

國立交通大學

資訊科學系

碩士論文

在 網 路 通 道 閘 道 器 上
提 供 差 別 服 務 品 質 之 請 求 排 程



Request Scheduling for Differentiated QoS at Access Gateway

研 究 生：歐陽銘康

指 導 教 授：林盈達 教授

中 華 民 國 九 十 三 年 六 月

在網路通道閘道器上提供差別服務品質之請求排程
Request Scheduling for Differentiated QoS at Access Gateway

研 究 生：歐陽銘康

Student : Ming-Kang Ou Yang

指導教授：林盈達

Advisor : Dr. Ying-Dar Lin

國 立 交 通 大 學

資 訊 科 學 系

碩 士 論 文



A Thesis
Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

在網路通道閘道器上提供差別服務品質 之請求排程

學生：歐陽銘康

指導教授：林盈達

國立交通大學資訊科學系

摘要

網際網路的交通流量成長的非常快速，造成網路發生擁塞的情形日益增加，現在企業存取網際網路的瓶頸通常都是發生在連線網路的頻寬上。故在連線網路上提供頻寬管理的功能是非常重要的，然而傳統在消費者端的網路通道閘道器上利用封包排程的技術提供頻寬管理功能有三個缺點：低擴充性、在下載瓶頸之後才作排程控制、同一時間內過量的傳輸。為了解決這些問題，我們提出了請求封包排程的架構。請求封包排程根據請求封包的三個參數：反應封包大小、反應封包傳輸率、反應封包回應延遲計算出請求封包對應的反應封包在連線網路中所佔用的頻寬，然後透過控制類別中請求封包的送出頻率，來提供公平而且可分享的頻寬，以及降低連線網路中發生碰撞的可能性。模擬的結果顯示不同類別之間的實際頻寬佔用比率跟指定的頻寬佔用比率相似度非常高，同時反應封包的回覆時間下降為原來的 68.75%，原因為空閒的頻寬可被其他類別所分享及封包在連線網路發生碰撞的情形降低，顯示所提的方法確實可以提供差別服務品質。

關鍵字：需求封包排程、頻寬管理、應用程式差別服務

Request Scheduling for Differentiated QoS at Access Gateway

Student: Ming-Kang Ou Yang Advisor: Dr. Ying-Dar Lin

Department of Computer and Information Science

National Chiao Tung University

Abstract

For ISP's customers, the access link is usually the bottleneck when accessing the Internet since they often delay upgrading the access link bandwidth to save costs. Thus the management of access link bandwidth is imperatively required. On managing the access link bandwidth, the common approach, packet-level scheduling at consumer-side access gateway, has three potential problems, namely, *low scalability*, *scheduling behind the downlink bottleneck*, and *excessive concurrent transmissions*. This work proposes a ReQuest Scheduler (RQS) to avoid these problems. RQS calculates the bandwidth of access link needed by the *response* traffic according to each request's three parameters, namely, response *size*, response *transmission rate*, and response *delay*. RQS controls the releasing rate of requests to provide proportionally shared bandwidth between various classes, and to reduce congestion occurring at the access link. The simulation results show the bandwidth usage ratio between classes can be close to the allocated quantum ratio, while the free bandwidth can be shared among active classes. Also the mean transmission time is decreased to 68.75% of the original value; and congestion occurring at the access link is reduced.

Keywords: *Request Scheduling, Bandwidth Management, Application QoS*

Contents

CHAPTER 1. INTRODUCTION	1
CHAPTER 2. ARCHITECTURE OF THE REQUEST SCHEDULER	4
2.1 OPERATION MODEL	4
2.2 RELEASE TIME CONTROLLER (RTC).....	5
2.3 RELEASE PROPORTION CONTROLLER (RPC).....	6
2.4 RESPONSE RECORDER (RR)	7
CHAPTER 3. IMPLEMENTATION OF REQUEST SCHEDULER.....	9
3.1 PROCEDURES WITHIN RQS	9
3.2 EXAMPLES	12
3.3 INACCURATE PARAMETERS AND COMPENSATION	14
CHAPTER 4. SIMULATION RESULTS.....	16
4.1 SIMULATION ENVIRONMENT.....	16
4.2 THROUGHPUT DIFFERENTIATION	17
4.3 USER-PERCEIVED LATENCY	18
4.5 FAIRNESS IMPROVED BY THE COMPENSATION MECHANISM	19
4.6 THE IMPACT OF INACCURATE $\text{DELAY}_{\text{RESP}}$ AND TRANSMISSION RATE.....	21
CHAPTER 5. MANAGING UPLINK BANDWIDTH BY RQS.....	23
5.1 OPERATION MODEL OF RQS IN MANAGING UPLINK.....	23
5.2 MODIFIED RTC AND RR ALGORITHM	23
5.3 SIMULATION SCENARIO AND RESULTS	24
CHAPTER 6. CONCLUSIONS AND FUTURE WORKS	26
REFERENCES.....	27

List of Figures

FIG. 1: OPERATION MODEL OF A GATEWAY WITH REQUEST SCHEDULER	5
FIG. 2: EXECUTION OF RTC	6
FIG. 3: PROCEDURE IN RTC	11
FIG. 4: PROCEDURE IN RPC	12
FIG. 5: PROCEDURE IN RR	12
FIG. 6: EXAMPLE OF RTC	12
FIG. 7: EXAMPLE OF RR	13
FIG. 8: PROCEDURE IN RR WITH COMPENSATION	14
FIG. 9: SIMULATION TOPOLOGY USED IN THIS WORK.....	16
FIG. 10: DOWNLINK BANDWIDTH USAGE OF EACH CLASS DURING 8 MINUTES	17
FIG. 11: AVERAGE USER-PERCEIVED LATENCY	18
FIG. 12: AVERAGE QUEUING TIME IN RQS GATEWAY	19
FIG. 13: THE SUM OF AVERAGE $DELAY_{RESP}$ AND RESPONSE TRANSMISSION TIME.....	19
FIG. 14: AVERAGE BANDWIDTH USAGE WITHOUT COMPENSATION	20
FIG. 15: AVERAGE BANDWIDTH USAGE WITH COMPENSATION.....	20
FIG. 16: THE IMPACT ON LINK UTILIZATION RESULTED FROM INACCURATE TRANSMISSION RATE	21
FIG. 17: THE IMPACT ON LINK UTILIZATION RESULTED FROM INACCURATE $DELAY_{RESP}$	22
FIG. 18: THE OPERATION MODEL OF UPLINK BANDWIDTH MANAGEMENT	23
FIG. 19: SIMULATION TOPOLOGY.....	24
FIG. 20: UPLINK BANDWIDTH USAGE OF EACH CLASS	24

List of Tables

TABLE 1: PARAMETERS OF RQS ALGORITHMS.....	10
TABLE 2: FUNCTIONS OF RQS ALGORITHMS	10
TABLE 3: DATA STRUCTURES OF RQS ALGORITHMS.....	11



Chapter 1. Introduction

Internet has evolved to provide a multitude of services, more and more enterprises and users use Internet to exchange various kinds of information, causing that the volume of traffic on Internet grows rapidly. Generally speaking, ISPs are willing to invest a lot of money to expand the backbone bandwidth to provide their customers good surfing. However, to minimize costs, their customers often delay upgrading the bandwidth of the access link for as long as possible, and the insufficient bandwidth at the access link results in the serious congestion and long transmission latency. Thus the access link is still the potential bottleneck for enterprises and users who want to access the Internet.

