Computer Networks 52 (2008) 3392–3404

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

On applying fair queuing discipline to schedule requests at access gateway for downlink differential QoS

Shih-Chiang Tsao [a,*], Yuan-Cheng Lai [b], Le-Chi Tsao [a], Ying-Dar Lin [a]

[a] Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan
[b] Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 17 December 2007
Received in revised form 13 August 2008
Accepted 4 September 2008
Available online 18 September 2008

Responsible Editor: Nelson Fonseca

Keywords:
Request scheduling
Access gateway
Fair queuing

ABSTRACT

Scheduling packets is a usual solution to allocate the bandwidth on a bottleneck link. However, this solution cannot be used to manage the downlink bandwidth at the user-side access gateway, since the traffic is queued at the ISP-side gateway but not the user-side gateway. An idea is scheduling the requests at the user-side gateway to control the amount of the responses queued in the ISP-side gateway. This work first investigates the possibility of applying the class-based fair queuing discipline, which was widely and maturely used in scheduling packets, to schedule requests. However, we found that simply applying this discipline to schedule requests would encounter the timing and ordering problems at releasing requests and may not satisfy high-class users. Thus, we propose a minimum-service first request scheduling (MSF-RS) scheme. MSF-RS always selects the next request from the class receiving the minimum service to provide user-based weighted fairness, which ensures more bandwidth for high-class users. Next, MSF-RS uses a window-based rate control on releasing requests to maintain full link utilization and reduce the user-perceived latency. The results of analysis, simulation and field trial demonstrate that MSF-RS provides fairness while reducing 23–30% of user-perceived latency on average. Besides, a MSF-RS gateway can save 25% of CPU loading.

1. Introduction

Numerous enterprises connect to the Internet with the access link of Internet service provider (ISP), a typical topology of which is depicted as Fig. 1. In general, ISPs are willing to invest money in expanding the backbone bandwidth to provide their customers better service. However, to minimize costs, their customers often delay upgrading the bandwidth of the access link, which consequently becomes a potential bottleneck to access the Internet. To guarantee key traffic getting enough bandwidth when passing through the bottlenecked link, their customers may employ a class-based fair queuing (FQ) discipline or other packet-based bandwidth management [1] at the *user-side access gateway* to scheduling packets.

Unfortunately, these packet scheduling solutions fail to provide such guarantee for key traffic when the downlink is the bottleneck. In this case, packets are queued at the ISP-side gateway, not at the user-side gateway, for traversing the bottleneck. Scheduling packets at the user-side access gateway is useless because the packets have passed the bottleneck. On the other hand, although scheduling packets at the ISP-side gateway is useful, classifying packets at this gateway may be troublesome because of the network address translation (NAT), which is widely deployed at the user-side gateway to allow multiple users in an intranet sharing a public IP address. The packets which intend to enter the intranet cannot be classified by the ISP-side gateway because the classification needs to refer to the destination IP address of these packets but unfortunately they own the same one before they enter the NAT-embedded user-side gateway.

* Corresponding author. Tel.: +886 3 5731899.
E-mail addresses: weafon@cs.nctu.edu.tw (S.-C. Tsao), laiyc@cs.ntust.edu.tw (Y.-C. Lai), lctsao@cs.nctu.edu.tw (L.-C. Tsao), ydlin@cs.nctu.edu.tw (Y.-D. Lin).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.09.004

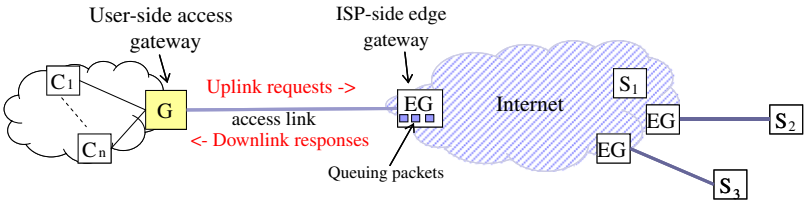


Fig. 1. A typical network topology that an enterprise accesses the Internet through ISP.

Scheduling requests, instead of packets, at the user-side access gateway may solve the above mentioned failure of packet scheduling. Such an idea is based on that applications running over the Internet mostly adopt the client–server model, i.e. the request/response model, such as HTTP, FTP, and E-mail. Requests sent from clients go through the access gateway and the uplink of the access link to remote servers, and the corresponding responses answered by the remote servers return to clients through the downlink of the access link and the access gateway. The downlink bandwidth could be managed by controlling the releasing of uplink requests.

Request scheduling was used in several studies to provide differential Web QoS for the requests of different types or the users of different classes. Such a usage was first introduced in [2] to schedule requests at a Web server. Then, the earliest deadline first scheduling was taken in [3] to ensure that requests of different types can be served within their specific deadlines. Besides, requests of a new session may be blocked from getting service in [4] to prevent the server from overloading and thus ensure the service quality of existing sessions. Compared with [2–4], which deployed request scheduling at a Web server, [5–7] deployed it at a Web-side gateway, i.e. a gateway in front of a group of Web servers. Requests were scheduled based on their resource requirements for load balance in [6] while this scheduling was for performance guarantee of users in different classes in [7].

Although many request scheduling algorithms were proposed for a Web server or a Web-side gateway, no published studies discussed how to schedule requests at the user-side access gateway to provide differential services for users. The key difference of scheduling requests at a server or a Web-site gateway from at a user-side gateway is that the destination Web servers in the former are specific and their statuses are easy to be measured or controlled for assisting in the scheduling operation. However, the servers in the latter are infinite in number, distributed over the Internet, and cannot be controlled.

In order to provide bandwidth sharing and weighted fairness among users of different classes on their downlink responses, this work studies how to schedule uplink requests at the user-side access gateway. We first investigate the possibility of applying the class-based FQ discipline, which was widely and maturely used in scheduling packets, to schedule requests. However, we found that simply applying this discipline to schedule requests would encounter three problems. The first two are determining the timing of releasing requests and selection of the next released request. The last one is that the class-based weighted fairness, achieved by a class-based FQ discipline, does not suit for the user-level differentiation, i.e. may not guarantee high-class users to get more bandwidth than low-class ones when more users appear in the high class. Based on the above investigation, we further propose a minimum-service first request scheduling (MSF-RS) scheme to provide bandwidth sharing and user-based weighted fairness, i.e. a policy that the ratio of the bandwidth allocated for each high-class user to that for each low-class user matches the ratio of their weights.

MSF-RS consists of a minimum-service order arbiter (MOA) and a window-based rate controller (WRC). MOA always selects the next request from the class which receives the lowest amount of responses while WRC determines the timing in releasing a request by monitoring the downlink utilization to control the number of *outstanding* responses. A response is regarded as *outstanding* if its corresponding request is released, but the response has not been received completely. MSF-RS is originally designed based on the assumption that the uplink traffic consists of requests only and the downlink traffic consists of their corresponding responses. However, MSF-RS also works well under the environment where the exceptive traffic coexists with the assumed traffic. We would further discuss the traffic-mixed case and show the robustness of MSF-RS by simulation in Section 6.

The remainder of the paper is organized as follows. Section 2 identifies the three problems occurring in scheduling requests with the class-based fair queuing discipline. Also, the user-based weighted fairness is introduced herein. Section 3 proposes the MSF-RS scheme. Section 4 proves that MSF-RS does shorten the user-perceived latency and also analyzes the worst-case fairness of MSF-RS, which are further demonstrated through the simulation results in Section 5. Besides, the affection of exception traffic on MSF-RS is discussed in Section 6. Section 7 demonstrates the effect of MSF-RS through field trail, where MSF-RS is implemented in Squid [8], an open-source Web proxy package. Section 8 gives the conclusions and future work.

