A Compressed and Dynamic-range-based Expression of Timestamp and Period for Timestamp-based Schedulers *

Dong Wei†, Nirwan Ansari†, and Jianguo Chen‡
† Advanced Networking Lab
New Jersey Institute of Technology
‡ Bell Labs
Lucent Technologies

Abstract – **Scheduling algorithms are implemented in high-speed switches to provision Quality-of-Service guarantees in both cell-based and packet-based networks. Being able to guarantee end-to-end delay and fairness, timestamp-based fair queuing algorithms have received much attention in the past few years. In timestamp-based fair queuing algorithms, the size of timestamp and period determines the supportable rates in terms of the range and accuracy. Furthermore, it also determines the scheduler's memory in terms of access bandwidth and storage space. An efficient expression can reduce the size of the timestamp and period without compromising the supportable rate range and the accuracy. In this paper, we propose a compressed and dynamic-range-based expression of the timestamp and period, which can be readily implemented in hardware for both high-speed packet-based and cell-based schedulers. As compared to fixed-point and floating-point number expression, when the size is fixed, the proposed expression has a better accuracy. Regarding to efficiency and relative error consistency, it is even better than our earlier proposal.**

I. INTRODUCTION

The current high-speed, service-integrated and packet-switched networks are able to support many kinds of services simultaneously. Packet switches are required to support a large number of sessions with diverse bandwidth requirements; for example, the supportable rate can go as low as 4 Kbits/s and as high as even 10Gbits/s (OC192). Packet switches are also required to support a wide range of packet sizes, from 40 bytes (such as TCP/IP ACK packets) to 64 Kbytes (such as maximum IP packet). Three important issues should be considered in the design of a scheduler: 1) end-to-end delay, 2) fairness, and 3) implementation complexity.

Based on the architecture of the schedulers, packet switches are classified into two types [1]: 1) frame-based, 2) sorted priority. Recently, sorted priority algorithms, also known as packet fair queuing (PFQ), have received much attention because they can approximate the idealized generalized processor-sharing (GPS) algorithm, which has desirable properties in terms of end-to-end delay and fairness [2].

In a PFQ algorithm, there is a global variable called virtual time, associated with outgoing sessions being scheduled. The virtual time is updated when a packet receives service. Each packet has its own timestamp in the system. All packets are sorted by their timestamps. Timestamp sorted algorithms [1]-[2] include weighted fair queuing (WFQ), self-clocked fair queuing (SCFQ), and worst-case weighted fair queuing such as WF^2Q and WF^2Q+. The virtual start time and the virtual finish time are the typical timestamps used in PFQs. Service interval is the time interval that the packet should stay in the scheduler. Given a virtual start time, the service interval is used to calculate the virtual finish time, and vice versa.

The size of the timestamp and period determines the supportable rate in terms of range and accuracy. In this sense, it seems to be tempting to use a larger size to represent them. However, the size of the timestamp and period determines the system memory in terms of bandwidth required to access and space to store. In this regard, the smaller size the better. To resolve this trade-off, an optimal representation of the timestamp and period, which can meet the required accuracy in the smallest size, is required. Note that it is very difficult to use normal fixed-point or floating-point to represent the large range of both the packet size and service rate efficiently. By allocating a range number to accommodate different ranges of service rates, we have developed a better representation in terms of accuracy and the size of timestamp and period (instead of the service interval in [2]). This representation is generically applicable to any timestamp-based scheduling algorithm.

The rest of the paper is organized as follows. Section II presents the background of timestamp and period of PFQs, analyzes the performance of fixed-point expression and floating-point expression in terms of accuracy and the total number of bits to represent timestamp and period. Our proposed expression for packed-based schedulers is discussed in Section III. The proposed expression is extended for cell-based (ATM) schedulers in Section IV. Implementation issues are discussed in Section V. This paper ends with some concluding remarks in Section VI.

II. BACKGROUND

PFQ algorithms are used to approximate the idealized generalized processor-sharing (GPS) algorithm. All PFQ algorithms have similar sorted-queue architecture. They differ in two aspects [2]: virtual time function and packet-selection policy.

