A Routing System Using Minimum Starting-tag Fair Queueing

Jen-Bang Feng
Institute of Computer Science
National Chung Hsing University
jbonf@cs.nchu.edu.tw

Yi-Hung Huang
Department of Information Networking Technology
Hsiuping Institute of Technology
ehhwang@mail.hit.edu.tw

Yen-Ping Chu
Institute of Computer Science
National Chung Hsing University
ypchu@nchu.edu.tw

Abstract

Minimum Starting-tag Fair Queuing (MSFQ) is an efficient packet scheduling algorithm with theoretically proved in performance metrics such as delay bound, fairness, and time complexity. However, MSFQ algorithm didn't have any implementation yet. In this paper, we established a configurable routing system based on Linux with MSFQ as the routing scheduling algorithm and provide an efficient way to build a testing environment.

In the process of implementation, we designed and improved the internal part of data structures, embedded MSFQ algorithm into Linux kernel according to structures of Linux Network Traffic Control extended from Linux advanced routing project called iproute2. In order to boot MSFQ from kernel, we also modified 'tc' package in iproute2 project.

An analysis model is built to evaluate the performance of proposed routing system. The comparison between MSFQ algorithm and other scheduling algorithms built in Linux kernel such as prio, sfq, etc. is also provided. The result of implementation proved the behaviors of MSFQ both theoretically and essentially. We also offered an easier and faster process to implement other algorithms so that any new algorithms will be implemented and analyzed more efficiently.

Keywords: MSFQ, QoS, Linux, Packet Scheduling Algorithm, Kernel

1. Introduction

Design of the packet scheduling algorithms is one of the most important issues in providing quality of service (QoS). As each connection interacting with others at each switching node, different packet scheduling algorithms cause different behaviors. The lower delay time reduces the better response time, the smaller the delay jitter increase the stabilization of the connections, and fairness avoid the starvation.

In order to provide a better-behaved service, many algorithms such like VC (Virtual Clock), GPS (Generalized process sharing), PGPS (Packet-by-packet GPS), Self-clocked Fair Queueing, Starting-time Fair Queueing, Stochastic Fair Queueing (SFQ, [8])… have been proposed. Minimum Starting-tag Fair Queuing (MSFQ, [1]) has been also proposed in the same theme after well design and mathematically analysis. This paper is proposed to test and verify the efficiency and performance of MSFQ, to make sure that MSFQ works as well as we suggested.

Linux is an open sourced operation system with well-designed network components, and is already working in many systems. If we can make MSFQ works on Linux, then it should work at any platform or any switching router. For this reason we choose Linux kernel 2.4 together with iproute2 project ([3]) to be our platform, and MSFQ will run inside the patched kernel with fixed tools to control it.

After building a MSFQ router, we test the delay time under varies algorithms, and analyze the performance in several situations. Also we compare MSFQ with other packet scheduling algorithms already built in Linux kernel.

This paper is organized as follows: introduction in Chapter 1 and MSFQ is introduced in Chapter 2. Chapter 3 is the Implementation of the Proposed System and we establish our test

bed in Chapter 4. Finally the testing results and conclusions are in Chapter 5 and 6.

2. Minimum Starting-tag Fair Queueing (MSFQ)

MSFQ belongs to the class of Guaranteed Rate (GR) scheduling algorithms ([9]) and has finishing tag to decide which packet to be served next from least to largest value. The formula of MSFQ is:

$$F^i(p_f^0) = 0, \tag{1}$$

$$v^i(t) = \min_{j \in B^i(t) \wedge j \neq f} S_j^i(t), \tag{2}$$

$$S^i(p_f^k) = \max\{v^i(A^i(p_f^k)), F^i(p_f^{k-1})\}, k \geq 1, \tag{3}$$

$$F^i(p_f^k) = S^i(p_f^k) + \frac{l_f^k}{r_f}, k \geq 1, \tag{4}$$

where p_f^k is the k-th packet of connection f, $B^i(t)$ is the set of all connections existed at time t, $S_j^i(t)$ is the minimal starting tag of connection j at time t in node i, $S^i(p_f^k)$ is the starting tag of p_f^k, $A^i(p_f^k)$ is the time that node i accept p_f^k, $F^i(p_f^k)$ is the finishing tag of p_f^k, l_f^k is the length of p_f^k, and r_f is the reserved bandwidth of connection f.

