論文
6-2-10

A Study on Discrete-Event Modeling of a Heterogeneous Web Server System

남의석(Eui-seok Nahm)[1] 강이구(E. G. Kang)[2] 정헌석(H. S. Chung) [3]
이준환(J. H. Lee)[4] 현득창(D. C. Hyun)[5]

Abstract

A heterogeneous webserver such as an HTTP server should be able to currently deal with numerous users. To the end, it is inevitable to formally analyze web traffics as well as a webserver itself. In particular, as most systems adopt HTTP 1.1 protocol instead of HTTP 1.0 protocol, it is more difficult to represent the system as a simple analytic mode. In addition, since most of previous models missed the detailed processes of the server, it is unsuitable for the current server based on HTTP 1.1 to tune itself with its own system parameters. On the basis of HTTP 1.1 protocol supporting persistent connections, we thus present an analytical end-to-end tandem queueing model considering specific hardware configurations inside the webserver, which ultimately covers from accepting the customer requests to completing the services.

Keywords :Web server, HTTP protocol, HTTP traffic, Queueing model

논무접수 : 2005. 6. 1.
심사완료 : 2005. 6. 22.

1) 정회원 : 극동대학교 정보통신공학부
2) 정회원 : 극동대학교 정보통신공학부
3) 정회원 : 극동대학교 정보통신공학부
4) 정회원 : 극동대학교 정보통신공학부
5) 정회원 : 극동대학교 정보통신공학부

I. Introduction

Among several protocols provided by a webserver, the HTTP protocol implemented through a certain web browser is taking up most of the web traffics. To characterize HTTP traffics, numerous studies have proposed analytical web traffic models including session requests, the distributions of the requested file sizes, transfer times. However, most previous works missed the detailed processes in the lower parts of TCP/IP layer, such as server hardware platform, network bandwidth, management of I/O buffers, and file sizes(Z. Liu, R.D. Van der Mei). Consequently, it seems unreasonable to evaluate the performance of the webserver just with such few parameters as service requests, total service time, and the number of the processors. There was an analytical model for the webserver based on HTTP 1.0 in order to evaluate the generic performance of a webserver(P. Barford, 1999). However, since it does not consider persistent connections, the model is no longer suitable for current webservers operating on the HTTP 1.1 protocol.

The presented model utilizes the form of the tandem queueing model that handles web requests through three consecutive subsystems: TCP subsystem, HTTP subsystem, and network subsystem. Our model also takes account of both the specific hardware and software configurations, while dealing with the number of the threads, TCP listen queue sizes, memory sizes, I/O buffer sizes, and the network bandwidth.

In Section 2, both HTTP traffic analysis and workload characterizations are described. In Section 3, a tandem queueing model with three subsystems for the webserver is proposed together with performance evaluation with regard to the parameters associated with each component. In Section 4, we validate the suggested model through a certain bench marking test with lab environments and predict the system performance while varying the parameters in terms of discrete event simulations. The paper concludes with Section 5.

II. Web Traffic Characterization

As depicted in Fig. 1, a user session begins with making a request for some page in the web server, which transfers successively all of the files embedded in the requested page and ends with expired persistent-connection time. It can be said that ON time corresponds to the transfer time and OFF time corresponds to the time between transfers. OFF time is composed of active OFF times and inactive OFF times(Z. Liu, N, 2000).

<Fig. 1> User session characterization

Inactive OFF time corresponds to the idle time that users spend on being idle while watching the browser, and active OFF time corresponds to the time that the web browser spends on parsing the web requests, preparing for establishing a new TCP

connection. HTTP 1.1 does not only maintain persistent connections until inactive OFF time does not exceed the predetermined timeout, but also support the pipeline that the web server transfers successive requests without waiting for an ACK message for the transferred file. Through the analysis of those OFF times, we built a web server model having several OFF times such as the thinking time for users to think while watching the browser and the processing time to prepare for a new TCP connection and to parse the web requests between transfers of files by the browser.