2. Problems on using class-based fair queuing

Three problems would occur when the class-based FQ is used to schedule requests. The former two are related to the FQ discipline while the last is about the class-based weighted fairness policy.

2.1. The timing for releasing requests

A FQ-based packet scheduler selects and sends the next packet via a link right after the last packet has finished its

transmission. The bandwidth of the link would be totally consumed by the scheduled packets themselves. That is, each packet transmission *fully uses* the link bandwidth.

However, a FQ-based request scheduler *cannot* send the next request immediately following the last request, because a request does not directly consume the bandwidth of the bottleneck downlink, which actually will be consumed when receiving its responses. Releasing requests one-by-one may bring a large number of concurrent response transmissions at the bottleneck downlink, because the transmission time of a response, due to its size, is often longer than that of a request. Each transmission, under such a condition, only shares small bandwidth, resulting in the serious congestion or the long user-perceived latency. On the other hand, the request scheduling cannot just wait to send the next request till the preceding request completely gets its response, because such a waiting may waste the downlink bandwidth. After a request is sent out, until the first packet of its response returns, the downlink will be idle. Besides, even when the response is transmitting, the transmission may not run out the whole downlink bandwidth, because the Internet bandwidth available for the transmission may be smaller than the downlink bandwidth.

Since requests cannot be sent out as packets, a mechanism is necessary to control the release of uplink requests based on the utilization of downlink bandwidth, in order to avoid the downlink from congestion and to keep it on high utilization.

2.2. The determination of the next request

A FQ-based packet scheduler selects the next packet which is the earliest one to be completely served, i.e., fully transmitted, in the fluid-based general processor sharing (GPS) model [9]. The order of service completion is easily determined when a packet arrives because the determination only involves two known parameters, packet arrival time and packet size. For two packets arriving at the same time, the packet size decides the order of service completion time. A smaller packet finishes service earlier.

However, in a FQ-based request scheduler, although the arrival time of each request is known, the size of a request does not affect the service time of the request, which however is counted from releasing a request to receiving its whole response and mainly contributed by the transmission time of the response. Because the transmission time is determined by the response size and the available bandwidth in Internet, it is uncertain *at the request-scheduling moment*. Therefore, the completion time is uncertain too and the request scheduling cannot select the next request simply by its completion order. Hence, this selection is a problem when the FQ discipline is applied to request scheduling.

2.3. The class-based weighted fairness policy

The class-based weighted fairness policy is originally proposed to provide differential QoS for different service types of connections. For example, the real-time connections and the best-effort connections would be classified into two distinct classes. Then, according to the weights of these classes, each class is allocated with a fixed proportion of bandwidth. When the policy is applied, to guarantee that each connection in a class gets enough bandwidth, controlling the number of the active connections in the class is necessary. Establishing a new connection will be rejected when the number of active connections exceeds the expected value.

However, when the class-based weighted fairness policy is applied on providing differential QoS for *different levels of users*, it may expose undesirable characteristics for high-class users. For example, a high-class user may be rejected from getting service when the number of users now in the high class exceeds the expected value. Besides, if the number of users in the high class is much more than that in the low class, each high-class user may get lower bandwidth than the low-class user. Therefore, another policy may be necessary to always provide the high-class users enough bandwidth particularly when more users are active in the high class than the low class. We call such a policy the user-based weighted fairness. The policy guarantees that the ratio of bandwidth allocated for each high-class user to that for each low-class user matches the ratio of their weights.

3. A request scheduling scheme in user-side gateway

The section proposes a minimum-service first request scheduling (MSF-RS) scheme, which is deployed at the user-side access gateway and can provide user-based weighed fairness, bandwidth sharing, full bandwidth utilization, and short user-perceived latency. As shown in Fig. 2, the MSF-RS scheme consists of a minimum-service order arbiter (MOA) and a window-based rate controller (WRC). The former decides which request is the next one while the latter determines the timing to release requests.

3.1. Minimum-service order arbiter (MOA)

As shown in Fig. 2, MOA includes a request selector, a request receiver, and three groups of variables. There are N classes and each class is allocated a queue Q, a user counter (UC), a service counter (SC), and a weight w. MOA selects the next request from one of the class queues, i.e. Q_1 to Q_N, based on the value in SCs, which are updated by considering UCs and w's.

(1) UC and SC: The UC of a class keeps the number of the active users now in the class, where the active user means an intranet user who has requests or outstanding responses in MSF-RS. The SC of a class keeps the amount of services which the class has received. Herein the service represents the received responses in bytes after normalized with w and UC. That is, every time when one class, say the class i, receives its partial response of length L_p, its SC, SC_i, is updated as

$$SC_i^{new} = SC_i^{old} + \frac{L_p}{w_i \times UC_i}. \quad (1)$$

By normalizing L_p with w and UC during the SC update, for any two classes with the same SC, the ratio of their received responses will match the ratio of their weights

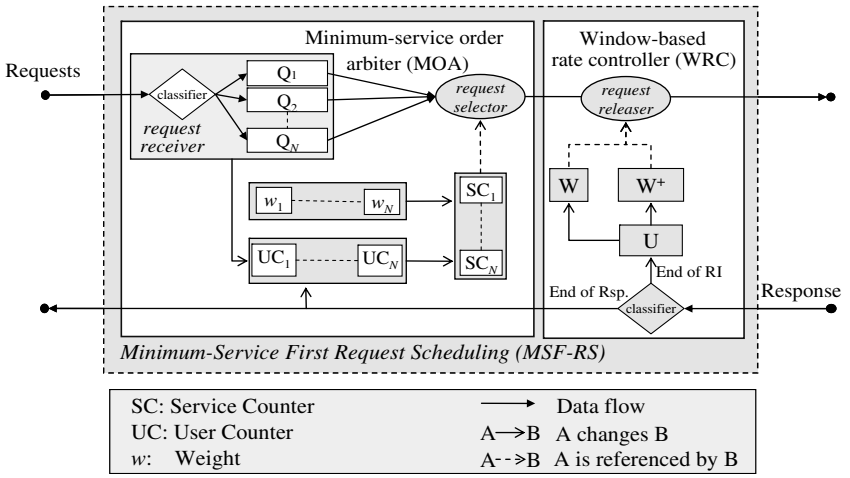


Fig. 2. The internal architecture of MSF-RS.

and their active users, respectively. That is, even when the number of active users in the high class is much more than that in the low class, the ratio of the responses got by one high-class user to that by one low-class user still matches the ratio of their pre-assigned weights.

(2) Next request selection: As shown in Fig. 2, when a request arrives, the classifier forwards the request into the corresponding *Q*. On the other hand, the *request selector* selects the head-of-line (HOL) request from the queue with the minimum SC. A class with the smaller SC represents that it received less services than other classes. Selecting a request from such a queue improves the fairness between classes, because it minimizes the difference of their SCs. Besides, if multiple classes have the same SC, the request selector selects the class with the highest product of the weight and UC. An idle class represents that the class has no outstanding responses and its request queue is empty. When a class idles for a long period, its SC may be far smaller than the SCs of active classes. Once the idle class has incoming requests, its far-small SC may cause the starvation of other classes. That is, until its SC is larger than any one SC of other classes, no request can be selected from other classes. To avoid this unfavorable condition, once the idle class becomes busy, its SC is updated to the minimal SC among all active classes. Such an update lets MOA follow the sharing concept used in the fair queuing discipline: the bandwidth freed from the idle classes would be shared by active classes, and these active classes will not be punished for the sharing, e.g. get less bandwidth when the idle classes become active. Notably, if all classes are idle, all SCs will be reset to 0.

