Prioritized Traffic-Dependent Task Scheduling Approach in Embedded Operation System

Jin-Chang Wang, Yau-Hwang Kuo, Shu-Mei Guo and Jar-Shone Ker*

Department of Computer Science and Information Engineering

National Cheng Kung University

and

Department of Electronic Engineering*

Kun Shan University of Technology

Abstract

This paper proposes a PRIoritized trafFIC-dependent (PRIFIC) task scheduling scheme based on the traffic prediction approach. The PRIFIC manages network tasks by dynamically allocating system resources, so that it can not only prevent the network tasks from starvation state but also improve the utilization of system resources. The PRIFIC is constructed by employing an eigentraffic feature extraction model, a traffic prediction model, and an adaptive task scheduling model. Besides, an additional reference algorithm which is called principal component analysis (PCA) is adopted to reduce the dimensions of traffic features. It can efficiently reduce the computation complexity of the proposed PRIFIC algorithm.

In summary, PRIFIC has the advantages of traffic prediction and QoS-enabling. According to the simulation results, the proposed PRIFIC exhibits excellent task scheduling performance in both constant traffic and random traffic applications.

1. Introduction

In a network embedded device, the embedded operation system usually has more than two kinds of tasks; one kind of tasks is the tasks for network communication jobs, and the second kind of tasks is the tasks for other jobs, such as the system jobs and user application jobs.

A task can be characterized by the following attributes,

1. Priority: A task's priority defines the scheduling preference of the task relative to other jobs in the OS.
2. Start time: The start time defines when a task can be executed. No task can be executed before its start time.
3. Finish time: The finish time of a task is its deadline. At a task's finish time, even if the task has not finished, it must be stopped.

These attributes must be properly assigned by a task scheduler. Besides, a task scheduler is responsible to allocate system resources to each task according to the task scheduling policy. So the task scheduling policy is very important in an operation system.

Task scheduling policy is the task scheduling method of the embedded OS that allows the scheduler adaptively manages the task service time according to different policy, priority or reasonable system resource allocation such as system memory, processing capability. Presently, the available task scheduling algorithms can't efficiently allocate the system resource for individual tasks, because they are not good approaches for network embedded systems.

Round-Rabin scheduling [3] and priority scheduling [4] are well-known task scheduling policies, they are ordinary and simple methods. When we apply these scheduling policies in network systems, it is very possible to result in some serious

problems, such as the data loss, because they don't consider the effect of network environment, for example, the burst and un-balanced heavy traffic. Hence, the task scheduling algorithm must be extended to let the scheduling policy more suitable in the multimedia network environment.

In general, the embedded network device usually has limited system memory and limited processing capability that might cause undesirable problems for the transmission and receive tasks, such as the transmission or receive buffer full, and the packet loss. Because the characteristic of network traffic is unpredictable, and the OS usually allocates few memory to the transmission or receive task, the scheduler must has adaptive scheduling capability and policy to avoid task buffer full.

To guarantee the quality of service [2], all of the task scheduling algorithms should not only devote to enhance the utilization of buffer but also to reduce the data loss probability. When the buffer overflows, the incoming data will be discarded. On the contrary, when the buffer underflows, no data is available in the receiver's buffer. When there is no available data in sender's buffer to transmit, the receiver will starve. As results of buffer overflow and underflow, some artifacts at the receiver end will be easy to observe. To efficiently manage the task service time, we need to develop a robust task scheduling mechanism to conquer all the problems mentioned above.

2. Prioritized Traffic-Dependent Task Scheduling

In the PRIoritized trafFIC-dependent task scheduling (PRIFIC) structure, there are three main models in the structure, including the Traffic Feature Extractor, Traffic Predictor and Traffic-Dependent Task Scheduler. The structure of the Prioritized Traffic-Dependent Task Scheduling is shown in Fig. 1.

