NORTHWESTERN UNIVERSITY

Characterization of User and Server Behavior In Web-Based Networks

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical Engineering

By

Amy Csizmar Dalal

EVANSTON, ILLINOIS

December 1999

ABSTRACT

Characterization of User and Server Behavior in Web-Based Networks

Amy Csizmar Dalal

The explosion in popularity of the World Wide Web over the past few years has changed the focus of Internet research. While this research has given us some insight into the behavior of this segment of Internet traffic, many holes remain in our understanding of the interactions between World Wide Web users and the web sites (servers) they frequent.

We seek to gain a better understanding of the interactions between a population of impatient web users accessing a population of 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?

Our approach is twofold. First, we derive a state diagram model describing a typical web session from both the user and server perspectives. We identify key actions, events, and times associated with the lifetime of a web session on both the user and the server ends. We briefly explain how the combination of these three entities can be used to construct a computer model of a web session.

Second, we explore the concept of "server performance", defining performance in terms of how much revenue a server earns from processing a request from a user. We quantify revenue as a decreasing function of the amount of time a user waits for the server to process its request. We then use this definition to derive an improved service ordering policy at a web server. We find that when the goal is to improve user-perceived performance by maximizing the revenue earned by the server on average per unit time, the optimal service ordering policy tends to be greedy rather than fair. We demonstrate these results both analytically and via simulation.

We conclude by applying the server performance results in a brief exploration of server dimensioning issues.

In memory of my grandfather, Paul V. Jolley.

Acknowledgments

First and foremost, I would like to thank my research advisor, Scott Jordan. He has been a valuable mentor to me, and has helped me to mature as a researcher and a scholar. I am extremely grateful to have had the opportunity to work with him.

I would also like to thank my committee members, C.C. Lee, Michael Honig, and Aggelos Katsaggelos. In addition, several other faculty members have served as mentors to me, most notably Alvin Bayliss, Valerie Taylor, and Jorge Nocedal. I appreciate their patience and guidance as they assisted me in developing my teaching skills.

My labmates have been wonderful colleagues and friends during my time here. I would like to thank in particular Gowri, Wayne, Xiao, Terry, Lance, and Carlos. Tom Kostas kept me sane as we were both finishing up our dissertations, and I am grateful to him for helping to keep me on track. I would also like to thank Terri Fry, Lisa Singh and Danielle Walker, who have been true friends throughout this entire experience.

Several people deserve mention here because they inspired me to pursue a career in engineering. I am grateful in particular to Sr. Mara Walton, Cindy Wilson, Joanne Liebler, and Dr. Roger Knapp, all of whom were instrumental in encouraging my interest in math and science.

I could not have made it to this point without the love and support of my family: my parents, Dan and Sue; and my siblings, Laura, Carrie, and Danny. They have always believed in me and have always encouraged me to pursue my dreams.

Last but not least, I would like to thank my husband, Michel. Without his love, patience, sense of humor, and understanding, none of this would have been possible.

December 1999

Amy Csizmar Dalal

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This chapter contains the background information and motivation for this Ph.D. dissertation research. We begin with a description of the rise in popularity of the World Wide Web and the resulting implications for Internet backbone traffic. We explain the general operation of the two main protocols, TCP and HTTP, involved in web-based file transfers. Next, we describe the architecture of a typical web server and define the inherent design problems in the architecture. We then outline the scope of the research problems and explore previous work on related research problems. The chapter ends with an outline of the rest of this document.

1.1 Motivation

The World Wide Web is a fairly recent phenomenon. Before 1994, web traffic made up only a small percentage of Internet backbone traffic. The majority of backbone traffic consisted of FTP and TELNET traffic. Around 1994, the web started to gain popularity, first in the scientific community and then quickly with the rest of the population. Since 1994, the web has enjoyed exponential growth. Today, web traffic makes up the majority of Internet traffic, and the number of users and hosts on the web and on the Internet as a whole continue to grow at an extremely rapid rate.

As an illustration of this point, Figures 1.1 and 1.2 show the growth in the estimated number of hosts and users on the Internet, respectively, by year. Figure 1.1 shows that since January 1994, the number of hosts has roughly doubled every year. In the past year alone, between July 1998 and July 1999, the number of hosts has risen from 30 million to 60 million. The pace of the growth of users on the Internet is slightly slower than that of the hosts. As Figure 1.2 shows, though, since 1996 the number of users worldwide has increased by roughly 50 million each year.

The rapid rise in the number of participants on the World Wide Web, and the Internet in general, worries those who maintain the Internet backbone. Lit-

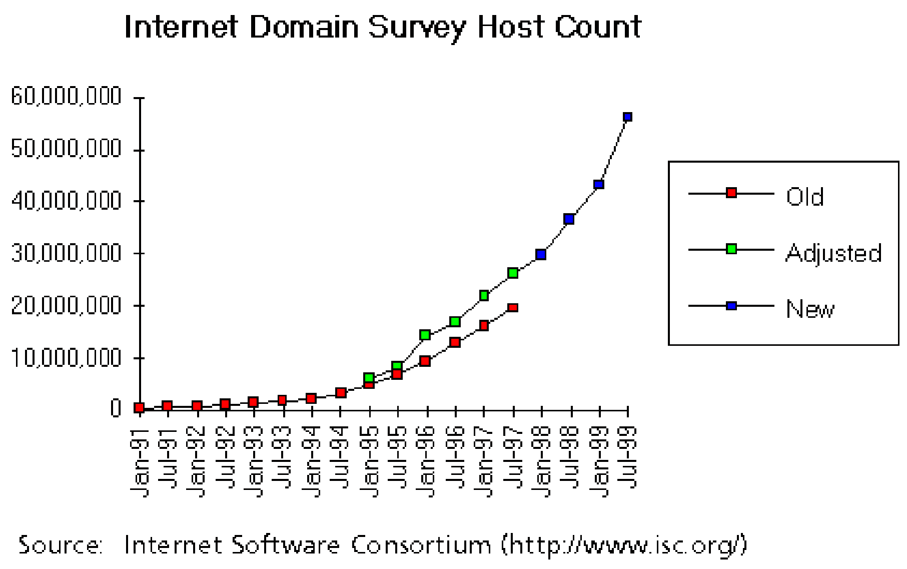

Figure 1.1: The number of Internet hosts by year [30]

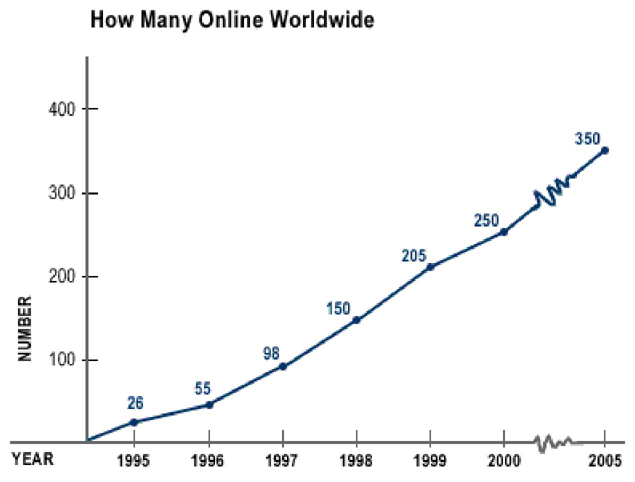

Figure 1.2: The number of users on the Internet by year, in millions [59].

tle is understood about the long-term implications of web traffic on the Internet backbone; however, understanding these implications is vital to the survival of the Internet. Steps must be taken to insure that the Internet has enough capacity to handle present and future access patterns, particularly those related to web traffic.

1.1.1 Web Server Protocols

In this section, we describe the main protocols involved in web transactions, TCP/IP and HTTP. TCP/IP 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.

Hypertext Transfer Protocol

Hypertext Transfer Protocol, or HTTP ([11], [22]), is a request-response protocol that operates between a web browser and a web server on the application layer. Figure 1.3 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.

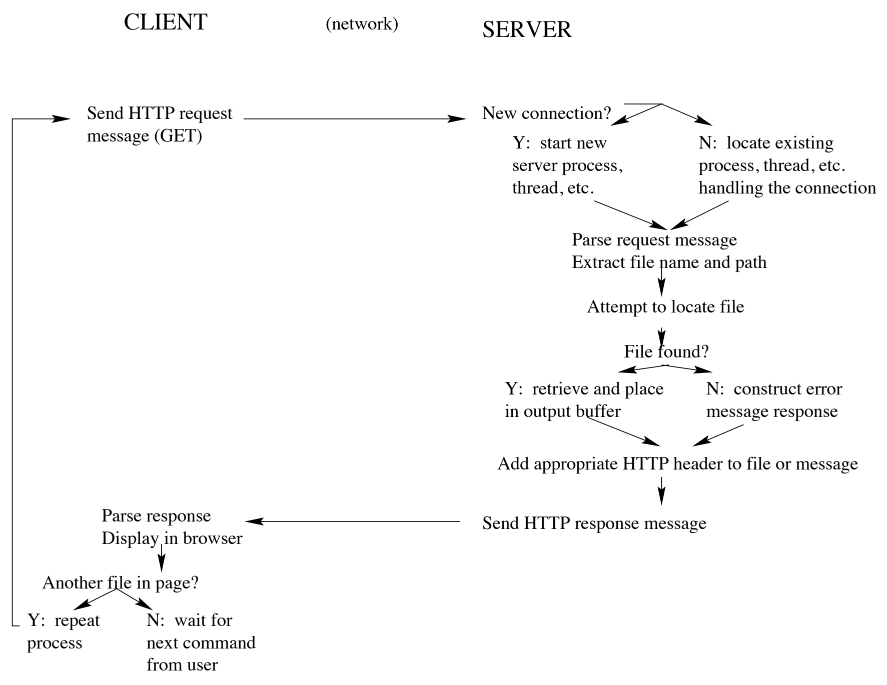
CLIENT
(network)
SERVER
Send HTTP request message (GET)
New connection?
Y: start new server process, thread, etc.
N: locate existing process, thread, etc. handling the connection
Parse request message
Extract file name and path
Attempt to locate file
File found?
Y: retrieve and place in output buffer
N: construct error message response
Add appropriate HTTP header to file or message
Send HTTP response message
Parse response
Display in browser
Another file in page?
Y: repeat process
N: wait for next command from user

Figure 1.3: 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.

TCP and HTTP

TCP and IP are the protocols used for data transfer on the Internet. TCP ([53],[20]) 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", illustrated in Figure 1.4. 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.

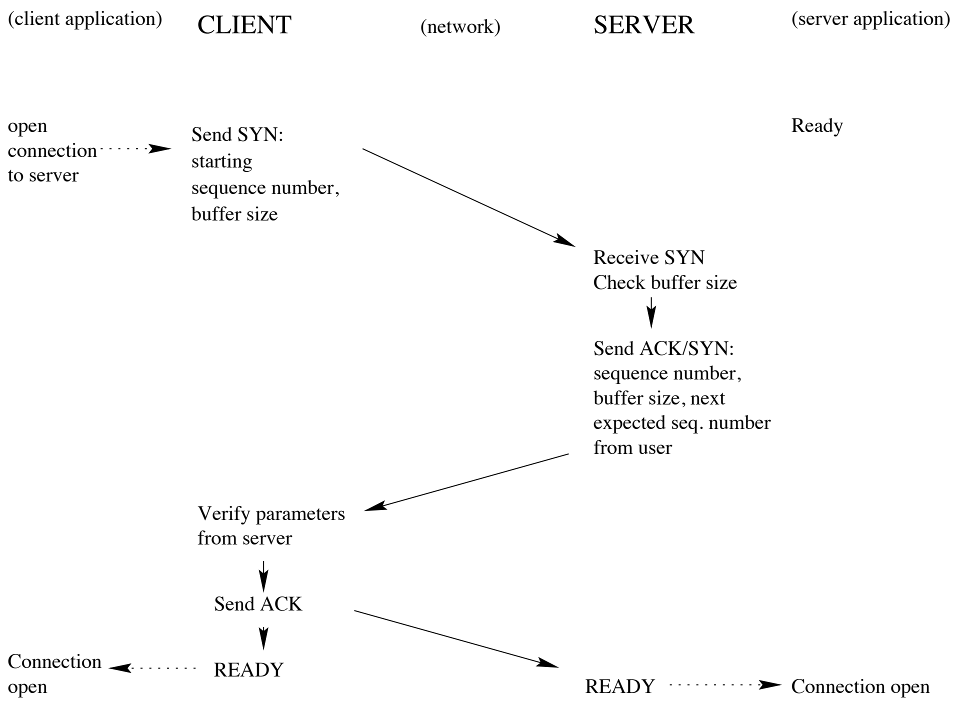
(client application)
CLIENT
(network)
SERVER
(server application)
open connection to server
Send SYN: starting sequence number, buffer size
Ready
Receive SYN Check buffer size
Send ACK/SYN: sequence number, buffer size, next expected seq. number from user
Verify parameters from server
Send ACK
Connection open
READY
READY
Connection open

Figure 1.4: 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”, illustrated in Figure 1.5. 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.

