Performance modeling of web servers

Rachel Waema, MSc.Math
Faculty of Information Technology, Strathmore University,
P.O. Box 59857 00200, Nairobi – Kenya
Email:rwaema@strathmore.edu

Freddie Acosta, PhD
Faculty of Information Technology, Strathmore University
P.O. Box 59857 00200, Nairobi – Kenya
Email:facosta@strathmore.edu

Abstract
A general model of a web server system comprising of the interactions between World Wide Web users and the web sites (servers) is analyzed and evaluated. Incoming requests, once admitted for processing, compete for the available resources (HTTP threads). An efficient approximate solution is provided; its accuracy is evaluated by comparing the model estimates with those obtained from simulations. The effect of several controllable parameters on the performance of the system is examined in a series of numerical and simulation experiments. In trying to understand the interactions between web users and web servers, we attempt to answer three key questions. How can we model user and server behavior on the World Wide Web ? How do users and web servers interact? Can we improve upon the ways in which web servers process incoming requests from users?
In our study we formulate a queueing model for the web server and from the queueing model we obtain expressions for web server performance metrics such as average response time, throughput and blocking probability. This model will be used evaluate the suitability of web servers to prospective users of web server systems. The foreseen end users of the model are corporate decision makers who faced by a variety of several web server systems, are interested in evaluating the suitability of the servers in market. We envision a situation in which a given manager has a set of his/her own requirements or analysis of the business requirements and needs to purchase a web server that can meet the demands/requirements of the situation at hand. Hence with the users requirements and server specifications, the model could predict the best web server for the user requirements. We model the web server as an M/M/1/K queue with FCFS queueing discipline. The arrival process of HTTP requests is assumed to be Poissonian and the service discipline First come First served (FCFS). The distribution of service time is assumed to be exponential. The total number of requests that can be processed at one time is limited to K. We obtain closed form expressions for web server performance metrics such as average response time, throughput and blocking probability.

Key words – Web server, blocking probability, response time, queueing model

1. Introduction

1.1 Background of the study

The study is in the broad area of applied statistics. Specifically the research falls under performance modeling with special interest in building mathematical models using stochastic processes. The research focuses on studying a web server and the factors, which influence its performance. The mathematical models formulated will include Performance predictions models such as the response time, blocking probability and throughput models. Simulation will be used to study the sensitivity of the model parameters. To be able to design an efficient overload control it is important to have a good and reasonable performance model of the web server. It also has to be simple enough to be able to use in practice. Traditional modeling of telecommunication systems means modeling the systems as queuing systems from classical queuing theory. Queuing models are well suited for modeling web servers.

A web server is a program that accepts requests from a user processes the request and then sends back a reply to the user. The requests which arrive to find the server busy (unavailable) have to wait for the server to be available and hence the queueing system. In a queueing system the important performance measure is the time a user takes waiting for a reply from the server (response time). In this research, we study the factors affecting the response time and their impact on the performance of a web server.

Web server performance is a complex interplay between a variety of factors (e.g., hardware platform, web server software, server operating system, file sizes, workload characteristics, network bandwidth, etc). Experience has shown that the performance of Web servers can be impacted tremendously by the proper tuning of the server components. In order to properly configure these different components, however, it is crucial to understand how these components interact and how they impact user-perceived end-to-end performance.

The objective of studying the web server is to maximize user perceived performance, which is a function of the amount of time the user spends waiting for a file to download from a web server. In this context, download refers to the actions from the time the user requests a file from a web server to the time the file (or an error message) is delivered to the user's browser. The shorter the download time, the higher the user's perceived performance. We assume that the network connecting the client and server is a static quantity, we therefore look solely at what the server does in processing requests.

In this paper we describe a web server model that consists of a one server with one queue attached to it. Our metric of interest in this process is a quantity we call "response time". We define response time as a measure of the perceived performance which is a function of the amount of time the user spends waiting for a file to download from a web server. Requests arrive at the system according to a Poisson process. They are numbered in the order that they arrive at the system. Once a request has entered the system, it does not leave until it completes service. The service time requirement of a request is directly proportional to the size of the file requested. These file sizes are independent and identically distributed exponential random variables with identical means. The service discipline is First come First Served. The total number of requests that can be processed at one time is limited to K. A system like this is called an M/M/1/K*FCFS queue. The average service time and the maximum number of jobs are parameters that can be determined through a maximum likelihood estimation. We also derive closed form

expressions for web server performance metrics such as throughput, average response time and blocking probability.