When the traffic characterization conforms to the leaky bucket policing mechanism ([5,6]), the delay bound of MSFQ is equal to GR algorithms:

$$d_f^k \leq \frac{\sigma_f + (M-1)l_f^{\max}}{r_f} + \sum_{i=1}^{M} \frac{l^{\max}}{C^i}, \tag{5}$$

where $l^{\max}$ is the maximal length of all packet in node i, C^i is the capacity of node i, andσ_f is the characteristic coefficient of connection f.

The fairness of MSFQ is presented in [1] as the different throughput between any two connections during any time interval [t1, t2] as:

$$\left|\frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m}\right| \leq \max \{\frac{l_f^{\max}}{r_f}, \frac{l_m^{\max}}{r_m}, \max_{i \in B(t)}(\frac{l_i}{r_i}) + \frac{l_f^{\max}}{r_f}, \max_{i \in B(t)}(\frac{l_i}{r_i}) + \frac{l_m^{\max}}{r_m}, \frac{l_f^{\max}}{r_f} + \frac{l_m^{\max}}{r_m}\}, \tag{6}$$

where $W_f(t_1,t_2)$ is the throughput of connection f during time interval $[t_1, t_2]$, r_f is the bandwidth reserved for connection f, and $l_f^{\max}$ is the maximal packet length of connection f. The detail discussion was already provided in Huang's researches.

3. Implementation of the Proposed System

Figure 1 shows the Linux traffic control structure designed by Werner Almesberger ([2]), a queueing discipline (Qdisc) is a model for implementing our algorithm. After upgrading kernel, we insert MSFQ as a choice of routing algorithm. Inside this structure, the major parts of all the process are en-queue and de-queue. En-queue process puts an arriving packet into the queue and de-queue process choose a packet to go, Figure 2 is the flow chart of the two processes.

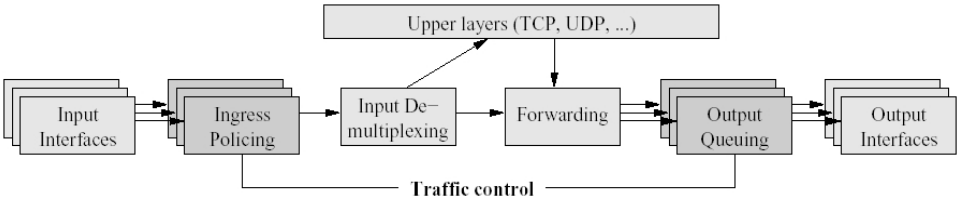


Figure 1. Processing of network data ([2])

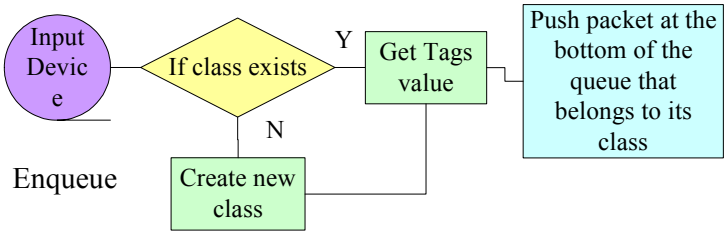


Figure 2.1 En-queue processes

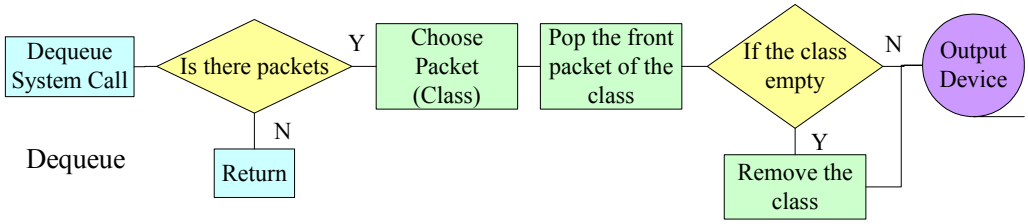


Figure 2.2 De-queue processes

Figure 2. Flow chart of en-queue and de-queue processes

In order to lower the time complexity, we use a hashing table to record the information about each connection and packet belonging to the connection itself. By using this table, we can reduce the access complexity to $O(1)$. MSFQ needs to query the least valued starting and finishing tag both, so we construct two minimal heap tables to record tags of each connection. This time complexity of finding the least valued connection is $O(1)$, but adjusting heap table costs $O(\log N)$, where N is the number of active connections. Thus the total complexity of en-queue process in the above figure is $O(1)$ when connection is already existed, $O(\log N)$ when connection does not exist. De-queue process costs $O(\log N)$ to choose a packet. Figure 3 is the simplified data structure we used.