In HTTP/1.0 [11], each request sent by the client requires a new TCP connection to the server. Once the server sends its response to the client, it closes the TCP connection between them. For example, if a web document contains four image files in addition to the main .html file, client software using HTTP/1.0 opens a total of five distinct TCP connections to the server (one for each file). Earlier

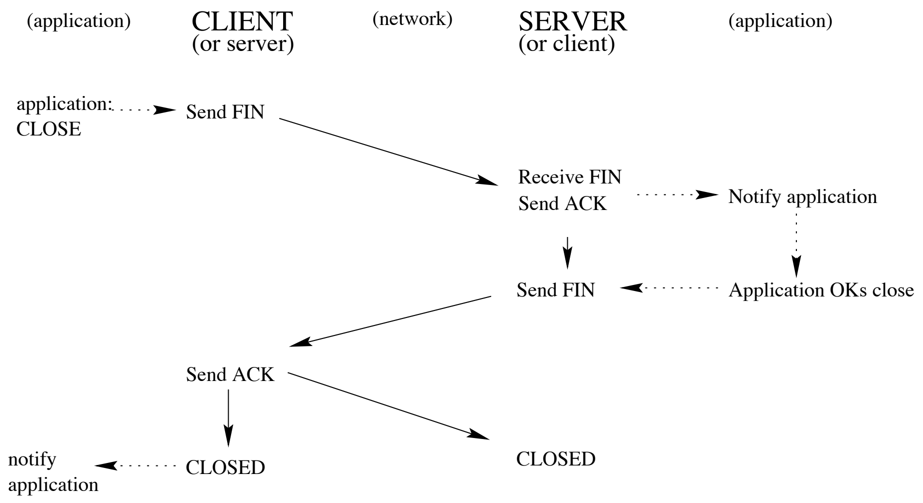

Figure 1.5: Three-way handshake to terminate a TCP connection.

versions of browser software would open these connections sequentially; later versions of some browsers (previous versions of Netscape Navigator [46], for example) implement simultaneous TCP connections for retrieval of multiple files. Returning to the previous example, if the client uses simultaneous connections to retrieve the document described above, it first opens one TCP connection to retrieve the .html file, then closes this connection once the file transfer completes. The client parses the .html file, determines that there are four more files to retrieve, and opens up four simultaneous connections to the server, one for each image file. The client closes each connection once the file transfer associated with that connection completes.

Using TCP in this manner is problematic for several reasons, and has been well-documented in the literature. (See, for example, [42], [48], and [56].) 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" [31], 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. This mechanism is ill-suited for HTTP transactions which tend to be of short duration and small in terms of the number of packets transmitted, because slow start takes up a significant portion of the total connection duration. Also, each time the client requests a new file from the web server, it must go through this process.

In 1997, the World Wide Web Consortium and Internet Engineering Task Force released the first version of the standard for HTTP/1.1 [22], a modification to the HTTP/1.0 protocol. With HTTP/1.1, the client has the ability to keep the TCP connection to the server open to make subsequent requests. When one TCP connection is used for multiple file requests, we call this *persistent connection TCP*. Persistent connections reduce retrieval latency by reducing the number of round-trip times required to retrieve a web page from a server. Returning to our example from the previous section, to retrieve the five files in the web document, the client opens just one TCP connection to the server. The client then either explicitly closes the TCP connection once all five files have been retrieved, or keeps the connection open if it anticipates retrieving more files from the same server in the very near future. The connection times out if it remains idle for an established period of time.

HTTP/1.1 represents an improvement over HTTP/1.0 for several reasons. First, the need for fewer setups and tear-downs of TCP connections means fewer round trip times are required per web document retrieval. The existence of fewer TCP connections means that less resources are used at the client and server overall to maintain the connections. Finally, since HTTP/1.1 reuses a TCP connection for several file retrievals, slow-start affects a smaller percentage of the connection duration time.

Because HTTP/1.1 utilizes persistent connections rather than simultaneous connections, in some cases file retrieval could take more time under HTTP/1.1 than HTTP/1.0. Under persistent connections, the client reuses one connection to retrieve multiple files, whereas using simultaneous connections, the client can send out multiple file requests at the same time. To address this shortcoming, HTTP/1.1 also implements *pipelined requests*. When a client pipelines a series of requests, it sends out the requests in succession on the same TCP connection, without waiting for a response. The server then processes each connection in turn, sending back the responses according to the order of the requests. This helps to reduce the total retrieval time associated with a web document without increasing the number of necessary TCP connections. [47] verifies the performance improvements from using persistent pipelined connections in HTTP/1.1 as compared to both HTTP/1.1 without pipelined connections and HTTP/1.0 with multiple simultaneous connections.

1.1.2 Web Server Architecture

In this section, we briefly describe the typical configuration of a web server. Specifically, we discuss the structure of a typical web server software program, or HTTP daemon, and the relationship between this daemon and the underlying operating system of the computer on which the software runs.

A web server software program is basically an implementation of the HTTP protocol on the server side. Figure 1.3 illustrates the basic operation of the HTTP protocol from the client and server sides. This section expands on the server side of this diagram. In the following discussion, we use the terms HTTP daemon (or httpd for short), web server software, web server daemon, and server daemon interchangeably.

The web server software interacts with the computer system's CPU, file system, and network interfaces in order to serve web documents to clients that request them. The software only provides an interface to these resources, and does not control them directly. The scheduling of processing time for requests, for example, is left to the operating system kernel. The software interfaces with the network connection only via reads and writes from a specific socket, commonly referred to as a port. Finally, the software interfaces with the file system in retrieving files and writing to various log files.

A web server daemon processes incoming requests in the following manner. The daemon listens to a specific TCP port, typically TCP port 80, for incoming requests. A client, after establishing a TCP connection with the server, sends an HTTP request to the server. Upon receiving this request, the server daemon parses the request message to determine the requested file and any other special instructions involved in the request. For example, the client may only be able to accept certain file types, called MIME types; it would then send a list of the acceptable types to the server as part of the request message. The server daemon next expands out the file name and locates the requested file, either in the file system or in the memory cache, using this extended path name. Lastly, the server daemon writes this file (or an error message), along with the proper headers, to the network queue to be sent out on TCP port 80 back to the client. If the server implements HTTP/1.1, the connection between the client and server remains open until either the client explicitly closes it or the connection times out.

One of the most important aspects of the server daemon is its ability to handle multiple simultaneous requests. There are two ways in which web server software accomplishes this. The first way is via cloning. In this method, one instance of the server daemon is created upon initialization. This parent process listens on TCP port 80, or some other port dedicated to HTTP traffic, for incoming web request messages. Upon arrival of a request, the parent process spawns a child process and passes the connection to this child process. The child process serves the connection for the entire lifetime of the connection. Once the connection closes, the child process is either killed or added to a process pool to be reused by a later connection.

Spawning child processes can be a time- and resource-consuming process, so in some cases the parent process pre-spawns a number of child processes to form a "process pool". The parent then passes incoming requests to one of the processes in this pool. Again, the processes may or may not be reused, or may have a "lifetime" associated with themselves to dictate how many requests they may service before being killed. The Apache server [2] operates in this manner.

Another way in which web servers handle multiple simultaneous requests is by using multiple threads. This method creates a thread for each HTTP connection. The threads are created and controlled by a main thread, which operates similarly to the parent process described above. As with the parent/child processes, the main thread may pre-create a thread pool to handle incoming requests faster. Examples of multithreaded servers include Netscape Enterprise [45], Zeus [62], Microsoft IIS [40], and PHTTPD [51].

The scheduling of the processes and threads is left largely up to the operating system kernel. Multithreaded servers tend to share resources among all open connections. Typically, one thread executes until it reaches a "pause" in execution, such as a disk read or a write to a log file. At each pause, the execution switches to another thread until that thread reaches a pause in execution, and so on. Process-based servers may operate by sharing resources as in the multithreaded case, or in a first-in, first-out manner. In most cases, however, the combination of web server and kernel scheduling results in some form of processor-sharing service ordering policy.

In general, the thread-per-request model is more efficient than the process-per-request model. It generally takes less time to switch between threads than between processes. Each thread retains its own information about the network connection it is handling and the memory associated with the request, but shares the web server program code and global data. Each process in the process-per-request model, on the other hand, is a complete copy of the web server daemon, and retains its own information about the connection, memory, and program code. Thus, switching between threads incurs much less overhead, because only the connection information has to be copied to runtime memory; whereas switching between processes entails copying the connection information, memory information, and program code to runtime memory.

Using threads has its disadvantages. First, a multithreaded server only works well on an operating system that supports threads. This is becoming less of an issue as most of the popular operating systems support threads now. Second, a multithreaded server is more complex than one that uses the process-per-request model, because it is difficult to program and coordinate the execution of threads, particularly when doing file input/output operations. Third, in some cases dynamically-generated pages cannot use threads, but must be generated by a process. Doing so eliminates the time-saving advantages of the threaded operation.

The main disadvantage to using the process-per-request model, besides being slower than the multithreaded model, is that care must be taken to ensure that too many processes are not generated/active at once. Each process, even when inactive, takes up memory and CPU time; thus, the pool of available processes must not exceed some maximum value (which varies from system to system).

1.1.3 Issues

One area of particular interest to Internet researchers is in identifying the sources of bottleneck, or excessive delay, within a web session. In many cases the backbone network is to blame for much of the bottleneck. Most of the congestion is related to the interactions between HTTP and TCP/IP, which we explored in Section 1.1.1. Even with the implementation of HTTP/1.1, which utilizes TCP connections more efficiently than its predecessor, web traffic stresses the backbone network, in terms of both the sheer volume of traffic and in the way traffic traverses the network.

Many researchers believe that the key to identifying and alleviating congestion on the Internet due to web traffic lies in developing a clear picture of web traffic and defining its characteristics. However, this task has proved difficult thus far.

There are several reasons for this, many of which are addressed in [50]. For one, the picture of web traffic is constantly changing. Not only is web traffic growing in terms of the number of users and hosts sending and receiving it, but it is also changing in terms of what constitutes "web traffic". The type and number of applications classified as web traffic has not remained constant, even over relatively brief periods of time. As a result, it is difficult to characterize what web traffic will look like in a few years or even a few months. The increasing use of Java, scripting languages, dynamically-generated content, and multimedia traffic is already changing the nature of web traffic. Another factor is the exponential growth of the user population on the web, which makes it increasingly difficult to profile the "typical" web user. In addition, the nature of the network itself is dynamic. New hosts and routers are added daily, and the paths traversed by Internet traffic, because they are dependent on the instantaneous congestive state of the network, change on the order of minutes or even seconds. All of these factors contribute to the difficulty in characterizing World Wide Web traffic on the Internet.

Some of the bottleneck can be traced to the architecture of the web server itself, and the web server software in particular. Most of today's web servers handle incoming web requests in a round robin manner. This method of scheduling requests is partly a consequence of operating system design and partly due to the web server software design. Traditionally, it has been held that processor sharing policies (such as round robin) yield the best "user-perceived" performance, in terms of fairness and response time, when service time requirements are variable. Processor sharing policies typically allow shorter jobs to complete before longer jobs, and the average time a job spends in a system that employs processor sharing is linearly proportional to its required service time. However, this theory falls short when applied to the construction of web server software. Web users tend to be very impatient and will abort, or cancel, a pending request if a response is not received in a matter of seconds. Users that time out in such a manner cause the server to waste resources on a request that never completes. At a lightly-loaded server handling a few requests per second, this may not pose a problem; but at a heavily-loaded server with many requests arriving per second, this may ultimately prove disastrous, and lead to a situation of "server deadlock", where the server works at capacity but does not complete any requests.