2. Related work

Performance modeling is an important part of the research area of web servers. Without a correct model of a web server it is difficult to give an accurate prediction of performance metrics.

In the literature, many researchers have done their study on workload characterization of the traffic on the Internet and in intranet environments, based on traffic measurements. Only a few studies in the literature are focused on the modeling of Web server performance. A validated model is the basis of web server capacity planning, where models are used to predict performance in different settings, see [6] or [11]. Several attempts have been made to create performance models for web servers. [10] modeled the web server as a tandem queuing network. The model was used to predict web server performance metrics and was validated through measurements and simulations. [14] made a performance analysis of web servers using colored Petri nets. Their model is divided into three layers, where each layer models a certain aspect of the system. The model has several parameters, some of which are known. Unknown parameters are determined by simulations [5] used layered queuing models in their performance studies. [4] used a model similar to the one presented in this paper, but with assumptions of deterministic service times and session based workload. [12] also used a model similar to the one presented in this paper, but with assumptions of a general distribution for service times and processor sharing service discipline. [2] proposed a generalized processor sharing performance model for Internet access lines which include web servers. Their model describes the low-level characteristics of the traffic carried. They established simple relations between the capacity, the utilization of the access line and download times of Internet objects. [13] proposes to model a Web server as an open queueing network. However, several of the previous models are complicated. It lacks a simple model that is still valid in the overloaded work region. A simple model renders a smaller parameter space thus easier to estimate, while a complicated model usually contains parameters that are difficult to obtain.

3. Web servers

A web server is a computer /program that is responsible for accepting HTTP requests from clients, which are known as Web browsers, and serving the files which form Web pages, (which are usually HTML documents and linked objects (images, etc.)) and then forwards the requests back to the client.

A web server contains software that offers access to documents stored on the server.

Any Web server machine contains, in addition to the Web page files it can serve, an HTTP daemon, a program that is designed to wait for HTTP requests and handle them when they arrive. Clients can browse the documents in a web browser. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Java script or Perl files. The communication between clients and server is based on Hypertext Transfer Protocol (HTTP) [13]. Apache [12], which is a well-known web server and widely used, is multi-threaded. This means that a request is handled by its own thread or process throughout

the life cycle of the request. Other types of web servers e.g. event-driven ones also exist [14]. However, in this paper we consider only the Apache web server. Apache also puts a limit on the number of processes allowed at one time in the server.