1. File size characterization

The service time of the web server is dominated by the transfer time of the files, is proportional to the file size. The distribution of the file sizes in the web server usually has a heavy-tailed distribution. This is commonly observed from the file sizes requested by customers, the network connection sizes, and the stored file sizes in the web server. If a random variable X has a heavy-tailed distribution, its cumulative distribution function is defined as $P[X > x] = x^{-\alpha}$ where $0 < \alpha < 2$. Usually, the body part of the file size distribution has lognormal distribution, and the tail part has a Pareto distribution whose probability density function is $\alpha k^{\alpha} x^{-(\alpha+1)}$ with a parameter of $0.40 < \alpha < 0.63$(Z. Liu, N, 2000). A Pareto model is suitable for the file size between 4KB and 4MB, and has a long-tail distribution(V. Paxson and S. Floyd, 1995). Since the random variable that has a heavy-tailed distribution shows high variation for normal file size, a few of big sized files take up the most of the loads in the web server. On the other hand, the most of requested file are occupied with small sized files

2. OFF time characterization

Supposing that the probability density function to describe file size is F(x), the expiration time of the persistent connection is T_{out} , and the random variable to describe the active OFF time is Y, the probability P to close a session resulting from expiration of the timeout value can be described as $P[Y > T_{out}]$, and the mean of active OFF time within persistent connection can be expressed as follows:

$$E[Y | Y < T_{out}] = \frac{1}{1-p} \int_0^{T_{out}} F(x)dx \quad (1)$$

A previous study shows that the active OFF time has a Weibull distribution like (2), and the inactive OFF time has a Pareto distribution [1,2].

$$P(y) = \frac{bx^{b-1}}{a^b} e^{-(\frac{x}{a})^b} \quad (2)$$

Through the analysis of those OFF times, we can apply it to web server model several OFF times, such as thinking time for users to think while watching the browser and the processing time to prepare for new TCP connection and parse the web requests between transfers of files by browser.

3. Popularity

The relative number of the requests for the individual file on the server has a potent influence on the role of a cache. The distribution of the request number for the files follows Zipf's Law(Z. Liu, N, 2000). It

means that if the files are ordered from most popular to least popular, the reference number P for a file is in inverse proportion to its rank r. As it were, $p = kr^{-1}$, where k is an positive number. This property can be observed well in spite of the unclear reason for this phenomenon in the web workload. Such a distribution of the references between web files shows that some files are greatly popular, while most of files have a relatively rare reference.

4. Temporal locality

Once a file is requested, the file inclines to be requested again by other users. The characterization of the temporal locality is required, because the cache is more effective when temporal locality exists. To characterize the temporal locality, the measurement of the stack distance is generally used. Assuming that files are stored in a push-down stack, the stack distance of the requested file is found when each request moves the requested file to the top of the stack and pushes down the other file. So to speak, short stack distance means that the requests for the same file are closely occurred each other. And typical distribution for request sequence in the web server is known as the lognormal distribution, which means significant temporal locality is often in existence in the sequence of the request for the web server.

5. HTTP protocol

HTTP protocol occupying most of web traffics makes a connection through 3-way handshaking, which consists of sending a SYN message to the server by client computer, replying back a SYN-ACK message from the server, and sending the ACK message from the client. Thereafter, it creates a proper data structure corresponding to service requests by allocating required system resources with a new port number. Since HTTP 1.0 releases a connection right after transferring a requested file, substantial overheads result from the same procedures that occurs in the connection establishment. Note

<Fig. 2> A tandem queueing model on HTTP 1.0

that a web file usually refers to image files, and thus a request for the single web file renders transferring multiple files from the web server. We call a web file including all these files to be transferred 'web object' accordingly.

Contrary to HTTP 1.0 transferring only one file at a connection, HTTP 1.1 diminishes the overhead from connection establishment while maintaining a persistent connection even after transferring the requested file. Additionally, by using the pipeline that sends the requests on the same connection without waiting for the completion of the previous transfer, the HTTP 1.1 not only eliminates wastes of unnecessary resources but also saves transferring time. It also supports caching scheme as well as document compression

over link level.

With respect to the service time for the web service requests, HTTP 1.1 is not always superior to HTTP 1.0, for example in overloaded states. In general, HTTP 1.1 shows reduced overheads by making new connections. In other words, the persistent connections of HTTP 1.1 need maintaining fewer connections than those of HTTP 1.0. As the concurrent connections approach to the maximum limit connections of HTTP 1.1, a new request becomes refused. Therefore, HTTP 1.1 yields performance deterioration in the response time under the overloaded conditions. In this aspect of HTML latency under overload condition, since HTTP 1.0 has lower latency through parallel multiple connections in spite of both three-way handshaking and TCP slow start phases for each file, it excels HTTP 1.1 that inquires all ACK messages for all embedded objects in a page on a single connection. When the performance of the web server is restricted to file transfer under the overload conditions, the persistent connections of HTTP 1.1 may result in degrading Quality of Service rather than maintaining a connection only during transferring a single object like HTTP 1.0. However, the normal condition of workloads brings HTTP 1.1 to support the persistent connections to have significantly better performance