The two main issues we address in this thesis are the user population characteristics and web server characteristics. To characterize the user population, we consider what a web session looks like from one user's and one web server's perspective, identifying the key stages and parameters involved in such a session. To further characterize the web server, we again consider one web server, and determine how requests from remote users are processed at that server via a queueing theory approach. We do so by exploring the concept of "server performance", defining what constitutes "good performance" and using this definition to derive an improved service ordering policy at a web server.

The objective of the web server in the system studied here 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 model, requests arrive at the system according to a Poisson process. They are numbered in the order that they arrive at the system. 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. Once a request has entered the system, it does not leave until it completes service. We assume that swap times are instantaneous.

Our metric of interest is a quantity we call "revenue". We define revenue as a measure of the perceived performance per user per file. The user "pays" the server an amount relative to the download time upon the server's completion of the user's request. Here, we ignore the transit time required to send the result of the request back to the browser, and assume that the revenue is collected instantaneously upon the file's departure from the server.

We describe revenue as a decaying exponential function of the time the request spends in the system before completing service. Thus, the longer the request remains in the system, the lower its probability of remaining at the server until its service is completed. Our goal is to find the service policies that maximize the average revenue earned per second by the server.

1.2 Prior Work

Despite the relative newness of the web, there has been a significant body of work established in Internet research. The prior research falls into three main areas: research into issues on the network-level and transport-level, or how HTTP traffic traverses the backbone network and interacts with other TCP traffic; research into issues at the session level, which includes characterization of browser and/or server behavior; and research into issues on the application level, most of which concerns defining and measuring the "performance" of the web server in terms of one or more metrics and finding ways to improve upon its performance (typically by modifying the order in which web requests are serviced).

1.2.1 Network- and Transport-level Research

Network- and transport-level research studies the interactions of HTTP and TCP and the effects that HTTP traffic has on Internet (TCP) traffic as a whole. Since 1995, HTTP traffic has dominated the traffic on the Internet backbone. By characterizing how HTTP and TCP interact on the backbone network and/or at the server, researchers hope to predict how Internet traffic will look in the future.

One of the premier works in this area is [41]. This work studied the HTTP and TCP characteristics of a heavily-trafficked web server, the 1994 election server in California.[1] The study measured the usage of network resources at the server level using server logs and packet traces of the LAN on which the distributed hosts resided. In addition to these measures, the study also measured some server characteristics, including peak and average hit rates, number of retrievals per client, file popularity, and load imbalances among the hosts. The study was primarily transport-layer-focused; in particular, the study was expanded upon in [42] to justify the revision of the HTTP protocol in order to utilize the TCP protocol more efficiently and cut down on the amount of backbone traffic.

A similar study [4] used packet traces from the Atlantic Summer Olympics web site in 1996. This, too, was a transport-layer-centric study. In particular,

[1]The site received heavy traffic as compared to most web sites in the same time period. It received nearly one million hits in a 24-hour period, which in 1994 was a phenomenal amount of web traffic.

this study was concerned with the relationship between the number of open TCP connections and the performance of the server in terms of throughput and loss rates.

The two papers mentioned above are similar in that they look at transport-level traffic from the server's perspective. Another related approach looks at this same traffic from the perspective of the backbone network, using packet traces from points on the backbone network or from stub networks in order to characterize the HTTP and/or TCP traffic. We discuss several of these studies below.

The original studies on the backbone network did not involve HTTP traffic. We mention them here because they in essence sparked much of the interest in Internet research, and many of the problems and characteristics they identified are still hotly debated today. The pioneering work in this area is described in [35], which measured packet-level traffic on an Ethernet between 1989 and 1992. The study concluded that traditional forms of modeling packet-level network traffic—using Poisson arrivals—are inaccurate. Instead, the study found that Ethernet traffic exhibits some properties of *self-similarity*, meaning that it shows burstiness across all time scales which does not average itself out on longer time scales, as exponential traffic does. [49] performed a similar study on Telnet and FTP traffic over the Internet backbone. They found evidence of self-similar characteristics, but also found some cases in which Poisson modeling was adequate for this traffic (in particular, for session arrivals).

Modern studies in this area examine the behavior of the Internet in the face of ever-increasing amounts of HTTP traffic. These studies attempt to characterize HTTP traffic on the backbone network in order to predict future traffic patterns and identify potential capacity problems and bottlenecks on the backbone network. The best examples of this research are [14] and [21], both of which use packet traces of backbone traffic to identify traffic "flows" that correspond to HTTP sessions between endpoints.

The works described above relate indirectly to our research. They represent one approach to characterizing web and Internet traffic. They are also important to our research in that they present ideas as to the shape of the traffic on the backbone, which we apply to our web session characterization.

1.2.2 Session Level: Web Characterization

The most extensive research has been done in the area of web session characterization. Web characterization research attempts to parameterize certain elements (usually traffic patterns or server actions) of a typical web session. Several approaches have been undertaken, ranging from server-centered characterization, to client-centered characterization, to network-centered characterization.

The pioneering work in this area was done by Arlitt and Williamson, who in [3] identified ten invariants of web traffic seen at a web server by studying server log files from 1994 and 1995. This web workload study involved six different servers that saw several orders of magnitude of traffic over two different orders of magnitude of time. The ten invariants represent traffic characteristics common to the log files from all six web servers. The invariants ranged in scope from file size distribution, to the distribution of requested file types, to locality of requests with respect to the client's IP address. This work was pivotal because it paved the way for similar studies of server and client traffic patterns, and also because most of the invariants have been confirmed by other researchers.

Others who have attempted to characterize web traffic from the server end include [34], who used server logs from 1994 from the NCSA server; and [38], who also used server logs in a workload development study for a new web server benchmark.

Another pivotal paper in this subgroup tackles the traffic characterization problem from the opposite end: the client side. [18] presents a traffic analysis from logs gathered from instrumented versions of Mosaic [44], the first popular web browser software. The study confirmed some of the invariants in [3], in particular the heavy-tailed distribution of file sizes and the concentration of file references. The study was updated in 1998, with the results presented in [6]. In the updated study, new traces were taken from a proxy server installed on the same workstations as the web browsers in the study (this time Netscape and not Mosaic) and compared to the original traces. The study found that the file size distribution was largely unchanged from the original study, but that the effects of file reference concentration were less pronounced in the later data set. A third study based on this data set, [15], uses the original data to establish the self-similarity of the client-side traffic and the heavy-tailed nature of the file size distributions. A similar but unrelated paper is [23], which used logs from a proxy at a modem bank to characterize the web traffic at that location.

Another similar study is described in [29]. This work measured traffic from an AOL caching proxy in 1997 and also from another instrumented version of Xmosaic at Georgia Tech in 1994. This study, unlike those mentioned previously, focused on characterizing the number of pages accessed per web site and the number of visits per web site over a time period.

Still other studies attempt to view and characterize traffic from the network level. Mah [36] used packet-level traces from a LAN, collected in 1995, to reconstruct web session traffic between clients and servers. From these traces, he calculated empirical distributions for such quantities as request size, response (file) size, user think time, number of files per web page, and number of documents retrieved per server visit. Even though it used packet-level traces, the study was not concerned with any packet-level characteristics of the traffic.

Some studies combine session-level characterization with network-level characterization. For example, [7] presents an analysis of the effects of HTTP traffic on the network and also on web server resources, such as CPU utilization and disk accesses; it studies both HTTP/1.0 and HTTP/1.1. An earlier study, [9], similarly looks at CPU utilization and its relationship to the TCP connections carrying HTTP traffic. [8] similarly proposes a model for measuring web client, server, and network performance simultaneously. [1] studies the interactions on three levels: the operating system level, the application level (HTTP), and the network level (TCP). It measures system level characteristics like CPU utilization, process lifetimes, and logging times; as well as file size distributions and the throughput in connections per second. [5] presents a model of a traffic generator that is mainly concerned with reproducing client access patterns. Finally, [37] uses a web proxy log to construct a server workload, characterizing such measures as CPU utilization and response times.

A study that does not fit the above categories is [39], which presents a “site characterization” from web server logs. It measures such factors as growth indicators, variation in site volume, and the differences in traffic patterns at different web sites.

These studies and others (see [52] for a summary of the most relevant web characterization studies through 1998) relate to our research in that they present various methods of characterizing a web session. The problem with most of these studies is that they rely on empirical data. While using empirical data does have its advantages, it may quickly render these studies obsolete since web traffic changes so rapidly. Our work, by contrast, relies on identifying known actions and events tied to the HTTP and TCP protocols to characterize web traffic. Using this method allows us to create a web session characterization that is valid under a variety of traffic patterns and types.

1.2.3 Application Level: Server Queueing Models

The research on the application level is most closely related to the majority of the work in this thesis. The majority of the work in this area has been done by the same group of researchers. The approaches for the works described in this section are similar. All of the studies model the web server as a queue or a system of queues to represent the web resources (CPU, file system, and network queue). They all define performance in terms of the latency involved in serving a request; typically, the studies use user-perceived latency of some form. In most cases, the analysis involves modifying the service ordering at the web server in an attempt to improve performance in terms of the chosen latency measure.

Two key papers in this area are [16] and [26], which compare less conventional service ordering policies to service orderings traditionally found in web servers. [16] compares the two most commonly-used service orderings in web servers, first-in, first-out and processor sharing, to a shortest-connection-first service ordering. The study compares two "fair" service orderings against a size-based policy, and quantifies the performance in terms of mean response time, where response time is defined as the sum of the waiting time and the service time. They find that the "unfair" policy performs four to five times better, in terms of mean response time, than either of the fair policies. [26] is also a comparison of a fair service ordering, processor sharing, against a more aggressive policy, shortest-remaining-processing-time. Like [16], this paper measures performance in terms of mean response time, but it also looks at mean slowdown as a performance measure, where mean slowdown is the ratio of waiting time to service time. Also like [16], this work finds that the "unfair" policy represents a performance improvement in both metrics as compared to the processor sharing policy.

The two papers described above are similar to our work. Each paper presents both an analytical analysis and a simulation analysis of the queueing system, as we do. Each of the papers measures performance, at least partially, in terms of user-perceived response time solely at the server level. While we do not define performance in the same manner, our measure of performance is a function of response time.

The key differences between our approach and the approaches presented in these two papers lie in the goals of the analyses. The papers described above select one service order policy that may perform better than one or more commonly-used service ordering policies. Then, using a "realistic" service time distribution, they determine the improvement in response time (or some other related metric) when this new policy is used in lieu of the traditional policies. The two papers, and much of the rest of their work, assume that the file size distribution is heavy-tailed, meaning that it is modeled by a bounded Pareto distribution with parameter $\alpha, 0 \leq \alpha \leq 2$; and that the server knows the size of each task upon its arrival to the server. They argue that this is an appropriate model for static files arriving at a web server, since the server can deduce file sizes from the HTTP GET message and since a number of studies (see, for example, [3], [18], [6], and [36]) verify the heavy-tailed nature of file sizes for static files. Thus, using such a model gives them a "realistic" picture of how the new policy improves upon the old policies, both analytically and in practice (empirically). By reducing mean response time and other mean time-related metrics, they infer that more pages on average can be served to the general population of web users.

By contrast, our approach is to consider *all* general policies (within a certain set of conditions), including traditional, well-defined policies and non-traditional policies that are not clearly defined. Once we determine the set of acceptable policies under consideration, we systematically eliminate the policies that do not maximize our metric of interest, average revenue per unit time, until we are left with a policy or a family of policies that does maximize average revenue per unit time under the limiting conditions. We define this as our optimal policy, because it optimally serves the pool of impatient users; by maximizing the revenue per unit time, we infer that we are maximizing the number of users that remain at the server until their service requirements are met. We find, in fact, that under most conditions the optimal policy can be described by one of a family of "greedy" policies, rather than one well-defined policy. In order to make our analysis tractable, we necessarily must use a less-accurate model for file size than the model used in the work described above. We thus assume a worst-case scenario in which file sizes are exponential, meaning that the server does not have any sense of the size of each incoming task.