Fig 3.1 web server

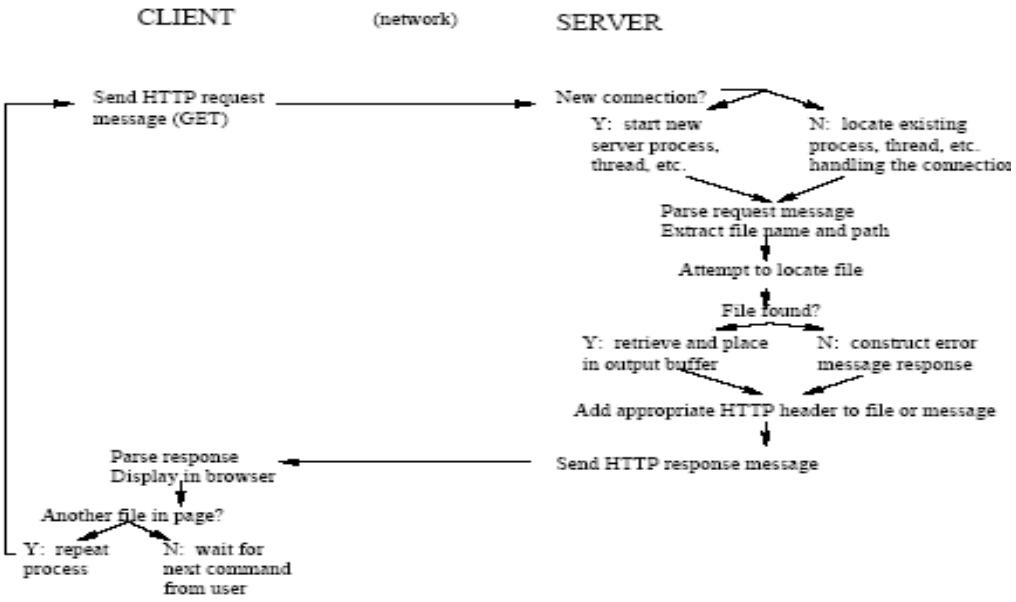
3.1 Web server protocols

In this section, we describe the main protocols involved in web transactions. TCP/IP and HTTP are the transfer and network layer protocols that control how data packets travel on the Internet backbone between hosts. HTTP is the application layer protocol that controls how web browsers and web servers communicate with each other.

3.1.1 Hypertext Transfer Protocol

Hypertext Transfer Protocol or HTTP is a request_response protocol that operates between a web browser and a web server on the application layer. The Figure below illustrates the basic format of an HTTP session. The HTTP transaction begins after the client and server establish a TCP connection, described in the next section. The client begins the session by sending an HTTP request message to the server. The most common request message type is GET, which is used to request a file from the web server. Other request types include POST, which sends data as in a form or email message to the server and OPTIONS which requests information about the web server's capabilities or the properties of a particular file or resource at the server. We assume in the rest of the description that the client has sent a GET message.

Figure 3.2 A basic HTTP session

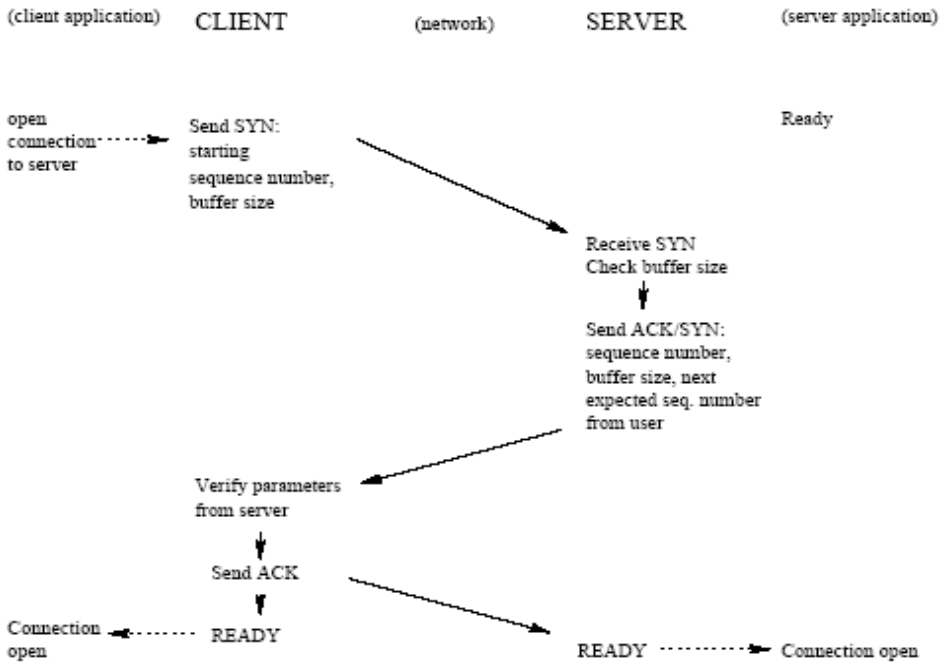


When the server receives the request message from the client, it searches for the file requested within its memory stores. If the server finds the file, it attaches an HTTP header to the file and then sends it back to the client. If the server does not locate the file, or if there is some problem with the file or the request message itself, the server sends back an error message. When the client receives a response from the server, it parses it and displays it in the browser window. If, when parsing the file, the client discovers that there are more files contained in the web document, such as inline images within an html file, the client sends a GET message for each remaining file in the document. The process continues until the client requests a file from another web server or ends its web session.