A. Notation

* This work has been supported in part by the New Jersey Commission on Science and Technology via the NJ Center for Wireless Telecommunications, and the New Jersey Commission on Higher Education via the NJ I-TOEWR project.

0-7803-7206-9/01/$17.00 © 2001 IEEE

$Z(.)$ – the number of bits
$\overline{P}$ – the idealized period with infinite bit expression
P – actual period representation with finite bit expression
$\overline{r}$ – the idealized service rate with infinite bit expression
r – actual service rate with finite bit expression
r_{max} – maximum supportable service rate
r_{min} – minimum supportable service rate
r_i – required service rate for session ***i***
r_{LC} – link capacity of the scheduling system
$\varepsilon(\alpha)$ – the relative error of α

$$\varepsilon(\alpha)=\left|\frac{\alpha-\overline{\alpha}}{\overline{\alpha}}\right|$$

T – timestamp
I_α – integer part of α
F_α – fractional part of α
M_α – mantissa part of α
E_α – exponent part of α
L– the packet size
M- number of bits to represent virtual system time
N- total number of bits of timestamp and period

B. PFQ Algorithms

PFQ algorithms have a global variable – system virtual time V(.), which is defined differently for different PFQ algorithms. They also maintain a virtual start time and a virtual finish time for each session. When the ***k^th^*** packet of session ***i*** arrives, the virtual start time $S_i(.)$, virtual finish time $F_i(.)$ and the service interval Φ_i^k of this packet are updated as follows:

$$S_i(t)=\begin{cases}\max(V(t),F_i(t-)) & \text{session } i \text{ in service},\\ F_i(t-) & p_i^{k-1} \text{ finished service}.\end{cases} \quad (1)$$

$$F_i(t)=S_i(t)+\Phi_i^k. \quad (2)$$

$$\Phi_i^k=\frac{L_i^k}{r_i}. \quad (3)$$

where L_i^k is the size of the ***k^th^*** packet of session ***i*** and r_i is the required service rate of session ***i***..

The worst-case fair index (WFI) [3] was introduced to characterize the fairness performance of PFQ algorithms. It was shown that PFQ algorithms with two tags (the virtual start time and the virtual finish time) could achieve better fairness performance than those with only a single tag.

C. Expression of Timestamp and Period

Let L_{min} be the minimum supportable packet size. In a packet scheduling system, the concept of time slot is introduced to normalize the time interval. One time slot τ equals to the time interval required to transmit a packet with the minimum supportable packet size at link rate r_{LC}:

$$\tau=\frac{L_{\min}}{r_{LC}}. \quad (4)$$

Usually, $r_{max}=r_{LC}$.

The system virtual time $V(t)$ is defined differently from one scheduling algorithm to another. $V(t)$ is expressed in the unit of time slot; for example, $V(t)$ is nothing but a counter in the virtual clock scheduling scheme, and it is incremented by one when a single packet with the minimum supportable packet size is sent out. The period of session ***i*** is defined as follows:

$$P_i=\frac{r_{\max}}{r_i}.$$

The period of a session is unit-less quantity. A session with $P=3$ means that the session should receive service of one time slot in every three time slots. The service interval and period are related as follows:

$$\Phi_i^k=\frac{L_i^k}{r_i}=\frac{L_i^k}{L_{\min}}\frac{L_{\min}}{r_{\max}}\frac{r_{\max}}{r_i}=\frac{L_i^k}{L_{\min}}\tau P_i \quad . \quad (5)$$

Denote $\underline{\Phi_i^k}$ as the normalized value of Φ_i^k. That is,

$$\underline{\Phi_i^k}=\frac{\Phi_i^k}{\tau}=\frac{L_i^k/r_i}{L_{\min}/r_{\max}}. \quad (6)$$

The period is stored in a processing table for each session and the timestamp is assigned for each packet. Using the virtual start time as the timestamp, the normalized service interval can be calculated from (3) and (6), and thus we can derive the virtual finish time; using the virtual finish time as the timestamp, the virtual start time can be similarly computed.