A related paper is [10]. The focus of this work is on avoiding "starvation" of incoming requests, or ensuring that requests are serviced within some maximum response time. The performance of the server is measured in terms of two metrics: response time per request and slowdown per request. Unlike [16] and [26], this work is concerned with satisfying both fairness and good performance; also, this work looks at the performance with respect to individual requests instead of the on-average behavior of the system. As a result, the optimal policy deduced in this paper is a family of earliest-deadline-first policies. The authors find that these optimal policies are too complex to implement in practice, however.

Another set of closely-related papers looks at service ordering among hosts in a distributed server system. In this system, the web content is duplicated at a number of hosts, and incoming requests are assigned to one of the hosts according to a "load-balancing" scheme. [25] compares four task assignment policies: round-robin, size-based, random, and dynamic-least-work-remaining. This work assumes that each host uses first-in, first-out service ordering. It finds that the best task assignment policy depends on the amount of variance in the task sizes (file sizes), and that the optimal policy is either size-based or dynamic-least-work-remaining. It measures performance in terms of mean waiting time, mean queue length, and mean slowdown, both overall and at each host. [17] studies the effects of running each host in a distributed server system at different loads, and assigning small incoming requests to the more lightly-loaded hosts. It measures performance in terms of mean slowdown and mean waiting time, and finds that the performance in terms of mean slowdown improves using such a scheme significantly, by up to several orders of magnitude; it also explores the tradeoff between reduction in slowdown and increase in waiting time at the more highly-loaded hosts. As in the service ordering studies, task sizes are heavy-tailed and known upon arrival at the server system.

[24] is a combination of the service order and load balancing studies. It compares two service orderings at each host, processor sharing and first-in-first-out, where tasks are assigned to each host based on their file sizes (which again are known to the server and are heavy-tailed). This work finds that in this case, first-in-first-out actually outperforms the more “fair” policy, processor sharing, by utilizing the CPU and server resources more efficiently. We mention this work here because like our research, it challenges the conventional wisdom as to the best way to service incoming requests.

In addition to the works mentioned above, [55] studies the performance of a web server by modeling it as an open Jackson network. Like the papers mentioned above, and like our research, this work measures performance in terms of response time at the server. [55], however, views how response time is affected by server parameters such as network bandwidth, processor speed, and adding additional hosts to a distributed server system. (We explore the latter two in our research.) Like our study, [55] assumes that file sizes are exponential, so that the server has no indication as to the service requirement of an incoming request. However, [55] only considers first-in, first-out service ordering, and does not study how alternate policies may affect response time.

1.3 About This Document

The rest of this document describes the research areas that comprise this doctoral dissertation.

Chapter 2 discusses the traffic modeling problem. The goal of this portion of the research is to characterize web traffic from the user's perspective and the server's perspective. The chapter covers the important background issues and discusses how the important parameters in a web session are identified. It ends by suggesting how these parameters can be measured and perhaps quantified in distribution form.

Chapter 3 discusses the basic server performance problem. The goal of this portion of the research is to identify ways in which server performance can be modified to improve the users' perceived performance (or perceived latency). We study this problem by deriving an optimal service ordering policy for a web server or server cluster.

Chapter 4 extends the basic model to include two permutations of the scenario presented in Chapter 3. First, we assume in the original model that all incoming requests pay the same amount of revenue to the server upon successful completion of the request. We modify the model so that this reward varies among the incoming requests. Second, we examine the case where the file sizes of incoming requests are drawn from a family of exponential distributions, each with a unique mean. This corresponds to the case where a web server contains several different types of content files (HTML files, image files, CGI scripts, et cetera) that are described by different exponential distributions. We look at the worst-case scenario in which each incoming request has a file size drawn from an exponential distribution with a mean unique to that request.

Chapters 5 and 6 present empirical results derived from simulations. The basic simulation model is described in detail in Chapter 5. The revisions made to implement the service orders derived for the extensions in Chapter 4 are described in Chapter 6. In each chapter, we describe and interpret the key simulation results obtained. The metrics of interest include total revenue per simulation, average revenue per request, and average response time per request.

Chapter 7 discusses the issue of server dimensioning. To handle high "hit rates", it may be necessary to consider upgrading web server components, either by replacing the CPU with a faster processor, using multiple CPUs in a single host, or using a distributed host system. We present several examples of revenue/cost tradeoffs and determine prudent courses of action to further improve server performance (as defined in previous chapters).

Finally, Chapter 8 summarizes the accomplishments of the thesis and discusses remaining unresolved issues.

Chapter 2

CHARACTERIZATION OF USER AND SERVER BEHAVIOR

2.1 Introduction

The first part of this thesis concerns the generic problem of characterizing a web session. We describe the interactions between a user, the user's web browser software, and a web server via a three-tiered state diagram. We also identify the relevant parameters associated with each tier of the state diagram and describe how to use these parameters to construct a computer model of the system.

In the descriptions that follow, the terms *client*, *web browser*, and *web browsing application* are used interchangeably to refer to the web browsing software (e.g., Netscape Navigator, Internet Explorer). *User* refers to the person accessing the World Wide Web, and *server* refers to the web server software. We also make no assumptions as to the physical setup of the web server.

2.2 Model

Consider a computer network consisting of a web browser and web server. The web browser and web server communicate over a TCP/IP network via the HTTP protocol. We wish to characterize the interactions that occur between the web browser software and the web server software, as well as the interactions between the human user and the browser software.

Each portion of the system (browser, user, and web server) goes through a series of "stages" during a web session. In each case, there is a series of setup stages in which the entity establishes a communication link to another entity (for example, when the browser connects to the server or the user starts up the browser). The setup stages are followed by the actual web page (or web document) retrieval and display. The session ends when the entities tear down the communication link between them and each entity retreats to an "idle" state.

We define a *state* to be one of the stages of a web session for either a user, browser, or server. The series of stages assigned to an entity during a web session comprises the *state diagram* for that entity. Our system consists of three such state diagrams: the user state diagram, the browser state diagram, and the server state diagram. We define an *event* as a transition in a state diagram of one entity that causes a transition in a state diagram of another entity. Transitions between states in any of the three state diagrams occur either as the result of an event or after a specified time value.

Figure 2.1 illustrates the overall state diagram for a web session. Transitions between states are denoted by solid arrow lines. Events are pictured as dotted arrow lines. Events allow states from the three diagrams (user, client, and server) to "communicate" with each other. The client is the only entity that communicates with both of the other entities; the server and user only communicate with each other via the client.

In our system model, we treat the underlying network connecting the browser to the server as a "black box". It is beyond the scope of this thesis to model all of the actions of the underlying TCP protocol; thus, we remove quantities associated with the TCP protocol and model each of these as a time delay. We model the

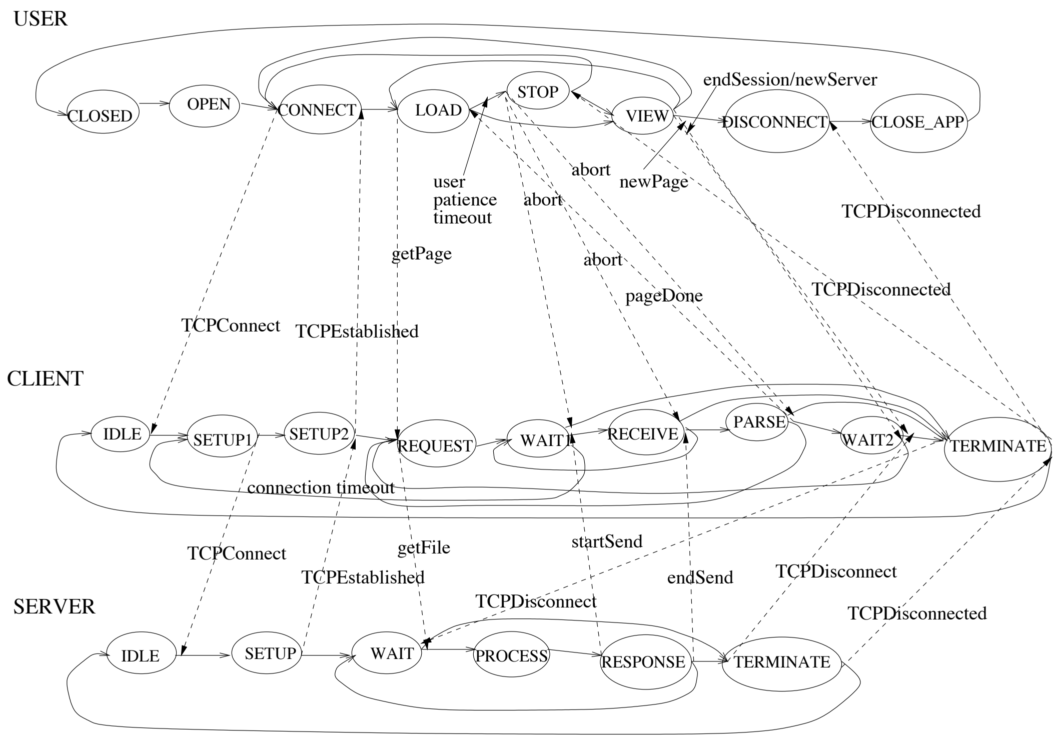

Figure 2.1: State diagram for a WWW session.

delay as a function of the level of congestion[1] on the network as well as the size of the file being transferred.

From the state diagrams, we derive the important parameters inherent in a web session and guess as to their distribution types. These parameters are identified within the descriptions of the states in each state diagram.

2.3 State Diagrams of a WWW Session

The state diagrams described below assume that the client and server use HTTP/1.1, which allows for persistent connections between the client and server. That is, the same TCP connection can be reused to service multiple requests from a single client to a single server within a time period specified by either the client or the server. We assume that persistent connections are always used when a client connects to a server.

2.3.1 User State Diagram

During a web session, the user interacts directly with the browser software and indirectly with the server. Figure 2.2 illustrates the portion of the session state

[1]The level of congestion accounts for things such as path length, queueing delay at the routers, and number of packet retransmissions.

diagram corresponding to the user state diagram. There are eight states, of which two are setup states and two are closing states. As in all of the diagrams that compose the larger state diagram, this diagram includes a null or idle state.

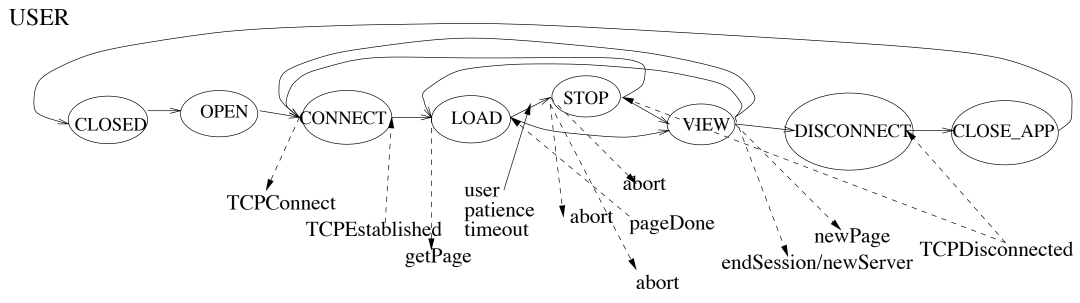

Figure 2.2: The user portion of the state diagram. This diagram includes the user states as well as the events originating and terminating at user states (from the client states).

Closed: In the closed state, the user is off-line, and the web browsing application is not running. This state is the user's "idle" state, or the inter-web-session state.

The time spent in this state, t_{idle}, is the time between distinct web sessions by a particular user, and varies from user to user. We assume that it is a measured random variable, in that we could derive its distribution empirically by measuring data from web sessions.

Open Browser: This is the first setup state in the user state diagram. In this state, the user opens and initializes the web browsing application and goes online. The user arrives at this state from the *Closed* state.

The time spent in this state, t_{init}, is the time required to open and initialize the web browser software. We model this time as a measured constant that is dependent on the browser software and the type and speed of the processor on the user's machine. After this amount of time elapses, the user transitions to the *Connect to Server* state.

Connect to Server: This state represents the second setup stage. The user chooses a web server from which to retrieve a document or web page, by selecting a URL (either by typing the URL in the browser window, or by clicking on a hyperlink, or by selecting a bookmark from a bookmark file). The user then waits for the connection between the browser and the server to be established before moving on to the next state.