III. Web Server Model

1. Tandem queueing model

Since most web server models dealt just with both of the request for the file and the file transfer ignoring essential lower-level details of HTTP and TCP/IP protocols, it is insufficient to show the end-to-end performance for the communications between the client and the server. On the contrary, the author of (Z. Liu, N, 2000) evaluated the performance of the web server by dividing a web service into TCP connection processing phase, HTTP transaction processing phase, and I/O processing phase as shown in Fig. 2. The author of (R.D. Van der Mei) expanded this result to the web service model to handle both static requests and dynamic requests on the basis of HTTP 1.0. However, since the work of was based on the HTTP 1.0 that one web request go through three tandem queues and then terminate the connection with resulting in repeatedly making another request, it was not appropriate for HTTP 1.1 models, guaranteeing persistent connections. In addition, supposing all transaction requests entering each subsystem come from the previous subsystem, both works did not consider that a blocked transaction returning to the previous queue(Z. Liu, R.D. Van der Mei).

The HTTP server model presented in this paper is described as a tandem queueing model going through three subsystems. On the other hand, we consider the situation of transaction reattempts being serviced in case of being blocked in the subsystem. And we observe the system performance according to the probability that the session is terminated by a persistent connection supported in HTTP 1.1. The entire system model can be analytically described as Jackson network with such three subsystems as TCP subsystem, HTTP subsystem, NETWORK subsystem, which are shown in Fig. 3.

<Fig. 3> A tandem queueing model on HTTP 1.1

Note that the net arrival process is composed of isolated external arrivals followed by a burst of feedback arrivals blocked due to being full of listen queues or being busy requested resources. We can use Jackson's theorem that the number of transactions in the queues at time t is independent random variable. In addition, if the vector of the number of transactions in all the queues, N(t)= (N1(t),N2(t),N3(t)), is a Markov process, Jackson's theorem states that the product of the steady state probabilities of the individual queues is equal to the steady state pmf for N(t). Therefore, it can be said for any possible state n=(n1,n2,n3) as follows.

$$P[N(t)=n]=P[N_1=n_1]P[N_2=n_2]P[N_3=n_3] \quad (3)$$

where N1 is the random variable for TCP queue, N2 is for HTTP queue, and N3 is for NETWORK queue.

Therefore, consider a network of three queues in which transaction arrive from outside the network to TCP queue according to independent Poisson processes of rate α. We assume that the service time of a transaction in queue k is exponentially distributed with the rate μ_k and independent of all other service times and arrival processes. We also suppose that P_{B1} and P_{B2} be the blocking probabilities of TCP subsystem and HTTP subsystem, P_{time} be the probability of session termination as persistent connection time expires, and π_r be the probability to retry without leaving the system. Corresponding equations on each subsystem can be obtained as follows.

$$\lambda_1=(\pi_r P_{B1})\lambda_1+(\pi_r P_{B2})\lambda_3+\alpha \quad (4)$$

$$\lambda_2=(1-P_{B1})\lambda_1+(1-P_{time})\lambda_3 \quad (5)$$

$$\lambda_3=(1-P_{B2})\lambda_2 \quad (6)$$

2. TCP subsystem

The TCP subsystem corresponds to TCP listen queues handled by the server daemon. A TCP connection is established by well-known three-way handshake procedures, if there is a slot available at the TCP listen queue. Otherwise, the server sends connection refusal message and drops the request. After establishing the TCP socket, the server daemon forwards the requests to the HTTP subsystem and closes the socket by sending FIN segment when the transaction is completely serviced.