(3) Basic procedures: Fig. 3 lists the pseudo codes of the two components in MOA. The *request_selector* picks the class queue with the minimum SC and releases the HOL request of this queue. The *request_receiver* classifies and en-queues all incoming requests. If an arrival request is classified into an idle class, i.e. the class's UC is zero, the *request receiver* resets the SC of this class. Next, the request is put into the belonging queue. If the request comes from a user *j* who has no request waiting for service or being served, i.e. ReqFromUser[*j*] = 0, then the UC of its belonging class will be added one, implying one more user arrives in this class. If the system is idle, i.e. no responses are outstanding, the *request_receiver* actively asks the *request_selector* to release the just coming request immediately.

```
Array ReqFromUser[j] : # of requests now in MOA and
                       coming from the j^th user

boolean Request_Selector // Called by Request_Receiver or WRC
{
  // Select the queue with min SC among all active queues,
  // i.e. their corresponding classes' PendingPpt are not zero.
  // return null if no active queues.
  Q_i ← ActiveQ_with_minSC()
  If (Q_i = null)
      return False          // imply there are no active classes
  Else
      SendHeadReq(Q_i)
      return True
}
Procedure Request_Receiver(Req)// Called when a request arrives
{
  i  ← GetClassNo (Req)
  j  ← GetUserNo(Req)
  If (UC_i=0)
      SC_i←GetMinSC()
  Enqueue(Req)
  If (ReqFromUser[j]=0)
      UC_i← UC_i+1
  ReqFromUser[j] ←ReqFromUser[j]+1;
  If (W = 0)
      Request_Selector ()
}
```

Fig. 3. Two procedures in *MOA*: request selector and request receiver.

3.2. Window-based rate controller (WRC)

As shown in Fig. 2, WRC controls the maximum of outstanding responses, W^+, according to the bandwidth utilization of the link, denoted by *U*. The variable *W* is used

to record the current number of outstanding responses. U is updated by the expression:

$$U_m = \frac{S_m/T}{C}, \tag{2}$$

where U_m is the utilization between the mth and $(m+1)$th updates, C is the link capacity, T is the time interval between two updates, and S_m is the responses in bytes received during T. Next, once U_m is changed, WRC updates W^+ by the equation

$$W^+_{m+1} = \min\left\{\frac{U^+}{U_m}, K\right\} \times W^+_m, \tag{3}$$

where W^+_m and W^+_{m+1} are the maximum of outstanding responses allowed after the mth and $(m+1)$th update, respectively. Also, $W^+_0 = 1$ and U^+ is the target utilization. K is a constant and assigned to 2 to avoid WRC from overestimating the new W^+ particularly when the old W^+ is small. When U_m is lower than U^+, W^+_{m+1} will be set to a larger value so that more outstanding responses can use the bandwidth and then raise the utilization U_{m+1}. For example, if the current W^+ is six and U is 60%, then the next W^+ will be set to 10 when $U^+ = 99\%$. On the contrary, when U_m is higher than U^+, W^+_{m+1} will be decreased so that fewer responses compete for the downlink bandwidth.

U^+ is constant and should be smaller than 100% ($U^+ < 100\%$). Otherwise, when $U^+ = 100\%$ and $U_m = U^+$, it cannot be distinguished whether the bandwidth required by the responses is larger than or just equal to the link capacity. Notably, W^+ should be recomputed only when $W = W^+$. When $W < W^+$, it is wrong to expect the raise of U by increasing W^+, because a low U is caused by insufficient arrival requests, but not too small W.

Fig. 4 lists the pseudo code of WRC. When WRC receives any part of a response, it looks for the class and the user which this response belongs to, and updates its class's SC. Once the received data includes the last packet of a response, W is decreased by one to imply that a request has been fully answered. Also, ReqFromUser of this user is decreased by one. If this is the last request, UC of this class is decreased by one also, because one user leaves the class. Next, the *request_selector* is invoked to release requests as more as possible, till $W = W^+$ or all queues are empty.

```
Procedure WRC                          // called when partial response returns
{
   Data  ← GetData()
   i     ← GetClassNo(Data)
   j     ← GetUserNo(Data)
   Len   ← Size(Data)
   SC_i  ← SC_i + Len / (w_i × UC_i) // update SC by (1)

   If ( IsTail_of_Rsp(Data)) {        // ending event of a transaction
      W ← W - 1
      ReqFromUser[j] ←ReqFromUser[j]-1
      If (ReqFromUser[j]=0)
         UC_i ← UC_i-1
   }

   While (W < W+)  // imply more transactions are expected
   {
      If (Request_Selector()=False) // ask for releasing a request
         Goto no_reqs                // if all req queues are empty
      W←W+1
   }
Label no_reqs:
}
```

Fig. 4. Procedure of window-based service-rate controller (WRC).

4. Analysis for latency and fairness

In the section, we first demonstrate that a MSF-RS gateway provides users shorter latency than an ordinary gateway on the average. An ordinary gateway means that it directly forwards the requests or responses once receiving them. Then, we analyze the fairness provided by MSF-RS in the worst case.

4.1. User-perceived latency

In general, the user-perceived latency of a request represents the time interval beginning when the client host sends out the request and ending when it receives the response of the request. As shown in Fig. 1, when a user-side gateway G is deployed at the gate of the Intranet to the Internet, we divide such a latency into four parts: (1) the transmission time of the request and its response between the client host and G, (2) the queuing time T_q of the request in G, (3) the transmission time of the request from G to the Internet destination site, and (4) the service time T_s of the request, i.e. the time of receiving its response from the site. The time of the first and third parts are ignored in the following comparison because they are small and unchanged no matter MSF-RS is deployed or not in G. Besides, the sum of T_q and T_s is called the active time of the request in G, denoted as T_a.

First, we conceptually explain why the T_s of a response in MSF-RS is shorter than that in an ordinary gateway when both gateways achieve the same high downlink utilization. In general, the link utilization is the most dominated factor to affect the packet's T_s, the transmission time over the link. However, since a Web response consists of a crowd of packets and its T_s spans from transmitting the first packet of the response to the end of its last packet's transmission, thus T_s depends on not only the utilization but also the concurrent number of the response transmissions. That is, a large number of concurrent transmissions would also increase the T_s of a response. Recall that the number of concurrent responses in MSF-RS is controlled to run out but not overload the downlink, while that in an ordinary gateway is uncontrolled and increased with the arrival rate of requests. When both MSF-RS and the ordinary gateway operate at the high link utilization, MSF-RS would have a smaller number of concurrent transmissions and thus provide a shorter T_s. For example, assume that 100 concurrent transmissions of responses can right exhaust the downlink bandwidth. Then, when 200 responses are concurrently transmitted through an ordinary gateway, although the downlink is exhausted too, the transmission time of the 200 responses would prolong doubly at least,

because each response transmission in the former only gets half bandwidth of that in the latter on average.

The following formally proves that MSF-RS provides shorter T_a on average than an ordinary gateway under a case that a batch of m requests arrives into the gateways. Assume MSF-RS has a fixed W^+ and the m requests would ask the responses of size L. Besides, the maximum bandwidth that W^+ responses can use would approximate to the downlink bandwidth C, because if a MSF-RS gateway allows the current transmission of W^+ responses, then these responses are expected to completely occupy but not overload the downlink. Therefore, the maximum bandwidth of each response can be expressed as C/W^+. Although the assumption of C/W^+ may be unrealistic, it is for the convenience in analysis and does not impose any constraint on the conclusions, which are validated by our simulations and experiments where no such an assumption is given.