For enterprises, in order to avoid such congestion from degrading the transmission of important connections, some levels of Quality of Service (QoS) must be provided. Currently the common solution to this requirement is deploying the packet-level bandwidth management [1] at an access gateway. The deployment at the access gateway is undoubted because enterprises do not need to install any software into and modify any configuration on their PCs and servers. However, managing the bandwidth of the access link by a packet-level scheduling mechanism at an access gateway has three problems:

Low scalability: The segment of a file or a web page is usually decomposed to a great amount of packets. To manage the link bandwidth, the packet scheduling has to decide the releasing *order* and *time* of each packet. This *per-packet* processing costs much CPU resource and increases the complexity on scheduling, causing the scalability problem.

Scheduling behind the downlink bottleneck: In general, without ISP's support, the

enterprise can only deploy the bandwidth management at the local gateway. Such a management mechanism schedules the *downlink* packets behind the bottleneck, the access link, which the downlink packets have already passed through. Therefore, the downlink scheduling behind the access link is *meaningless*. Actually, these packets should be delivered to end hosts immediately without any scheduling because the bandwidth within enterprise's intranet is far larger than the bandwidth of the access link.

Excessive concurrent transmissions: The packet-level scheduling mainly considers packet delay differentiation between various classes, but does not care which packets belong to one file within a class. When too many files in one queue are transmitted simultaneously, each file may only grab a little bandwidth, implying that all users in this class have to experience a long transmission time before they get the complete file.

The above three problems reveal the inefficiency and ineffectiveness of packet scheduling. Fortunately, scheduling packets is not the only way for managing bandwidth in the today Internet. Most applications running over the Internet adopt the request-response model; that is, a client sends a request to a server, and then the server returns a response to the client. Request scheduling is used in some recent studies to provide Web QoS [2]. These studies provided QoS services on a single Web server [3-6], caching proxies [7-8] and server farms [9-10]. No studies discussed how to design request scheduling at the access gateway. This work proposes a novel approach, the mechanism at an access gateway, ReQuest Scheduling (RQS) which schedules requests to manage the bandwidth of the access link. The RQS achieves three objectives:

1) *High scalability:* According to [11], the size of one request is 1/10 long as that of one response averagely. The ratio indicates that scheduling requests is more *efficient* than scheduling packets of responses.

2) *Differentiated and shared bandwidth:* By controlling the amount of *requests* released from various classes, RQS can allocate the bandwidth of the access link to

different classes for transmitting the *response* packets. Besides, by adopting the Deficit Round-Robin scheduling algorithm [12], free bandwidth can be shared among all active classes.

3) *Reducing congestion*: By controlling the releasing time of a request RQS can roughly estimate the return time of its response. Thus RQS can avoid too many responses competing for the access link simultaneously, reducing the congestion occurrence at the access link and then shortening the mean transmission time of a response.

Note RQS not only can schedule outgoing requests to manage the downlink bandwidth, but also can schedule incoming requests to manage the uplink bandwidth. In fact, the complexity and effect of scheduling incoming requests is easier and stabler, respectively, because the servers accessed by the incoming requests are very close to the access gateway. However, for clearer description, this work first emphasizes on scheduling outgoing requests, and then considers handling incoming requests.

The rest of this paper is organized as follows. Chapter 2 presents the operation model and three components of RQS. Chapter 3 describes the algorithms used in each component and proposes a compensation mechanism for the case that the response size cannot be accurately estimated. The simulation results, in Chapter 4, demonstrate that RQS actually achieves the above three objectives and reveal how the dynamic transmission rate and response delay affect RQS on fairness and link utilization. Chapter 5 discusses the scenario that RQS manages the uplink bandwidth of the access link. Finally, the conclusions and future works are given in Chapter 6.

Chapter 2. Architecture of the ReQuest Scheduler

2.1 Operation Model

Fig. 1 is a typical network topology that an enterprise accesses the Internet services. Requests sent from multiple clients go through the access gateway and the uplink of the access link to the servers in the Internet; then, the corresponding responses returned from the servers come back through the downlink of the access link. To supply differentiated QoS among clients, a request classifier and a ReQuest Scheduler (RQS), deployed in the access gateway, classifies and schedules the requests sent from clients, respectively, as shown in Fig. 1. The request classifier puts every arriving request into a request class queue according to the user-defined classification policies. The RQS mechanism *controls the sending rate of requests* to allocate the differentiated downlink bandwidth for each class. This rate control involves the decision of the sending *order* and the sending *time* of a request. Under the operation of RQS, the user-perceived latency consists of the time queued in the gateway with RQS, $Delay_{resp}$, and response transmission time, where $Delay_{resp}$ is the delay from releasing the request to receiving the first packet of its response at RQS and can be considered as the total of the RTT of the gateway and the processing time of the server. In fact, $Delay_{resp}$ can be seen as the response time of one request [13] experienced by RQS. RQS is designed to handle the application protocols that use request-response model and the response size can be known after receives the first response packet. For example, HTTP, IMAP4, POP3, and some streaming video services are such application protocols that fit these two assumptions.

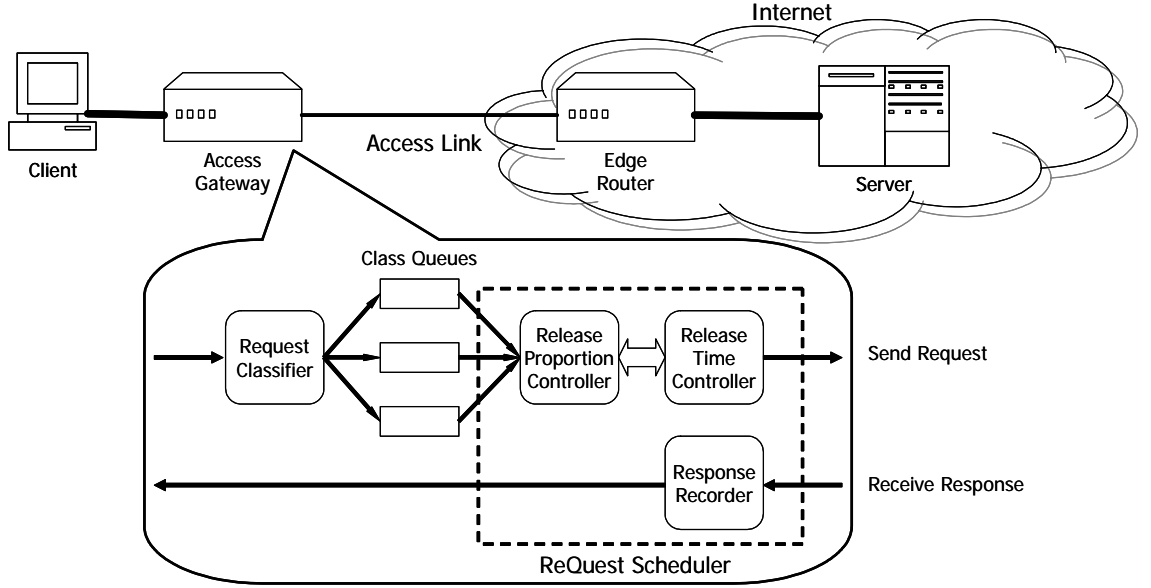


Fig. 1: Operation Model of a gateway with ReQuest Scheduler

RQS consists of three components, namely, Release Proportion Controller (RPC), Release Time Controller (RTC), and Response Recorder (RR). RTC releases the request selected by RPC at a "suitable" time. Releasing the request at this suitable time will let the corresponding response occupy at a "right" time, a downlink bandwidth which is equal to the bandwidth supplied by the Internet. RPC controls the released amount of requests for each class, and thus determines its occupied proportion on the downlink bandwidth. To estimate the releasing time of a request and decide the released proportion for each class, RTC requires two parameters, the transmission *rate* and $Delay_{resp}$ of the response, and RPC require one parameter, the response *size* of the request. RR is responsible to monitor the response traffic and then update these parameters.