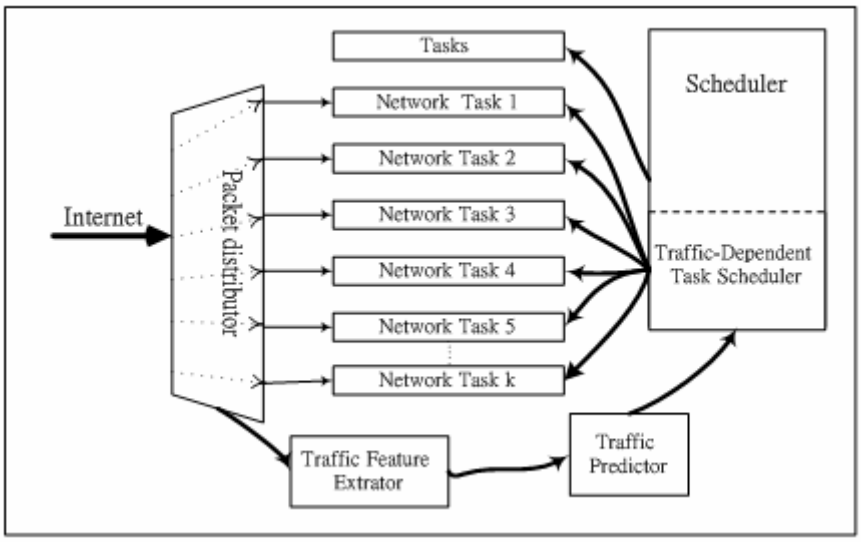

Fig. 1. Structure of Prioritized Traffic-Dependent Task Scheduling

2.1 Prioritized Network Tasks

Based on the requirement of Quality-of-Service networking on the multimedia Internet, a prioritized characteristic is defined in IEEE 802.1P standard [1]. The IEEE 802.1P standard realizes the Quality of Service protocol on the MAC level. Today, the Internet only provides best effort service. Since rise of multimedia communication application, the importance of providing a quality of service mechanism ought not to be neglected.

Today, the network devices supporting IEEE 802.1P standard [1] are more and more numerous, so we propose a prioritized traffic-dependent task scheduling algorithm where the priority is based on IEEE 802.1P standard. Our algorithm can process not only prioritized packets but also normal packets. The normal packets will be distributed to the lowest priority buffer that will let our algorithm adapt network environment. According to the user application requirement, the user can define proper priority for the different application.

Thus, our task scheduling algorithm is very elasticity, the priority of task depends on the buffer size of each task, the meaning is that when the task buffer size is smaller

than other tasks, the task has larger priority. Each task has different priority, and the task scheduler will adaptively schedule all tasks according to the prioritized traffic-dependent algorithm.

If user defines k priorities, we must maintain k network tasks. The k network tasks process different priority packets respectively, when the packets arrived different network interface, the packet distributor must distribute the packets to appropriate network task according to its priority.

2.2 Traffic-Dependent Task Scheduling

Presently, most frequently used task scheduling algorithms don't suit the multimedia communication network, such as round-robin task scheduling algorithm, and priority task scheduling algorithm. The reason is that those algorithms don't consider the variation characteristic of traffic and the concept of Quality of Service.

In this paper, a traffic-dependent scheduling algorithm is developed to model the relationship between the traffic states and the task scheduling policies. The traffic states will be a compound statement that involves the number of bytes, the number of packets, the average byte count of four latest sampled points, the byte count differences of two adjacent samples among four latest sampling points. The traffic predictor will use such traffic information to predict traffic flow, and the traffic-dependent task scheduler will use the prediction result to adaptively schedule the tasks.

2.3 Traffic Prediction Algorithm

In this section, we will discuss the proposed traffic prediction scheme. The proposed traffic prediction scheme is called TPEE because it is based on the eigenfeature extraction scheme [6].

The TPEE comprises three models. They are the Traffic Feature Extractor, Traffic Predictor, and the Traffic-Dependent Task Scheduler.

2.3.1 Traffic Feature Extractor

This model will extract traffic features in a sampling time, and it also provides feature set to the traffic predictor. With sufficient traffic features and adopting robust control scheme, the system resource will be utilized more efficiently. Fig. 2 shows the block diagram of the traffic feature extractor.

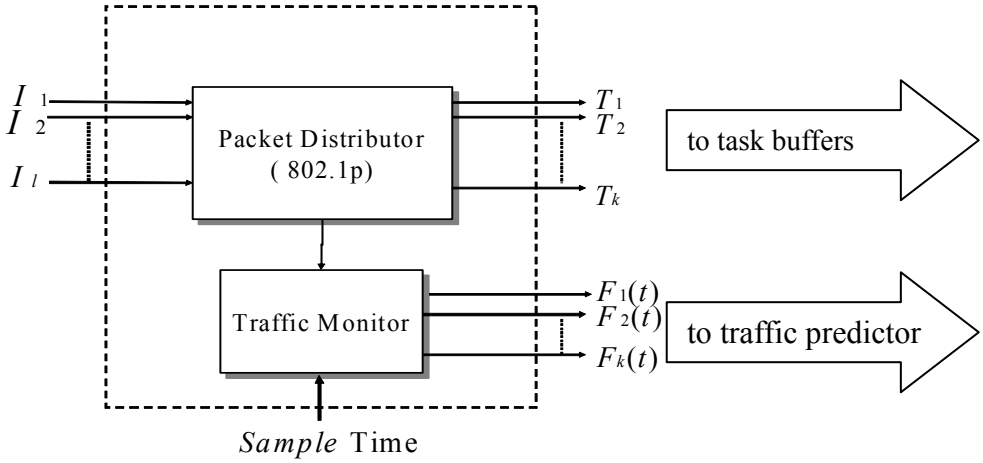

Fig. 2. Block diagram of traffic feature extractor.

As liiustrated in Fig. 2, the Packet Distributor can distribute all packets to different priority tasks according to the priority information of each received packet from network interfaces, $I_1, I_2 \ldots, I_l$, and can provide traffic information to the traffic monitor.