Upon entering this state, the user sends an "open TCP connection" event to the client state diagram. It exits this state when it receives a "TCP connection established" event from the client state machine, and moves to the *Page Loading* state.

The time spent in this state, t_{select}, is the time required for the client and server to establish a TCP connection. The transition between this state and the next state, *Page Loading*, is event-driven. The event causing the transition depends on the TCP setup time required by the client and the server, described in sections 2.3.2 and 2.3.3, respectively.

Page Loading: In this state, the user requests a document or web page from the web server it selected in the previous state. The user then waits for this document or for an error message to appear in the browser window.

When the user diagram enters this state, it sends a "get page" event to the client state diagram. The user moves to the next state either when the page finishes loading or when the user grows too impatient to allow the page to finish loading, and aborts the transaction. In the former case, the next state is the *View* state, and the state transition is event-driven. In the latter case, the next state is the *Stop* state, and the transition is timer-driven.

The time spent in this state, t_{request}, is given by

$$t_{\text{request}} = \min(t_{\text{retrieve}}, t_{\text{timeout,user}})$$

where t_{retrieve} is the time required for the browser to retrieve and display a web page and $t_{\text{timeout,user}}$ is a timeout value representing the user's "patience level". The value of t_{retrieve} depends on timer values from the client and server state machines, while $t_{\text{timeout,user}}$ is a measured random variable.

Stop: The user enters this state when it aborts the transfer of a web page or document before the document finishes loading in the browser window. A transition to this state occurs when the user is in the *Page Loading* state and $t_{\text{request}} = t_{\text{timeout,user}}$. In this state, the client and server break the TCP connection between them. The user sends an “abort” event to the *Wait 1*, *Receive*, and *Parse* states in the client state diagram. The user can reconnect to the same or another web server immediately, in which case it moves to the *Connect to Server* state. Otherwise, the user moves to the *View* state. The former case occurs if, for example, the user selects a new URL before the present document finishes loading; the latter case occurs when the user presses the “stop” button on the browser. In either case, the transition to the next state results from the reception of a “TCP disconnected” event from the client state diagram.

The time spent in this state, t_{stop}, is dependent on the time required for the client and the server to close the TCP connection between them. Thus, the transition from this state to either of the next two possible states is event-driven.

View: The user enters this state after the page finishes loading, or after the user manually stops the page loading, in the browser window. The user enters this state from either the *Page Loading* or *Stop* states.

There are three possible transitions from this state. If the user requests another page from the same server, the next state is the *Page Loading* state, and the user diagram sends a “get next page” event to the client diagram. If the user requests another page from a different server, the next state is the *Connect to Server* state, and the user diagram sends a “new server” event to the client diagram. The third possibility is that the user decides to end the web session; in this case, the next state is the *Close Connection* state, and the user diagram sends a "end session" event to the client diagram. All three transitions are timer-driven and occur with some nonzero probability.

The time spent in this state, t_{view}, is the time between user actions within a web session, where the three possible user actions are defined in the previous paragraph. t_{view} is a measured random variable. It is dependent to some extent on the size of the web document and also on the user.

Close Browser Connection: This stage is the first of two stages in which the user closes the browser and ends the web session. In this stage, the browser closes its connection to the web server. The user arrives in this state from the *View* state and transitions to the *Close Application* state once the TCP connection between the client and server is broken, signified by the receipt of a "TCP disconnected" event from the client diagram. Thus, the transition to the next state is event-driven.

The variable t_{close1} describes the time spent in this state. Its value is dependent on the time needed for the client and server to close the TCP connection, which is defined in sections 2.3.2 and 2.3.3 .

Close Browser Application: This is the second of two stages in which the user closes the browser and ends the web session. This stage occurs after the client and server break the TCP connection between them. In this state, the browser software closes. From this state, the user returns to the *Closed* state.

The transition from this state to the *Closed* state is timer-driven. The time value associated with this state, t_{close2}, is a function of the browser software, operating system, and processor speed, and is a measured quantity.

2.3.2 Client State Diagram

The client is the liaison between the user and the server. The client handles the details of connecting to the server and retrieving the files in the requested web page. (Recall that a web document typically consists of one or more files.) The client also parses these files and displays them in a viewable format. Figure 2.3 illustrates the client portion of the state diagram.

We assume a one-to-one relationship between the user and the client in our analysis. We realize that some browsers are associated with a particular computer or IP address and are shared among a pool of users; we ignore this for now to simplify our analysis.

Idle: The *Idle* state represents the time in between TCP connections to a server by a particular web user, either within or in between web sessions. In this state, the client does not maintain a connection to any web server. The client leaves this

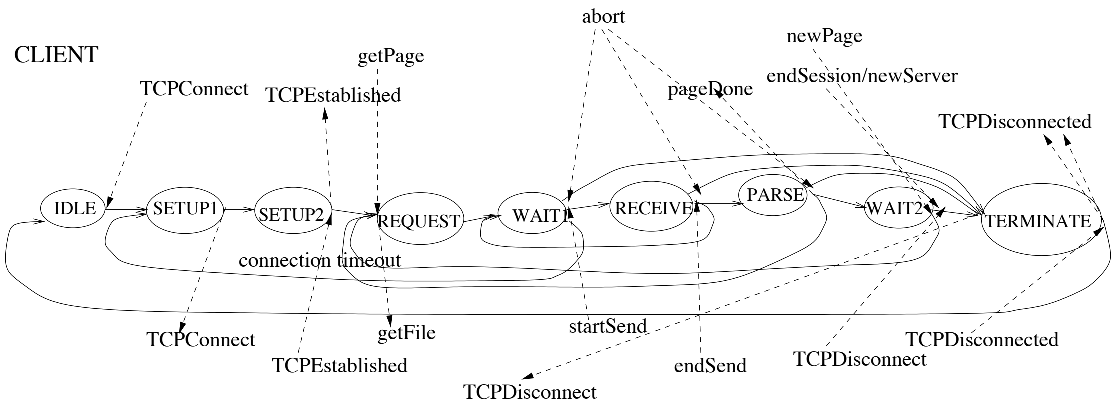

Figure 2.3: The client portion of the state diagram. Events in the top half of the diagram correspond to the user diagram, while those in the bottom half correspond to the server diagram.

state upon receipt of a "TCP connect" event from the user diagram, requesting a TCP connection to a particular web server. This event is sent when the user state diagram transitions to the *Connect to Server* state. The client then transitions to the *Setup 1* state.

The time spent in this state, t_{idle}, represents the time between TCP connections by a single user. This time could be either within a web session or between distinct web sessions for a single user. It is a slave to the user state machine.

Setup 1: This is the first of two states in which the client establishes a TCP connection with a web server. To setup a TCP connection, the client and server exchange a "three-way handshake". The three-way handshake is described in Section 1.1.1.

In this first setup state, the client sends the initial SYN packet to the server. The client immediately transitions to the *Setup 2* state. This transition is completely timer-driven.

The time spent in the first state represents the initial processing by the client to set up the connection. This time is given by

$$t_{\text{setup1}} = t_{\text{trans, client}}$$

where $t_{\text{trans, client}}$ is the transmission time required for a SYN packet and is given by

$$t_{\text{trans, client}} = \frac{20}{r_c}$$

Here, 20 is the size of a SYN packet in bytes and r_c is the source rate of the client in bytes/sec. We assume that r_c is a known quantity. Thus, t_{setup1} is dependent only on the source rate of the client.

Setup 2: This is the second of two states in which the client and server establish a TCP connection. In this stage, the client and server exchange the SYN/ACK and ACK messages. The client leaves this state after it receives the final ACK message from the server. The client diagram receives a "TCP connection established" event from the server diagram at this point and moves to the *Request* state. Upon this transition, the client diagram also sends a "TCP connection established" event to the user diagram.

The time spent in this state, t_{setup2}, is determined completely by the time spent in the *Setup* state in the server state diagram.

Request: The next series of states represents the actions required in retrieving a web page from a server. The *Request* state is the first of these stages. In this state, the client requests a file from the web server. The file makes up part or all of the web page or document requested by the user. The client diagram sends a "get file" event to the server at the transition into this state. The client leaves this state once it finishes sending the request and moves to the *Wait 1* state.

The time spent in this state is the time required for the client to send the file request to the server. This time is given by the random variable

$$t_{\text{request}} = t_{\text{trans, client}} + t_{\text{delay}}$$

where

- $t_{\text{trans,client}} = \frac{k_r}{r_c}$;
- k_r is the size of the request message in bytes, and is a measured random variable;[2]

[2]Studies such as [36] suggest that the distribution of this variable is bimodal; we assume in our preliminary studies that the request size k_r is a constant value

- t_{delay} is the time required for the message and the corresponding ACKs (and retransmissions) to travel through the TCP network connecting the client and server.

Wait 1: This is the second of the page-retrieval states. In this state, the client waits for the server to process its request.

There are three possible transitions from this state. If the user grows impatient and sends an "abort" event to the client, the client transitions to the *Terminate* state. If the server does not send any response within a specified timeout value, the client assumes that the TCP connection to the server has been broken. In this case, the client moves to the *Setup 1* state and attempts to reestablish the TCP connection with the server. If, however, the server sends a response to the client's request, the client diagram receives a "start response" event from the server diagram and as a result moves to the *Receive* state. In the first and third cases, the transition is event-driven; in the second case, it is timer-driven.

The time spent in this state, t_{wait1}, is given by

$$t_{\text{wait1}} = \min(t_{\text{timeout,user}}, t_{\text{timeout,client}}, t_{\text{response}})$$

where $t_{\text{timeout,user}}$ is the timeout value associated with the "abort" event from the user; $t_{\text{timeout,client}}$ is the no-response timeout value; and t_{response} is the time between the transition to the *Wait 1* state and the receipt of the "start response" event from the server.

Receive: In this state, the client receives the file or an error message from the server and begins to parse the file. The client remains in this state until it receives either an "abort" event from the user or a "response finished" event from the server. In the former case, the next state is the *Terminate* state; in the latter case, the next state is either the *Wait 1* state (if there is a pipelined request[3] pending) or the *Parse* state otherwise.

The time spent in this state, t_{receive}, is a random variable representing the time period between the first packet the client receives from the server and the last packet received from the server corresponding to the requested file:

$$t_{\text{receive}} = \min(t_{\text{send}}, t_{\text{timeout,user}})$$

where t_{send} is the time required for the server to send the file over the network to the client; it is a function of the TCP network dynamics and the file size.

Parse: In this state, the client, after receiving a file or error message from the server, parses this file to determine if other files need to be retrieved from that server or from other servers, and makes subsequent file requests accordingly. File parsing actually begins in the previous state; the time spent in this state is

[3]Pipelined requests are defined in Section 1.1.1.

the residual time required to parse the file after the transport time is taken into account.

The client can transition from this state to one of three states. If the user sends an “abort” event while the client is in this state, the next state is the *Terminate* state. If the client successfully parses the file, it transitions either to the *Wait 2* state if no further files need to be retrieved for that page, or to the *Request* state if there is another file that needs to be retrieved from the server within that document.

The client spends an amount of time in this state specified by the minimum of the two variables t_{parse} and $t_{\text{timeout,user}}$. t_{parse} is a function of both the browser/operating system speed and the size of the file or error message.

Wait 2: The client enters this state after it finishes parsing and displaying the page or document. The client then waits for the next request or action from the user.

From this state, the client transitions to one of two states, both of which result from events received from the user. If the user sends a “new server” or “end session” event, the next state is the *Terminate* state. If the user sends a “get next page” event, the client moves to the *Request* state. The time spent in this state, t_{wait}, is completely dependent on the user state machine.

Terminate: In the final state in the client state diagram, the client closes the TCP connection with the current web server. Either the client or the server initiates this TCP close. From this state, the client enters the *Idle* state after sending a "TCP disconnect" even to the server diagram and receiving a "TCP disconnected" event from the server diagram.

To close a TCP connection, the client and server execute a three-way handshake, similar to the one used to establish the TCP connection. This process is described in Section 1.1.1.