Modular comparison is used to select 1) packets eligible to enter the scheduler; 2) packets which should receive service.

Packets are allowed to enter the sorted-queue of the scheduler by comparing the system virtual time and their virtual start time. Packets are scheduled for transmission based on the virtual finish time.

By employing modular comparison, two binary numbers represented by ***n+1*** bits can be compared without ambiguity if the difference between them is less than 2^n. Using the notation X[*i:j*] to represent the binary number extracted from the ***i^th^*** through ***j^th^*** bits of X, with the convention that the LSB bit is the ***0^th^*** bit. For example, given X=1011101, X[5:2]=0111. A modular arithmetic comparison X>Y can be computed by the following pseudo code:

```
Boolean Modular_Comparison (X,Y)
1  if X[n-1:0] > Y[n-1:0]
2    then result = TRUE
3    else result = FALSE
4  if X[n]=Y[n]
5    then return result
6    else return NOT result
```

X[n-1:0] represents the binary number, and X[n] (the ***n^th^*** bit of X) is used to discern wraparound ambiguity [2][5]. The following condition must be satisfied:

$$|X-Y|<2^n.$$

For example, when X=110 and Y=001, which represent 6 and 1, respectively. Since the above condition must be met,

Y=X+011 rather than X=Y+101; in other words, Y>X. Thus, the result of Modular_Comparison(X,Y) is *FALSE*, implying that X is not greater than Y because of wraparound.

With this property, Reference [4] suggests that the size of timestamps has to be at least one bit larger than the largest normalized service interval $\underline{\Phi}_{\max}$.

$$\underline{\Phi}_{\max} = \frac{L_{\max} / r_{\min}}{L_{\min} / r_{\max}}.$$

Therefore:

$$Z(I_T) \geq \left\lceil \log_2 \frac{r_{\max}}{r_{\min}} + \log_2 \frac{L_{\max}}{L_{\min}} \right\rceil + 1. \qquad (7)$$

The largest period is $p_{\max} = r_{\max} / r_{\min}$. Thus, the number of bits to represent the integer part of the period must satisfy

$$Z(I_P) \geq \left\lceil \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil. \qquad (8)$$

In a cell-based scheduling system, it is tempting to use an integer representation of the timestamp, so that the system virtual time is simply increased by one each time a cell is transmitted. However, this would adversely affect the provisioning of those sessions with high bandwidth requirement. In a scheduling system with integer representation of virtual time, the period of 1,2,3 … represents service rate of 1, 1/2, and 1/3… times the link capacity. As a result, session rates between 1 and 1/2, 1/2 and 1/3 of the link capacity cannot be represented. Therefore, we need more bits after the decimal point to represent the timestamp and period of high-rate sessions.

The relative error of the service rate can be expressed in terms of the period representation with finite bits as follows:

$$\varepsilon(r_i) = \left| 1 - \frac{r_i}{\overline{r_i}} \right| = \left| \frac{\frac{1}{r_i} - \frac{1}{\overline{r_i}}}{\frac{1}{r_i}} \right| = \left| \frac{\frac{r_{\max}}{r_i} - \frac{r_{\max}}{\overline{r_i}}}{\frac{r_{\max}}{r_i}} \right| = \left| \frac{P_i - \overline{P_i}}{P_i} \right|.$$

Define $\hat{\varepsilon}(P_i)$ as the approximation of relative error of P_i

$$\hat{\varepsilon}(P_i) = \left| \frac{P_i - \overline{P_i}}{P_i} \right|.$$

Thus, $\hat{\varepsilon}(P_i) = \varepsilon(r_i)$. (9)

Therefore, to guarantee the accuracy of the service rate we need to maintain the accuracy of the approximation of the period $\hat{\varepsilon}(P_i)$. The timestamp and period are stored and used together for each packet in the system. The accuracy of both of them determines the accuracy of the service rate.