In the TCP connection processing phase, the server is able to service the requests as many as the number of slots in the listen queue, and drops requests arriving after all slots are occupied due to no waiting room. Therefore, as a queueing model, m_{tcp} the number of slots corresponds to the number of servers and RTT(Round Trip Time) corresponds to the service time because the service time corresponds to the time from

sending SYN-ACK message by the server till receiving ACK message from the client. Hence, this subsystem can be described as a $M/M/m_{tcp}/m_{tcp}$ queueing model, if RTT has an exponential distribution with mean of μ_1 and the size of queue is m_{tcp}, and λ_1 the rate of the request arriving in this subsystem can be derived from solving the simultaneous equations (4)-(6). μ_1 the service rate, P_{B1} the blocking probability of dropped additional requests after all slots are occupied, and $E[N_{tcp}]$ the mean transaction requests in the TCP subsystem can be obtained as follows.

$$\lambda_1 = \frac{a}{(1-P_{B1})(1-\pi_r)}, \quad \mu_1 = \frac{1}{RTT} \quad (7)$$

$$P_{B1} = \frac{a^{m_{tcp}}}{m_{tcp}! \sum_{k=0}^{m_{tcp}} \frac{a^k}{k!}}, \quad a = \frac{\lambda_1}{\mu_1} \quad (8)$$

$$E[N_{tcp}] = \frac{\lambda_1}{\mu_1} = \frac{a}{\mu_1(1-\pi_r)(1-P_{B1})} \quad (9)$$

3. HTTP subsystem

The HTTP subsystem simply consists of an HTTP listen queue and several HTTP threads. A HTTP daemon listens to the arrival of a transaction request and allocates a HTTP thread to handle the arrived request. The thread fetches the file requested by the transaction from a file system or a cache and copies it into the memory buffer. This thread is then released to serve another transaction. If all of the memory buffers are busy, the HTTP thread remains idle until the memory buffer becomes available. In case several threads wait for an available buffer, the waiting threads catch the buffer by a certain allocation rule with the proper priority. At this time, the utility of the memory buffer depends on whether or not the data in the memory buffer can be put into a network I/O buffer of the network subsystem. The data in the network I/O buffer should be held until receiving an ACK message from the client. Depending on the network congestion, it is natural that the availability of the memory buffer is regarded as a random variable.

In case the HTTP daemon cannot find a thread available to handle the transaction request, it should wait in the so-called HTTP listen queue until any thread becomes available. If all HTTP listen queues have no more room to hold arriving transaction requests, the TCP connections replying to those transaction requests are cancelled, which makes clients leave the system or enter the TCP listen queue repeatedly with delivering the refusal message to the clients.

Denoting the number of HTTP threads by m_{http} and the size of HTTP listen queue by Q, this HTTP subsystem can be described as $M/M/m_{http}/(m_{http}+Q)$ system. The service time in this subsystem corresponds to the sum of fetching time for requested file, copying time into the memory buffer and the time which it takes for thread to wait until the memory buffer is available if all the memory buffers are occupied.

λ_2 the request rate in the HTTP subsystem can be obtained by getting the solution from (4)-(6). And, μ_2 the service rate, P_{B2} the probability to be blocked from HTTP listen queue, $E[N_{Q\_http}]$ the average number of transactions to wait for an HTTP thread available, and the whole system time

in this HTTP subsystem are as follows;

$$\lambda_2 = \frac{a}{(1-\pi_r)\{1-(1-P_{time})(1-P_{B2})\}} \quad \text{-- (10)}$$

$$\mu_2 = \frac{1}{T_{fetch} + T_{wait\ I/O} + T_{write\ I/O}} \quad \text{------- (11)}$$

$$P_{B2} = \frac{a^{m_{http}}}{m_{http}!}\left(\sum_{k=0}^{m_{http}-1}\frac{a^k}{k!} + \frac{a^{m_{http}}(1-\rho^{Q+1})}{m_{http}!(1-\rho)}\right)^{-1} \quad \text{(12)}$$

$$E[N_{Q\_http}] = \sum_{k=m_{http}}^{m_{http}+Q}(k-m_{http})\rho^{k-m_{http}}P_{B2} \quad \text{---- (13)}$$

$$T_{http} = \frac{E[N_{Q\_http}]}{\lambda_2} + \frac{1}{\mu_2} \quad \text{---------- (14)}$$

where $a = \frac{\lambda_2}{\mu_2}$ $\rho = \frac{\lambda_2}{m_{http}\mu_2}$

4. Network Subsystem

In the Network subsystem, files in the memory buffers are splitted into MSS(Maximum Segment Size) blocks which are the transmission units for TCP/IP based network. A file stored in the memory buffer is splitted into MSS blocks and put into network output buffer to transmit by network controller. When each block is sent to the client through the network, the client reads the block through network card and sends ACK message. Supposing that F is the size of the file, and B_{net} is the size of the output buffer in the network subsystem, k the number of MSS blocks splitted from a file can be given by

$$k = \left\lceil \frac{F}{B_{net}} \right\rceil \quad \text{(15)}$$

As the data on the output buffer should be held on the output buffer before getting ACK message from client owing to connection oriented property of TCP, we can see that $T_{wait\ I/O}$ the waiting time to use available memory buffer in the HTTP subsystem relies on the network congestion condition.