3.1.2 TCP and IP

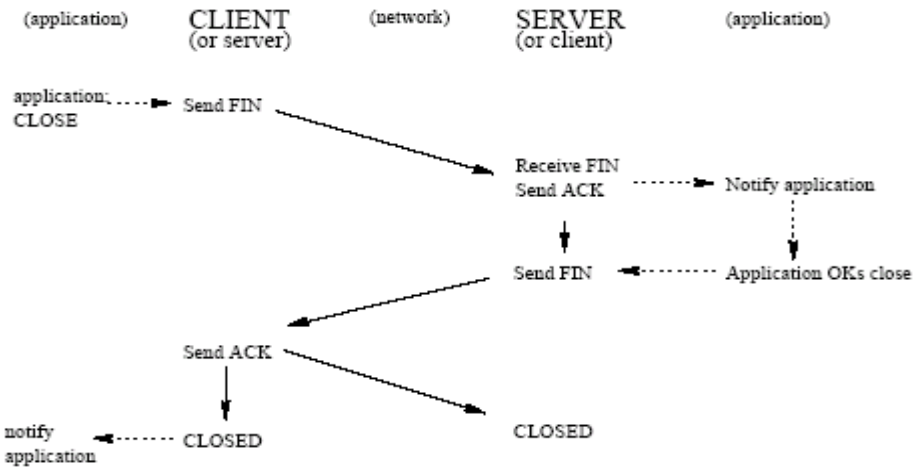
TCP and IP are the protocols used for data transfer on the Internet. TCP is the connection oriented transport layer protocol and IP is the connectionless network layer protocol. TCP ensures that packets sent from one host arrive error free and in a timely fashion at the destination host. As mentioned previously, HTTP runs on top of TCP at the application layer. To establish a TCP connection, two hosts exchange a three way handshake as illustrated in the Figure below. The initiating host sends a TCP SYN (synchronize Segment) packet to the second host. The SYN packet contains information about the size of the data segments that the first host will send and how much buffer space it has to store data sent by the second host. The second host responds with a TCP SYN_ACK (synchronize acknowledge) packet to indicate that it has received the SYN from the initiating host. The first host then ACKs (acknowledges) this packet completing the connection establishment process.

Fig 3.3 Three_way handshake to open a TCP connection



After completing the three way handshake, the two hosts can send packetized data to each other. In transferring data, one host sends a number of data packets limited by the minimum of the destination host's receive window and the congestion window described below to the destination host and waits for a reply. When the other host receives these data packets, it checks to see if any packets were lost in transmission. It does so by checking the received packet sequence numbers against the packet sequence numbers it expects to receive. The destination host then sends an ACK to the original host for the received packets. Once the first host receives this ACK, it transmits the next set of packets until the message transfer completes and all packets in that message have been ACKed by the receiver. If the first host does not receive an ACK within a specified timeout period, it retransmits any data packets that have not yet been ACKed by the Receiver. To tear down an existing TCP connection, the hosts exchange another three way handshake as illustrated in the Figure below. The host initiating the close sends a TCP FIN (final segment) packet to the second host. The second host responds with an ACK and then a FIN packet to acknowledge the close. Once the initiating host receives this packet, it ACKs the second host's FIN packet and breaks the TCP connection on its end. The second host breaks the TCP connection on its end once it receives this ACK packet. Either host may initiate the close.

Fig 3.3 Three_way handshake to terminate a TCP connection



Using TCP in this manner is problematic for several reasons and has been well documented in the literature. First, maintaining , establishing and tearing down TCP connections is expensive in terms of the server and client resources required. Second, setting up and tearing down TCP connections is expensive in terms of the number of round trip times needed for the task. For a single TCP connection, the number of round trip times is not significant. As the number of files in a web document increases, and subsequently as the number of TCP connections required to retrieve the document increases, the number of round trip times becomes more significant. Third, TCP contains some inherent congestion control mechanisms, in particular , slow start designed to benefit longer duration TCP connections such as those associated with TELNET sessions. Briefly, the slow start algorithm ensures that a host does not suddenly flood a possibly congested network with a large number of packets by dictating a ramp up process for transmitting packets. Slow start defines a congestion window that limits the number of packets a client is allowed to send onto the network at any one time. Initially, slow start sets this window to one packet. Each packet that is ACKed by the receiving end increases the congestion window by one packet. The number of packets the sender is allowed to transmit is always the minimum of the congestion window and the receiver's receive buffer size, the congestion window can never exceed this receive window size.