The traffic monitor is responsible to calculates the features $F_k(1), F_k(2) \ldots F_k(m)$, where $F_k(m)$ represents the *m*-th traffic feature set, for each sampling period. In our design, the traffic feature set, *F,* is represented in a six-tuple vector, $[P_1\ P_2\ P_3\ P_4\ P_5\ P_6]$, so that we can perform some vector operations on *F*.

In this paper, the traffic feature set involves six features to represent. The first feature is the number of packets between two sampled points, t can reveal the number of packet variation state. The second feature is the number of packet bytes, it is the traffic flow efor last sampling period, and it can reveal the traffic load variation state.

The third feature is the average byte count of four latest sampled points, it can also capture the variation of traffic flow. The remainder three features are the byte count differences of two adjacent samples among four latest sampling points. The intention of these differences is the traffic variation among four latest sampling points; it can reveal the traffic variation. Once the defined traffic features are extracted, we can use the traffic feature vectors to derive the Eigentraffics Feature vectors, with some vector operations.

2.3.2 Calculating Eigentraffics and Traffic Prediction

The idea of using Eigentraffics features was motivated by a technique developed by Sirovich and Kirby [6] for efficiently representing pictures of faces using eigenfaces analysis. They argued that a collection of face image can be approximately reconstructed by storing a small collection of weights for each face and a small set of standard pictures.

It occurred to us that if a multitude of traffic feature can be reconstructed by weighted sums of a small collection of characteristic features, then an efficient way to learn and predict the traffic can be used to build the characteristics-specific traffic by comparing the feature weights with the weights of known individuals.

The following steps summarize the prediction process:

1. Initialization: Acquire the history set of traffic feature and calculate the Eigen-traffics, which define the Eigentraffic space.
2. When a new traffic feature is encountered, a set of weights based on the input traffic feature and the M traffic feature set must be calculated by projecting the input traffic onto each of the Eigentraffics.
3. Determine the most similar feature by checking the shortest distance between the feature and other features in the feature space, and use the most similar feature to determine the next feature in feature space.

4. If the prediction error, i.e., the difference between prediction result and real traffic flow, is seen several times over the tolerable bound, the Eigentraffics must be re-calculated to ion scheme.

The Eigentraffics are a set of orthonormal basis vectors computed from a collection of history traffic features. Let the traffic feature F be a two–dimensional N by N array of intensity values, or a vector of dimension N^2.A traffic feature of size 1 by 6 describes a vector of dimension 6, or, equivalently, a point in 6-dimensional space. We can find the vectors which account for the distribution of traffic feature within the entire traffic feature and vector is of length N^2, describes an N by N traffic feature, and is a linear combination of the original traffic feature.

Because these vectors are the eigenvectors of the covariance matrix corresponding to the original traffic features, and because they are traffic features similar in appearance, we refer them as “Eigentraffics”.

Let the history set of traffic features be $F_{j,1}, F_{j,2}, F_{j,3}, ..., F_{j,m}$ and the average feature of the set is defined by $\overline{F_j} = \frac{1}{m}\sum_{i=1}^{m} F_i$, Each feature differs from the average by the vector $\phi_j = F_{j,i} - \overline{F_j}$, and PCA [7] is applied to the set of history traffic features to find the *N* eigenvalues of the covariance matrix

$$\frac{1}{m}\sum_{i=1}^{m}(F_i - \overline{F_j})(F_i - \overline{F_j})^T \qquad (1)$$

where $\overline{F_j} = \frac{1}{m}\sum_{i=1}^{m} F_i$ is the average of the ensemble. The eigenvalues of the covariance matrix are calculated. Let $\Phi_{j,k}$ be the eigenvector corresponding to the *k*th largest eigenvalue. Each history traffic feature F_j is projected on the eigentraffic space as a point

$$x_i = \Phi_j{}^T (F_{j,i} - \overline{F_j}) \quad (2)$$

where $\Phi_j = [\Phi_{j,1}, \Phi_{j,2}, \cdots, \Phi_{j,n}]$, and is used as a prototype feature point.

Given a query traffic feature F_c, which will be used to predict the next traffic feature. At first, its projection on the eigentraffic space is calculated as

$$x_c = \Phi_j{}^T (F_c - \overline{F_j})$$

Let $x_1, x_2, \ldots, x_m$ be *M* distinct prototype feature points belonging. The feature Euclidean distances between x_c of the query and each prototype feature point x_i are calculated for each query point. The distances are sorted in ascending order, each being associated with a similar identifier. The feature distance is the first rank distance

$$d(x, x_{i*}) = \underset{1 \le j \le M}{Min}\, d(x, x_i) \quad (3)$$

The first rank gives the traffic prediction composed of the best matched feature point x_{i^*}. The best matched feature point x_{i^*} has the most similar feature to the query point x, then we select x_{i^*+1} to be the final prediction result.

After we get the prediction result, we will compare the result with the real traffic flow. When the prediction error exceeds the tolerable bound, we will perform the Eigentraffics re-calculation algorithm.