Thus, overall system time in the Network subsystem corresponds to the sum of $T_{partition}$ the time to split the file on the memory into MSS blocks, T_{buf} the time to put those blocks into output buffer, $T_{transfer}$ the transfer time from the server to the client, T_{client} the time to read the block through network card, and T_{ACK} the time to take to receive ACK message from the client.

Assuming that the file is splitted into k blocks, since all the time except $T_{partition}$ need the more time to be multiplied by k, λ_3 request rate in Network subsystem and μ_3 the service rate can be obtained as follows.

(16)

(17)

When the network controller sends the data on the memory buffer to the output buffer, controller makes it possible for infinite corresponding transactions to wait until the output buffer is available in case where no output buffer is available. The network controller can send to network only one data out of m_{net} output buffers at a time, and hold the output buffers as many as m_{net}. Therefore, the network subsystem can be described as $M/M/1/m_{net}$. In steady state, denoting the $\frac{\lambda}{\mu}$ by ρ, the probability with which k transactions are in the network subsystem is given by

$$P[N=k] = \frac{(1-\rho)\rho^{k}}{1-\rho^{m_{net}+1}} \quad (18)$$

Considering transactions waiting for an output buffer available when the transactions arrive to the network subsystem going through HTTP subsystem, $E[N_{net}]$ the number of transactions and T_{net} whole system time in Network subsystem are as follows.

$$E[N_{net}] = \sum_{k=0}^{m_{net}} kP[N(t)=k] \quad (19)$$

$$T_{net} = \frac{E[N_{net}]}{\lambda_3} = \frac{\rho}{(1-\rho)\lambda_3} - \frac{(m_{net}+1)\rho^{m_{net}+1}}{(1-\rho^{m_{net}+1})\lambda_3}$$

----------(20)

IV. Simulation for Result Validation

For the purpose of accessing the validity of the model presented in this paper, we compare the both web server performances evaluated by the discrete event simulation and the WEB BENCH 4.1(http://www.etestinglabs.com.), benchmarking in the LAN environment of test lab. All the parameters used in the simulation are configurated similar to what are based on the values used by the real web server

First, as shown in Fig.4, the test environment is composed of one controller responsible for load generating and monitoring, one web server, and 8 client-computers connected via Fast Ethernet(100Mbps).

<Fig. 4> Test environment configuration

The controller computer generates the load by making client-computers request the files stored in the web server and measures the output bandwidth of the server as the request rates vary by adjusting the number of the client-computers. The web server has Pentium-Ⅱ300MHz processor, 128M RAM, 10Mbps Lan card, and operating system of Linux kernel 2.4.2 and operates the web server application of Apache with version 1.3.19.

<Table 58> Parameters of the web server model

Parameter	m_{tcp}	m_{http}	Q	T_{fetch}	m_{net}	B_{net}
Value	64	30	10	10ms	60	1024

As shown in Table. 1, m_{tcp} is set as 64 by configuring the value of baacklog on Linux, and μ_1 is set by measuring RTT in the test lab. T_{fetch}, the time which it takes for an HTTP thread to fetch the requested file from the disk can be considered as the average of approximate 10ms, because most of disk access time is seeking time if 5400RPM hard disk processes about 100 random I/O job per second. In addition, supposing that the file size has an exponential distribution with mean of 7KB and hard disk has the transmission rate of 10MB/sec, $T_{write\ I/O}$ the time to write the data into the memory buffer for a file can be ignored since it is just 0.7μs. As we assume that the average file size on the web server is 7KByte, both T_{buf}, the time to put a file into output buffer and $T_{Partition}$, the time to partition is so small as they can be approximated as 0(Patreick Killelea, 1998).