4. Queueing theory

As is often the case in computer systems, Web servers typically process many simultaneous jobs (i.e., file requests), each of which contends (competes) for various shared resources: processor time, file access, and network bandwidth. Since only one job may use a resource at any time, all other jobs must wait in a queue for their turn at the resource. As jobs receive service at the resource, they are removed from the queue; all the while, new jobs arrive and join the queue. Queueing theory is a tool that helps to compute the size of those queues and the time that jobs spend in them.

In this paper, we are concerned with the number of simultaneous HTTP GET file requests handled by a server, and the total time required to service a request.

4.1 M/M/1/K*FCFS Queue

Consider an M/M/1/K queue with First come First served service discipline. The arrival of jobs is according to a Poisson process with rate λ. The service time requirements have an exponential distribution with mean μ. An arrival will be blocked if the total number of jobs in the system has reached a predetermined value K. When a job has received the amount of service required, it leaves the queue. The probability mass function (pmf) of the total number of jobs in the system has the following expression,

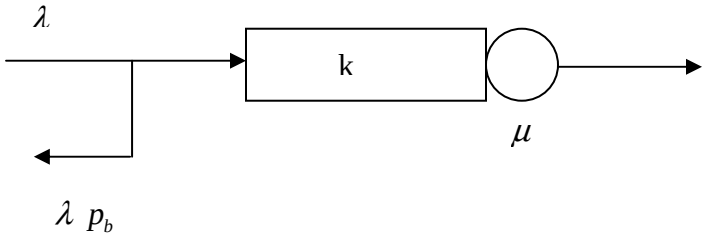
$$P[N=n] = \frac{(1-\rho)\rho^n}{1-\rho^{k+1}}, \quad n = 1, 2, ...k \tag{1}$$

Where ρ is the offered traffic and is equal to $\frac{\lambda}{\mu}$

5. Web Server Model

We model the web server using an M/M/1/K*FCFS queue as Fig. 4.1 shows. The requests arrive according to a Poisson process with rate λ. The average service requirement of each request is μ. The service can handle at most K requests at a time. A request will be blocked if the number has been reached. The probability of blocking is denoted as p_b Therefore the rate of blocked requests is given by $\lambda\ p_b$. Requests arrive at listener process and are dispatched to one of a pool of server processes/threads. From (1) we can derive the following three performance metrics, average response time, throughput and blocking probability.

Fig. 4.1: An M/M/1/K-FCFS model of web servers



5.1 Assumptions for the model

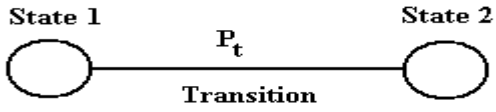
- The arrivals are assumed to follow a poisson distribution
- The population from where the arrivals originate is infinite (there are infinite number of web browsers)
- The queue length is limited to k requests .
- The queuing mechanism is FIFO
- One request atmost arrives/departs during any time interval

- The service times are assumed to follow an exponential distribution
- Each HTTP request is served by only one server

5.2 Birth - and - Death Processes

Most queuing models in which we are interested are based on birth-and-death processes. These processes can be represented by a state transition diagram, as shown in Figure 5.1.

Figure 5.1 State Transition Diagram



This diagram is intended to represent a single server (*i.e.*, an *M/M/1*) queue. In Figure 5.2, each state represents the position that a request has in the queue, Thus, the HTTP request in State 0 is the Request currently being processed, As new requests enter the system, they occupy the lowest available state; this means that the rate at which state transitions occur from a lower to a higher state, (*i.e., births*). More formally, an arrival transforms the system from its current state, *n*, to the next state, *n+1*. The transition rate from a lower to a higher state is therefore λ. Similarly, as requests are processed by the server, the system undergoes transitions from higher states (*n* to lower states (*n*-1), a process often referred to as *departures* or *deaths.* The rate at which these departures occur is just the service rate μ.

