using nested deficit round robin. Proceedings of the IEEE Workshop on High Performance Switching and Routing, May 29-31, 2001, Dallas, TX., USA., pp: 6-10.

Kumar, D. and V. Priyameenal, 2011. Adaptive packet scheduling algorithm for real-time services in Wi-MAX networks. Proceedings of the International Conference on Recent Trends in Information Technology, June 3-5, 2011, Chennai, Tamil Nadu, pp: 342-347.

Mansy, A., M. Ammar and E. Zegura, 2011. Deficit round-robin based message ferry routing. Proceedings of the Global Telecommunications Conference, December 5-9, 2011, Houston, TX., USA., pp: 1-5.

Parekh, A.K. and R.G. Gallager, 1993. A generalized processor sharing approach to flow control in integrated service network: The single node case. IEEE Trans. Network., 1: 344-357.

Shreedhar, M. and G. Varghese, 1996. Efficient fair queuing using deficit round-robin. IEEE/ACM Trans. Networking, 4: 375-385.

Shuqing, Y. and P. Jinye, 2012. QoS multi-users packet scheduling algorithm in IEEE 802.16e system. Proceedings of the International Conference on Computer and Automation Engineering, January 21-23, 2011, Chongqing, China, pp: 112-116.

Sivakumar, G. and A.V. Ramprasad, 2012. Analysis of FiWi networks to improve TCP performance. Proceedings of the International Conference on Computing, Communication and Applications, February 22-24, 2012, Dindigul, Tamil Nadu, pp: 1-6.

Sklower, K., B. Lloyd, G. McGregor, D. Carr and T. Coradetti, 1996. The PPP Multilink Protocol (MP). http://tools.ietf.org/html/rfc1990

Sleem, M.Y., H.M. ElBadawy and M.S. Abo-El-Seoud, 2011. Two layer channel aware scheduling for QoS support in IEEE 802.16/WiMAX networks. Proceedings of the 8th International Conference on Wireless and Optical Communications Networks, May 24-26, 2011, Paris, pp: 1-5.

Xiaodong, T. and L. Lemin, 2002. LL-DRR: An efficient scheduling algorithm for packet network. J. Electron. Inf. Technol., 24: 361-369.