2.3.3 The Eigentraffics Calculation Scheme

In our algorithm, we define three parameters: the threshold of tolerable prediction error ($E_{\max}$), threshold of successive times of bad prediction (ε), and tolerable recalculation rate (β). The first one, $E_{\max}$, is a maximum tolerable threshold parameter; the second one, ε, is tolerable successive times of prediction error over threshold parameter. When the prediction error exceeds $E_{\max}$ and the successive times of prediction error is greater than β, we must add ε feature sets into the history set and then we must re-calculate the eigentraffic space.

In our design, the algorithm has three stages to adjust the size of history set, *m*. In the fist stage, the size of *m* will increase with ε sets per one re-calculation until the re-calculation count is less than β. After the first stage finished, the second stage will start. In the second stage, the size of *m* will be fixed. When ε sets are added to the history set per each re-calculation, the algorithm will also omit the forefront ε sets in the history set, but if the algorithm finds that the status doesn't have satisfied β condition, the second stage will stop and the algorithm stage will reenter first stage, if the status has satisfied β condition yet, the algorithm stage will enter the third stage, the third stage is that when algorithm status has satisfied once β condition, the *m* will decrease of forefront ε history sets, and if the algorithm finds that the status doesn't have satisfied β condition, the third stage will stop and the algorithm stage will reenter first stage.

2.4 Adaptive Task Scheduling

The Task Scheduler unit of the proposed PRIFIC model is governed by an adaptive scheduling algorithm. In the algorithm, we do adaptive scheduling according to different priorities and variations in the traffic flow.

We use the eigentraffic prediction to predict next possible arriving traffic load, we consider that the next arriving traffic that can have large influence on tasks, so we will predict next arriving traffic feature according to recently arrived traffic load. That is, a set of traffic features $F_j(i), F_j(i-1), F_j(i-2), ..., F_j(i-m)$, extracted during the past sampling time periods $T_j(i), T_j(i-1), T_j(i-2), ..., T_j(i-m)$, will be used to predict the number of traffic load $F_j(i+1)$ which will arrive in the next time period $T_j(i+1)$.

Since the traffic load is just an element of network traffic feature, once we get the

feature $F_j(i+1)$, we can know the traffic load, $\hat{A}_j(i+1)$. In addition, we have to find the possible queue length at the end of $T_j(i+1)$ with the following expression:

$$\hat{Q}_j(i+1) = \min[B_j, \max(0, Q_j(i) + \hat{A}_j(i+1) - C_j(i))] \quad (4)$$

where B_j is the total queue size of task j, $Q_j(i)$ is the measured queue length of task j at end of time period $T_j(i)$, $\hat{A}_j(i+1)$ is the predicted traffic load of task j in the next time period $T_j(i+1)$ and $C_j(i)$ is the number of bytes of task j to be served in one sampling time period.

Equ. 5 is then used to find the regulation parameter $\hat{r}_j(i+1)$, which represents the estimation of the proportion of service time to serve the task j in time period $T_j(i+1)$.

$$\hat{r}_j(i+1) = \hat{w}_j(i+1) \times \hat{q}_j(i+1) \quad (5)$$

where $w_j(i+1)$ is a weighting function,

$$\hat{w}_j(i+1) = 1 \Big/ 1 + B_j - \hat{Q}_j(i+1) \quad (6)$$

and $\hat{q}_j(i+1)$ is the queue state,

$$\hat{q}_j(i+1) = \hat{Q}_j(i+1) \Big/ B_j \quad (7)$$

In our design, we define a buffer size base, *BSB*, and the buffer size of priority level 1 is $k_1 * BSB$, the buffer size of priority level 2 is $k_2 * BSB$, the buffer size of priority level 3 is $k_3 * BSB$, the buffer size of priority level 4 is $k_4 * BSB$, the buffer size of priority level 5 is $k_5 * BSB$, the buffer size of priority level 6 is $k_6 * BSB$, and $k_1 = 1$, $k_2 = 2$, $k_3 = 3$, $k_4 = 4$, $k_5 = 5$, $k_6 = 6$.

Suppose there are k network tasks, so we have k transmission buffers, and therefore, we can get $\hat{r}_1(i+1), \hat{r}_2(i+1), \ldots,$ and $\hat{r}_k(i+1)$ from equation (5). Finally, we can use equation (8) to allocate the available processing capability, APC, to each task $C_1(i+1), C_2(i+1), \ldots.,$ and $C_k(i+1)$ respectively.

$$C_k(i+1) = APC \times \hat{r}_k \Big/ \sum_{j=1}^{k} \hat{r}_j \tag{8}$$

3 Analysis of Dynamic Prediction Algorithm

In the real network environment, the random traffic is a common used traffic model. Regarding to random traffic, the number of bytes and number of packets between two samples are characterized by two random variables

We consider that each task buffer has random arrival rate, the observation of traffic rate at time t is represented as $\lambda(t)$, where $\lambda(t)$ is the random variable of traffic rate. To run the simulation, we set the parameters as the values shown in Table 1 and simulate with the testing data in Fig. 3. Table 2 summarizes the corresponding simulation results. In our experiments, the testing data has 2000 sampling points and after the 1100th sampling points, the testing data within 0 and 900 are duplicated, so after the 1100th sampling points, we can explore the inside testing performance of the prediction algorithm.