2.2 Release Time Controller (RTC)

RTC is invoked in the two cases. The first case is when a new request arrives in a queue and the other queues are empty. The other case is at the scheduled instant, as shown in Fig. 2, which is determined by RR. Specifically, the scheduled time is the *lookahead* instant before the finishing time of a response transmission that is at $(T_{minFT} -$

$T_{lookahead}$), where $T_{lookahead}$ is the advance time and T_{minFT} is the minimum finishing time among all active response transmissions. Once invoked, RTC computes the available downlink bandwidth after T_{minFT} . Next, RTC gets requests from RPC and sends them out if the free bandwidth is sufficient to transmit responses.

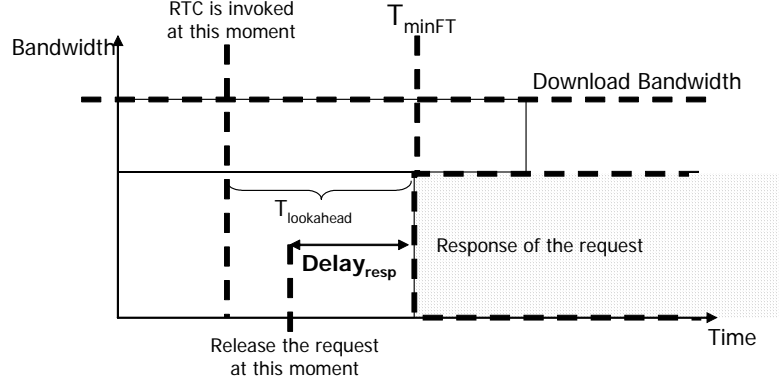


Fig. 2: Execution of RTC

In the following, the choice of $T_{lookahead}$ is explained. If a response is expected to return and fill up the bandwidth available beginning from T_{minFT} , RTC needs to release its request at $T_{minFT} - Delay_{resp}$. In fact, $Delay_{resp}$ among connections are different, so RTC computes the available bandwidth at $T_{minFT} - T_{lookahead}$ where $T_{lookahead}$ should be equal to the maximum of the $Delay_{resp}$ of all connections. If $T_{lookahead}$ is smaller than some requests' $Delay_{resp}$, the response of these requests won't come back by T_{minFT} , the time when the downloading response finishes. Since $T_{lookahead}$ and T_{minFT} are estimated by RR, the scheduled time at invoking RTC is scheduled by RR.

2.3 Release Proportion Controller (RPC)

Every time when RPC is invoked by RTC, RPC selects a request from queues. The amount of requests selected by RPC in long term for one class decides the proportion on the downlink bandwidth occupied by the corresponding response traffic in this class. The selecting algorithm applied in RPC is derived from the Deficit Round Robin (DRR) algorithm [12]. DRR is one of packet fair queuing algorithms, which guarantees all of

the flows passing through a link proportionally share the bandwidth of the link. DRR serves the queues in a round-robin manner. As a queue is served, the scheduler allows a given quantum in bytes or bits to send out from this queue. If the quantum is insufficient for sending out the first packet in the queue, the deficit counter of this queue saves the residual quantum for using in the next round. Then the scheduler serves the next queue.

However, applying DRR in RPC requires two modifications. First, the proportion on quantum size among classes should decide the proportion on the *response* traffic among classes, not on the *request* traffic. Thus, sending a request is determined on whether the quantum is sufficient for the size of its corresponding response, not the size of itself. Second, in RPC transmitting requests does not cost the downlink bandwidth, the target to be scheduled. Therefore, if RPC sends requests one-by-one as the way DRR sends packets, the returning response traffic may overflow the downlink bandwidth. For this, in RQS, RPC only decides from which queue to get the next request, but RTC decides when to send the request. When RTC forecasts there are free downlink bandwidth, it get the appropriate request from RPC and then send it out. In this way, the sending rate of requests in RPC actually depends on the target bandwidth to be scheduled.

2.4 Response Recorder (RR)

For scheduling a request having been forwarded by RQS, RR provides the historical size, rate and $Delay_{resp}$ of the response transmission corresponding to this request. On the other hand, for a new request, RR needs to provide the initial estimation of these three parameters. $Delay_{resp}$ can be predicted by monitoring the TCP 3-way handshaking when client connects to server. Response size can be assigned by the average value of the response type and transmission rate can be assigned by the average value by using historical records. Besides providing the three parameters, another important job in RR

is to schedule the next invoked time of RTC, $T_{minFT} - T_{lookahead}$. To get T_{minFT} , the finishing time of each response transmission is necessary and calculated by RR from the size and rate of the response.



Chapter 3. Implementation of ReQuest Scheduler

3.1 Procedures within RQS

Fig. 3, 4 and 5 lists the pseudo codes executed in RTC, RPC, and RR, respectively. Table 1, 2 and 3 list the variables, functions, and data structures, such as H_{FT} , $H_{RELEASE}$, and Q_{RPC} , used in the pseudo code, respectively.

a) *The procedure in RTC:* As presented in Fig. 3, when RTC is invoked, the first loop from H_{FT} removes all of the requests whose finishing time of response transmission equals to T_{minFT} to accumulate the bandwidth going to be released by these requests and thus available after T_{minFT} . In the second loop, if the available bandwidth, BW , is enough for the head request in Q_{RPC} to receive its response with the rate offered by the Internet, denoted as $HEAD(Q_{RPC}).BW$, RTC calls $DEQUEUE(Q_{RPC})$ function to get this header request from RPC, and then subtracts the bandwidth occupied by its response, denoted as $Req.BW$, from BW . Next, RTC determines the releasing time of this request as $T_{minFT} - Delay_{resp}$ of the request. Finally, this scheduled request is put into $H_{RELEASE}$, and going to be released when time is up. If the available bandwidth is still enough for the next requests in Q_{RPC} , RTC continues to arrange them. Otherwise, it sleeps. Since the responses of these scheduled requests are returning in the future, RTC also puts them into H_{FT} , but their finishing time are set to infinite at present since the times are unknown before RR receives the first packets of their responses.

b) *The procedure in RPC:* RPC, invoked by $DEQUEUE(Q_{RPC})$ in RTC, is responsible for selecting a request from queues. Fig. 4 lists the pseudo codes of RPC, which is similar to the DRR algorithm. An index, i , is maintained in RPC to point the next queue to be served. If the queue Qu_i is empty, then the deficit counter of this queue, DC_i , will be set to 0. Otherwise, the quantum of this class queue, Q_i , will be added to

DC_i . The Heading-of-line (HOL) request in queue is picked only when the value of deficit counter is bigger than the response size of this request. After the request is dequeued from the queue, the deficit counter must be subtracted from its response size.

c) *The procedure in RR*: RR is invoked when the first packet of a response comes back to the access gateway. RR receives a response packet then calculates the transmission time by measuring the response size and bandwidth usage. Finally RR decides the next trigger time of RTC by a look ahead time before the minimum finishing time among all active response transmissions.

