A SCHEDULING ALGORITHM USING COMPENSATING ROUND ROBIN IN PACKET-SWITCHING BROADBAND NETWORKS

Lan Julong  Wang Binqiang  Li Ou  Wu Jiangxing
(National Digital Switching System Eng. & Technological Research Center, Zhengzhou 450002)

Abstract A new approximation of fair queuing called Compensating Round Robin (CRR) is presented in this paper. The algorithm uses packet-by-packet scheduler with a compensating measure. It achieves good fairness in terms of throughput, requires only $O(1)$ time complexity to process a packet, and is simple enough to be implemented in hardware. After the performances are analyzed, the fairness and packet loss rate of the algorithm are simulated. Simulation results show that the CRR can effectively isolate the effects of contending sources.

Key words Broadband networks; Round robin; Scheduling; Bandwidth allocation; Queuing

I. Introduction

When there is contention for resources in packet switching networks, it is important for resources to be allocated fairly. We need firewalls between contending users, so that the “fair” allocation is followed strictly. Unfortunately, most networks are susceptible of contending sources. A contending source that sends at an uncontrolled rate can seize a large fraction of the buffers at an intermediate router; this can result in dropping packets from other sources sending at more moderate rates. A solution to this problem is needed to isolate the effects of contending sources.

Fair Queuing (FQ)\textsuperscript{[1–5]} is a technique that allows each flow passing through a network device to have a fair share of network resources. But general fair queuing\textsuperscript{[1]} schemes that achieved good fairness are expensive to implement for the time complexity required to process a packet in these schemes is $O(\log(n))$, where $n$ is the number of flows. This is expensive at high speeds. On the other hand, cheaper approximations of fair queuing that have been reported in the literatures\textsuperscript{[6–8]} exhibit unfair behavior.

In this paper we provide an isolation mechanism that achieves good fairness. And the mechanism takes $O(1)$ processing work per packet. The algorithm is simple to be implemented at high speeds in a router or gateway. It uses a packet-by-packet scheduler with a compensating measure. When there is contention for resources, it only discards packets belonging to the contending sources.

II. Compensating Round Robin Algorithm

1. Basic concepts

We first define two measures: FairIndex (that measures the fairness of the queuing discipline) and Work (that measures the time complexity of the queuing algorithm). The measures are not only specific to Compensating Round Robin (CRR), but also applied to other forms of fair queuing. To define the Work measure, we assume the queuing model of a router as follows: There are a number of input queues and one output queue. And there are an enqueuing process and a dequeuing process. Enqueuing process finds the number of the input queue used by the flow and appends the packet of the flow to the tail of the queue. Dequeuing process decides which queue should be served, how many packets should
be transmitted from the queue in current round, and transmits the packets. Thus the Work to process a packet involves two parts: enqueuing and dequeuing.

**Definition 1** Work is defined as the maximum of the time complexities in enqueuing and dequeuing a packet from the router. The time complexity is the operating times of a scheduler to schedule a packet.

To define the throughput fairness measure, we assume that we start sending packets on the outgoing link at time 0. Let $S_{i,t}$ be the total number of bytes sent by flow $i$ by time $t$; let $S_t$ be the total number of bytes sent by all $n$ flows by time $t$.

**Definition 2** The bandwidth ratio obtained by flow $i$ is

$$FQ_i = \max \left( \lim_{t \to \infty} \frac{S_{i,t}}{S_t} \right)$$

where $\max(\ )$ is taken across all possible input packet size distributions for all flows.

Next, we use $f_i$ to express the bandwidth allocated to flow $i$ by a manager. Thus the ideal fairness quotient for flow $i$ is

$$IFQ_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

Finally, we measure how far a fair queuing implementation departs from the ideal by measuring the ratio of actual fairness quotient achieved to the ideal fairness quotient. We call this the fairness index.

**Definition 3** The fairness index for a flow $i$ in a fair queuing implementation is

$$FairIndex_i = \frac{FQ_i}{IFQ_i} = \frac{FQ_i \sum_{j=1}^{n} f_j}{f_i}$$

**2. Compensating round robin**

On the basis of the definitions of some basic concepts, we introduce a nearly fair queuing scheme. The basic ideas are as follows:

1. We assume there are $n$ flows coming to a router and there are $m$ queues in the scheduler. We use McKenney’s idea of stochastic queuing to bound the number of queues required. Scheduling algorithm and hash algorithm affect the performances in different way. To clearly separate these issues, during the analysis of CRR, we will first ignore the effects of hash algorithm, and assume that flows are mapped uniquely into different queues. That is to say, the number of flow $n$ is equal to the number of queue $m$. And then the effects of hash algorithm on the FairIndex and Work will be analyzed.

2. Sending packets in a packet-by-packet round robin fashion. Pure packet-by-packet round robin serves every queue a packet in a round. It fails to guarantee a fair allocation of bandwidth because of variations in packet sizes. Therefore, we define a threshold $T$ for every queue. When the length of a queue does not reach the threshold, send a packet from the queue in every round if there are packets in it. If a queue has no packet to send in its turn, turn to the next queue. When the length of a queue exceeds the threshold $T$, send $C_i\max$ byte in its turn in a round, where $ST_i$ is

$$ST_i = \min(x | x \geq C_i\max, \ x \text{ includes integral packets}) \text{ (Byte)}$$

where $C_i$ is directly proportional to $f_i (C_i = \alpha f_i \geq 1, \alpha$ is a constant) and $\max$ is the maximum packet size. In principle, whenever the length of a queue is greater than $T$, the scheduler should send $C_i\max \ (T > C_i\max)$ byte from the queue. But the scheduler serves the queues in a packet-by-packet fashion, it can not leave a fraction of a packet at the head of the queue after the queue is served. Therefore when sending out $C_i\max$ byte from a queue,