IMPLEMENTATION OF A NEW SCHEDULING POLICY IN WEB SERVERS

Ahmad S. Al Sa'deh
Computer Engineering Department, Prince Fahad Bin Sultan National University, Tabuk, Saudi Arabia

Adnan H. Yahya
Computer Systems Engineering Department, Birzeit University, Birzeit, Palestine

Keywords: Web Server Performance, Request Scheduling Policy, Remaining Response Time Scheduling.

Abstract: Recently, the Shortest-Remaining-Processing-Time (SRPT) has been proposed for scheduling static HTTP requests in web servers to reduce the mean response time. The SRPT assumes that the response time of the requested file is strongly proportional to its size. This assumption is unwarranted in Internet environment. Thus, we proposed the Shortest-Remaining-Response-Time (SRRT) that better estimates the response time for static HTTP. The SRRT prioritizes requests based on a combination of the current round-trip-time (RTT), TCP congestion window size (cwnd) and the size of what remains of the requested file. We compare SRRT to SRPT and Processor-Sharing (PS) policies. The SRRT shows the best improvement in the mean response time. SRRT gives an average improvement of about 7.5% over SRPT. This improvement comes at a negligible expense in response time for long requests. We found that under 100Mbps link, only 1.5% of long requests have longer response times than under PS. The longest request under SRRT has an increase in response time by a factor 1.7 over PS. For 10Mbps link, only 2.4% of requests are penalized, and SRRT increases the longest request time by a factor 2.2 over PS.

1 INTRODUCTION

Today busy web servers are required to service many clients simultaneously, sometimes up to tens of thousands of concurrent clients (Kegel, 2006). If a busy web server's total request rate increases above the total link capacity or the total server concurrent users, the number of rejected requests increases dramatically and the server offers poor performance and long *response time*, where the *response time* of a client is defined as the duration from when the client makes a request until the entire file is received by the client. The slow response times and difficult navigation are the most common complaints of Internet users (King, 2003). Research shows the need for fast response time. The response time should be around 8 seconds as the limit of people's ability to keep their attention focus while waiting (Nielsen, 1997). The question arises, what can we do to improve the response time at busy web servers?

It is possible to reduce the mean response time of requests at a web server by simply changing the order in which we schedule the requests. A traditional scheduling policy in web servers is Processor-Sharing (PS) scheduling. In PS each of *n* competing requests (processes) gets *1/n* of the CPU time, and is given an equal share of the bottleneck link. It has been known from queuing theory that Shortest-Remaining-Processing-Time (SRPT) scheduling policy is an optimal algorithm for minimizing mean response time (Schrage, 1968). However, the optimal efficiency of SRPT depends on knowing the response time of the requests in advance, and under the assumption that preemption in SRPT implies no additional overhead.

The SRPT scheduling policies on web servers (Crovella et al, 1999; Harchol-Balter et al, 2003; Schroeder and Harchol-Balter, 2006) used the job size, which is well known to the server, to refer to

[*] This paper is based on Ahmad Al Sa'deh's Master Thesis in Scientific Computing work carried out at Birzeit University, Birzeit, Palestine under the supervision of Prof. Adnan Yahya.

S. Al Sa'deh A. and H. Yahya A. (2008).
IMPLEMENTATION OF A NEW SCHEDULING POLICY IN WEB SERVERS.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 22-29
DOI: 10.5220/0001524200220029

processing time (response time) of the job to implement SRPT for web servers to improve user-perceived performance. In the Internet environment, depending only on the file size for estimating the response time is not enough since it does not take into consideration the client-server interaction parameters over the Internet like Round-Trip-Time (RTT), bandwidth diversity, and loss rate. Dong Lu et al. (Lu et al, 2005) have shown that the correlation between the file size and the response time are low, and that the performance of SRPT scheduling on web servers degrade dramatically due to weak correlation between the file size and the response time in many regimes.

To better estimate the user response time we proposed a new scheduling policy in web servers which is called Shortest-Remaining-Response-Time (SRRT) to improve the mean response time of clients. The proposed method estimates the response time for a web client by benefiting from the TCP implementation at the server side only, without introducing extra traffic into the network or even storing historical data on the server. The SRRT estimates the client response time in each visit to a server, and then schedules the requests based on the shortest remaining response time request first. SRRT uses RTT and TCP congestion window size (cwnd) in addition to the size of the requested file for estimating the response time. The *getsockopt()* Linux system call is used by SRRT to get the RTT value and the cwnd "on-the-fly" for each connection. See section 3 for the complete description of SRRT algorithm.