The time associated with this state is given by

$$t_{\text{terminate}} = 2t_{\text{trans, server}} + 2t_{\text{trans, client}} + t_{\text{delay}} + t_{\text{process, TCP, end}}$$

where

- $t_{\text{trans, client}} = \frac{k}{r_c}$;
- $t_{\text{trans, server}} = \frac{k}{r_s}$;
- k is the size of the FIN or ACK packet (20 bytes);
- r_c is the source rate of the client in bytes/sec and r_s is the source rate of the server in bytes/sec;
- t_{delay} is a function of the TCP network dynamics;

- $t_{\text{process, TCP, end}}$ is the overhead time in seconds required to close a TCP connection. (This quantity may be negligible.)

Since this event may originate in either the client or the server, we assume that the *Terminate* states in both the client and server state diagrams are identical and occur simultaneously.

2.3.3 Web Server State Diagram

In a web session, the server interacts with the user via the client. The server state diagram, in Figure 2.4, is the simplest of the three state diagrams in that it contains the fewest number of states and state transitions.

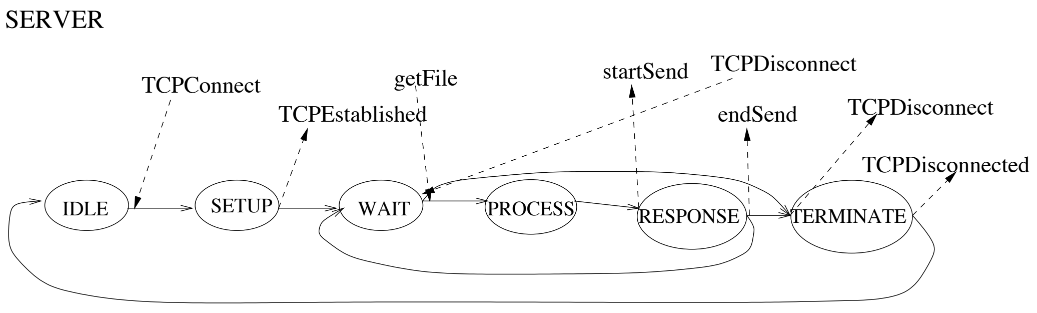

Figure 2.4: The server state diagram. Events correspond to the client diagram.

Idle: The *Idle* state represents the time in between TCP connections by the same user (client) to the server. As in the client state diagram, this time represents either the inter-session time or the intra-session time between separate TCP connections from the same client. From this state, the server moves to the *Setup* state once it receives a "TCP connect" event from the client.

The time spent in this state, t_{idle}, is dependent on both the client and user state diagrams.

Setup: In this state, the server responds to a request for a TCP connection from a client. The server establishes a TCP connection with the client, according to the three-way handshake described in Section 1.1.1, as long as resources are available for such a connection. Once the server completes its portion of the connection establishment process, the server diagram sends a "TCP connection established" event to the client diagram and transitions to the *Wait* state.

The time spent in this state is the time required for parts two and three of this three-way handshake:

$$t_{\text{setup, server}} = t_{\text{trans, client}} + t_{\text{trans, server}} + t_{\text{delay}} + t_{\text{process, TCP}}$$

where $t_{\text{process, TCP}}$ is the overhead time in seconds required to establish a TCP connection.

Wait: In this state, the server waits for the next action from the client after establishing the TCP connection with the client. Client actions include requesting a file and closing the TCP connection. From this state, the server either moves to the *Process* state (if it receives a "get file" event from the client) or to the *Terminate* state (if it receives a "disconnect" event from the client).

The waiting time, t_{wait}, is a measured random variable, and depends on the action taken by the client. t_{wait} is either the time between file requests within a web page, the time between web page requests, or the time between the last request and the request to close the TCP connection. The upper bound on this waiting time is the time an inactive TCP connection is allowed to remain open. (Values for this timeout are defined in the TCP specifications [53].)

Process: In this state, the server responds to a request for a file from a client. The server searches for the file in its memory stores, retrieves the file, attaches an appropriate HTTP header, and places it in its send queue. If the file can not be located, or an error occurs such that the server cannot service the request, the server composes a HTTP header containing the error message and places this header in its send queue. From this state, the server transitions to the *Send Response* state.

The time spent in this state is given by:

$$t_{\text{process}} = t_{\text{queue}} + t_{\text{retrieve}} + t_{\text{overhead}}$$

where

- t_{queue} is the time a request spends waiting at the server for service, and is a function of server load and server processing time;
- t_{retrieve} is the time required to search for, locate, and retrieve the requested file; or, if the file cannot be located, the time to assemble an appropriate error message. This time is a function of the requested file size, processor speed, and the operating system of the server;
- t_{overhead} includes the time required to attach a header to the file or message and to prepare the message to be transported over the network (breaking the message into packets, etc.). It is a function of the requested file size and the operating system.

Send Response: In this state, the server sends the file or error message assembled in the previous state (*Process*) over the network to the client. When the server transitions into this state, it sends a "start response" event to the client. Upon completion, the server sends a "response finished" event to the client and moves to either the *Terminate*[4] state or the *Wait* state.

[4]The HTTP/1.1 specifications [22] leave the termination of a TCP connection between the client and server up to the implementor. Either entity has the ability to explicitly close the connection by including a "close" token in the header of a request or response message. If the client includes this token, the server will close the TCP connection after sending the entire response. If the server includes this token, it will close the connection at the end of the response it is currently sending.

The time spent in this state is the time needed to transport the file or error message to the client:

$$t_{\text{send}} = t_{\text{trans, server}} + t_{\text{delay}}$$

where $t_{\text{trans, server}}$ is defined as above, with k a random variable representing the file size in bytes.

Terminate: In the final state in the server state diagram, the server closes the TCP connection with the client. The close may be initiated by either the client or the server, as described in Section 2.3.2. The server moves to this state upon receipt of a "disconnect" event from the client, or from the *Send Response* state. When leaving this state, it sends a "TCP disconnected" event to the client and goes to the *Idle* state.

The time spent in this state, $t_{\text{terminate}}$, is defined in the previous section under the *Terminate* state description.

2.4 Summary

A summary of the state diagrams in tabular form is listed in the appendix. Table A.1 lists the states for each of the three state diagrams, along with a brief description of each state. Table A.2 lists the event names associated with each of the state diagrams, and Table A.3 lists the timer values.

The transitions within the state diagrams and the interrelations among them are combined in Tables A.4 through A.6, also in the appendix. The information in these tables represents the same information contained within the state diagrams, but in a format that is more like “pseudocode” and thus more easily implementable into a computer model.

Chapter 3

OPTIMAL SERVICE ORDERING OF A WWW SERVER

3.1 Introduction

We wish to determine an optimal service ordering policy for a web server processing requests for files from remote clients. We define our system as the combination of a WWW server and the HTTP requests for files, or *jobs*, coming in from remote clients (users running WWW browser software).

The web server wants to process the requests in the best way possible. In particular, the server wants 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.[1] 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.

We define *revenue* as a measure of the perceived performance per user per file. The user "pays" the server an amount relative to the download time upon the server's completion of the user's request.[2]

The server's objective is to maximize the average revenue it collects from all jobs served during a busy cycle. Busy cycles are time periods in which there is at least one job in the system, either in service or awaiting service. Idle cycles, by contrast, are time periods in which the system is empty. In our system, busy and idle cycles repeat.

[1]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.

[2]Here, we ignore the transit time required to send the result of the request back to the browser, and assume that the revenue is collected instantaneously upon the file's departure from the server.

3.2 System Model

Assume that we have a system where jobs arrive according to a Poisson process. The service times[3] for all jobs in the system are independent and identically distributed exponential random variables with parameter μ. Once a job has entered the system, it does not leave until it completes service (according to its service time). The time it takes the server to switch between jobs in service is negligible, so that we can assume that swap times are instantaneous.

Let state $\emptyset$ represent the "null" state, where no jobs await service. When the system is in state $\emptyset$, we say that it is in an *idle* cycle. The beginning of a busy cycle occurs when a job arrives at a server in state $\emptyset$. Let $[Y_j, Z_j]$ denote the jth *busy cycle*, where Z_j marks the start of the jth idle cycle. Then busy cycles are i.i.d., idle cycles are i.i.d., and busy and idle cycles are independent.

Jobs are numbered in the order that they arrive at the system. Let x_i be the arrival time of job i at the server and t_{i_p} be the departure time of job i under some policy p. Then job i has a *response time* under a policy p equal to $t_{i_p} - x_i$, where we define response time as the total time spent in the system, both waiting for service and in service.

[3]In actuality, we assume that the sizes of the files requested by the user are independent and identically distributed random variables. However, we assume that the file size and service time are related by a constant multiplier; thus, the service times are also exponential. For the remainder of the discussion, we refer to "service times" rather than "file sizes"

Suppose that the server earns an amount of revenue from serving job i under policy p equal to $g_{i_p}(t_{i_p}) = e^{-c(t_{i_p}-x_i)}$. The server collects this amount once job i departs the system. Let N denote the number of jobs served in the first busy cycle (i.e., the number served in the interval $[Y_1, Z_1]$. By the Renewal Reward Theorem [54], the average reward per unit time earned by the server under policy p is

$$V_p(\emptyset) = \frac{E\left[\sum_{i=1}^{N} g_{i_p}(t_{i_p})\right]}{E[Z_1]} \tag{3.1}$$

The optimal server policy satisfies

$$V = \max_p V_p(\emptyset) \tag{3.2}$$

At any time t prior to t_{i_p}, we define the *potential revenue function* for job i in the system as $g_{i_{\text{pot}}}(t) = e^{-c(t-x_i)}$. This is the amount of revenue the server would earn from job i if it departed the system at t, given that i is still in the system at t.

In addition, we assume that the distribution over the first busy/idle cycle is identical to those of later cycles; so it suffices to consider just the first busy/idle cycle in our analysis.

Finally, we define $\mathcal{S}$ to be the set of policies under consideration. In this discussion, the set of policies includes all preemptive and non-preemptive policies,

policies that work on one request at a time, policies that work on multiple requests simultaneously, and both idling and non-idling policies.

3.3 Optimal Service Ordering Policy Requirements

Given the above assumptions, we can determine an optimal service ordering from the set of policies in $\mathcal{S}$. Such a policy will satisfy the following criteria.

3.3.1 Non-idling Policies

Lemma 3.3.1 *An optimal policy can be found among the class of non-idling policies*

Proof: Suppose there exists a policy p which is optimal but not non-idling over a sample path in the first busy cycle. Define $[a_1, a_2)$ to be a time interval in this sample path in which the server remains idle under policy p during a busy cycle (while jobs are queued). At time a_2, the server switches from idle to busy, and works on some job $j*$. Define a_3 as the next completion time during this busy period, where $a_3 > a_2$.

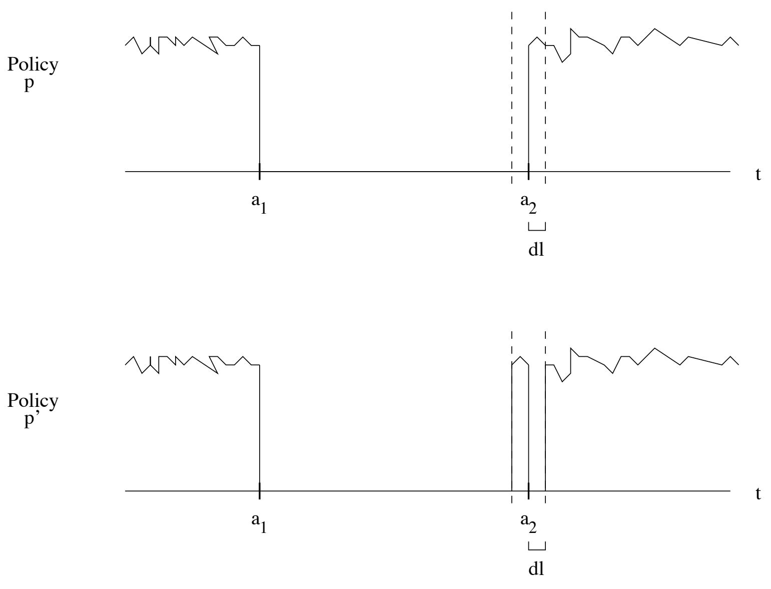

Figure 3.1: Illustration of policies p and p' for proof of Lemma (3.3.1).