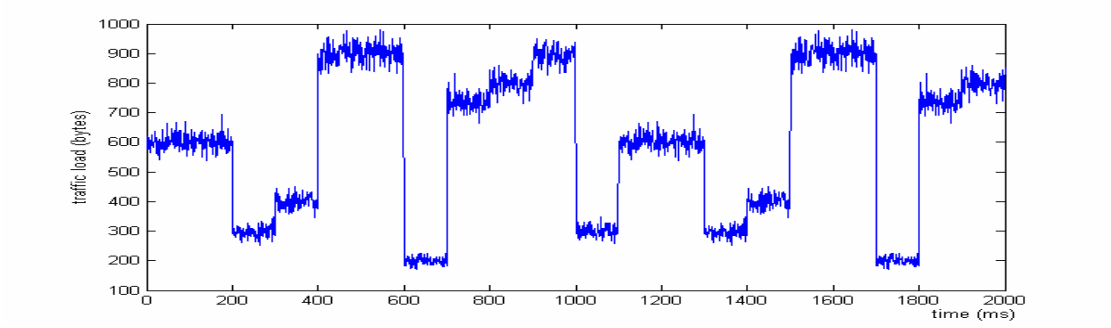

Fig.3. Testing data for simulation.

Table 1. Simulation parameters

Parameter	Value
The initial size of history data set (m)	0
Tolerable recalculation rate (β)	3
Threshold of tolerable prediction error ($E_{\max}$)	$\pm 5\%$
Threshold of successive times of bad prediction(ε)	2

Table 2. Summary of simulation results of reduction degree

Reduction Degree	0	1	2	3	4	5
Size of history set	425	425	421	473	471	479
Run time (second)	31.3	27.8	25.1	26.1	24.4	20
Mean prediction error (%)	5.2	5.3	5.4	5.8	5.9	5.9
Re-calculation times	211	211	209	235	234	238

After analyzed the simulation results, we find some exciting results. The inside testing has smaller prediction error than that of outside testing. When an unknown traffic flow is fed to our prediction algorithm, the traffic information will be extracted, and stored in the history set. The reduction degree doesn't have obvious influence in the mean prediction error, but it has obvious influence in the simulation run time, size of the history set, and number of re-calculation times. It saves about thirty percents of simulation run time under the maximum reduction degree, with slightly increasing in the size of the history set and the number of re-calculation times.

If we reduce the feature degree, we know that the traffic feature information must have some loss. When the algorithm performs prediction, the larger reduction degree is, the approximate result may be led to larger error. However, according to the simulation results, the prediction error doesn't directly proportion to reduction degree; it mainly depends on the actual traffic flows. In the following paragraphs, we do further experiments to get the relations among the important parameters $E_{\max}$, ε and the mean prediction error. For such an experiment, β equals 3, the reduction degree

is 5, and the testing data is shown in Fig 3. The simulation result is summarized in Table 3.

Table 3. Relations among the mean prediction error, $E_{\max}$ and ε.

	$E_{\max} \pm 2\%$	$E_{\max} = \pm 5\%$	$E_{\max} = \pm 8\%$
$\varepsilon = 1$	3%	4.9%	7.4%
$\varepsilon = 3$	4.7%	6.5%	26%
$\varepsilon = 5$	6%	26%	27%
$\varepsilon = 8$	7%	40.2%	42%

According to Table 3, if we want to get more precise prediction result, the parameters $E_{\max}$ and ε must be set to smaller number. However, such a setting will lead to larger size of the history set and higher re-calculation possibility.

In our design, the $E_{\max}$ is $\pm 5\%$ and in the following, we do further experiments to get the relations among the important parameter ε, the mean prediction error, run time and re-calculation. For such an experiment, β equals 3, the reduction degree is 5, and the $E_{\max} = \pm 5\%$ and the testing data is shown in Fig. 3. The simulation result is summarized in Table 4

Table 4. Simulation results for various ε (with $E_{\max} = \pm 5\%$)

ε	1	2	3	4	5	6
Size of history set	668	479	316	115	28	15
Run time (second)	54.5	20	6.2	4.5	2.5	1.6
Mean prediction error (%)	4.9	5.9	6.5	7.9	26	39
Re-calculation times	665	198	87	40	48	53

According to Table 4, if we want to get least prediction speed, the parameters ε must be set to larger number. However, such a setting will lead to larger mean prediction error. In our design, we consider the prediction algorithm must be finished

in a short time and the prediction error must be restricted in the smaller value, the value of ε is set to 2.