For our experiment, we use a web workload generator to generate requests with certain distribution and focus only on static HTTP requests which form a major percentage of the web traffic (Manley and Seltzer, 1997; Harchol-Balter et al, 2003). In 2004, logs from proxy servers show that 67-73% of the requests are for static contents (IRCache, 2004). The experiment uses the Linux operating system and Apache web server. Network Emulator represents the WAN environment.

The SRRT is compared to the PS and SRPT scheduling policies in web servers. We find that the SRRT gives the minimum mean response time. We conclude that the client response time is affected by the Internet conditions. So the priority based scheduling policy in web servers should take into consideration the Internet conditions to prioritize the requests.

The rest of the paper is structured as follows. Section 2 discusses relevant previous work in web server requests scheduling. The SRRT scheduling algorithm is presented in section 3. The modifications for Apache web server and Linux operating system to implement SRRT are covered in section 4. The experiment setup and results analysis are given in section 5. Section 6 summarizes the results obtained and discusses possible future work.

2 LITERATURE REVIEW

It is well known from scheduling theory literature (Schrage and Miller, 1966; Schrage, 1968; Smith, 1976; Goerg, 1986) that if the task sizes are known, the SRPT scheduling is optimal for reducing the queuing time, therefore reducing the mean response time. The work based on the SRPT algorithm for web server scheduling can be divided into three categories: web server scheduling theoretical studies, scheduling simulation studies, and scheduling implementation.

N. Bansal and M. Harchol-Balter (Bansal and Harchol-Balter, 2001) compare the SRPT policy and the PS policy analytically for an M/G/1 queue with job size distributions that are modelled by a Bounded Pareto distribution. They show that with link utilization 0.9, the large jobs perform better under the M/G/1 SRPT queue than the M/G/1 PS queue. Then they prove that for link utilization 0.5, the SRPT improves performance over PS with respect to mean response time for every job and for every job size distribution. For the largest jobs, the slowdown (response time divided by job size) under SRPT is only slightly worse than under PS (Bansal and Gamarnik, 2006). In (Bansal, 2003) and (Bansal and Gamarnik, 2006), interesting results on the mean response in heavy traffic were obtained that show that SRPT performs significantly better than FIFO if the system is under heavy traffic.

C. Murta and T. Corlassoli (Murta and Corlassoli, 2003) introduce and simulate an extension to SRPT scheduling called Fastest-Connection-First (FCF) that takes into consideration the wide area network (WAN) conditions in addition to request size when making scheduling decisions. This scheduling policy gives higher priority to HTTP requests for smaller files issued through faster connections. This work is done only by simulation without providing a clear idea on how to implement it in real web servers. M. Gong and C. Williamson (Gong and Williamson, 2003) identify two different types of unfairness: endogenous unfairness that is caused by an inherent property of a job, such as its size. And exogenous unfairness caused by external conditions, such as the number of other jobs in the system, their sizes, and their arrival times. They then continue to evaluate SRPT and other policies with respect to these types of unfairness. E. Friedman et al. (Friedman and Henderson, 2003) propose a new protocol called Fair-Sojourn-Protocol (FSP) for use in web servers. FSP orders the jobs according to the processor sharing (PS) policy and then gives full resources to the job with the earliest PS completion time. The FSP is a modified version of SRPT and it has been proven through analysis and simulation that FSP is always more efficient and fair than PS given any arrival sequence and distribution. Their simulation results show that FSP performs better than SRPT for large requests, while the SRPT is better than FSP for small requests.