We can construct a policy p' that is identical to p except over the interval $[a_1, a_3)$. Define dl to be an infinitesimal time interval, where $dl << 1$. We can then define policy p' as the policy that makes a "swap" of two intervals of length dl from policy p. In particular, policy p' serves job $j*$ in the interval $[a_2 - dl, a_2)$, remains idle in the interval $[a_2, a_2 + dl)$, and then returns to the service order defined in policy p at time $a_2 + dl$, and continues following policy p until the end of the busy cycle. Figure 3.1 illustrates the original policy and the new policy.

We define the optimal policy as one that maximizes the average revenue per unit time per cycle, as given by (3.1). Thus, under policy p,

$$V_p(\emptyset) = \frac{E\left[\sum_{i=1}^{N} g_{i_p}(t_{i_p})\right]}{E[Z_1]}$$

and under policy p',

$$V_{p'}(\emptyset) = \frac{E\left[\sum_{i=1}^{N} g_{i_{p'}}(t_{i_{p'}})\right]}{E[Z_1]}$$

Since the expected value of the duration of the busy/idle cycle is the same under both policies, we can drop the denominator from our analysis and just consider the numerator in each expression.

The difference in the expected revenue under the two policies is

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] = E[g_{j*_{p'}}(a_2)] - E[g_{j*_p}(a_2 + dl)]$$

dl is small enough so that at most one departure can occur in either $[a_2 - dl, a_2)$ or $[a_2, a_2 + dl)$. The probability of job $j*$ departing within the interval $[0, dl)$ is $\mu\, dl + o(dl) \approx \mu\, dl > 0$. Thus,

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, [g_{j*_{p'}}(a_2)] - \mu\, dl\, [g_{j*_p}(a_2 + dl)] \\ &= \mu\, dl\, [e^{-c(a_2 - x_{j*})} - e^{-c(a_2 + dl - x_{j*})}] \\ &= \mu\, dl\, e^{-c(a_2 - x_{j*})}(1 - e^{-c\, dl}) \qquad (3.3) \end{aligned}$$

The last term in the above equation is positive, so we conclude that

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

and therefore

$$V_{p'}(\emptyset) > V_p(\emptyset)$$

Thus, with some nonzero probability, policy p' will earn more than policy p, so p cannot be an optimal policy according to our definition of optimal.

3.3.2 Piecewise-constant Policies

Lemma 3.3.2 *An optimal service ordering can be found in the class of policies that switch between jobs in service only upon an arrival to or departure from the system.*

Proof: Suppose there exists an optimal policy p which is not piecewise-constant over some interval of a sample path in the first busy cycle. At some time interval $[a_1 - dl, a_1)$ during this busy cycle, where dl is defined in the proof of Lemma 3.3.1, the server works on job $j*$. At time a_1, the server switches to job $(j+1)*$, and processes that job for at least the interval $[a_1, a_1 + dl)$. We define a_1 to be a time at which no arrivals to or departures from the system occur.[4] At some future time a_2, the server switches back to serving job $j*$. We make no assumptions as to the jobs served in the interval $[a_1 + dl, a_2)$.

We construct a policy p' which is identical to policy p in the intervals $[0, a_1 - dl)$, $[a_1 + dl, a_2)$, and $[a_2 + dl, \infty)$. Our construction of p' depends on the potential revenues of jobs $j*$ and $(j+1)*$ at time a_1. Recall that the potential revenue of a

[4]This definition of a_1 is necessary to yield a policy that is not piecewise-constant.

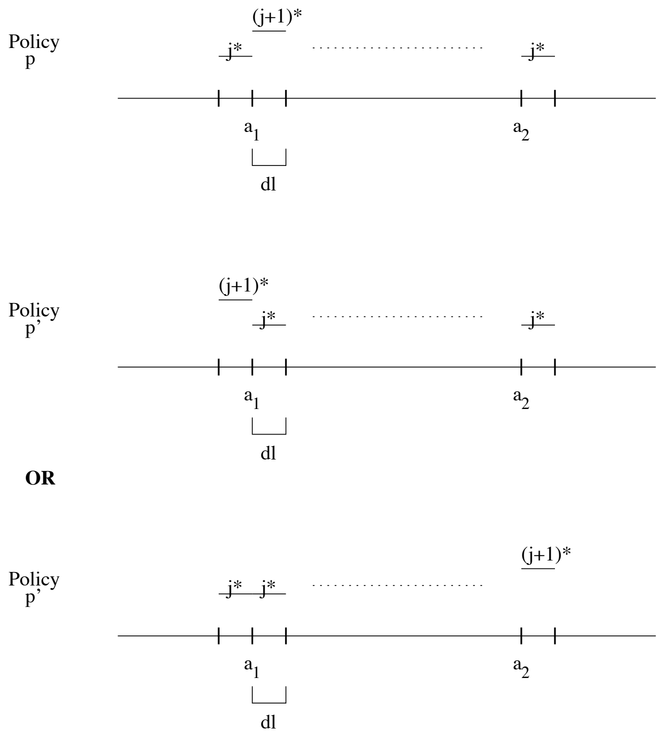

Figure 3.2: Policy p and the two alternate p' policies used in the proof of Lemma 3.3.2.

job i at time t, or the revenue the server would generate from serving that job if it left the system at t, is given by $g_{i_{\text{pot}}}(t) = e^{-c(t-x_i)}$. Thus, the potential revenue of job $j*$ at a_1 is $g_{j*_{\text{pot}}}(a_1) = e^{-c(a_1 - x_{j*})}$ and the potential revenue of job $(j+1)*$ is $g_{(j+1)*_{\text{pot}}}(a_1) = e^{-c(a_1 - x_{(j+1)*})}$. Figure 3.2 illustrates the original policy and the two alternate policies which will be described below.

If $g_{(j+1)*_{\text{pot}}}(a_1) > g_{j*_{\text{pot}}}(a_1)$, then we construct p' as follows: In the interval $[a_1 - dl, a_1)$, serve job $(j+1)*$. At a_1, switch to job $j*$ and serve $j*$ in the interval $[a_1, a_1 + dl)$. Then, serve $j*$ in the interval $[a_2, a_2 + dl)$. In this case, p' differs from

p only over $[a_1 - dl, a_1 + dl)$.

Again, we ignore the denominator of each term in Equation (3.1) in our analysis for the reasons cited in the proof of Lemma 3.3.1.

The difference in the expected revenue under the two policies is

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= E[g_{(j+1)*}(a_1) + g_{j*}(a_1 + dl)] \\ &\quad - E[g_{j*}(a_1) + g_{(j+1)*}(a_1 + dl)] \end{aligned}$$

We know that there is no departure at time a_1 under policy p by our definition, but there may be a departure at a_1 under policy p'. Also, because the server processes job $j*$ again in the interval $[a_2, a_2 + dl)$, we know that job $j*$ cannot depart the system prior to time $a_2 + dl$; thus, the expected revenue the server generates from job $j*$ in the interval $[a_1, a_1 + dl)$ under policy p' is zero. Recall that dl is small enough so that at most one departure can occur in any time interval of length dl, with probability approximately equal to $\mu\, dl > 0$. Thus,

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, [g_{(j+1)*}(a_1)] - \mu\, dl\, [g_{(j+1)*}(a_1 + dl)] \\ &= \mu\, dl\, e^{-c(a_1 - x_{(j+1)*})}(1 - e^{-c\,dl}) \end{aligned}$$

The last term in the above equation is positive. Therefore, for this particular definition of p', we conclude that

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

If, however, $g_{(j+1)*_{\text{pot}}}(a_1) < g_{j*_{\text{pot}}}(a_1)$, then we construct p' such that p' serves job $j*$ in the interval $[a_1 - dl, a_1 + dl)$ and serves job $(j+1)*$ in the interval $[a_2, a_2 + dl)$. Here, p' differs from p over $[a_1, a_1 + dl)$ and $[a_2, a_2 + dl)$.

The difference in the expected revenue under the two policies is

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= E[g_{j*}(a_1 + dl) + g_{(j+1)*}(a_2 + dl)] \\ &\quad - E[g_{(j+1)*}(a_1 + dl) + g_{j*}(a_2 + dl)] \end{aligned}$$

Again, there is no departure at time a_1 under policy p, but there may be a departure at either $a_1 + dl$ or $a_2 + dl$ (or both) under policy p'. The expected revenue is thus

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, g_{j*}(a_1 + dl) + \mu\, dl\, g_{(j+1)*}(a_2 + dl) \\ &\quad - \mu\, dl\, g_{(j+1)*}(a_1 + dl) - \mu\, dl\, g_{j*}(a_2 + dl) \\ &= \mu\, dl\; e^{-c(dl - x_{j*})}(e^{-ca_1} - e^{-ca_2}) \\ &\quad + \mu\, dl\, e^{-c(dl - x_{(j+1)*})}(e^{-ca_2} - e^{-ca_1}) \\ &= \mu\, dl\, e^{-c\,dl}(e^{-ca_1} - e^{-ca_2})(e^{cx_{j*}} - e^{cx_{(j+1)*}}) \end{aligned}$$

Since $g_{(j+1)*_{\text{pot}}}(a_1) < g_{j*_{\text{pot}}}(a_1)$, we know that x_{j*} must be greater than $x_{(j+1)*}$. Therefore, the last term in the above equation is positive, and

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

Thus, for each policy p', we have shown that with some nonzero probability,

$$V_{p'}(\emptyset) > V_p(\emptyset)$$

and therefore p cannot be an optimal policy.

3.3.3 Non-processor-sharing Policies

Lemma 3.3.3 *There is no policy outside of the class of non-processor-sharing (NPS) policies that generates a strictly higher revenue than the best policy from the NPS class of policies.*

Proof: Suppose there exists a generalized processor-sharing (PS) policy that generates a higher revenue than the best NPS policy. Then, over some interval $[a, b]$, the server splits its resources between at least two jobs.

We consider the case where the processor shares resources between two jobs. The n-job case is a natural extension of this case, since the service times of all n jobs are independent.

Prior to time a, job 1 has spent an amount of time equal to τ_1 in the system, and job 2 has spent a time equal to τ_2 in the system. After time a, the distribution for the total revenue that the server earns from serving the two jobs under policy

p is

$$\begin{aligned} g_p(t) &= g_{1_p}(t) + g_{2_p}(t) \\ &= c_f e^{-cT_f} + c_s e^{-cT_s} \end{aligned}$$

where

- T_f is the time at which the first job to complete service departs the system, with respect to time a, given by

  $$T_f = \min(T_1', T_2')$$

  with T_1' and T_2' defined as the remaining service times of jobs 1 and 2, respectively;

  $$T_f \sim \exp(q\mu + (1-q)\mu) \sim \exp(\mu);$$

  and q defined as the probability that job 1 departs the system first.

- T_s is the time at which the remaining job departs the system, with respect to time a, with a distribution

  $$T_s = T_f + Z$$

  where Z is an exponentially-distributed random variable with parameter μ.

- c_f is the previous revenue function decay of the first job to depart, and is given by the distribution

$$c_f = \begin{cases} e^{-c\tau_1} \triangleq c_1, & T_f = T_1' \\ e^{-c\tau_2} \triangleq c_2, & T_f = T_2' \end{cases}$$

- c_s is the previous revenue function decay of the second job to depart, and is given by the distribution

$$c_s = \begin{cases} e^{-c\tau_1} \triangleq c_1, & T_f = T_2' \\ e^{-c\tau_2} \triangleq c_2, & T_f = T_1' \end{cases}$$

The expected value of the revenue gained over this portion of the sample path is

$$E[g_p(t)] = \int_0^\infty \int_0^\infty (c_f e^{-ct_f} + c_s e^{-ct_s}) f_{T_f,T_s}(t_f, t_s)\, dt_f\, dt_s$$

where $f_{T_f,T_s}(t_f, t_s)$ is the joint distribution of the service times of the two jobs, given by

$$f_{T_f,T_s}(t_f, t_s) = \mu^2 e^{-\mu(t_s - t_f)} e^{-\mu t_f} \tag{3.4}$$

We make the substitution $z = t_s - t_f$ and proceed:

$$\begin{aligned} E[g_p(t)] &= \int_0^\infty \int_0^\infty (c_f e^{-ct_f} + c_s e^{-c(t_f+z)}) \mu^2 e^{-\mu t_f} e^{-\mu z}\, dt_f\, dz \\ &= \tfrac{\mu}{c+\mu} c_f + \left(\tfrac{\mu}{c+\mu}\right)^2 c_s \end{aligned}$$

There are two possible values for c_f and c_s, depending on which job departs the system first. Thus, if job 1 departs the system first with probability q, the expected revenue from the PS policy is

$$E[g_p(t)] = q\left[\tfrac{\mu}{c+\mu}c_1 + \left(\tfrac{\mu}{c+\mu}\right)^2 c_2\right] + (1-q)\left[\tfrac{\mu}{c+\mu}c_2 + \left(\tfrac{\mu}{c+\mu}\right)^2 c_1\right] \tag{3.5}$$

If, instead, the server chooses to work on one job first and then the other, there are two possible policies. Define policy NPS1 as the policy that serves job 1 and then serves job 2, and policy NPS2 as the policy that serves job 2 first and then serves job 1. Prior to time a, each job has been in the system for a time τ_1 or τ_2, respectively. After time a, the remaining service time required by job 1 is t_1' and the remaining service time required by job 2 is t_2'.