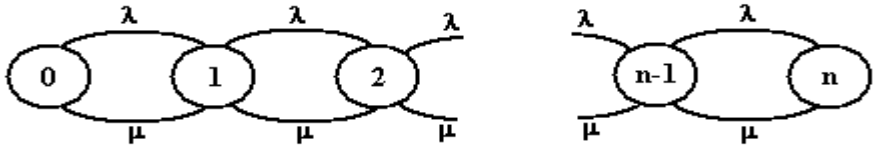


Figure 5.2 Single Server Queue State Transition Diagram

Given this, we can now begin to perform some calculations, with the goal of discovering our mean value parameters of interest [blocking probability p_b, Server throughput H and average response time T] . Based on the performance of a queuing system described above, we can write, in the steady state that

$$\lambda p_{n-1} = \mu p_n, \qquad \text{n = 1, 2,…..,k} \qquad (2)$$

(i.e, What goes into the system is the same as what leaves the system)

Where the probability of being in state *n* is p_n

Or

$$p_n = \frac{\lambda}{\mu} p_{n-1}, \qquad \text{n = 1, 2,…..,k}$$

This is the steady state probability; it does not change with time for a given traffic intensity. As the traffic intensity increases, the probability that more requests will be in the queue is greater.

Returning to the derivation at hand, since we could also represent p_{n-1} in similar fashion to the above equation, we can write:

$$p_{n-1} = \frac{\lambda}{\mu} p_{n-2}, \qquad \text{n = 1, 2,…..,k}$$

or

$$p_n = \left(\frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right) p_{n-2} = \left(\frac{\lambda}{\mu}\right)^2 p_{n-2}$$

Recursively this will give

$$p_n = \left(\frac{\lambda}{\mu}\right)^n p_0, \qquad \text{n = 1, 2,…..,k} \qquad (3)$$

This can be written as

$$p_n = \rho^n p_0, \qquad \text{n = 1, 2,…..,k} \qquad (4)$$

Where $\rho = \frac{\lambda}{\mu}$

Note that ρ is the traffic intensity for the web server. Furthermore, the utilization of the web server, ρ is just the probability that the server is busy, $1 - p_0$ where (p_0 is the probability that no requests to the server). Basic probability theory states the sum of all probabilities in the system must equal one, or

$$\sum_{n=0}^{k} p_n = 1$$

but, $p_n = \rho^n p_0$

so

$$\sum_{n=0}^{k} p_n = \sum_{n=0}^{k} \rho^n p_0 = p_0 \sum_{n=0}^{k} \rho^n = 1$$

The summation is a geometric series, *i.e.*,

$$\sum_{n=0}^{k} \rho^n = 1 + \rho + \rho^2 + \rho^3 + ...\rho^{k-1} + \rho^k$$
$$= \frac{(1-\rho^{k+1})}{1-\rho} \tag{5}$$

so we can now write

$$p_0 \sum_{n=0}^{k} \rho^n = \frac{p_0(1-\rho^{k+1})}{1-\rho} = 1$$

Therefore,

$$p_0 = \frac{1-\rho}{1-\rho^{k+1}} \tag{6}$$

Note that p_0 is the probability that there are no requests to the system, *i.e.*, the web server is idle. Thus the probability that the server is busy, or the server *utilization*, is just

$$1 - p_0 = 1 - \frac{1-\rho}{1-\rho^{k+1}} \tag{7}$$

Continuing with our calculation, if we substitute this value of p_0 back into the fourth equation, we can write

$$p_n = \rho^n \frac{(1-\rho)}{(1-\rho^{k+1})}, \quad \text{n = 0,1,2,…,k} \tag{8}$$

where P_n = probability that there are n HTTP requests in the system

5.3 Modeling the blocking probability

The blocking probability p_b is equal to the probability that there are K requests in the system, i.e. the system is full,
Therefore

$$p_b = P(N = k) = \frac{\rho^k(1-\rho)}{(1-\rho^{k+1})} \tag{9}$$

Where

$$\rho = \frac{\lambda}{\mu}$$

5.4 Modeling the Throughput

The throughput H is the rate of completed requests. When web server reaches equilibrium, H is equal to the rate of accepted requests,
Completed requests = Requests that arrived to the system minus Those were rejected
Therefore

$$H = \lambda(1 - p_b) = \lambda\left(1 - \frac{\rho^k(1-\rho)}{(1-\rho^{k+1})}\right) \tag{10}$$

(Everything that arrives *and is not blocked* must eventually depart.)