The work that implements scheduling for web servers based on the SRPT was done on both the application level, and at the kernel level to prioritize HTTP requests. M. Crovella et al. (Crovella et al, 1999) experimented with the SRPT connection scheduling at the application level. They get an improvement in the mean response times, but at the cost of drop in the throughput by a factor of almost 2. This drop comes as a result of no adequate control over the order in which the operating system services the requests. M. Harchol-Balter et al. (Harchol-Balter et al, 2003) implemented SRPT connection scheduling at the kernel level. They get much larger performance improvements than in (Crovella et al, 1999) and the drop in the throughput was eliminated. B. Schroeder et al. (Schroeder and Harchol-Balter, 2006) show an additional benefit from performing SRPT scheduling for static content web requests. They show that SRPT scheduling can be used to alleviate the response time effects of transient overload conditions without excessively penalizing large requests. SWIFT algorithm (Rawat and Kshemkayani, 2003) extend the work in (Harchol-Balter et al, 2003) based on SRPT, but taking into account in addition to the size of the file, the RTT to represent the distance between the client and the server. With this technique they obtained a response time improvement for large-sized files by 2.5% to 10% additional to the SRPT. In the SWIFT algorithm implementation, they assumed that the HTTP requests are embedded with the RTT in their trace driven experiment. This assumption is not a realistic scenario. Moreover, the implementation of the SWIFT requires additional modifications on the web server to support functions that are parses request to extract the RTT that assumed to be part of client requests. Accordingly, we did not implement the SWIFT to compare it with SRRT. SRRT gets the RTT and congestion window size (cwnd) at the server side for each connection "on-the-fly" by using *getsockopt()* Linux system call to use it with the file size to better estimate the response time in a WAN environment.

3 SRRT ALGORITHM

The SRRT algorithm benefits from TCP implementation to address most of the client-server interaction on the Internet. Due to TCP's congestion control mechanism, TCP window sizes (cwnd) can be bound to the maximum transfer rate *R* = (cwnd/RTT) bps despite the actual bandwidth capacity of the network path. Also, the TCP congestion control mechanism involves Time-outs that cause retransmissions. RTT is monitored and Time-out is set based on RTT (Karn and Partridge, 1995; Jacobson, 1995).

After processing an HTTP request, the server code uses the *getsockopt()* to get these useful information about the network condition (cwnd, RTT) that will be used in estimating the remaining response time of the request on the server side. The requested file size is already known by the server. Hence, the remaining response time (RRT) can be approximated as follows:

$$RRT \approx RTT + \frac{RFS}{R}$$

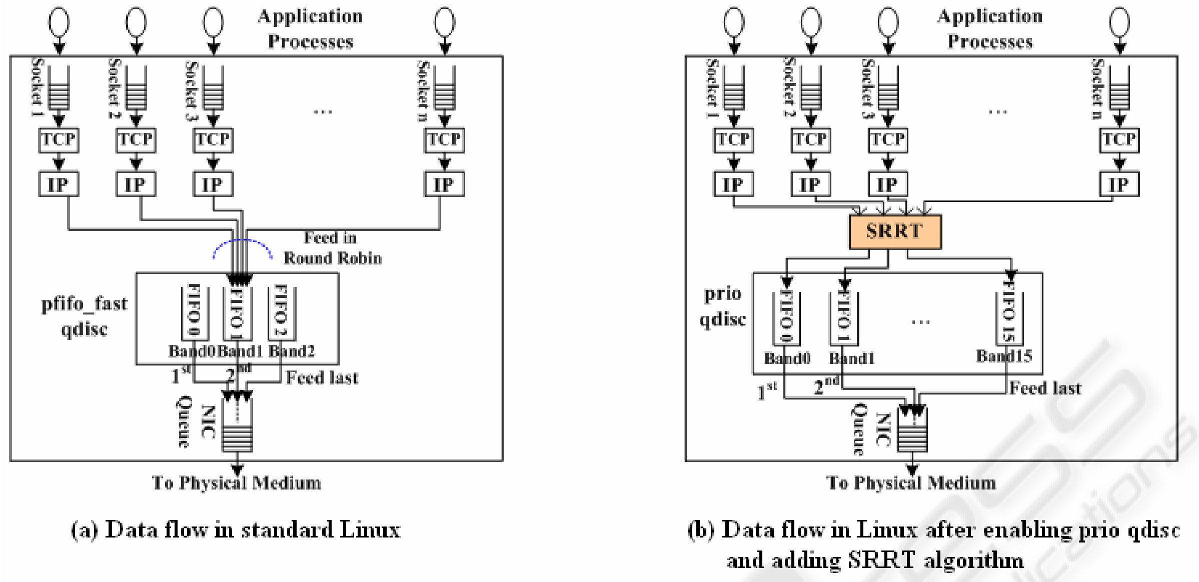
$$RRT \approx RTT\left(1 + \frac{RFS}{cwnd \times MSS}\right)$$

Where RFS is remaining length of the requested file(s) in bytes, R is the approximated TCP transfer rate, and MSS is the maximum segment size for the connection in bytes.