In network subsystem, we can see that the web server transfers files to client-computer at the bandwidth of about 900Kbps by measurement. Since we assumed the file size has an exponential distribution with mean of 7 KByte, it can be said that transferring time to client has an exponential distribution with mean of 60ms. Besides, RTT, the time that it takes for client to read the received packet and send ACK, is also so small compared with transferring time as it can be neglected. Therefore, we performed the simulation assuming that total service time in Network subsystem has an exponential distribution with mean of 60ms. For the purpose of visually expressing the performance of the web server, we define the performance of the web server as total system time per transaction give by

$$T_{system} = \frac{1}{\mu_1} + T_{http} + T_{net} \quad (21)$$

where T_{http} and T_{net} can be obtained by (14) and (20)

The throughput and response time of the web server via both WEB BENCH 4.1 and simulation are shown in Fig. 5.

<Fig. 5> Validation of test result with simulation

Among several parameters, HTTP 1.1 heavily depends on the timeout probability $P_{timeout}$ resulting from that inactive OFF time exceeds persistent connection time before the session completes, the retry probability π_r to try to make connection for downloading the page when the session terminates. Thus, in order to validate the presented model, we had to apply various $P_{timeout}$ and π_r according to the system characteristic, which heavily relies on the both configured persistent connection time and content of

web server, e.g. whether it is web commercial site to, or common homepage. As a result of simulation, we could have similar simulation result to WEB BENCH on $\pi_r = 0.15$, and $P_{timeout} = 0.3$

We have changed the number of HTTP thread and the size of HTTP listen queue in order to observe the system performance variety according to system parameters. At first, the total system time variety between the number of HTTP thread and the size of HTTP listen queue is constant not related to each other as shown in Fig. 6.

<Fig. 6> HTTP thread vs HTTP buffer

And, the total system time variety between the size of TCP listen queue and the number of HTTP thread is shown as in Fig. 7. When m_{http} / m_{tcp} is small, the system does not show good performance owing to large system time. And as m_{http} / m_{tcp} increases, the performance is enhanced due to decrease of system time. But we can see almost constant system time if m_{http} / m_{tcp} exceeds 0.1.

<Fig. 7> TCP listen queue vs HTTP listen queue

Therefore, the web server is able to maintain settled performance if it has more than 10% of the size of TCP listen queue, can not improve the performance in spite of increasing the number of HTTP thread . This phenomenon results from that $T_{wait\ I/O}$ the time to get memory buffer available depends on the network congestion. The reason is that the data in the network IO buffer should be held after transfer till receiving an ACK message from client.

V. Conclusion

In this paper, we present the web service as a tandem queueing model consisting of TCP, HTTP, and Network sub systems, and analytically describe the specific processes occurred to each subsystem through queueing theory. This model is validated by both benchmarking in test LAN environment via WEB BENCH 4.1 and discrete event simulation whose parameters are based on similar environment of test lab. In addition, we could find some conditions to support

optimized web service by changing system parameters to affect the performance of each subsystem. Since this model is based on HTTP 1.1, because $P_{timeout}$ the probability to be timeout and π_r the probability to retry undergo potent influence according to configuration of persistent connection timeout in the web server application such as Apache, therefore, the performance analysis for the configuration should precede the performance evaluation of web server.

It is as follows what we should more consider to generalize and make up for the presented web server model. First, the web traffic does not have Poisson distribution(V. Paxson and S. Floyd, 1995). Second, the distribution of requested file size has not exponential distribution but heavy-tailed distribution. Third, the distribution of service time on each subsystem has not exponential distribution but general distribution. Forth, the cache has an influence on the performance of web server. According to those effects, this model has a room for expansion.

Reference

[1] Z. Liu, N. Niclause, C. Villaneva, "Traffic model and Performance Evaluation of web servers," Performance Evaluation vol. 46, pp 77-100, 2000

[2] P. Barford, M.E. Crovella, "A Performance Evaluation of Hyper Text Transfer protocols," proceedings of the ACM Sigmetrics'99, 1999

[3] R.D. Van der Mei, R. Hariharan, P.K. Reeser, "Web Server Performance Modeling,"Proceedings of 4th Informs Telecom Conference, Special Issue of Telecommunication.

[4] Alberto Leon-Garcia, "Probability and Random Processes for Electrical Engineering,"second edition, Addison Wesley. Patreick Killelea, "Web Performance Tuning ," O'Reilly, 1998 http://www.etestinglabs.com.

[5] V. Paxson and S. Floyd, "Wide-Area Traffic : The Failure of Poisson Modeling, "IEEE / ACM Transactions on Networking, vol. 3, no. 3, pp. 226 244, June 1995.