Let us first consider the situation under an ordinary gateway. Since this work concerns the condition when the downlink is the bottleneck for an enterprise to connect the Internet, we assume the uplink bandwidth is far higher than the bandwidth necessary for the transmission of requests. Also, because the ordinary gateway simply forwards any received requests without considering the utilization of downlink and a request is usually far smaller, in length, than its corresponding response, the T_q's of these requests are closed to zero, compared to their T_s's. Then, since the responses of the m requests will concurrently share the downlink bandwidth, the bandwidth got by each response can be written as $\frac{C}{m}$. Therefore, the average T_a of requests under an ordinary gateway is

$$\mathrm{avg}(T_a^{\mathrm{ordinary}}) = 0 + \frac{L}{\frac{C}{m}} = \frac{mL}{C}. \tag{4}$$

Next, consider the case under MSF-RS. The T_q's of the first W^+ requests are zero because they are forwarded immediately. Then, others requests will be queued until any of the W^+ requests have been served. In the worst case, the W^+ requests end concurrently. Thus, the T_q's of the next W^+ requests would equal to the T_s's of the first W^+ requests, i.e. $\frac{W^+L}{C}$. Next, the T_q's of the following W^+ requests would equal to $2\frac{W^+L}{C}$. The T_q's of the residual requests could be derived from the same way. Thus, by summing up the T_q's of W^+ requests in each round and considering the possibility that the number of requests in the last round may be less than W^+, the average T_q of the m requests could be calculated as

$$\mathrm{avg}(T_q^{\mathrm{MSF\text{-}RS}}) = \frac{1}{m}\left(W^+\left(0+1+2+\cdots+\max\left\{\left\lfloor\frac{m}{W^+}\right\rfloor-1,0\right\}\right) + (m \bmod W^+)\left\lfloor\frac{m}{W^+}\right\rfloor\right)\frac{W^+L}{C}. \tag{5}$$

Similarly, the mean T_s of the m requests could be expressed as

$$\mathrm{avg}(T_s^{\mathrm{MSF\text{-}RS}}) = \frac{1}{m}\left((m-(m \bmod W^+))\frac{W^+L}{C} + (m \bmod W^+)\frac{(m \bmod W^+)L}{C}\right). \tag{6}$$

Therefore, we get the average T_a of requests under a MSF-RS by summing up Eqs. (5) and (6).

To compare the average T_a of requests over the two gateways, Fig. 5 plots the ratio of the MSF-RS gateway to an ordinary gateway on T_a over different m and W^+. Fig. 5 shows that the ratio is smaller than 1 always, i.e. the average time to queue and serve requests under MSF-RS is no more than that under an ordinary gateway. For example, as plotted by the dotted line, MSF-RS can reduce 25% of T_a when the number of arrival requests is two times of W^+. These results demonstrates that MSF-RS does not cause additional delay on T_a through MSF-RS does queue requests, i.e prolong T_q, to prevent the downlink from being the bottleneck of the response transmission.

The following gives an example to further clarify the fact. Assume there are 12 requests waiting for being forwarded. Also, we suppose that the downlink bandwidth can support four response transmissions in 1 s, i.e. $T_s = 1$, when four responses are received concurrently.

When an ordinary gateway is deployed, all the 12 requests would be immediately forwarded. Since there are 12, instead of four, response transmissions concurrently competing the downlink, each transmission will get only 1/12, instead of 1/4, of downlink bandwidth. Therefore, T_s of the 12 requests becomes three and their user-perceived latency is $T_q + T_s = 0 + 3 = 3$ s on average.

In contrast, when a MSF-RS gateway is deployed, since it monitors the downlink utilization to control the number of the concurrent response transmissions, only four responses would be allowed to be transmitted concurrently. Thus, the first four requests are forwarded without any delay and their responses are received in one second; meanwhile, the other eight requests are queued in the gateway for 1 s. Then, the next four requests are forwarded and their responses are received in the 2nd second, while the last four requests are still queued. Finally, the last four requests are forwarded.

Therefore, the average T_q for the 12 requests can be easily calculated as $(0*4+1*4+2*4)/12 = 1$ s. Besides, T_s of the 12 request is 1 s. Then, their average user-perceived latency is equal to 1 + 1 = 2 s, which is obviously shorter than that achieved by an ordinary gateway.

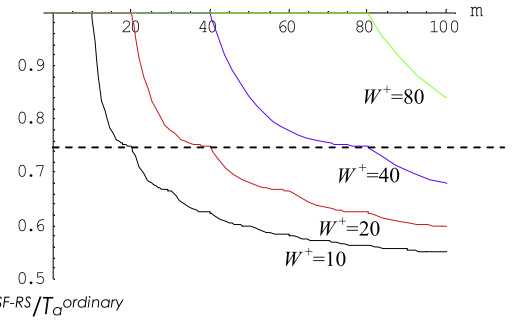


Fig. 5. The ratio on T_a of a MSF-RS gateway to an ordinary gateway.

4.2. Fairness

The fairness parameter that we use is based on the definition presented by Golestani [10] for the analysis of self-clocked fair queuing. The parameter is defined as the maximum difference of the service got by any two backlogged classes over arbitrary time intervals, which can be written in a mathematical formula as

$$\mathrm{MAX}\left\{\left|\frac{D_i(t_1,t_2)}{w_i}-\frac{D_j(t_1,t_2)}{w_j}\right|\right\}(t_2>t_1),$$

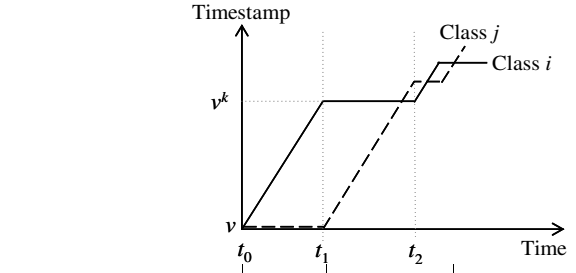
where t_1 and t_2 are two arbitrary instants when the two classes are backlogged. w_i is the weight of Class i, and $D_i(t_1,t_2)$ represents the received service of Class i during $[t_1,t_2]$. A scheduling algorithm has a zero value of fairness if it always provides equal service for any two classes even in a short time interval. We consider two classes, Class i and Class j, through the following analysis because the definition of fairness only concerns two classes. The existence of more backlogged classes does not affect the difference of services got between two classes.

As shown in Fig. 6, assume the MSF-RS is idle before the time t_0, i.e. $W=0$. Then, at the time t_0 more than W^+ requests of Class i arrive. Let v denotes the timestamp of the first request of Class i, where the timestamp represents the value in SC_i when the request arrives into Class i. Upon these requests arrives, MSF-RS will forward the first W^+ ones and W is set to W^+. Let request k be the $(W^+ + 1)$th one of Class i, i.e. it would be the first request in the queue of Class i after t_0. Since W^+ requests of Class i would be served before k, k will have a timestamp v^k expressed by

$$v^k = v+\frac{1}{w_i}\sum_{h=1}^{W^+}L_i^h \leqslant v+\frac{W^+L_i^+}{w_i},$$

where L_i^h is the size of the hth response of Class i and L_i^+ is the maximum size of response of Class i.

Assume that the requests of Class j arrive right after the time t_0, and the first request should have the timestamp v since the timestamp of the request latest released by the server right after the time t_0 is equal to v. However, although the first request of Class j has timestamp v smaller than that of Class i, i.e. v^k, no requests can be forwarded from Class j because all W^+ sub-links are busy for Class i.



Time	t_0~t_1	t_1~t_2	t_2~
Max. difference of service between Class i and j	$\frac{W^+L_i^+}{w_i}$	$\frac{W^+L_i^+}{w_i}+\frac{L_i^+}{w_i}$	$\frac{L_i^+}{w_i}$

Fig. 6. The difference of the service between Class i and Class j.