As seen above, the estimating of RRT depends on three variables; RFS, current RTT, and the current cwnd. Thus we consider almost all aspects that affect data transfer over the Internet since the RTT and the cwnd change dynamically according to network conditions. The estimated RRT is influenced by network conditions. The highest priority is given to the connection that has the best-estimated performance: the connection that needs to transfer small file through an un-congested path, which has short RTT and large cwnd.

4 SRRT IMPLEMENTATION

The experiments have been done using Apache web server since it is the most popular web server (Netcraft, 2007). To build SRRT based on Apache



(a) Data flow in standard Linux

(b) Data flow in Linux after enabling prio qdisc and adding SRRT algorithm

Figure 1: Data flow in Linux operating system.

running on Linux, basically two things are needed. First, to set up several priority queues at the Ethernet interface. Second, to modify the Apache source code to assign priorities to the corresponding requests.

The data being passed from user space is stored in socket buffers corresponding to each connection. When data streaming passes from the socket buffers to TCP layer and IP layer, the TCP headers and the IP headers are added to form packets. The packet flow corresponding to each socket is kept separate from other flows (Harchol-Balter et al, 2003). After that, packets are sent from IP layer to queuing discipline (qdisc). The default qdisc under Linux is the pfifo_fast qdisc. Figure 1(a) shows the default data flow in standard Linux. pfifo_fast qdisc is a classless queuing discipline, so it cannot be configured. The packet priorities are determined by the kernel according to the so-called Type-Of-Service (TOS) flag and priority map (priomap) of packets. However, all packets using the default TOS value are queued to the same band (band 1 in the Figure 1(a)). So the three bands appear as a single FIFO queue in which all streams feed in a round-robin service: all requests from processes or threads are given an equal share of CPU time and share the same amount of link capacity, Processor Sharing (PS). Packets leaving this queue drain in a network device (NIC) queue and then out to the physical medium (network link).

To implement SRRT, we need several configurable priority queues. This can be achieved by Priority (prio) qdisc with 16 priority queues which can be configured. The prio qdisc works on a very simple principle. When it is ready to dequeue a packet, the first band (queue) is checked for a packet. If there is one, it gets dequeued. If there is no packet, then the next band is checked, until the queuing mechanism has no more classes to check. Figure 1(b) shows the prio queuing discipline to implement SRRT.

In the SRRT implementation, the Apache code is responsible for assigning the priorities to the corresponding connection by using s*etsockopt ()* to determine in which band a packet will be enqueued. Therefore, we made changes to the Apache HTTP Server code to prioritize connections. The installation of the SRRT-modified Apache server is the same as the installation of standard Apache. The only thing that might need to change when experimenting with SRRT server is the priority array values, in the form of response time ranges, to determine the priority class of the socket according to the type of the load.

TCP SYN-ACKs gets by default into the highest priority band (band 0). Here, we will take into consideration the recommendation given by (Harchol-Balter et al, 2003). Because the start up of the connection is an essential part of the total response delay, especially for short requests before the size of the file is known, no sockets are assigned to priority band 0, but are assigned to other bands of lower priority, to prevent packets sent during the connection start up waiting in a long queue. The