Alternate way to compute blocking of M/M/1/K: Look at the output side

- P(server is busy) = $1-p_0$
- When the server is busy, the output rate = μ
- When the server is idle, the output rate = 0
- So the average output rate = $\gamma = \mu(1-p_0)+0\,p_0$

  Also at steady state what goes in the system should be the same as what leaves the system. i.e, $\gamma = \lambda(1-p_b)$

- Equating our two formulas for γ we get

$$\mu(1-p_0) = \lambda(1-p_b)$$

- Solving for p_b we get

$$p_b = 1-\frac{1-p_0}{\rho} = \frac{(1-\rho)\rho^k}{(1-\rho^{k+1})}$$

5.5 Modeling the Response time

Total response time includes the time to connect, the time to process the request on the server, and the time to transmit the response back to the client. In our study several simplifying assumptions are built into the model. The effect of the HTTP requests on the network are ignored, since the requests are typically much smaller than the files that are served. Also, it is assumed that the size of requested files (and thus the service times) are distributed exponentially.

The average response time T is the expected sojourn time of a job. Following the Little's law, we have that

$$T = \frac{N}{H} = \frac{\rho^{k+1}(k\rho-k-1)+\rho}{\lambda(1-\rho^k)(1-\rho)}$$

6.0 Conclusion

In this paper we presented a queueing model for the performance analysis of web server, which predicts the system performance such as system throughput, blocking probability and the response time of requests as the system parameter such as number of connection per second and number of client the server services changes. The model describes the impacts and interactions of the TCP subsystem, HTTP subsystem, I/O subsystem, and network. We noticed that, when the number of HTTP threads are increased, the response times decrease as well. The model described in this paper presents many opportunities and challenges to perform further research in various directions. First, in order to identify performance tuning

guidelines, we must obtain a better understanding of the impact of the different system parameters (e.g., the arrival process, the buffer sizes, the file-size distribution, the number of HTTP threads, the number and size of the I/O buffers, etc.) on the performance of Web servers. Simulation runs must be performed to compare the performance of Web servers under these many configuration settings. In addition, the model can be extended in several directions. For example, in order to process transaction requests involving dynamic content (e.g., CGI/API scripts, Java servlets), many servers are equipped with a script engine (with a set of dedicated threads). The model may be extended to explicitly cover the impact of such a script engine implementation.

References

[1] "Apache. Apache web server". http://www.apache.org.

[2] Beckers, J., I.Hendrawan, R.E.Kooij, and van der Mei, R. (2001). Generalized processor sharing performance model for internet access lines. In 9th IFIP Conference on Performance Modelling and Evaluation of ATM and IP Networks. Budapest.

[3] Cao, J. and Nyberg, C. (2002). On overload control through queue length for web servers. In 16th Nordic Teletra_c Seminar. Espoo, Finland.

[4] Cherkasova, L. and Phaal, P. (2002). Session-based admission control: A mechanism for peak load management of commercial web sites. IEEE Transactions on computers, 51, no.6 pp:669-685, june 2002.

[5] Dilley, J., Friedrich, R., Jin, T., and Rolia, J. (1998). Web server performance measurement and modeling techniques. Performance Evaluation, Vol.33, pp. :5-26, 1998

[6] Hu, J., Mungee, S., and Schmidt, D. (1998). Principles for developing and measuring high-performance web servers over ATM. In Proceedings of INFOCOM '98, March/April 1998.

[7] Jain, R. (1991). The Art of Computer Systems Performance Analysis. John Wiley & Sons.

[8] King, P. J. B. (1990). Computer and Communication Systems Performance Modelling. Prentice Hall.

[9] Kleinrock, L. (1975). Queueing Systems, Volume 1: Theory. John Wiley & Sons.

[10] Mei, R. D. V. D., Hariharan, R., and Reeser, P. K. (2001). Web server performance modeling. Telecommunication Systems

[11] Menasc, D. A. and Almeida, V. A. F. (2002). Capacity Planning for Web Services.Prentice Hall.36

[12] Mikael Andersson (2002). "Web server performance modeling using an M/G/1/K*PS Queue".

[13] Slothouber, "A Model of Web Server Performance",

[14] Wells, L., Christensen, S., Kristensen, L. M., and Mortensen, K. H. (2001). Simulation based performance analysis of web servers. In Proceedings of the 9th International Workshop on Petri Nets and Performance Models (PNPM 2001), pages 59{68. IEEE Computer Society.

[15] Widell, N. (2002). Performance of distributed information systems. Technical Report 144, Department of Communication Systems, Lund Institute of Technology. Lic. Thesis.