Let the sub-links become idle at t_1. Then, in the worst case all requests of Class j with a timestamp *no larger* than v^k will be forwarded before k. Assume the Class-j request with v^k asks a response with size equal to L_j^+. Then, between t_1 and t_2 the maximum of the total responses received by Class j will be

$$w_j\left(\frac{1}{w_i}W^+L_i^+\right)+L_j^+. \tag{7}$$

However, as shown in Fig. 6, Class i does not get any service during the period. Thus, between t_1 and t_2, the difference of the service between the two classes is $\frac{W^+L_i^+}{w_i}+\frac{L_j^+}{w_j}$, got by dividing (7) by w_j.

For the time later than t_2, when the service of a class already equals to another one, the additional service which the class can get must smaller than $\frac{L_i^+}{w_i}$ since $\frac{L_i^+}{w_i}\geqslant\frac{L_j^+}{w_j}$ is supposed. Since the difference of service after t_2 and before t_1 is smaller than that between t_1 and t_2, the difference of the service between t_1 and t_2 is the worst-case fairness of MSF-RS, that is

$$\frac{W^+L_i^+}{w_i}+\frac{L_j^+}{w_j}.$$

5. Simulation results

This section first verifies the effects of MSF-RS by *ns-2* [11] in terms of the fairness and bandwidth sharing, user-perceived latency and the relationship between U and W^+. Then, the effect of U^+ on latency is investigated.

5.1. Topology

The *HTTP/Cache* in *ns-2* acts as a Web proxy cache and sits between clients and Web servers. It intercepts the requests sent from clients and forwards them to the remote servers if the requested data is not cached yet. This work disables the cache function and implements MSF-RS in *HTTP/Cache*. Fig. 7 shows the topology used in the simulation. The MSF-RS gateway provides three classes, Class 1, Class 2, and Class 3, with the weights, 4, 2, and 1, respectively. Each class involves four clients and each client repeatedly requests pages from the 12 remote Web servers through the MSF-RS gateway. For each client, the time interval between two requests is an exponential distribution with mean equal to 5 s.

The link between the MSF-RS gateway and every client is 10 Mbps with 2 ms propagation delay. The ISP gateway connects to twelve servers with twelve independent links. These servers are classified into two equal numbers of groups, representing overseas servers and domestic servers. Links between the gateway and these servers have a *uniform* distribution, as shown in Fig. 7. By the statistics from the real Internet [12], the Web response size has a lognormal distribution with $M=9.357$ and $S=1.318$, where the probability density function $P(x)$ of the lognormal distribution can be written as

$$P(x)=\frac{1}{S\sqrt{2\pi}x}e^{-(\ln x-M)^2/(2S^2)}.$$

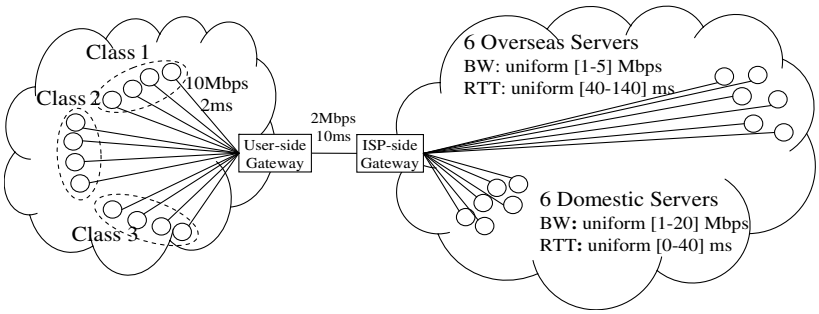


Fig. 7. Simulation topology for three classes with service ratio 4:2:1.

The average response size is $e^{M+S^2/2}$ bytes, i.e. 27,656 bytes. The U^+ in WRC is set to 98% and the time interval between two updates is set to 5 s. Section 5.6 would further reveal the effects of different U^+'s on link utilization, but that of different update intervals are ignored to show because of their insignificant effects. Besides, we use TCP SACK and assume no delayed acknowledgments. Over the simulation the packet size is 1000 bytes and the maximum congestion window of TCP is 200. The queues at the two gateways are managed by Drop-Tail and their sizes are 1.5 bandwidth-delay products.

5.2. Weighted fairness and bandwidth sharing

First, we demonstrate that when all classes have the same users, MSF-RS provides weighted fairness between classes and the idle bandwidth is shared by active classes. Four phases are included in the simulation and the duration of each phase is 200 s. In the first phase, all of the three classes have backlogged requests. In the next two phases, Class 1 and 2 stop requesting individually, and then both of them have backlogged requests again in the last phase.

Fig. 8a shows the average throughput under MSF-RS in each phase. During the first phase, the three classes get proportional bandwidth in ratio 3.96:1.98:1, which is close to the expected ratio 4:2:1. In the second phase, the idle bandwidth freed by Class 1 is shared by Class 2 and Class 3 proportionally. Both of the bandwidth obtained by Class 2 and Class 3 increase in this phase, and the usage ratio between them is still 2:1. After Class 2 stops requesting in the third phase, Class 3 occupies all bandwidth until the end of this phase. During the second and the third phases, Class 1 and Class 2 still obtain a bit of bandwidth separately due to their unfinished responses at the 200th and 400th s, respectively. Once all idle classes have requests again in the last phase, the three classes obtain the bandwidth in the expected proportion, 4:2:1, again.

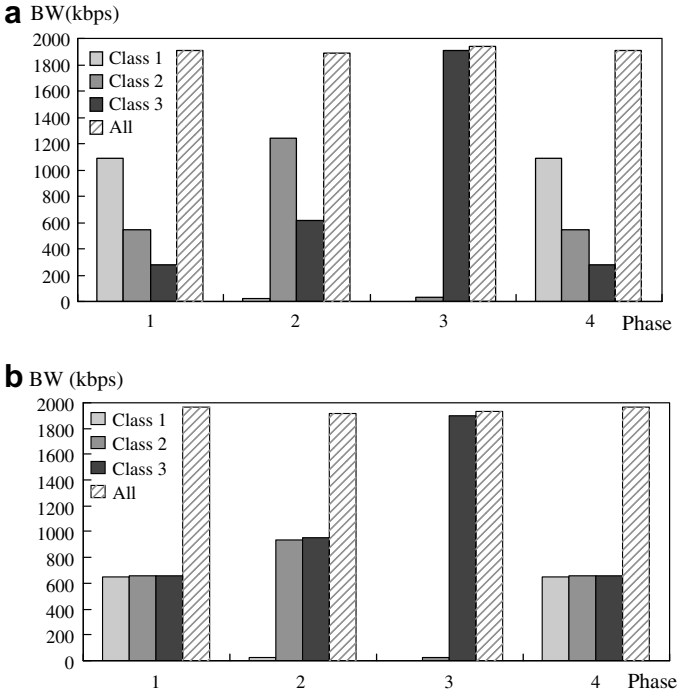


Fig. 8. The average throughput of three classes over the four phases under (a) MSF-RS and (b) DRR.

As mentioned in Section 1, packet scheduling algorithms fail to allocate the downlink bandwidth at the user-side gateway, because the packets of responses have passed the bottleneck and can be immediately forwarded to the clients. Under the situation, a packet scheduling algorithm would degrade into a first-in-first-out scheduling. To demonstrate such degradation, we employ a deficit round robin (DRR) [13] instead of MSF-RQ at the user-side gateway. DRR is a widely used packet scheduling algorithm because it is easy to be implemented. Fig. 8b shows the bandwidth allocation managed by DRR. Obviously, over the four phases, Class 1 and Class 2 do not get higher bandwidth than Class 3, even though both classes have larger weights than Class 3. Further observation shows that the request queues of three classes in DRR are empty during the simulation, which verifies that requests are forwarded upon their arrival, i.e. a FIFO order.