Table 1: Parameters of RQS algorithms

Parameter Name	Parameter Description
Req	A request object
$req.BW$	Bandwidth usage for transmitting the response relative to the request object
$req.FT$	Transmission finish time for the response relative to the request object
$req.D$	The $Delay_{resp}$ of the request object.
$req.SIZE$	Response size of the request object
DC_i	The deficit counter of class $_i$
Q_i	The quantum size of class $_i$
T_{minFT}	The minimum finishing time among all active response transmissions
$T_{lookahead}$	The period between the finish time of a response and the execution time of RTC
BW	The variable for accumulating available bandwidth

Table 2: Functions of RQS algorithms

Function Name	Function Description
$EMPTY(DS_{xxx})$	Check if the data structure DS_{xxx} is empty or not
$HEAD(DS_{xxx})$	Get parameter from the first object of DS_{xxx}
$DEQUEUE(DS_{xxx})$	Remove the first object of DS_{xxx}
$INSERT(req, DS_{xxx})$	Insert the request object req into DS_{xxx}
$SCH_RTC(Time)$	Set the next trigger time of RTC

Table 3: Data Structures of RQS algorithms

Data Structure Name	Data Structure Description
H_{FT}	This min-heap tree mounts all requests whose responses are transmitting. The root of this heap presents the request whose response will be transmitted completely earliest.
$H_{RELEASE}$	The min-heap tree mounts all un-transmitted requests whose released time has be decided by RTC. The root in this heap presents the request has the earliest released time.
Q_{RPC}	This queue is not a real queue presented in RQS. The order of requests in it shows the sequence of requests released from RPC. That is, the first object in this queue is the next request selected by the modified DRR algorithm in RPC
Qu_i	The queue for queuing the request arriving from Class $_i$

```

Procedure_in_RTC
// The first loop
TminFT = HEAD(HFT).FT
do {
    req = DEQUEUE(HFT)
    BW = BW + req.BW
} while(HEAD(HFT).FT = TminFT)

// The second loop
while (HEAD(QRPC).BW < BW)
{
    req = DEQUEUE(QRPC)
    BW -= req.BW
    req.release = TminFT - req.D
    INSERT(req, HRELEASE)
    INSERT(req, HFT)
}
sleep()

```

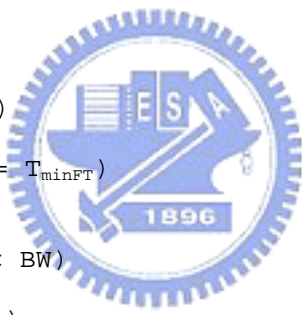


Fig. 3: Procedure in RTC

```

Procedure_in_RPC
do {
  if (EMPTY(Qi)) {
    DCi = 0
  } else {
    DCi = DCi + Qi
    if (HEAD(Qi).SIZE <= DCi) {
      req = DEQUEUE(Qi)
      DCi = DCi - req.SIZE
      return req
    }
  }
  i = (i + 1) mod n
} while (!(EMPTY(Qi) for all i))
sleep()

```

Fig. 4: Procedure in RPC

```

Procedure_in_RR
{
  Get req.SIZE
  req.FT = TCURR + req.SIZE/req.BW
  SCH_RTC(HEAD(HFT).FT - Tlookahead)
}

```

Fig. 5: Procedure in RR

3.2 Examples

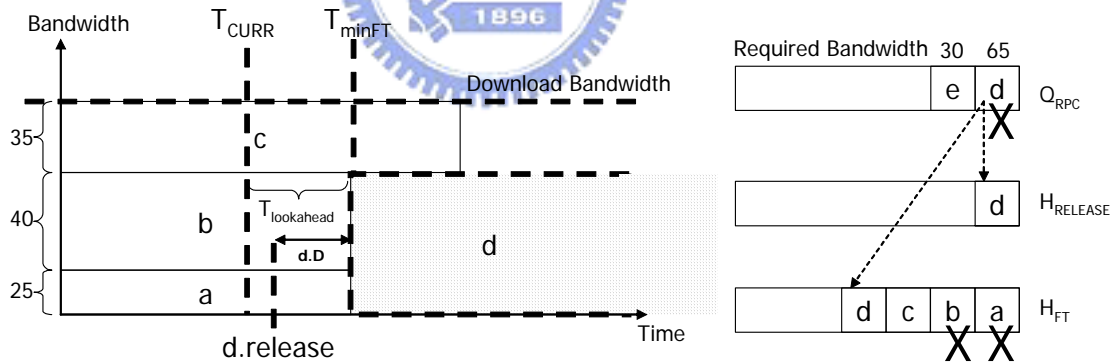


Fig. 6: Example of RTC

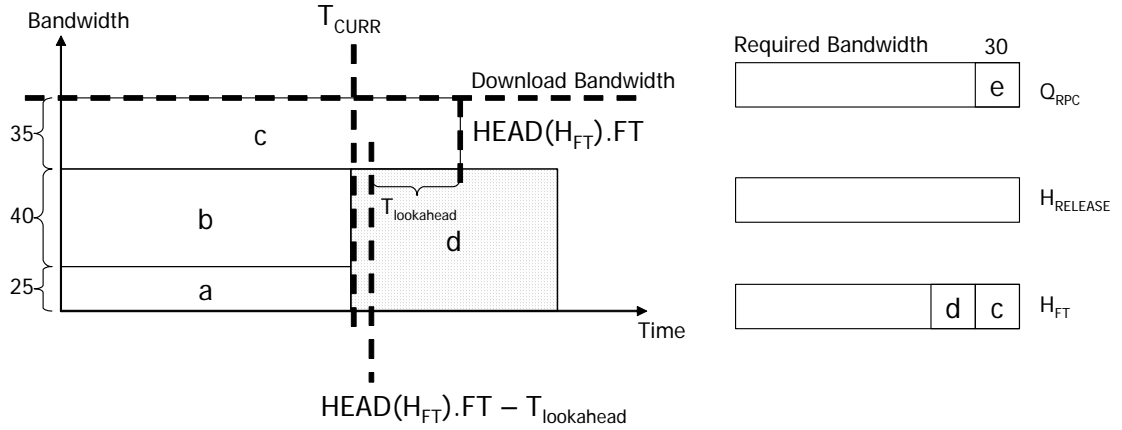


Fig. 7: Example of RR

To illustrate the interactive operation between RTC, RPC and RR, an example is given under the assumption that the exact transmission rate and required time on transmitting a response are known in advance. As shown on the left of Fig. 6, assume that 3 requests: a , b , and c , have been released before the time T_{CURR} , and their corresponding responses are transmitting and exhausts all downlink bandwidth. Among them, the finishing time of the responses corresponding to a and b should be at the variable for accumulating available bandwidth which is available at T_{minFT} and before that to c . Thus, at current time, T_{CURR} , i.e., $T_{minFT} - T_{lookahead}$, H_{FT} should contain three requests and a is its root node since the request a arrives before b and will finish the transmission before c . Meanwhile, RTC is also invoked at this time, scheduled by RR. Once invoked, RTC pops up a and b from H_{FT} to update the available bandwidth after T_{minFT} from 0 to $(25+40)$. Obviously, the available bandwidth is enough for transmitting the response of the request d , the head of Q_{RPC} .

Thus, as shown in Fig. 6, RTC gets request d from RPC and then calculates its releasing time. In this example, the best releasing time of request d is $T_{CURR} + T_{lookahead} - d.D$, since the response of d will come back after $d.D$ and is expected to fill the bandwidth released by the response of a and b . Next, the request d is inserted into $H_{RELEASE}$ and into H_{FT} . The finishing time of d is set to infinite since the time is

unknown before RR receives the first packet of its response. Since the available bandwidth is not enough for the response of the next request e , RTC sleeps.