(1) The Fixed-Point Expression

Using the fixed-point expression, the minimal number of bits to represent the timestamp and period ((7) and (8)) are:

$$\begin{cases} Z(I_T) = \left\lceil \log_2 \frac{L_{\max}}{L_{\min}} + \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil + 1. & (10) \\ Z(I_P) = \left\lceil \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil. & (11) \end{cases}$$

Since scheduling involves arithmetic operations on both timestamp and period, they should maintain the same accuracy, thus having the same number of fractional bits.

$$Z(F_P) = Z(F_T) \quad . \qquad (12)$$

Therefore, the generalized accuracy of the fixed-point expression is:

$$\hat{\varepsilon}(P) = \begin{cases} \frac{2^{-(Z(F_P)+1)}}{P}, & if \quad Z(F_P) \leq Z(F_T), \\ \frac{2^{-(Z(F_T)+1)}}{P}, & if \quad Z(F_P) \geq Z(F_T). \end{cases} \qquad (13)$$

The total number of bits to represent the timestamp and period is: $N = Z(I_T) + Z(F_T) + Z(I_P) + Z(F_P)$ (14)

Consider a packet scheduling system ***S***, which is required to support service rates from 4 Kbits/s to 622 Mbits/s, and the packet size ranged from 40 bytes to 64 Kbytes. Then, $Z(I_T)$ = 29 and $Z(I_P)$ = 18. With $Z(F_T) = Z(F_P)$ = 2, i.e., ***N*** = 51, the maximum relative error of the service rate is shown in Fig. 2.

(2) The Floating-Point Expression

Similarly, in order to achieve unambiguous modular comparison, the following inequality must be satisfied:

$$Z(M_T) \geq \left\lceil \log_2 \frac{L_{\max}}{L_{\min}} + \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil + 1. \qquad (15)$$

Let δ be the number of additional bits:

$$Z(M_T) = \left\lceil \log_2 \frac{L_{\max}}{L_{\min}} + \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil + 1 + \delta. \qquad (16)$$

To maintain the same degree of accuracy of the timestamp,

$$Z(M_P) = \left\lceil \log_2 \frac{r_{\max}}{r_{\min}} \right\rceil + \delta. \qquad (17)$$

Thus,

$$Z(E_T) = \lceil \log_2 Z(M_T) \rceil, \qquad (18)$$

$$Z(E_P) = \lceil \log_2 Z(M_P) \rceil, \qquad (19)$$

$$N = Z(M_T) + Z(E_T) + Z(M_P) + Z(E_P). \qquad (20)$$

The size of the binary equivalent of the timestamp and period are $Z(M_T)+1$ and $Z(M_P)+1$, respectively. The values of E_T and E_P determine how many bits the decimal point should shift from the position just after the most significant bit. Denote $\lambda(P)=Z(M_P)-E_P$ and $\lambda(T)=Z(M_T)-E_T$. $\lambda(P)$ and $\lambda(T)$ are the number of bits after the decimal point in the period and timestamp, respectively. Thus, the relative error can be expressed as:

$$\hat{\varepsilon}(P) = \begin{cases} \frac{2^{-1}}{\sum_{i=0}^{Z(M_P)-1} B_i(M_P) \cdot 2^i + 2^{Z(M_P)}} & if \quad \lambda(P) \leq \lambda(T), \\ \frac{2^{\lambda(P)-\lambda(T)-1}}{\sum_{i=\lambda(P)-\lambda(T)}^{Z(M_P)-1} B_i(M_P) \cdot 2^i + 2^{Z(M_P)}} & else. \end{cases} \qquad (21)$$

With $\lambda(T) < \lambda(P)$, if the timestamp has more bits after the decimal point, the accuracy of the period is higher. The value of the timestamp ranges from 0 to $\underline{\Phi}_{\max}$, implying that $\lambda(T)$ ranges from δ to $Z(M_T)$. The worst case, corresponding to

$\lambda(T) = \lambda_{min}(T) = Z(M_T)-\delta$, yields the following largest relative error:

$$\hat{\varepsilon}_{\max}(P) = \begin{cases} \dfrac{2^{-1}}{2^{Z(M_P)}} & if \quad \lambda(P) \le \lambda_{\min}(T), \\ \dfrac{2^{\lambda(P)-\lambda(T)-1}}{2^{Z(M_P)}}. & else \end{cases} \qquad (22)$$

Consider the same packet scheduling system ***S*** as before. Using (18) and (19), we have $Z(E_T) = Z(E_P) = 5$.