5.3. User-perceived latency

The simulation scenario here is the same as that used in the first phase in Section 5.2. Fig. 9 illustrates the user-perceived latency for the three classes, the average latency of all classes, and the latency if no MSF-RS is deployed, denoted as non-MSF-RS. The latency is decomposed into the queuing time and the service time of the requests, as introduced in Section 4.1.

First, by comparing the left three bars, the three classes in MSF-RS experience the different user-perceived latencies, mainly caused from different queuing time since they have different weights. Second, by comparing the right two bars, the average latency (6.76 s) in MSF-RS is shorter than that in non-MSF-RS (8.83 s) by 23.44%. It is because the average service time in MSF-RS (1.44 s) is far shorter than that in non-MSF-RS (8.83 s). The service time in MSF-RS is reduced because it has the well-controlled number of concurrent outstanding responses and thus each response can be received in a short time.

5.4. User-based weighted fairness

Next, we show the MSF-RS gateway provides the high-class users more bandwidth than the low-class users regardless of the number of users in the high class. The same testing scenario as that in Section 5.2 is used, but the number of users in Class 1 is increased from 4 to 24. Also, all of the three classes have backlogged requests during the whole testing time, 800 s. Fig. 10a plots the difference of the average bandwidth allocated for the users in each class. When there are 4 users in Class 1, each user in this class owns 270 Kbps, which is the two and four times of that allocated for the user in Class 2 and Class 3, respectively. The fixed ratio of the allocated bandwidth among the three classes is kept even when the number of users in the first class is increased. Fig. 10b plots the result provided by the MSF-RS gateway without considering the number of users when updating SC, i.e. the L_p in Eq. (1) is not divided by UC. Obviously, under such a gateway, the users in Class 1 cannot be ensured to get more bandwidth than that in other classes when more users are active in Class 1.

5.5. Adjustment of outstanding responses

The subsection observes the adjustment of W^+ when the arrival of requests is not backlogged always, i.e. MSF-RS may be idle sometimes. In a 1500-s test, clients in Class 1 and 2 send requests every 1 s while that in Class 3 sends one every 10 s. Besides, during the middle 500 s, clients in Class 1 and 2 stop sending requests, which results in insufficient requests so that the MSF-RS gateway has no request to send out. Fig. 11 reveals the relation between U and W^+. The utilization of access link stays around 0.98 as the expected U^+ in the first and last 500-s periods because of sufficient arrival requests.

In the 500th–1000th s, the utilization falls apparently and the value of W^+ keeps constant as described in Section 3.2. Increasing W^+ for raising the utilization during the period is in vain because the low utilization results from the fact that the incoming requests are too few to occupy all sub-links. Besides, the value of W varies with a wide range, determined by the dynamically arrival requests. During the period, any requests are forwarded immediately once they arrive, since there are always free sub-links. Notably, at the 1000th s, once the two stopped classes restart sending requests, all requests can be released soon and the utilization jumps to the expected value.

5.6. Effect of U^+ on latency

Fig. 12 depicts the user-perceived latency and the queuing time spent in the MSF-RS access gateway and in the

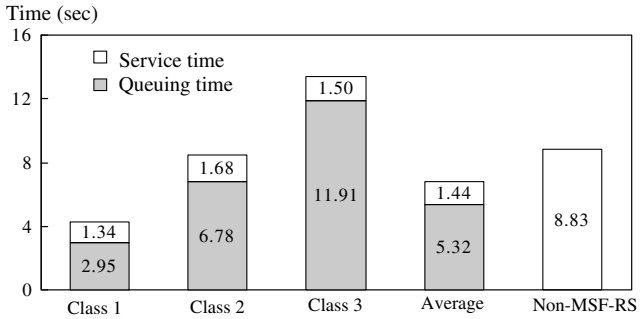


Fig. 9. The comparison on the user-perceived latency among classes and between MSF-RS and non-MSF-RS.

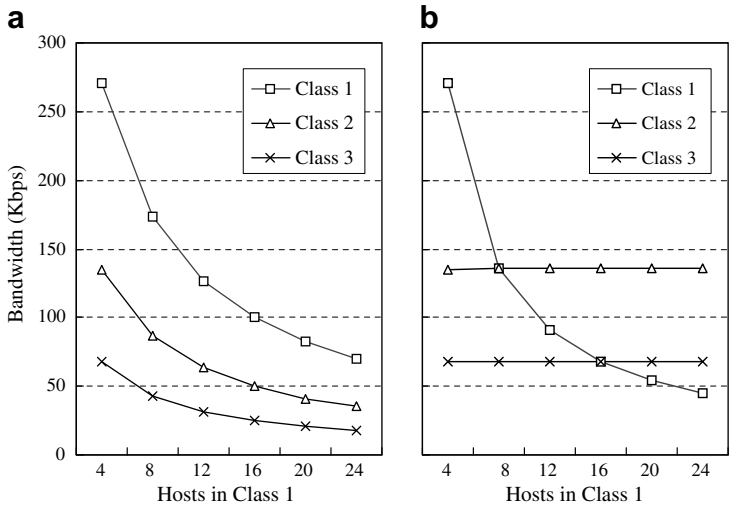


Fig. 10. The difference on the bandwidth allocated for the high-class host between the (a) host-based and (b) class-based weighted fairness.

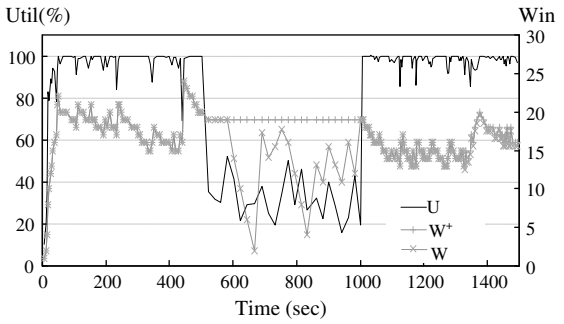


Fig. 11. The size of W^+ is fixed in the period with insufficient traffic (the 500th–1000th s).

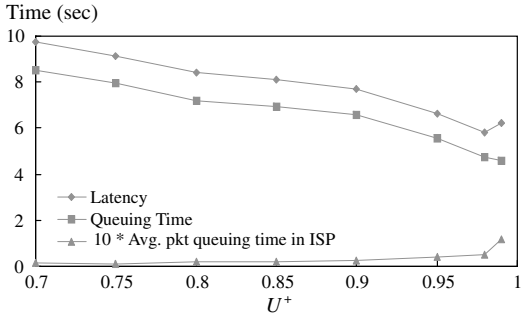


Fig. 12. The user-perceived latency, queuing time, and the packet queuing time in ISP-side gateway under the different U^+.

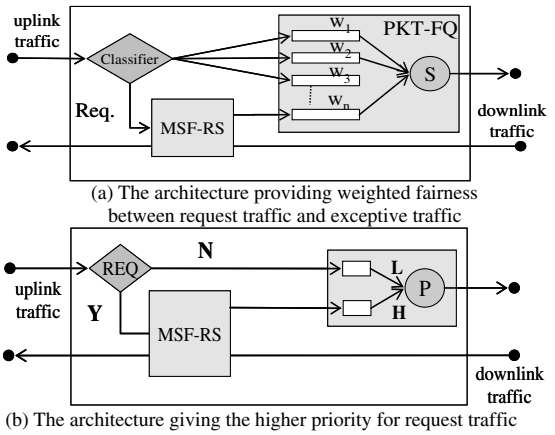
ISP-side gateway when U^+ is assigned from 0.65 to 0.99. Raising U^+ follows shorter user-perceived latency because more responses can be concurrently transmitted and the bandwidth can be more utilized. However, the raise also causes packets to be queued in the ISP-side gateway because of less free bandwidth to eliminate the queued packets as U^+ is high. By the observation in Fig. 12, the value of U^+ is suggested to be set to 98%.