Fig. 7 shows the time that RR receives the first response packet of d . The first job of RR is to re-compute the finishing time of d , $d.FT$. H_{FT} rewrite it into H_{FT} , and reconstruct H_{FT} . In this case, the first request in H_{FT} is still c , meaning $c.FT$ is the next time that the available bandwidth will be increased. Finally, RTC is scheduled to be invoked at $c.FT - T_{lookahead}$.

3.3 Inaccurate Parameters and Compensation

Unfortunately, the parameters provided by RR are not always correct for two reasons. First, the parameters of new requests cannot be actually estimated. Second, even for the forwarded requests, their historical parameters recorded by RR may affect by the flexibility of the Internet. Inaccurate $Delay_{resp}$ and transmission rate may affect the access-link utilization because RTC needs these two parameters in order to feed some responses to the available bandwidth of downlink. Inaccurate response size may affect the fairness of access-link bandwidth shared between various classes because in RPC the response size of a request determines the releasing order of the request.

A revised RR with a compensation mechanism is proposed to prevent the unfair sharing between various classes resulted from inaccurate response size.

```

Get req.SIZE
if (req.SIZE > the value used in RPC) {
    DCi = DCi - abs(req.SIZE - the value used in RPC)
} else {
    DCi = DCi + abs(req.SIZE - the value used in RPC)
}
req.FT = TCURR + req.SIZE/req.BW
SCH_RTC(HEAD(HFT).FT - Tlookahead)

```

Fig. 8: Procedure in RR with Compensation

Fig. 8 shows the modified RR algorithm with compensation. This revision

compensates the corresponding deficit counter in RPC after the accurate response size is known. Generally, the size is known after the first response packet is received or the transmission is finished. As shown as pseudo codes with the bold style in Fig. 8, if the real size is bigger or smaller than the value used in RPC, the corresponding deficit counter must be compensated with this difference.



Chapter 4. Simulation Results

4.1 Simulation Environment

The network simulation tool, *ns-2* [14], is used to simulate the operation of RQS. The class *Http/Cache* in *ns-2* models behaves as a simple HTTP cache with infinite size, so it will intercept the requests sent from clients and then will forward them to the web server. Thus, we implement the RQS algorithm on the class *Http/Cache*.

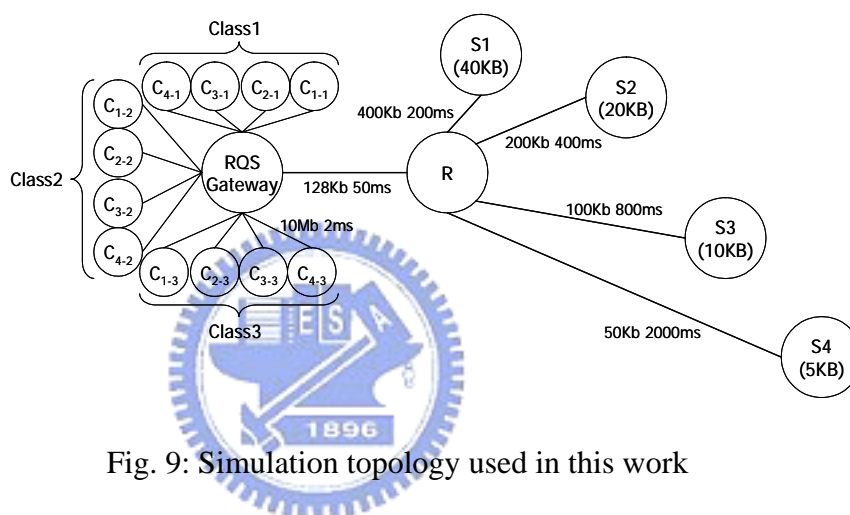


Fig. 9: Simulation topology used in this work

Fig. 9 shows the simulation topology used in this work. The RQS gateway provides three classes. Twelve clients connecting to the RQS gateway are equally divided into the three classes. The link between the RQS gateway and a client is 10Mb/s with 2 ms propagation delay. Besides, a 128Kb link with 50 ms propagation delay connects the RQS gateway with an ISP's edge route R to simulate the access link. Next, the route R connects to four servers with four independent links, whose configurations are presented in Fig. 9. The fixed response sizes in the four servers are 40KB, 20KB, 10KB, and 5KB, respectively. The client named as C_{x-y} represents that it only send requests to server x and their responses belong to class y. In other words, each server has to serve three clients belonging to three different classes. Note that the client can only send the next request until the whole response packets requested by the last request

return.

4.2 Throughput Differentiation

First, we demonstrate that RQS can actually allocate the proportional bandwidth to different classes and let active classes share the free bandwidth. The proportion of the bandwidth allocated for three classes is given as 4:2:1. At the beginning, clients in Class1 and Class2 send requests and after four minutes, clients in Class3 join. Fig. 10 shows the bandwidth usage of three classes during 8 minutes. At 0~4 minutes, Class1 and Class2 occupy the downlink bandwidth and their average bandwidth usages are equal. The reason that the ratio is not 4:2 is that the mean transmission rate of the response traffic in Class1 (55 Kbps) is not high enough to occupy $2/3$ bandwidth of the access link (256/3, or 85.33 Kbps). Thus, the Class2 fairly shares the bandwidth with Class1. However, at the 4th min, Class3 starts to occupy the downlink bandwidth. The proportion of the bandwidth usage occupied by three classes is 57:37:18. Compared to the average target ratio between various classes, $(4/2 + 2/1)/2$, or 2, that under RQS is $((57/37+37/18)/2)$, i.e., 1.798. The accuracy is $1.798/2=89.9\%$

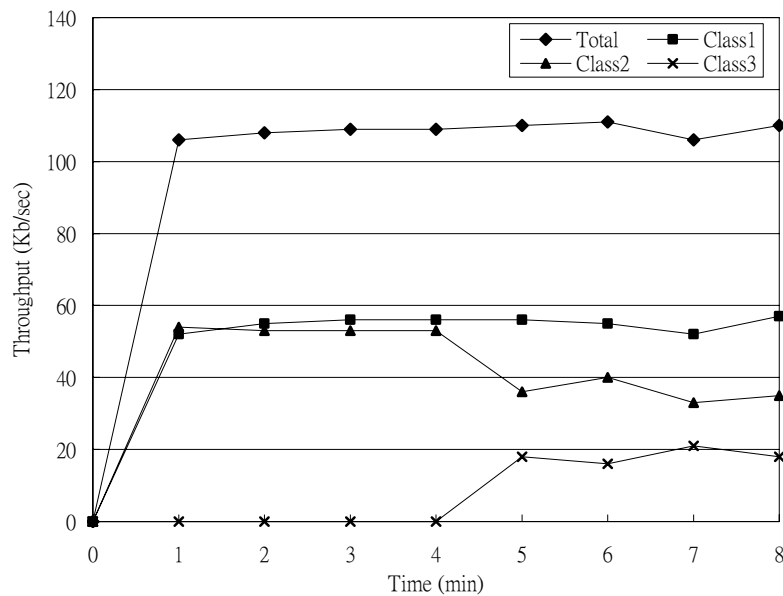


Fig. 10: Downlink bandwidth usage of each class during 8 minutes

4.3 User-perceived Latency

Next, we observe the user-perceived latency experienced by the clients in three classes during 8 minutes. The simulation scenario here is the same as that in Chapter 4.2, except that all clients send requests from the beginning. For the clients accessing the same server, as shown in Fig. 11, the higher class the client belongs to, the shorter user-perceived latency the client experiences. Besides, the differences of latency between various classes decrease with the increase of server's number, and this phenomenon can be further observed from Fig. 12 and 13. Fig. 12 shows the time queued in RQS and Fig. 13 shows the sum of $Delay_{resp}$ and response transmission time. Obviously, as shown in Fig. 12, the queuing delay in RQS for different classes determines the differences of latency between classes. That is because RQS delays the release of a request according to its response size, and the response size in Server1 is the larger than other servers. In other words, all clients connecting to Server1 get larger response sizes, i.e. longer transmission times, and therefore have longer waiting times for the next service than the waiting times experienced by clients of Server4.