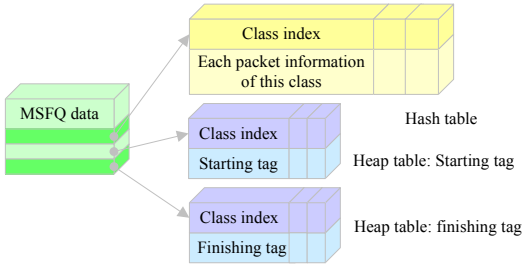


Figure 3. Simplified data structure in MSFQ

In order to evaluate the performance, we use two selective parameters. One is the maximum connection (*flow*) size we recorded and the other is the buffer size of each connection (*buffer*). The total memory size used for MSFQ module in Linux system is $105 + 40\,flow + 8\,flow \times buffer$ bytes.

Our achievement is already published on Internet under the GNU license. Anyone can download this source code of MSFQ in Linux kernel 2.4 at web[1] site.

[1] Please visit our web page at http://www.cs.nchu.edu.tw/~phd9213/MSFQ.htm and download the patch files to update system with MSFQ.

4. Analysis Models

We construct two models for analyzing MSFQ, as shown in Figure 4. Each server runs Linux and is equipped with tbf queuing discipline ([3]) to make the network traffic characterization conformed to leaky bucket model. And every router between the connections is also a Linux machine running MSFQ algorithm. Benchmark programs are running on Clients to retrieve the results.

The first analysis model is a strait path between source and destination. Extension from Analysis Model 1, we let another connection path overlap the original one. Then every router takes double the connections flow and connections interact to each other across the middle part of Analysis Model 2.

Figure 4. Analysis Model 1

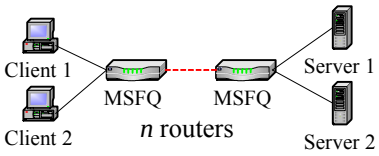


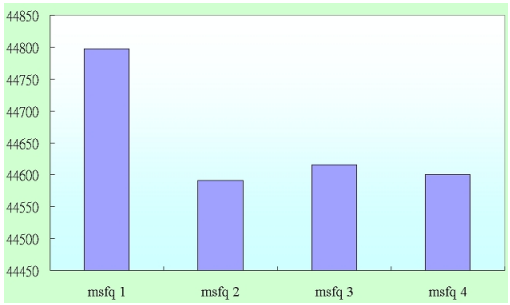
Figure 5. Analysis Model 2

5. Test Bed and Benchmark Results

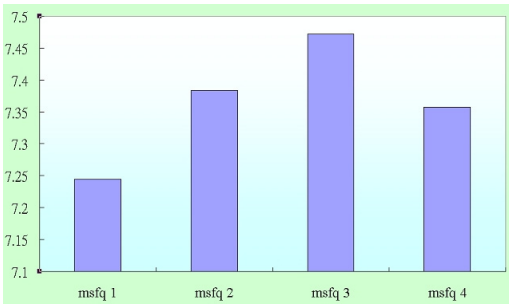
We use WebBench 4.1, a licensed PC Magazine benchmark program with Ziff-Davis as the vendor, to measure the performance of Web servers. WebBench 4.1 reports the results of average connections per second, average throughput per second, average delay time, etc.

First we use our Analysis Model 1 to measure our program performance under varies settings. Every server has traffic characteristic withσ=3Kbytes, ρ=50Kbytes/Second and maximum bandwidth equals to 50Kbytes/Second. The

following figure shows the result of Analysis Model 1.