4 Simulation Results and Performance Evaluation

The following simulations will evaluate the performance of PRIFIC comparing with priority task scheduling (PTS) [4] and fairness task scheduling (FTS) [3]. We will compare the mean loss rate, mean buffer length, and mean turnaround time under the limited memory size and limited processing capability for PREFIC, PTS, and FTS.

The simulation environment has six independent network interfaces and the packet distributor distributes the arriving packet to different task, according to IEEE 802.1P priority [1].

The required information of different network task generated by the Poisson process are the packet size and number of packets. Let a traffic load arrival, $\{TL(t),\ t \geq 0\}$, be a Poisson process with rate λ_r and let a packet arrival, $\{PA(t),\ t \geq 0\}$, be a Poisson process with rate λ_p. Let $0 < x_1 < x_2 < \ldots$ be the successive packet arrivals.

From above definition, we first have the distribution of received traffic load rate of a class buffer given by

$$P(t_n \leq t) = 1 - e^{-\lambda_r t} \quad \text{for } t \geq 0 \tag{9}$$

and the distribution of received packet rate of a class buffer given by

$$P(t_n \leq t) = 1 - e^{-\lambda_p t} \quad \text{for t} \geq 0 \tag{10}$$

In the definition of packet arrivals and traffic load arrivals, the traffic flow is defined as a sequence of number of bytes, and number of packets between the two adjacent sampling times is regarded as a random variable with an exponential distribution.

We use equ. 9 and 10 to generate 6 testing traffic flows, each testing traffic flows have 100 sampling points. We have 6 history traffic flows and those history traffic flows are extracted in past history time, each history traffic flow has 100 traffic features.

In this simulation, we use the largest reduction degree of traffic feature, because we consider the algorithm must be finished in the shortest time, so the reduction degree is 5. The simulation parameters are shown in Table 5.

Table 5. Simulation parameters

Parameter	Value
Size of initial history data set (m)	100
Reduction degree (RD)	5
Tolerable recalculation rate (β)	3
Threshold of tolerable prediction error ($E_{\max}$)	$\pm 5\%$
Threshold of successive times of bad prediction (ε)	2

In this simulation, when the available memory size is 63K bytes, we get the results shown in Fig. 4. The figure indicates that our algorithm has smaller processing capability requirement than the other two algorithms, and according to Fig. 4(c), we know that if available processing capability is greater than 7500 bytes/ms, the mean loss rate approximates zero, so we select the value of 7500 bytes/ms as the available processing capability value for the rest simulations in this section.

When the available processing capability is fixed to 7500 bytes/ms, the simulation results for the required memory are shown in Fig. 5. Fig.5(c) indicates that when we allocate about 7000 bytes memory to buffer 5, there is no any data loss on each buffer. Under our buffer allocation scheme, we need (7000/2) x 21 = 73500 bytes for the overall available memory. Fig. 5(c) indicates that when we allocate over 20K bytes to buffer 1, the data loss still exists for the PTS algorithm. Fig. 5(a) indicates

that when we allocate over 10K bytes to buffer 5, the data loss still exists for the FTS algorithm. Comparing Fig. 5(a), Fig. 5(b) and Fig. 5(c), we know that our algorithm has the smallest available memory requirement than the other two algorithms.

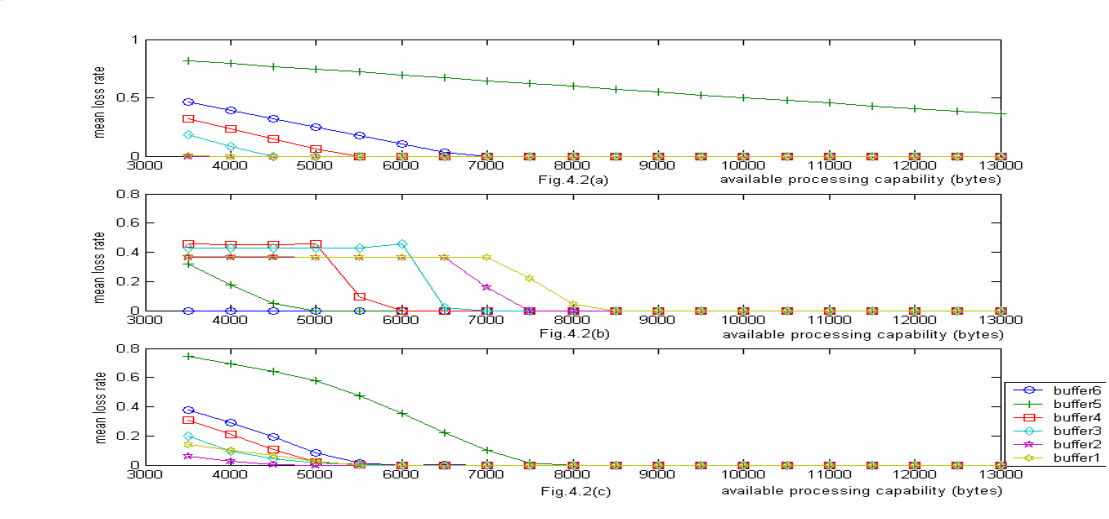

Fig. 4. Mean loss rates for the FTS, PTS and PRIFIC algorithms under different processing capabilities.