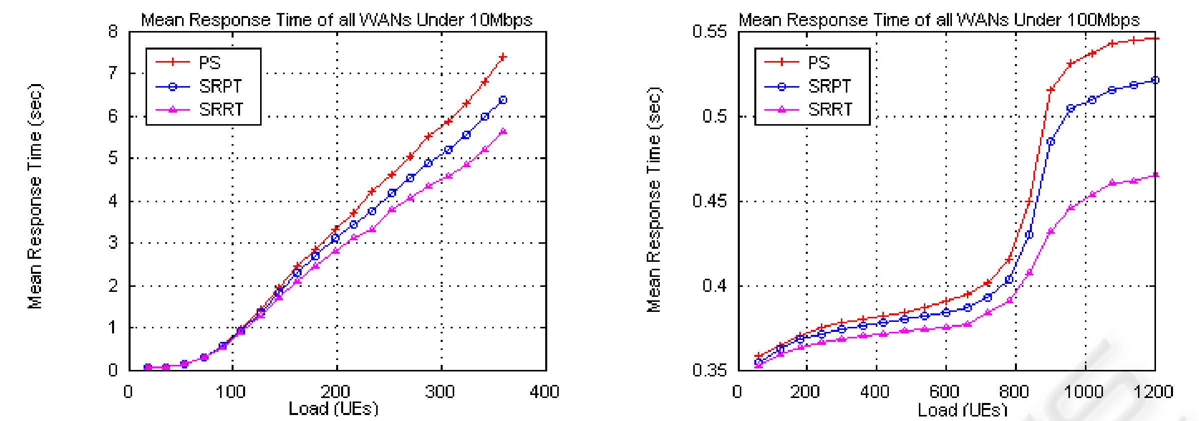


Figure 2: Mean response time of all WANs under 10Mbps and 100Mbps.

SYN-ACKs constitute a negligible fraction of the total load. Thus assigning them to higher priority does not affect the performance.

5 SETUP AND RESULTS

5.1 Experiment Setup

The experimental setup consists of seven machines connected by 10Mbps hub in the first experiment and by 100Mbps Fast Ethernet connection switch in the second experiment. Each machine has an Intel Pentium 4 CPU 3.20 GHz, 504 MB of RAM. We used the Linux 2.6.18. One of the machines (the server) runs Apache 2.2.3. The other machines act as web clients. The client machines generate loads using the Scalable URL Request GEnerator (SURGE) (Barford and Crovella, 1998). On each client machine, Network Emulator (netem) (Linux Foundation, 2007) is used to emulate the properties of a Wide Area Network (WAN).

Request sizes in the WWW are known to follow a heavy-tailed distribution (Harchol-Balter et al, 2003; Crovella et al, 1998). We chose SURGE to generate the HTTP 1.1 requests to the server such that they follow the heavy-tailed request size distribution. More than 300,000 requests were generated in each experiment run. We used 2000 different file sizes at the server by running *files* program from SURGE package at the server machine. Most files have a size less than 10KBytes. The requested file sizes ranged from 77B to 3MB. We represent the system load by the number of concurrent users, defined as the number of user's equivalents (UEs) generated by the SURGE workload generator. For each run we measure the mean response time at the client side by using the *pbvalclient* program from the SURGE package.

In our experiments, we assume that clients experience heterogeneous WANs. We have divided our experimental space into six WANs; where each of the six client machines represents a different WAN that shares common WAN parameters by setting the netem parameters. The WAN factors on each client machine are shown in Table 1. We experiment with delays between 50ms and 350ms and loss rates from 0.5% to 3.0%. This range of values was chosen to cover values reported in the Internet Traffic Report (Network Services & Consulting Corporation, 2007).

Table 1: WANs parameters.

WANs	RTT(ms)	Loss (%)
WAN1	50±10	0.5
WAN2	100±20	1.0
WAN3	150±30	1.5
WAN4	200±40	2.0
WAN5	250±40	2.5
WAN6	350±50	3.0

5.2 Results

We compare SRRT with the existing algorithms, namely PS and SRPT. We analyze our observations from the client's point of view in terms of mean response time under the 10Mbps and 100Mbps link capacity. The graphs in Figure 2 show the mean response time for all WANs as a function of server load (number of UEs) for the 10Mbps and 100Mbps link capacities. SRRT an SRPT show an improvement in the mean response time over PS.