With ($Z(M_T)$ = 33, $Z(M_P)$ = 4), i.e., N= 47, δ=4, the maximum relative error of the service rate using the floating-point expression is shown in Fig. 2.

(3) Our Earlier Proposed Expression

We have developed an expression [6], which is more efficient than the fixed-point and floating-point expressions. However, the performance in terms of number of bit can still be improved.

III. The Proposed Expression For Packet-based Schedulers

By employing the modular comparison, we propose a new expression by locating the decimal point according to the range number C of a session. Each range number C determines the decimal point, and thus the range of the period. Let C_i denote the range number for session ***i***.

A. The Compressed Timestamp Expression

For session ***i***, the timestamp-based algorithms have the globally bounded timestamp (GBT) [2]:

$S_i(t)+L_i^k/r_i \ge V(t) \ge S_i(t)-L_i^k/r_i$, *if queue of session* ***i*** *is not* ***empty*** (23)

Thus (23) can be re-written as:

$V(t) +\Phi_i^k \ge S_i(t) \ge V(t)- \Phi_i^k$, *if queue of session* ***i*** *is not* ***empty*** (24)

For example, consider session ***i*** which requires period $\boldsymbol{P_i}$=12.5. If $L_i^k = 2L_{min}$, then, $\underline{\Phi_i^k} = \boldsymbol{2P_i}$=25. Assume at the moment of V(t)= 200.75, this packet is allowed to enter the queue. Then $S_i(t)=200.75$ *and* $F_i(t)=225.75$.

0	0	1	1	0	0	1	0	0	0	.	1	1	0	0		***V(t)***
0	0	0	0	0	0	1	1	0	0	.	1	0	0	0		***P_i***
0	0	1	1	0	0	1	0	0	0	.	1	1	0	0		***$S_i(t)$***
0	0	1	1	0	1	0	1	0	1	.	0	1	0	0		***$F_i(t)$***
9	8	7	6	5	4	3	2	1	0	.	-1	-2	-3	-4		
					x	x	x	x	x	.	x					

Figure 1 The relationship of V(t), $S_i(t)$, $F_i(t)$ *and* P_i

In Fig.1, note that only those bits marked 'x' (bit 4 to bit –1) in $S_i(t)$ *and* $F_i(t)$ are different from those in $V(t)$. The number of 'x' depends on the number of significant bits of $\boldsymbol{P_i}$ and packet size L_i^k as well. Intuitively, we only need to save those bits marked 'x' as the timestamp, and compare this timestamp with those corresponding bits of the virtual system time. The ***compressed timestamp expression*** is defined as follows [2]:

If the period of session ***i*** is $\boldsymbol{P_i}$ and it is expressed with bits from bit ***m*** to ***n***, the timestamp of packet ***k*** of this session can be expressed with bits from bit $\boldsymbol{m}+l_i^k+\boldsymbol{1}$ to $\boldsymbol{n}+l_i^k$, where $l_i^k = \left\lfloor \log_2 \dfrac{L_i^k}{L_{\min}} \right\rfloor$. One extra bit in the timestamp is used to discern the wraparound ambiguity.

By using the compressed timestamp expression, the size of the timestamp can be reduced dramatically, especially for those sessions requiring higher service rates. *T'* denotes the compressed timestamp expression.