Average Throughput (Bytes/seconds)



Average Requests/Second

Figure 6. Results of analysis model 1

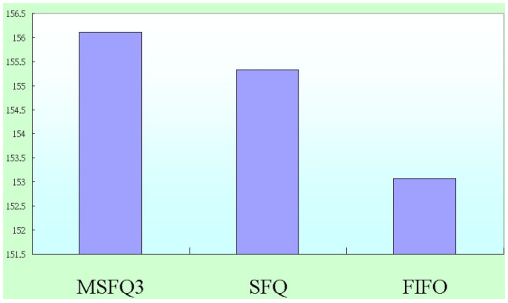
Table 1. Parameter settings in Analysis Model 1

	MSFQ 1	MSFQ 2	MSFQ 3	MSFQ 4
Flow size	127	23	17	17
Buffer size	128	32	20	18
Memory used	136204	7050	3836	3236

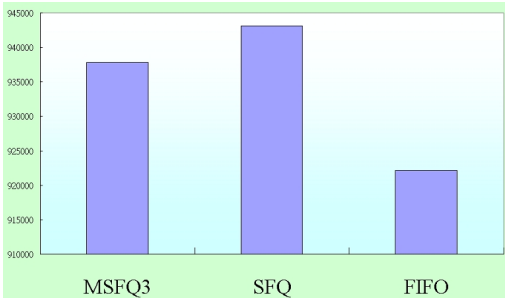
Most of the modern operation systems use page memory management method. Since the page size in Linux cache memory is 4Kbytes, the settings of the proposed system with memory used around 4KB is focused.

Also according to the result of different MSFQ settings, we can find that flow size to be 17 and buffer size to be 20 are better. The analysis result in the first model also shows that MSFQ is capable for sharing connection bandwidth. We use MSFQ3 and set flow size to 17 and buffer size to 20 in the following tests to get the best performance.

We use the Analysis Model 2 by reducing the bandwidth between routers to 500kbit/second and enlarge the server traffic with throughput at 10Mbits/second, also we set our clients sending requests to the different servers. This will produce a bottleneck in Analysis Model 2. Figure 7 is the result using Analysis Model 2 for FIFO, SFQ, and MSFQ.



Average Throughput (Bytes/second)



Average Requests/Second

Figure 7. Results of Analysis Model 2

The result shows that although FIFO is the simplest algorithm, it provides fewer successful connections than other algorithms. We can derive a reasonably conclusion that both MSFQ and SFQ can share delay time to each class of connections better than FIFO. After we made all the analysis, we showed that MSFQ behaves well in varies situations.

6. Conclusions and Future Works

From the result of our tests we can figure that system architecture is a critical factor of overall performance. When the memory we allocated is over 4K bytes, which is the size of a single page in Linux, total throughput decreases significantly. If we want to use MSFQ in other operating systems, we need to realize the system architecture and design a specially designated structure for MSFQ.

When a new method or new solution in networking was proposed, it usually needs an experimental system to prove the feasibility. Linux advanced routing and traffic control suit most of the requirements. We just used a part of all the functions in this paper and other parts of Linux advanced routing will be used to other architectures further.

Reference

[1] Y. P. Chu & E. H. Hwang (1997). *A New Packet Scheduling Algorithm: Minimum Starting-tag Fair Queueing.* IEICE transactions on Communications, Vol.E80-B, No. 10.

[2] Werner Almesberger (2001). *Linux Network Traffic Control -- Implementation Overview. EPFL ICA.*

[3] *Linux 2.4 Advanced Routing HOWTO: Introduction to iproute2*

[4] LARTC -- Linux Advanced Routing & Traffic Control list.

[5] ATM Forum. *ATM forum traffic management specification version 4.0*. Contribution 95-==13R11.

[6] J. Turner (1986). *New directions in communications (or which way to the information age?),* IEEE Comm. Mag., Vol. 24, pp.8-15.

[7] S. Keshav & AT&T Labs-Research (1997). *An engineering approach to computer networking: ATM networks, the internet, and the telephone networks*, Addition-Wesley.

[8] A. Sen, I. Mohammed, R. Samprathi, & S. Bandyopadhyay (2002). *Fair queuing with round robin: a new packet scheduling algorithm for routers*. Computers and Communications, 2002 Proceedings. of Seventh International Symposium.

[9] P. Goyal, S. Lam, & H. Vin (1995). *Determining end-to-end delay bounds in heterogeneous networks*. Proceeding of the 5th international Workshop on Network and Operating System Support For Digital Audio and Video, pp. 287-298, Durham, New Hampshire.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, & Clifford Stein (2001). *Introduction to Algorithms, Second Edition*, Massachusetts Institute of Technology.