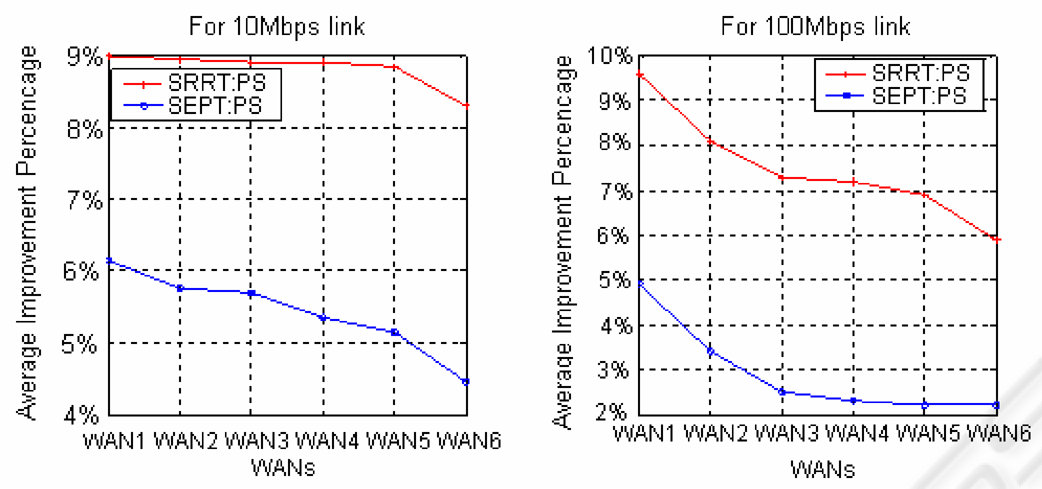


Figure 3: Average improvement of SRRT and SRPT over PS for 10/100Mbps links.

Also, the SRRT shows an improvement over SRPT. Table 2 shows the improvement percentage of SRRT over SRPT and PS, in addition to the percentage improvement of SRPT over PS for the two different link capacities; 10Mbps and 100Mbps.

Table 2: Percentage improvement of SRRT and SRPT.

Link	Algorithms	Improvement	
		Average	Max.
10Mbps	SRRT:SRPT	7.5%	13.2%
	SRRT:PS	13.6%	24.2%
	SRPT: PS	6.8%	13.7%
100Mbps	SRRT:SRPT	7.4%	11.6%
	SRRT:PS	7.1%	16.2%
	SRPT: PS	2.6%	5.8%

SRRT and SRPT show an improvement in the mean response time over PS. This comes from the fact that the bandwidth is shared for all requests under PS. Therefore, all incomplete requests still take fair share of the bandwidth from other requests. Hence, the mean response time of short requests increases. While under the SRRT and SRPT, long requests do not receive any bandwidth and short requests are completely isolated from the long requests. Therefore, completing short requests first and then long requests do not increase the mean response time by giving the chance to the small requests to complete first without competition from long requests. As a result, the PS shows a faster increase in mean response time than under SRRT and SRPT.

SRRT has the best results especially at high loads. This is likely because our approach better estimates the response time by taking into consideration the client-server interaction over the WAN environment. For low loads, the three algorithms show almost similar mean response time. Since for low load the available link capacity is large enough to serve all requests, which in turn results in keeping the number of packets in the transmission queue small so that the effect of scheduling is not noticeable. However, in the low load case the RTT dominates the total communication delay so SRRT shows better behaviour over SRPT in this region since SRRT takes into account RTT in estimating response time. For high load but before the link saturates, the improvement of SRRT over SRPT starts to become noticeable. For high load, the SRRT shows a great improvement over the SRPT for all WANs.

The over all requests average percentage improvement of SRRT and SRPT over PS for 10/100Mbps for all WANs is shown in Figure 3. The network WAN1 has the best network conditions (delay and loss) compared to other WANs, so the requests get higher priorities under SRRT and therefore minimize the mean response time. So WAN1 has the best average improvement percentage in SRRT over PS compared to the other WANs. Also, we can see that bad network conditions decrease the improvement of both SRRT and SRPT scheduling techniques over PS. However, SRPT is more affected by bad network conditions than SRRT since it uses only the file size to approximate the expected response time. Server delay dominates the response time for the case of a network with no loss, and in which we ignore RTT. In contrast, under bad condition WANs (large RTT and high loss rate) the transmission and retransmission delays are the dominant parts of the communication delay rather than the delay at the server. The mean response time increases as the

RTT and the loss rate increase. Higher RTTs make loss recovery more expensive since the retransmission time-outs (RTO) depend on the estimated RTT. Hence, lost packets cause very long delays based on the RTT and RTO values in TCP. SRRT takes these into consideration indirectly, TCP throughput for a connection being inversely proportional to the square root of the loss (Padhye et al, 2000), by decreasing the cwnd. When losses increase the cwnd decreases. Accordingly, the estimated response time in SRRT increases, so the corresponding connection receives less priority. Therefore, SRRT improvement is slightly decreased by the poor network conditions. As mentioned in (Harchol-Balter et al, 2003), "While propagation delay and loss diminish the improvement of SRPT over PS, loss has a much greater effect". SRRT considers the user's network conditions by benefiting from the TCP interaction between the server and the network to take into consideration the realistic WAN factors that can dominate the mean response time.