B. The Compressed Period Expression with Fixed-size

If all bits of the period can determine the accuracy, implying that the timestamp maintains the same degree of accuracy, then the maximum relative error of this expression is

$$\varepsilon_{\max}(r) = \hat{\varepsilon}_{\max}(P) = 2^{-(Z(P)+1)}. \qquad (25)$$

Let $\varepsilon^*(r)$ denote the acceptable relative error of the service rate. Select the number of bits of the period Z(P) such that the maximum relative error is less than $\varepsilon^*(r)$, then the relationship between Z(P) and $\varepsilon^*(r)$ is as follows:

$$Z(P) = \left\lfloor -\log_2 \varepsilon^*(r) \right\rfloor \qquad (26)$$

In a scheduler, period P must be larger than 1. Therefore, by using the hidden '1,' which is used in IEEE floating-point expression, one more bit can be saved. *P'* denotes the compressed period expression. Then (23) can be rewritten as:

$$Z(P') = \left\lfloor -\log_2 \varepsilon^*(r) \right\rfloor - 1 \qquad (27)$$

For example, if $\varepsilon^*(r)$ =1%, 5 bits are needed to represent the period. The relative error with 6 bits (including the hidden '1') expression < 2^{-7} = 0.78%. Note that in order to discern the wraparound ambiguity, Z(T) should be one bit larger than Z(P). To take the hidden '1' into consideration, the number of bits to represent the timestamp can also be calculated as

$Z(T')=Z(P')+2$. (28)

Note that using the compressed period instead of the service interval can reduce the access number of off-chip memory. When a session becomes active, the header of the newly packet is passed to network processor, with the period of this session, the interval can be computed and saved in on-chip memory.

C. Range Number

When the timestamp and the period maintain the same degree of accuracy, the maximum relative error is consistent over the whole range of service rates. In order to generate the compressed timestamp, a range number is also needed to indicate the position of the decimal point of the timestamp and period. For example, when C_i=4, then $2^4 \le P_i < 2^5$. Thus the maximum range number can be computed as

$C_{max} = \lceil \log_2(r_{max}/r_{min}) \rceil$. (29)

Thus, the number of bits to represent the range number

$Z(C) = \lceil \log_2 C_{max} \rceil$. (30)

P_i can be obtained from P_i' by right shifting the decimal point C_i bits. The timestamp can be similarly obtained with its compressed form and packet size. With these properties, we 1) compute the period for each session and save it in the

compressed form, 2) compute the timestamp and save it in the compressed expression, 3) rebuild the timestamp to the complete form from the compressed expression, and 4) the scheduler sorts packets by comparing their complete timestamps.

$$N = Z(C) + Z(P') + Z(T'). \quad (31)$$

Consider the same packet scheduling system ***S*** as before. From (25)-(31), the relationship between the range number and corresponding range of the period and the maximum relative error are shown in TABLE I.

With Z(C) =5, Z(T')=7, and Z(P')=5 (i.e., N=17), the maximum relative error of the service rate with the compressed expression is shown in Fig. 2.

TABLE I

RANGES OF THE PROPOSED EXPRESSION FOR SCHEDULER S

Range # C	P_{min}	P_{max}	Z(P')	Z(T')	N	ε_{max}
00000	1	2	5	7	17	2^{-7}
00001	2	4	5	7	17	2^{-7}
00010	4	8	5	7	17	2^{-7}
00011	8	16	5	7	17	2^{-7}
00100	16	32	5	7	17	2^{-7}
00101	32	64	5	7	17	2^{-7}
00110	64	128	5	7	17	2^{-7}
00111	128	256	5	7	17	2^{-7}
01000	256	512	5	7	17	2^{-7}
01001	512	1024	5	7	17	2^{-7}
01010	1024	2048	5	7	17	2^{-7}
01011	2048	4096	5	7	17	2^{-7}
01100	4096	8192	5	7	17	2^{-7}
01101	8192	16384	5	7	17	2^{-7}
01110	16384	32768	5	7	17	2^{-7}
01111	32768	65356	5	7	17	2^{-7}
10000	65356	130712	5	7	17	2^{-7}