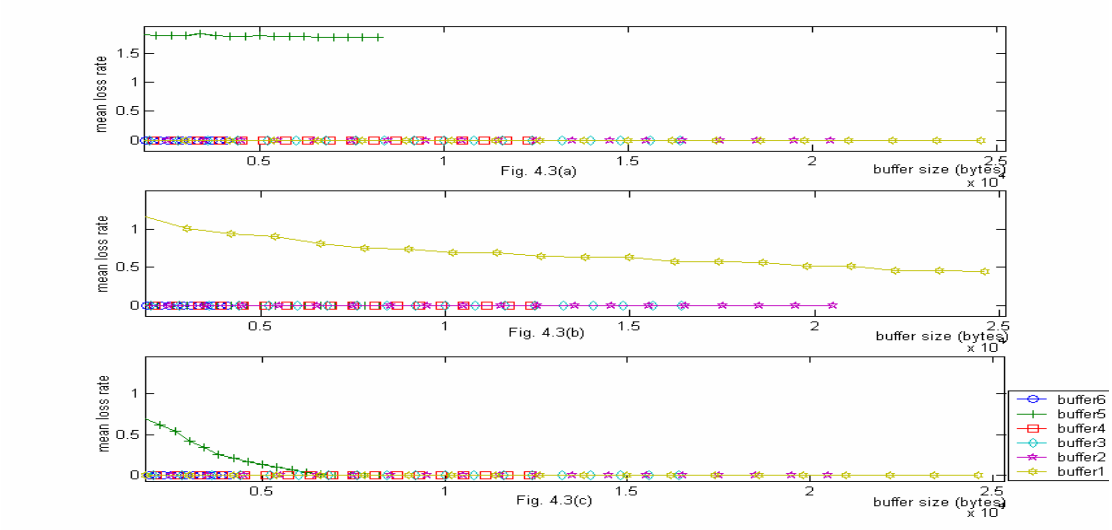

Fig. 5. Mean loss rates for the FTS, PTS and PRIFIC algorithms under different available buffer size conditions.

Fig. 6 shows the simulation results of the mean buffer length. It indicates that our algorithm usually has the longer mean buffer length than the other two algorithms, but when the buffer size is greater than a certain value, our mean buffer length will

approximate to constant length, the other two algorithms don't have this feature and we can use the feature to guarantee the turnaround time.

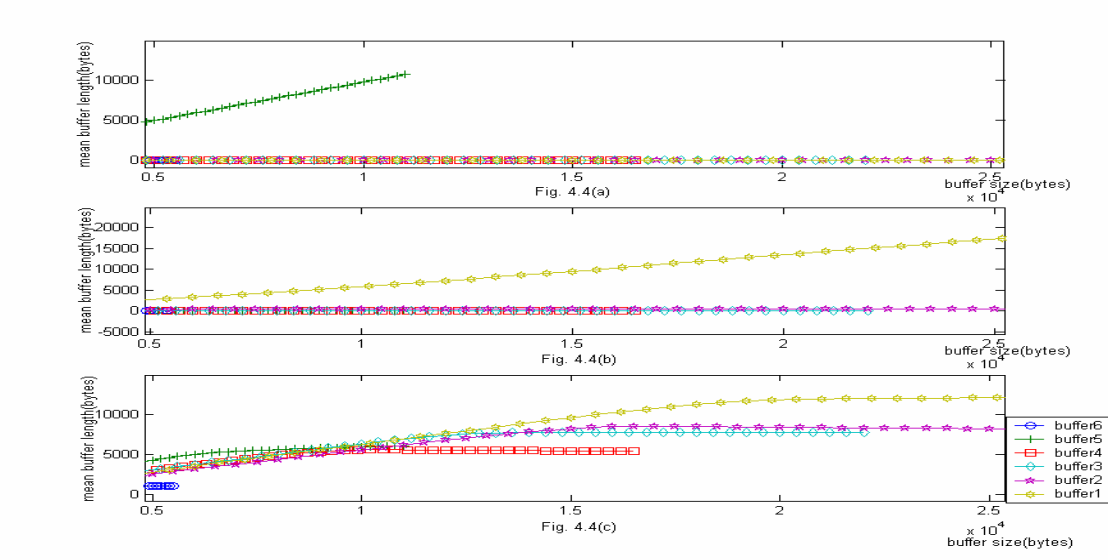

Fig. 6. Mean buffer length for the FTS, PTS and PRIFIC algorithms under different available buffer size condition.