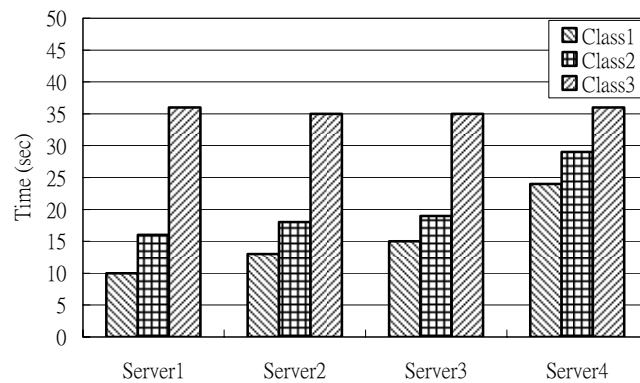


Fig. 11: Average user-perceived latency

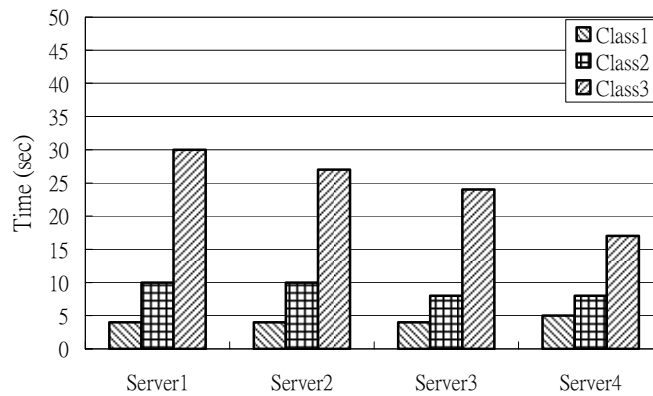


Fig. 12: Average queuing time in RQS gateway

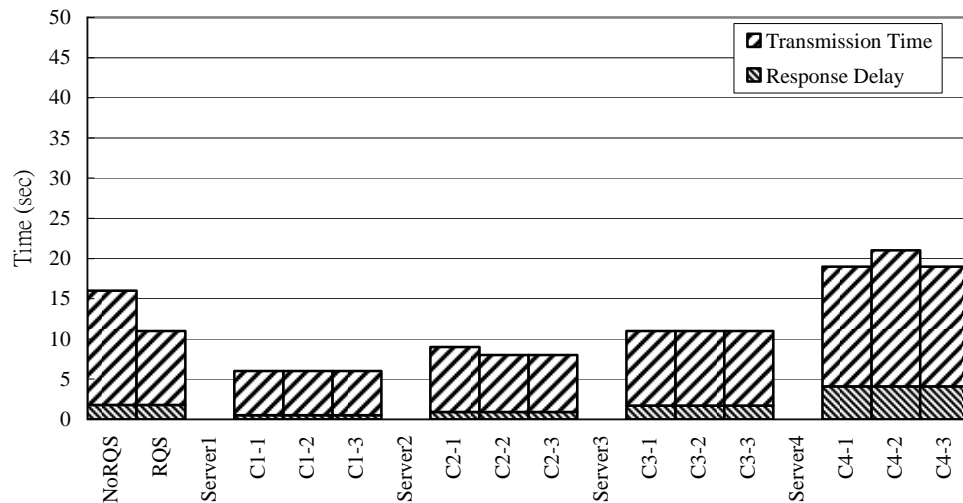


Fig. 13: The sum of average $Delay_{resp}$ and response transmission time

4.4 Response Transmission Time

Finally, as shown in the most left group of Fig. 13, the average transmission time of user-perceived latency for all classes under RQS is 68.75% shorter than that under no RQS. Thus RQS can decrease the transmission time, that is, it actually avoids too many connections competing for the access link simultaneously.

4.5 Fairness Improved by the Compensation Mechanism

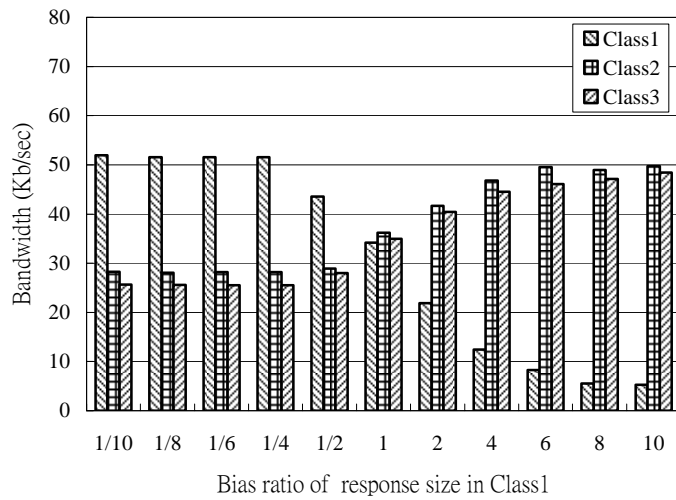


Fig. 14: Average bandwidth usage without compensation

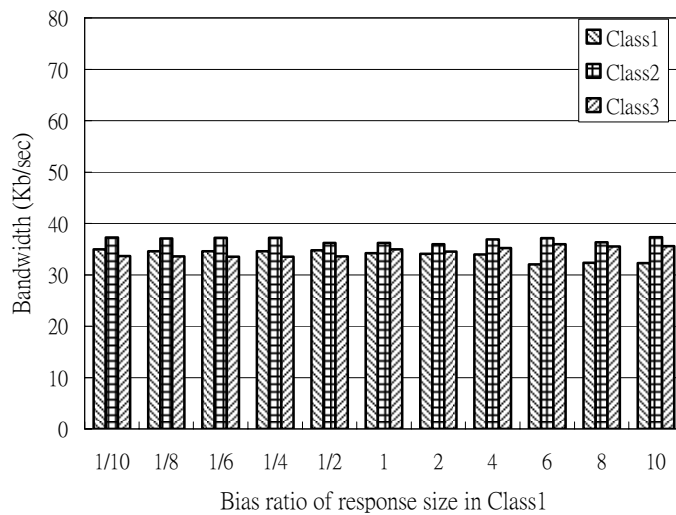


Fig. 15: Average bandwidth usage with compensation

In this evaluation, all clients of three classes send requests from the beginning and the bandwidth allocated for three classes are equal, i.e. the quantum ratio is 1:1:1. Next, assume that RQS can accurately estimate the size of the response, *except* the response belonging to Class1. The biasing ratio (size used in RQS/ real size) for the clients in the Class1 ranges from 1/10 to 10. For example, the biasing ratio being equal to 10 presents that RQS thinks the size of a response is 100KB although its real size is only 10KB.