The SRRT/SRPT add an additional overhead compared to PS since they need to assign priorities to the request by invoking *setsockopt()* system call. In addition to *setsockopt()* call, SRRT uses *getsockopt()* system call to get the RTT and the cwnd. However, the additional overhead is not critical under the assumption that the CPU is not the bottleneck. We found about 1% increase in the CPU utilization under SRRT over the PS.

5.3 Starvation Analysis

To see if the improvement in mean response time comes at the expense of starvation for long requests, we look to the response time for each individual request under SRPT and SRRT scheduling algorithms. To quantify the starvation, we use the starvation stretch metric, which is introduced in (Jechlitschek and Gorinsky, 2007). Starvation stretch $S_x(r)$ of request r under algorithm X is the ratio of response time $RT_x(r)$ under X to response time $RT_{ps}(r)$ under PS ($S_x(r)=RT_x(r)/RT_{ps}(r)$). The starvation occurs under the algorithm X if $S_x(r)>1$.

Under SRPT, we found that 2.3% of the requests have starvation stretch greater than 1 under the 100Mbps link capacity, and the largest file (3119822B) has a starvation stretch of 2.1. Under the 10Mps capacity, 2.6% of the requests starved. The largest file has a starvation stretch of 2.4. The SRRT shows better performance than SRPT since it has more information about the response time. For SRRT only 1.5% of the long requests starved under the 100Mbps link. The longest response has a starvation stretch 1.7. Under the 10Mbps, 2.4% of the requests starved. The longest response has a starvation stretch 2.2.

6 CONCLUSIONS

The performance of SRPT degrades dramatically in the Internet environment which has high diversity in bandwidth, propagation delay and packet loss rate. Thus, we proposed SRRT to better estimate the response time by getting useful TCP information, which is available at web server about the connection, in addition to the file size, without producing additional traffic. The SRRT uses the RTT, the congestion window size, and the file size to approximate the response time. The request with shortest SRRT receives the highest priority.

We proposed, implemented and evaluated the SRRT scheduling policy for web servers. The SRRT improves the client-perceived response time in comparison to the default Linux scheduling (PS) and the SRPT scheduling policies. The SRRT performs better than SRPT and PS at high and moderate uplink load and especially under overload condition. The performance improvement is achieved under different uplink capacities, for a variable range of network parameters (RTTs and loss rate). This improvement does not unduly penalize the long requests and without loss in byte throughput. The implementation of SRRT was done on an Apache web server running Linux to prioritize the order in which the socket buffers are drained within the kernel. The priority of the requests is determined based on the priority array values we have coded in the Apache source code. The choice of these values is based on the experiment trials. But we do not claim that this choice is optimal. Also, it is better to make these values configurable by the Apache configuration file to be able to change them as needed or even learn them during experimentation.

Another improvement on SRRT may be done by trying to take other factors that may affect the response time like queue delay approximation and the TCP connection loss rate. To check the validity of this algorithm, it is better to test it on a real web server. Also, it is good to evaluate the SRRT algorithm analytically to examine the validity of the experimental results if possible.

The SRRT is applied to static web requests. Future work can be enhancing it to also schedule dynamic requests where the approximation of the response time is not as easy as for static requests.

Also, this work may extend to other operating systems and other web servers. Also, SRRT algorithm may combine with other quality of service. For example, if connectivity quality is bad for one client, the server selects a lower quality image to send to the client to improve the response time.

We believe that SRRT scheduling will continue to be applicable in the future, although better link speeds become available and the bandwidth cost decreases. Due to financial constrains, many users will not upgrade their connectivity conditions. Also, the variance in network distance and environment will persist and diversity in delay will be continued to exist.