6. Affection of exceptive traffic

MSF-RS is designed under the assumption that the uplink traffic comprises requests only and the downlink traffic comprises their corresponding responses only. However, the exceptive traffic does coexist with the assumed traffic in the real environment. We classify the exceptive traffic into three types and explain why they do not affect the fairness or link utilization provided by MSF-RS.

(1) Uplink exceptive traffic: The type of traffic may include the uplink responses and the packets actively sent from the internal users. If this traffic is heavy enough to turn the uplink to a bottleneck, a packet scheduler with the FQ discipline is suggested to be deployed at the access gateway first. MSF-RS can coexist with the uplink FQ discipline well, as shown in Fig. 13a. Also, if the weighted fairness on uplink is not a concern, the combination of MSF-RS and priority queues is a simple solution, as shown in Fig. 13b. The solution gives the request traffic higher priority since they are smaller than responses usually.

(2) Downlink exceptive traffic belonging to some classes: Such traffic is still the downlink responses, but their requests are not recognized by the implementation of MSF-RS. For example, POP3 mail traffic for someone's host belongs to Class *i*. It is possible since only the Web request is recognizable for the present implementation of MSF-RS.The MOA in MSF-RS regards these exceptive packets as the received service of classes. That is, when the packets not triggered by an (recognized) request arrive from the Internet, their sizes are accumulated into the SC of the class which the packets belong to, as other response packets triggered by requests. That is, if a user of Class *i* receives a crowd of such packets from the Internet, the sizes of packets would be accumulated into SC_i. Although the additional value in SC_i caused by these exceptive packets brings the fewer requests sent out from Class *i* by MSF-RS, it does not affect the weighted fairness and the link utilization.



(a) The architecture providing weighted fairness between request traffic and exceptive traffic

(b) The architecture giving the higher priority for request traffic

Fig. 13. Two potential integrated architectures for handling the network where the uplink is a bottleneck.

(3) Downlink exceptive traffic not belonging to any class: Exceptive traffic not belonging to any class may include requests from outside network for the responses provided by the internal servers, or the malicious attacks. The former case is possible for the enterprises having Web servers for their customers. The exceptive traffic contributed from such requests is usually small, compared with other response traffic running on the downlink. The latter case may rudely and fully occupy the downlink, resulting in the failure for all transmissions. However, the latter case is a security problem and out of the scope of this work.

Since this exceptive traffic is not belonging to any class and not counted in any SC, it will not affect the weighted fairness between classes. Besides, although it would be counted by WRC in the utilization of the downlink, it does not degrade the high link utilization which WRC can ensure. Such traffic only causes WRC to have a smaller W^+ than that in the case without the exceptive traffic. That is, WRC may think such a small W^+ is enough to fully utilize the downlink bandwidth. Moreover, when the exceptive traffic passed, because WRC would quickly adjust W^+ according to the new U, the link utilization is not affected in this case.

We use the following simulation to demonstrate that the utilization is not affected by the downlink exceptive traffic. The on–off CBR traffic not belonging to any classes with different rates during different periods are generated to demonstrate the responsiveness of WRC is fast enough to keep the high link utilization. Five on/off periods are tested: 40, 80, 160, 320, and 640 s. During the on-period, three rates of CBR traffic are tested individually: 20%, 40%, and 60% of the downlink capacity. Each test is run for 3000 seconds and U^+ is set to 98%. Fig. 14 shows the case where on/off period is 640 s and CBR rate is 40% of link capacity, i.e. 0.8 Mbps. At 1280 and 2560, once the CBR traffic stops, WRC immediately resets W^+ from 7 to 13 in order to release more requests. The bandwidth freed by CBR traffic is fully and fast occupied by the response traffic. In fact, the results in all tests, as shown in Table 1, reveal

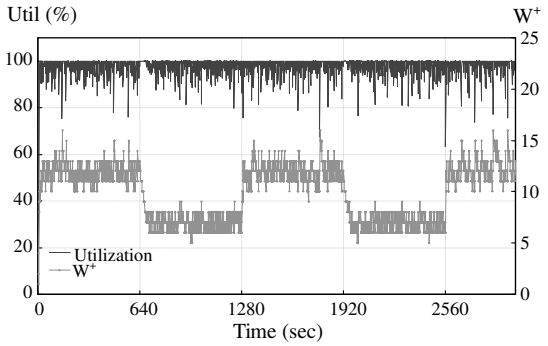


Fig. 14. Fast-responsive W^+ and full utilization of access link under oscillate CBR traffic.

that WRC keeps the utilization on 97.84% averagely, closing to the designed goal, 98%.

Although the exceptive traffic does not degrade the fairness and link utilization achieved by MSF-RS, it reduces the bandwidth available for response transmissions and thus increases the user-perceived latency of these responses. As shown in Section 4.1, MSF-RS has the advantage of shorter user-perceived latency than an ordinary gateway under no exceptive traffic. Actually, the advantage still holds when exceptive traffic appears, as explained in the following. When the exceptive traffic exists, the downlink bandwidth C used in Eqs. (4)–(6) is decreased to a smaller value, and thus a smaller W^+ is adopted in MSF-RS, as shown in Fig. 14. From Fig. 5, we know that the ratio on T_a of a MSF-RS gateway to an ordinary gateway under a small W^+ is smaller than that under a large W^+. That is, a MSF-RS gateway can provide a much shorter user-perceived latency than an ordinary gateway when exceptive traffic exists, although the latencies in both gateways are indeed increased in this circumstance.

7. Field trial

We implemented MSF-RS in *Squid* [8], which is an open-source package of Web proxy cache, and performed a field trial in an open network environment. Fig. 15 illustrates the test bed for evaluating MSF-RS in Squid. An application-layer traffic generator named *Avalanche* [14] is used to emulate the behaviors of multiple clients and send requests to the Web servers in the Internet. Avalanche is imported with a URL list, a historical record logged by an enterprise in a couple of days.

The access gateway installed with MSF-RS is configured as a transparent proxy with iptables [15]. All HTTP requests destined to port 80 are directed to port 3128, the service port of Squid. A layer 3 switch is acted as the ISP-side gateway. The bandwidth of the access link between the access gateway and the layer 3 switch is limited to 2 Mbps. As the configuration in simulation, three classes are provided with service ratio 4:2:1. Notably, the cache function is disabled to avoid getting responses directly from caches. The effects of MSF-RS Squid are observed in terms of weighted fairness, user-perceived latency, and CPU loading as follows.

Table 1
The utilization of link under oscillating CBR traffic (U^+ = 98%)

On/off period (s)	The rate of CBR during on-period		
	0.4 Mbps (%)	0.8 Mbps (%)	1.2 Mbps (%)
40	97.94	97.91	97.62
80	97.90	97.95	97.77
160	97.94	97.92	97.68
320	97.83	97.80	97.76
640	97.93	97.83	97.85

(1) Weighted fairness: The amounts of bandwidth allocated to three classes for a 200-s test are 1.03, 0.52, and 0.26 Mbps, respectively, when backlogged requests are applied. The result quite obeys the configured service ratio 4:2:1.

(2) User-perceived latency: Table 2 shows the latency provided by the original Squid and the MSF-RS Squid. The original Squid case represents all requests are immediately released by the proxy. The MSF-RS Squid reduces (1686-1175)/1686, or 30%, of the average user-perceived latency in the original Squid case, although the user-perceived latency in MSF-RS includes the additional queuing time, 515.5 ms.