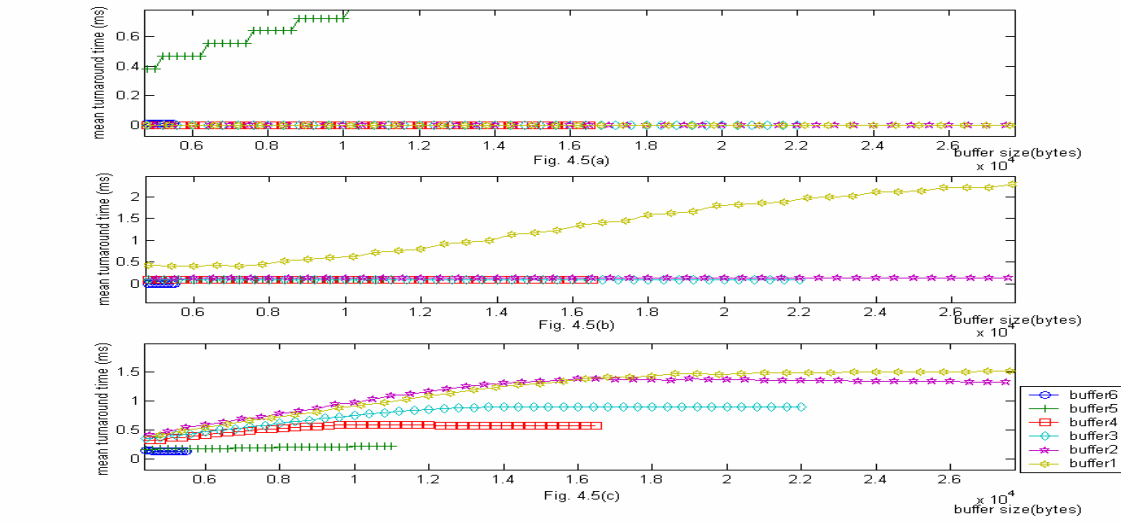

Fig. 7. Mean turnaround time for the FTS, PTS and PRIFIC algorithms under different available buffer size condition.

Fig. 7 illustrates the simulation results of the mean turnaround time. It indicates that our algorithm usually has larger mean turnaround time than the other two algorithms, but when the buffer size increase over a certain value, our mean

turnaround time will approximate to constant time, the other two algorithms don't have this feature and we can use the feature to guarantee the turnaround time.

5. Conclusions

In this paper, we have presented a prioritized traffic-dependent task scheduling algorithm called PRIFIC. PRIFIC adaptively adjusts the available resource to respond with respect to the variance of the incoming packet rate. According to the buffer state and the traffic features from a traffic feature extractor, PRIFIC can adaptively adjust the service rate for each task. A reference algorithm called the principal component analysis (PCA) [7] is adopted to reduce the degree of traffic feature; it can efficiently reduce the computation complexity of the PRIFIC algorithm.

For the viewpoint of operating systems, in order to support the QoS feature, IEEE 802.1P standard can be adopted. So that higher level applications do not need to care about the QoS issue.

According to the simulation results about average data loss rate, average buffer length and average turnaround time, PRIFIC indeed has the following advantages,

1. more efficient utilization of available memory space,
2. more efficient utilization of available processing capability,
3. lower data loss rate, and
4. guaranteed turnaround time.

To improve the performance of PRIFIC, we may need a mechanism to adjust the algorithm parameters $E_{\max}$, ε and β. Furthermore, if we can dynamically determine the buffer size of each task according to different delay requirement, the QoS functionality of PRIFIC will be improved.

References

[1] Lidinsky, W., IEEE Standard P802.1D Information technology Telecommunications and information exchange between systems - Common specifications - Part 3: Media Access Control (MAC) Bridges: Revision, 24.11.1997 [defered 14.3.1999]

[2] Shenker S. et. al., Specification of Guaranteed Quality of Service. RFC 2212, September 1997.

[3] Yu-Chung Wang, Kwei-Jay Lin, "Implementing a General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel," IEEE Real-Time Systems Symposium 1999, 246-255.

[4] C.L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, 20(1), 46-61, 1973.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," Journal of Internetworking Research and Experience, 3-26, October 1990.

[6] M. A. Turk and A. P. Pentland, "Eigenfaces for recognition," J. Cognitive Networks, vol. 3, no. 1, 71-86, Mar. 1991.

[7] K. Fukunaga, Introduction to statistical Pattern Recognition , 2nd ed. Boston, MA: Academic, 1990.

[8] L. Sirovich and M. Kirby, "Low-dimensional procedure for the charactization of human faces," J.Opt. Soc. Amer. A, vol. 4, no 3, 529-524, Mar. 1987.

[9] John Wiley and Sons, Queueing Systems, WILEY, 1997.

[10] M. Barabanov and V. Yodaiken, "Introducing Real-Time Unix," Linux Journal, No. 34, Feb 1997.

[11] Y.C. Wang and K.J. Lin, "Enhancing the real-time capability of the Linux kernel," Proc. of 5th RTCSA'98, Hiroshima, Japan, Oct 1998.

[12] J. Hyman, A Lazar, and G. Pacitici, "Real-Time Scheduling with quality of Service Constraints," IEEE JSAC, vol. 9, no. 9, 1052-63, September 1991.