This simulation runs in 8 minutes, and the average bandwidth usage is measured from the second minute for stability.

Fig. 14 shows the effect of inaccurate estimation of response size on the fairness between classes, and Fig. 15 demonstrates the improvement of using a compensation mechanism in RR. It is clear in Fig. 14 that wrong response size used in RPC seriously degrades the fairness on bandwidth usage between various classes. In the cases that the biasing ratio is larger than one, the bandwidth usage of Class1 is lower than the expected because its deficit counter in RPC is over-decreased. In the cases that the biasing ratio is smaller than one, the situation is reversed. When RR applies the compensation mechanism, as shown in Fig. 15, the bandwidth usage between three classes is quite close to the targeted ratio, 1:1:1, in all cases.

4.6 The Impact of Inaccurate $\text{Delay}_{\text{resp}}$ and Transmission Rate

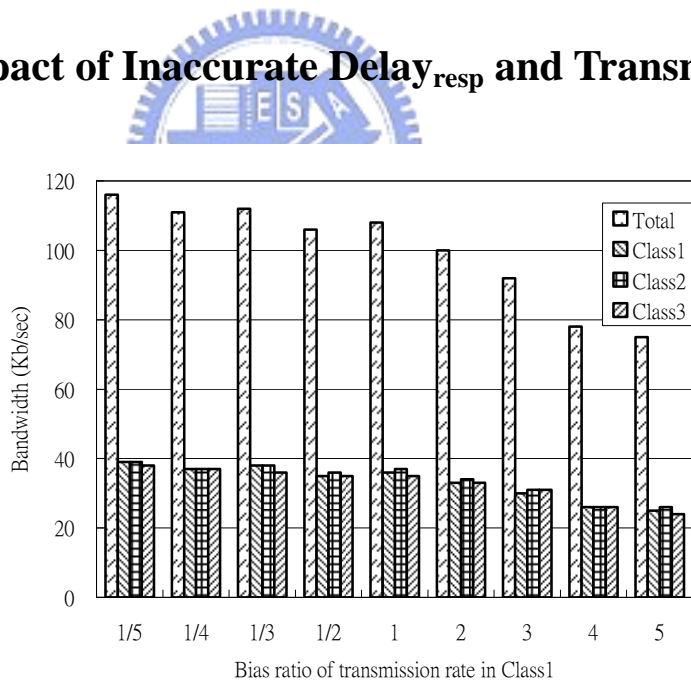


Fig. 16: The impact on link utilization resulted from inaccurate transmission rate

Fig. 16 shows the impact of inaccurate estimation of the mean transmission rate on link utilization. All clients of three classes send requests from the beginning. The quantum ratio between three classes is 1:1:1. The first observation is the total link utilization is decreased with the increase of the biasing ratio of transmission rate, i.e. the

actual sending rate of the response traffic is smaller than the estimation used in RQS. In addition, although the utilization is higher when the ratio is smaller than 1, serious congestion will occur at the access link, and it is not preferred.

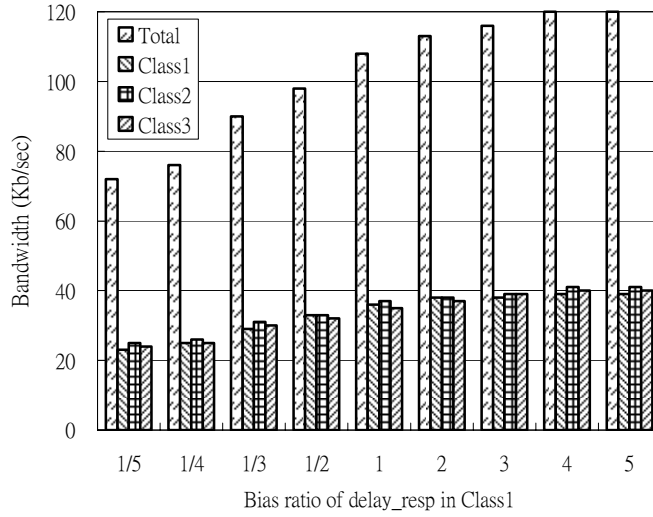


Fig. 17: The impact on link utilization resulted from inaccurate $Delay_{resp}$

Fig. 17 shows the results as $Delay_{resp}$ cannot be estimated accurately. The biasing ratio larger than one means the value estimated in RQS is larger than the actual $Delay_{resp}$, leading to *unexpected* response traffic competing for the access link. Contrarily, when the ratio is smaller than one means the actual $Delay_{resp}$ is smaller than the value used in RQS, causing one response cannot follow another finished response well. Therefore, available bandwidth is wasted and the link utilization is going down.

Although inaccurate $Delay_{resp}$ and transmission rate affect the link utilization, but the bandwidth usage ratio of three classes still close to the targeted ratio of three classes, i.e., 1:1:1.

Chapter 5. Managing Uplink Bandwidth by RQS

5.1 Operation Model of RQS in Managing Uplink

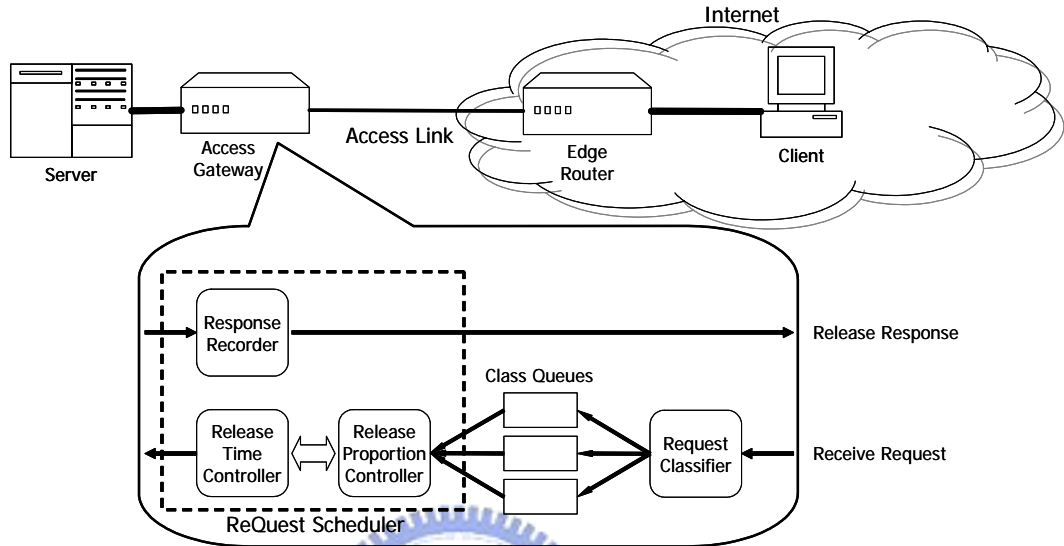


Fig. 18: The operation model of uplink bandwidth management

In the previous chapters, as shown in Fig. 1, the discussion of RQS focuses on how to manage the downlink bandwidth of the access link by controlling the outgoing requests in an environment where the web servers are in the Internet. In such a scenario, the traffic on uplink is usually light because the size of the request packet is very small. That is, the bottleneck is the downlink of the access link when internal clients in an enterprise access the web servers in the Internet. However, as shown in Fig. 18, the uplink may turn to be the bottleneck if the enterprise places a web server in the internal network and allows clients coming from the Internet to access it. Thus, in this chapter RQS is introduced to manage the bandwidth of uplink by skillfully scheduling the *incoming* requests.