REFERENCES

King, A., 2003. *Speed up your site: web site optimization,* New Riders. Indiana, 1st edition.

Nielsen, J., 1997. The need for speed. <http://www.useit.com/alertbox/9703a.html>.

Kegel, D., 2006. The C10K problem. <http://www.kegel.com/c10k.html>.

Crovella, M., Frangioso, R., 1999. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems.*

Rawat, M., Kshemkayani, A., 2003. SWIFT: Scheduling in web servers for fast response time. In *Second IEEE International Symposium on Network Computing and Applications.*

Jechlitschek, C., Gorinsky, S., 2007. Fair Efficiency, or Low Average Delay without Starvation. In *Computer Communications and Networks, 2007.*

Bansal, N., Harchol-Balter, M., 2001. Analysis of SRPT scheduling: investigating unfairness. In *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 279-290.

Manley, S., Seltzer, M., 1997. Web facts and fantasy. In *Proceedings of the 1997 USITS-97.*

Murta, C., Corlassoli, T., 2003. Fastest connection first: A new scheduling policy for web servers. I*n The 18th International Teletra#c Congress (ITC-18).*

Schrage, L., Miller, L., 1966. The queue M/G/1 with the shortest remaining processing time discipline. In *Operations Research*, vol. 14, no. 4, pp. 670-684.

Schrage, L., 1968. A proof of the optimality of the shortest remaining processing time discipline. In *Operations Research*, vol. 16, no. 3, pp. 678-690.

Smith, D., 1976. A new proof of the optimality of the shortest remaining processing time discipline. In *Operations Research*, vol. 26, no. 1, pp. 197-199.

Goerg, C., 1986. Evaluation of the optimal SRPT strategy with overhead. In *IEEE Transactions on Communications*, vol. 34, pp. 338-344.

Harchol-Balter, M., Schroeder, B., Agrawal, M., Bansal, N., 2003. Size-based scheduling to improve web performance. In *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 207-233.

Schroeder, B., Harchol-Balter, M., 2006. Web servers under overload: How schedule can help. In *ACM TOIT*, vol. 6, no. 1, pp. 20-52.

Schroeder, B., Harchol-Balter, M., 2006. Web servers under overload: How schedule can help. In *ACM TOIT*, vol. 6, no. 1, pp. 20-52.

Gong, M., Williamson, C., 2003. Quantifying the properties of SRPT scheduling. In *MASCOTS*, pp. 126-135.

Friedman, E., Henderson, S., 2003. Fairness and efficiency in web server protocols. In *ACM SIGMETRICS*, pp. 229-237.

Lu, D., Sheng, H., 2005. Effects and implications of file size/service time correlation on web server scheduling policies. In *MASCOTS*, pp. 258-267.

Bansal, N., 2003. On the average sojourn time under M/M/1 SRPT. In *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 2, pp. 34-35.

Bansal, N., Gamarnik, D., 2006. Handling load with less stress. In *Queueing Systems*, vol. 54 , no. 1, pp. 45-54.

Barford, P., Crovella, M., 1998. Generating representative web workloads for network and server performance evaluation. In *ACM joint international conference on Measurement and modeling of computer systems*, pp. 151-160.

Linux Foundation, 2007. Network Emulation (Netem). <http://www.linux-foundation.org/en/Net:Netem>

Karn, P., Partridge, C., 1995. Improving round-trip time estimates in reliable transport protocols. In *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, pp. 66-74.

Jacobson, V., 1995. Congestion avoidance and Control. In *ACM SIGCOMM Computer Communication Review*, vol. 25, no.1, pp. 157-187.

Network Services & Consulting Corporation, 2007. Internet Traffic Report. <http://www.internettrafficreport.com>

Padhye, J., Firoiu, V., Towsley, D., Kurose, J., 2000. Modeling tcp reno performance: A simple model and its empirical validation. In *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 2, pp133-145.

Crovella, M., Taqqu, M., Bestavros, A., 1998. Heavy-tailed probability distributions in the World Wide Web. *In A Practical Guide To Heavy Tails*, pages 3–26. Chapman & Hall, New York.

IRCache Home, 2004. The trace files. <ttp://www.ircache.net/Traces>

Netcraft, 2007. Internet monitoring company. <http://news.netcraft.com/>