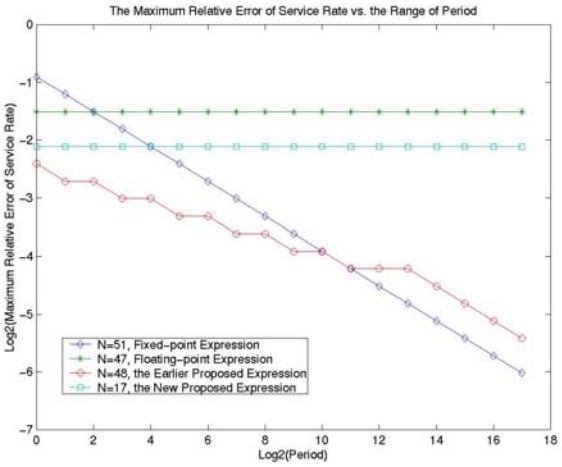

Fig. 2 The comparison of various expressions in terms of the maximum relative error

IV. THE PROPOSED EXPRESSION FOR CELL-BASED (ATM) SCHEDULERS

The cell-based scheduler can be considered as a special case of the packet-based scheduler, with $L_{max}=L_{min}$= constant. Equations derived for the fixed-point, floating-point and our proposed shifting fixed-point representations still hold. Note that in this case,

$$\log_2 \frac{L_{\max}}{L_{\min}} = 0 \text{ and } \underline{\Phi_i} = P_i.$$

Equations (25)-(31) still hold.

Suppose that the service rate of the cell-based scheduling system S' ranges from 4kbps to 622Mbps and the cell size is 40 bytes. Again, 1% relative error of service rate is considered acceptable. With Z(C) =5, Z(T')=7, and Z(P')=5 (i.e., N=17), comparable result similar to the one shown in Fig. 2 is obtained, implying that the proposed expression has a higher accuracy for cell-based schedulers.

V. IMPLEMENTATION ISSUES

In timestamp-based schedulers, there are always three arithmetic and relational operations: 1) addition, 2) modular comparison, and 3) multiplication and division. We introduce two more operation in the proposed method: 1) generating the compressed timestamp; 2) rebuilding the timestamp. With the fixed-point representation, it is very easy to perform operations such as addition and modular comparison. With the floating-point expression, we need to first shift mantissa of both timestamp and period, and then perform addition or comparison. The proposed expression uses two additional operations to adjust the decimal point by range number as compared to the fixed-point expression. This can be readily realized by hardware with some extra logic operations.

VI. CONCLUSIONS

In this paper, we have developed a more efficient expression, which can be implemented in high-speed switches, to represent timestamp and period in packet-based and cell-based scheduling systems. It is applicable to any timestamp-based scheduler.

In comparison with the normal fixed-point, floating-point representations, and our earlier proposal, the newly proposed expression can achieve better performance in terms of the number of bits and accuracy. We have also derived the formula to calculate the minimum number of bits to represent the timestamp and period that meets the system requirement (i.e., $\varepsilon_{max}(r)$, L_{max}, L_{min}, r_{max} and r_{min}).

REFERENCES

[1] A. Varma and D. Stiliadis, "Hardware Implementation of Fair Queuing Algorithms for Asynchronous Transfer Mode Networks," *IEEE Communication Magazine*, December 1997, pp. 54-68.

[2] D.C. Stephens, J.C.R. Bennett and Hui Zhang, "Implementation Scheduling Algorithms in High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, Vol.17, No.6, June 1999, pp. 1145-1158.

[3] J.C.R. Bennett and Hui Zhang, "WF^2Q: Worst-case fair weighted queuing," *Proc. IEEE INFOCOM'96*, San Francisco, CA, pp. 120-128

[4] J.L. Rexford, A.G. Greenberg, and F.G. Bonomi, "Hardware-efficient fair queuing architecture for high-speed networks," *IEEE INFOCOM'96*, San Francisco, CA, pp. 638-646.

[5] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated Volume 2: The Implementation*, Reading, MA: Addison-Wesley, 1995, pp. 807-812.

[6] D. Wei, J. Chen and N. Ansari, "An Efficient Expression of Timestamp and Period in Packet-based and Cell-based Schedulers," to be presented at ICC 2001, Helsinki, Finland, June 11-14, 2001.