5.2 Modified RTC and RR Algorithm

In the original operation of RQS mechanism, three parameters of a request are needed, namely, response size, transmission rate of response and $Delay_{resp}$. As shown in

Fig. 18, the server in the intranet is very close to the access gateway of enterprise, and thus the value of $Delay_{resp}$ should be small enough to be ignored in the procedure of RTC. Therefore, the advance time $T_{lookahead}$ can also be eliminated from the procedure in RR since $T_{lookahead}$ is the maximum value of $Delay_{resp}$. By such two simplifications, RQS can be easily applied to manage the uplink bandwidth.

5.3 Simulation Scenario and Results

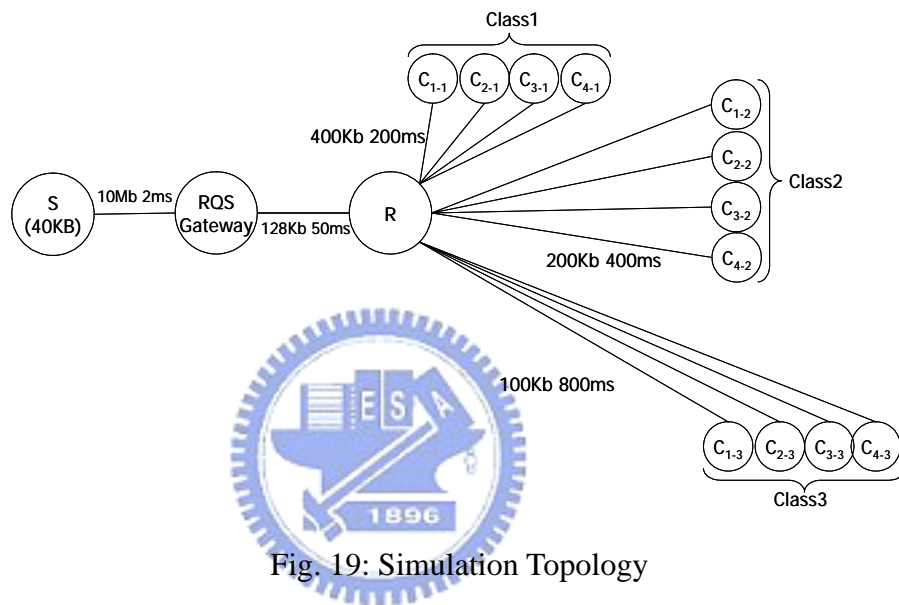


Fig. 19: Simulation Topology

Fig. 19 shows the simulation topology that clients in the Internet access the web server in the Intranet. Twelve clients, divided into three classes uniformly, access the single server near the RQS gateway. The quantum ratio of three classes is 4:2:1.

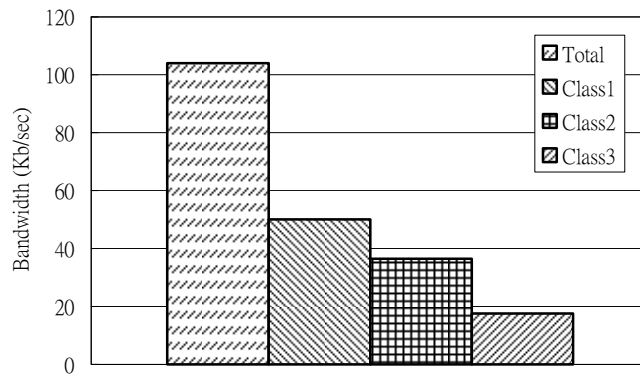


Fig. 20: Uplink Bandwidth usage of each class

Fig. 20 shows the bandwidth usage between three classes during eight minutes. This result shows that RQS is capable to manage uplink bandwidth by controlling the releasing of incoming requests. The ratio on bandwidth usage of three classes, 50:36:18 is close to the targeted ratio.



Chapter 6. Conclusions and Future Works

This work proposes the ReQuest Scheduler (RQS) at consumer-side access gateway to overcome the three problems of packet-level scheduling on managing the access-link bandwidth. RQS controls the releasing rate of requests in one class to arrange concurrent transmissions and pre-allocate the bandwidth of the access link occupied by the response traffic corresponding to this class. Thus, RQS can provide proportional and sharing bandwidth between various classes, and reduce the congestion occurring at the access link.

The simulation results show the ratio on bandwidth usage between classes scheduled by RQS is close to the targeted ratio (The similar degree is 89.75%) and the free bandwidth can be shared for active classes. Besides, the mean transmission time is decreased to 68.75% of the original value, revealing that RQS avoids too many responses competing for the access link simultaneously and thus reduces the congestion. Finally, by the compensation mechanism, RQS is robust to achieve the fairness between classes even when the response size is inaccurately estimated.

In the future, we will study how to reduce the impact on link utilization as the transmission rate of a response or the interval between sending a request and received its first response packet cannot be accurately estimated. On the other hand, handling application protocol that do not use request-response model and testing RQS in real world environment are the works to be done.

References

- [1] F. P. Kelly "Effective bandwidths at multi-class queues," *Queueing Systems: Theory and Applications*, v.9 n.1-2, p.5-16, Oct. 1991
- [2] M. Conti, M. Kumar, S. K. Das, and B. A. Shirazi, "Quality of Service Issues in Internet Web Services," *IEEE Transactions on Computers*, Vol.51, No.6, June 2002
- [3] R. Pandey, J. Fritz Barnes, and R. Fritz Barnes, "Supporting Quality of Service in HTTP Servers," *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 247-256, 1998.
- [4] N. Bhatti, A. Bouch, A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design," *Proceedings of the 9th International World Wide Web Conference*, 2000.
- [5] L. Cherkasova and P. Phaa, "Session Based Admission Control: a Mechanism for Web QoS," *Proceedings of the International Workshop on Quality of Service*, 1999.
- [6] L. Eggert and J. Heidemann, "Application-Level Differentiated Services for Web Servers," *World Wide Web Journal*, vol. 3, Issue 2, pp. 133-142, 1999.
- [7] T. P. Kelly, S. Jamin, and J. MacKie-Mason, "Variable QoS from Shared Web Caches: User-Centered Design and Value-Sensitive Replacement," *Proceedings of IEEE Conference on Computer Communications*, 2002.
- [8] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao, "An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services," *Proceedings of the International Workshop on Quality of Service*, 2002.
- [9] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli, "Enhancing a Web-Server Cluster with Quality of Service Mechanisms," *Proceedings of IEEE International Performance Computing and Communications Conference*, 2002.
- [10] E. Casalicchio and M. Colajanni "A Client-Aware Dispatching Algorithm for Web Clusters Providing Multiple Services," *Proceedings of the 10th International World Wide Web Conference*, 2001.
- [11] F. Donelson Smith, F. Herndndez Campos, K. Jeffay, and D. Ott "What TCP/IP Protocol Headers Can Tell Us About The Web," *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*
- [12] M. Shreedhar, and G. Varghese "Efficient Fair Queuing using Deficit Round Robin," *IEEE/ACM Transactions on Networking (TON)*, 1996, vol. 4, no. 3, pp. 375-385
- [13] P. Mills and C. Loosley, "A performance Analysis of 40 e-Business Web Sites," *CMG Journal of Computer Resource Management*, Issue 102, spring 2001.
- [14] The Network Simulator ns-2, <http://www.isi.edu/nsnam/ns/>