(3) CPU loading: Table 3 shows the benchmark results on CPU time occupied by the MSF-RS Squid process and the original Squid process when both processes provide the same link-speed throughput during 200 s. As expected, the CPU time increases as the number of classes or the access link bandwidth increases. Notably, the time under MSF-RS is always lower than that under the original Squid. Under the original Squid, all requests are immediately released by the proxy, bringing great number of concurrent responses. However, a proper number of concurrent responses are allowed by MSF-RS. It is believed that the number of concurrent responses dominates the cost of CPU computing.

8. Conclusions and future work

Scheduling the uplink requests is a potential method to manage the bottlenecked downlink at the *user-side* access gateway. Because the class-based fair queuing (FQ) discipline is widely and maturely used in scheduling packets,

Table 2
User-perceived latency comparisons

Item	Case	
	Original Squid	MSF-RS Squid
User-perceived latency	1686.1 ms	1174.9 ms (include queuing time 515.5 ms)

Table 3
Comparison between MSF-RS and the original Squid on CPU time

Case	Link capacity	
	2 Mbps	10 Mbps
MSF-RS Squid with 10 Classes	44.8 s	56.34 s
MSF-RS Squid with 100 Classes	46.02 s	58.04 s
Original Squid	63.22 s	84.90 s

we first investigate the possibility of applying this discipline to schedule requests. However, we found that three problems occur at applying the class-based FQ discipline to schedule requests: the *timing* and *ordering* to release requests and the suitability of class-based weighted fairness for user-level differentiation. Based on the investigation on the three problems, we propose the minimum-service first request scheduling (MSF-RS) scheme to manage the access link bandwidth at user-side access gateway. To achieve high bandwidth utilization while avoiding congesting the link, the window rate control module in MSF-RS determines the releasing rate of requests and the number of outstanding responses. To perform user-based weighted fairness and bandwidth sharing, the minimum-service arbiter module in MSF-RS always selects the request from the class receiving the least normalized responses.

The analysis first proves that MSF-RS shortens 25% of the user-perceived latency on average, compared with an ordinary gateway, because the number of concurrent transmissions is controlled, even though this control may queue requests in the MSF-RS gateway. Besides, the analysis on worst-case fairness represents that the MSF-RS gateway does provide the differential services among classes while avoiding the low-class users from long latency. The results in the simulation and in the field trial show that the bandwidth usage between classes conforms to the targeted ratio and the idle bandwidth is proportionally shared

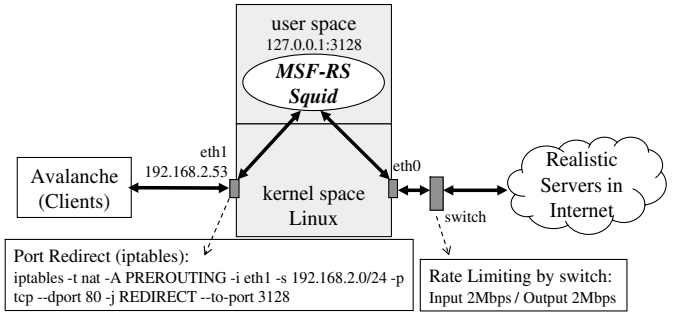


Fig. 15. The test bed for field trial in the Internet.

by all active classes. Besides, MSF-RS reduces 23.44% and 30% of user-perceived latency in the simulation and the field trial, respectively.

Currently, http traffic is growing due to the explosion of video streaming services. For streaming services, MSF-RS is still better than an ordinary gateway. The reason is explained as follows. MSF-RS does not provide a high-class user to use a significantly high throughput to download the video. In fact, it simply releases requests from high-class queue more frequently than low-class queue, but does not control the downloading rate of a single transmission. If the video server can properly determine transmitting rate, the transmission will not overuse the bandwidth. Actually, many existing servers can adjust this rate according to the size of client's buffer. In contrast, if the video server sends data as soon as possible, only a few high-class requests will be released since MSF-RS provides weighted fairness among high and low classes. Thus the low-class users do not encounter the starvation. Therefore, for streaming services, MSF-RS is still better than an ordinary gateway where all requests are released and no differentiation exists. Under an ordinary gateway, either high-class or low-class users will suffer insufficient bandwidth. The viewing of videos will be suspended to wait more data arrival, although users may be quickly initialized.

In the future, we will further enhance MSF-RS to support other request–response protocols, such as FTP and POP3. Finally, to further reduce the overhead, implementing MSF-RS in the kernel space may be considered.

References

[1] H.Y. Wei, S.C. Tsao, Y.D. Lin, Assessing and improving TCP rate shaping over edge gateways, IEEE Transactions on Computers 53 (3) (2004) 259–275.

[2] R. Pandey, J. Fritz Barnes, R. Fritz Barnes, Supporting quality of service in HTTP servers, in: Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing, 1998, pp. 247–256.

[3] N. Bhatti, A. Bouch, A. Kuchinsky, Integrating user-perceived quality into Web server design, in: Proceedings of the 9th International World Wide Web Conference, 2000.

[4] L. Cherkasova, P. Phaa, Session based admission control: a mechanism for Web QoS, in: Proceedings of the International Workshop on Quality of Service, 1999.

[5] V. Cardellini, E. Casalicchio, M. Colajanni, M. Mambelli, Enhancing a Web-server cluster with quality of service mechanisms, in: Proceedings of IEEE International Performance Computing and Communications Conference, 2002.

[6] E. Casalicchio, M. Colajanni, A client-aware dispatching algorithm for Web clusters providing multiple services, in: Proceedings of the 10th International World Wide Web Conference, 2001.

[7] C. Li, G. Peng, K. Gopalan, T. Chiuch, Performance Guarantee for Cluster-Based Internet Services, State University of Stony Brook (2001).

[8] Squid Web Proxy Cache. <http://www.squid-cache.org/>.

[9] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking (1993) 344–357.

[10] J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of the IEEE INFOCOM, Toronto, June 1994.

[11] The Network Simulator – ns-2. <http://www.isi.edu/nsnam/ns/>.

[12] P. Barford, M. Crovella, Generating representative Web workloads for network and server performance evaluation, ACM SIGMETRICS Performance Evaluation Review 26 (1) (1998) 151–160.

[13] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round-Robin, IEEE/ACM Transactions on Networking 4 (3) (1996) 375–385.

[14] Avalanche. <http://www.spirentcom.com/documents/664.pdf?wt=2&az-c=dc>.

[15] The Netfilter/Iptables project. <http://www.netfilter.org/>.

Shih-Chiang Tsao was born in Taiwan in 1975. He received the B.S. and M.S. in Computer and Information Science from National Chiao Tung University in 1997 and 1999, respectively. He worked as an associate researcher in Chung-Hwa Telecom from 1999 to 2003, mainly to capture and analyze switch performance. He received the Ph.D. degree in Computer Science from National Chiao Tung University in 2007, and is in High Performance Computing Research Department of Lawrence Berkeley Laboratory for his post-doctoral research. His research interests include TCP-friendly congestion control algorithms, fair queuing algorithms, and Web QoS.

Yuan-Cheng Lai received the Ph.D. degree in Computer Science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in 2001 and has been a professor since 2008. His research interests include wireless networks, network performance evaluation, network security, and content networking.

Le-Chi Tsao was born in Taipei, Taiwan in 1977. She received the B.S. and M.S. degree in Computer Science from National Chiao Tung University in 1999 and 2005, respectively. Her research interests include network QoS and request scheduling. She can be reached at lctsao@cis.nctu.edu.tw.

Ying-Dar Lin is a professor of Computer Science at National Chiao Tung University, where he is also the director of Computer and Network Center and Network Benchmarking Lab (NBL). His research interests include quality of services, deep packet inspection, and hardware software co-design. He is spending his sabbatical year, from July 2007, at Cisco, San Jose. Dr. Lin graduated from National Taiwan University (B.S. 1988) and UCLA (Ph.D. 1993).