Under NPS1, the total potential revenue is

$$g_{\text{NPS1}}(t) = e^{-c(\tau_1+t_1')} + e^{-c(\tau_2+t_1'+t_2')} = c_1e^{-ct_1'} + c_2e^{-c(t_1'+t_2')}$$

and under NPS2 it is

$$g_{\text{NPS2}}(t) = e^{-c(\tau_1+t_1'+t_2')} + e^{-c(\tau_2+t_2')} = c_1e^{-c(t_1'+t_2')} + c_2e^{-ct_2'}$$

The expected revenue under policy NPS1 over this interval is

$$\begin{aligned} E[g_{\text{NPS1}}(t)] &= \int_0^\infty\int_0^\infty e^{-ct_1'}(c_1 + c_2e^{-ct_2'})\mu^2e^{-\mu t_1'}e^{-\mu t_2'}\,dt_1'\,dt_2' \\ &= \tfrac{\mu}{c+\mu}c_1 + \left(\tfrac{\mu}{c+\mu}\right)^2 c_2 \end{aligned} \tag{3.6}$$

and under policy NPS2, the expected revenue is

$$\begin{aligned} E[g_{\text{NPS2}}(t)] &= \int_0^\infty \int_0^\infty e^{-ct_2'}(c_1 e^{-ct_1'} + c_2)\mu^2 e^{-\mu t_1'} e^{-\mu t_2'}\, dt_1'\, dt_2' \\ &= \left(\tfrac{\mu}{c+\mu}\right)^2 c_1 + \tfrac{\mu}{c+\mu} c_2 \qquad (3.7) \end{aligned}$$

But (3.5) is just a weighted sum of (3.6) and (3.7). Therefore, (3.5) cannot be greater than both (3.6) and (3.7). Thus, the PS policy cannot earn a strictly higher revenue than both policy NPS1 and policy NPS2.

The above analysis assumes that the percentage of resources the processor expends on the two jobs in service under the PS scheme is fixed. We can show that the result obtained for the fixed probability case also holds when the server expends a varying percentage of resources on the two jobs in service. Regardless of the individual probabilities at any time over the interval in which processor sharing is employed, one of the two jobs will still complete first with probability q'. The weighted sum is then represented by (3.5), with q replaced by q'. But this is also just a weighted sum of (3.6) and (3.7); just as in the fixed probability case, the PS policy cannot earn a strictly higher revenue than both NPS1 and NPS2.

3.3.4 Markov Policies

Lemma 3.3.4 *An optimal policy can be found among the class of Markov policies.*

Proof: If not, then there exists an optimal policy p that relies on all past history of the system to make its decisions as to which job to serve next.

Let G_{t_p} denote the state vector of the system at time t under policy p, where

$$G_{t_p} = (g_{1_p}(t)\ g_{2_p}(t)\ \ldots\ g_{M_p}(t))$$

M denotes the number of jobs that have arrived at the system since time $t = 0$, and $g_{i_p}(t)$ is given by the equation

$$g_{i_p}(t) = \begin{cases} e^{-c(t_{i_p} - x_i)}, & \text{if job } i \text{ departed prior to } t \\ e^{-c(t - x_i)}\ , & \text{if job } i \text{ is still in the system at } t \end{cases}$$

Here, t_{i_p} is the departure time of job i under policy p and x_i is the arrival time of job i.

The state vector G_{t_p} contains information about the arrival time of the job at the system and, if applicable, the departure time of the job from the system. Since each $g_{i_p}(t)$ is an exponential function, each element of the state vector will decay by the same ratio over each time interval, until the job departs the system, at which point its revenue function ceases to decay. Thus, each $g_{i_p}(t)$ for the jobs in the system at time t "looks" the same over an interval regardless of whether or not the server processes that job over the interval.

From Lemmas 3.3.1 and 3.3.3, we know that in each interval in which there is at least one job in the system, the server will select one job to process in that interval. The server bases its selection on its determination of which job will maximize its total revenue. It makes its selection from the pool of jobs that have not completed service (and are still in the system) at time t. Since jobs that have already departed do not affect the service selection, we can assume without loss of generality that these jobs depart the state vector as well upon departure from the system.

Because service times are memoryless, the probability of a job i completing its service in the interval $[t, t + dt)$, given that it is still in the system at time t and has already received an amount of service z_i, is

$$\begin{aligned} P(t_i < z_i + dt \mid z_i) &= P(t_i - z_i < dt) \\ &= 1 - e^{-\mu t_i'} \end{aligned}$$

where $t_i' = t_i - z_i$. This is identical to the distribution of the job's original service time. Thus, knowledge of how much service a job has already attained prior to t will not yield any new information as to the chances of that job departing prior to $t + dt$, given that the server chooses to process job i in this interval. We can then replace each $g_{i_p}(t)$ in the state vector with the constant $c_{i_p} \overset{\triangle}{=} e^{-c(t-x_i)}$. Doing so removes all knowledge of past history without removing the information necessary to make the decision as to which job to serve. The decisions made in policy p, therefore, are the same decisions that would be made if only the present state of the system is known.

Corollary 3.3.4.1 *An optimal policy over the interval $[t, x_N)$ can be found among the class of policies that are independent of future arrivals to the system.*

Proof: The proof to this corollary is similar to the proof of Lemma 3.3.4. From this lemma, we know that the state vector G_{t_p} contains information about the arrival times of jobs already in the system at time t. Based on this information, the server selects a job to process; the job selected is determined based on its constant c_i, a function of the arrival time of the job.

If this corollary does not hold, then there exists an optimal policy p that relies on future arrival times to make its decisions as to which job to serve. But arrivals to our system are memoryless; thus, the server does not have any prior knowledge of the arrival times of future jobs. According to Lemmas 3.3.1 and 3.3.3, the server must work on one job in each time interval in which there is at least one job in the system. Since the server has no knowledge of future arrival times, it must assume that no future arrivals will occur, and choose from the jobs presently in the system based on the state vector values. Thus, p makes decisions that are independent of future arrival times to the system.

3.4 Construction of an Optimal Policy

Since we have shown that an optimal policy satisfies the above lemmas, we can construct an optimal service ordering within these limitations. We make one further assumption about the optimal service ordering before we state what this service ordering is.

Definition 3.4.1 *The **best job** is the job corresponding to the highest value in the state vector at time t.*

From Corollary 3.3.4.1, we know that future arrivals do not affect the service order of an optimal policy. So, we will assume here that no arrivals occur to the system after time t, and at time t there are N jobs in the system.

We also know from Lemma 3.3.4 that an optimal policy will be a Markov policy. Thus, the state vector at time t is $G_t = (c_1\, c_2\, \ldots\, c_N)$, where c_i indicates the maximum revenue the server can obtain from job i from time t onward (or, conversely, the revenue the server would earn from job i if it departed the system at time t).

Because the distribution of remaining service time is identical for every job in the system at time t, the expected revenue gained from serving any one job i

beyond time t with remaining service time t_i' is

$$\begin{aligned} E[g(t_i')] &= E[c_i e^{-ct_i'}] \\ &= c_i E[e^{-ct_i'}] \\ &= \tfrac{\mu}{c+\mu}\, c_i \end{aligned}$$

The job with the highest expected payoff satisfies

$$\max_i \tfrac{\mu}{c+\mu}\, c_i = \max_i\, c_i \text{ , } i \in 1, 2, \ldots, N$$

Thus the job with the highest expected payoff is the job with the largest constant c_i in the state vector G_t. We define this job as the "best first job" to serve starting at time t.

With this definition, we can now state the following theorem about the optimal policy.

Theorem 3.4.1 *An optimal policy is greedy. That is, it chooses the "best first job" to serve, regardless of the order chosen for the other jobs in the system.*

Proof: Assume there exists an optimal policy p that is work-conserving, piecewise-constant, non-processor-sharing, and Markov, but not greedy. That is, at some time t in a sample path during a busy cycle, the server does not choose the best job to serve. Instead, it will choose to serve job $1*_p$ at time t, complete that job, and then serve job $2*_p$ to completion, where $c_{2*_p} > c_{1*p}$.

The state vector at time t is given in the proof of Lemma 3.3.4. We can reorder these values in terms of the order in which the jobs are served under policy p. The state vector then becomes $G_{t_p} = (c_{1*_p}\, c_{2*_p}\, \ldots\, c_{N*_p})$.

The revenue obtained from policy p from t until the end of the current busy cycle is

$$\begin{aligned}\sum_{j=1}^{N} g_{j_p}(t_{j_p}) &= c_{1*_p}e^{-ct'_{1*p}} + c_{2*_p}e^{-c(t'_{1*p}+t'_{2*p})} + \ldots + c_{N*_p}e^{-c(t'_{1*p}+t'_{2*p}+\ldots+t'_{N*p})} \\ &= \sum_{j=1}^{N} c_{j*_p}e^{-c\sum_{k=1}^{j} t'_{k*p}}\end{aligned}$$

where t'_{i*_p} is the remaining service time of the ith job served after time t under policy p.

We construct a policy p' that performs an interchange of jobs $1*_p$ and $2*_p$, such that the server processes job $2*_p$ at time t to completion and then processes job $1*_p$ to completion.

The revenue earned under policy p' from t until the end of the current busy cycle is

$$\sum_{j=1}^{N} g_{j_p}(t_{j_p}) = c_{2*_p}e^{-ct'_{2*p}} + c_{1*_p}e^{-c(t'_{1*p}+t'_{2*p})} + \ldots + c_{N*_p}e^{-c(t'_{1*p}+t'_{2*p}+\ldots+t'_{N*p})}$$

$$= \quad c_{2*_p} e^{-ct'_{2*p}} + c_{1*_p} e^{-c(t'_{1*p}+t'_{2*p})} + \sum_{j=3}^{N} c_{j*_p} e^{-c\sum\limits_{k=1}^{j} t'_{k*p}}$$

The remaining service times for job $1*_p$ and job $2*_p$ are the same under policy p and policy p'. Since service times are identically distributed, we let t'_{1*} denote the service time of the *first* job served after time t in both policy p and policy p' and t'_{2*} denote the service time of the second job served after time t under both policies.

The expected revenue under policy p is thus $E\left[\sum\limits_{j=1}^{N} g_{j_p}(t_{j_p})\right]$ and under policy p' is $E\left[\sum\limits_{j=1}^{N} g_{j_{p'}}(t_{j_{p'}})\right]$.

The difference in revenue between the two policies is

$$\begin{aligned} E\left[\sum_{j=1}^{N} g_{j_{p'}}(t_{j_{p'}})\right] - E\left[\sum_{j=1}^{N} g_{j_p}(t_{j_p})\right] &= E\left[c_{2*_p} e^{-ct'_{1*}} + c_{1*_p} e^{-c(t'_{1*}+t'_{2*})}\right] \\ &\quad - E\left[c_{1*_p} e^{-ct'_{1*}} + c_{2*_p} e^{-c(t'_{1*}+t'_{2*})}\right] \\ &= E\left[c_{2*_p} e^{-ct'_{1*}} + c_{1*_p} e^{-c(t'_{1*}+t'_{2*})}\right. \\ &\quad \left. - c_{1*_p} e^{-ct'_{1*}} - c_{2*_p} e^{-c(t'_{1*}+t'_{2*})}\right] \\ &= E\left[c_{1*_p} e^{-ct'_{1*}}(e^{-ct'_{2*}} - 1)\right. \\ &\quad \left. + c_{2*_p} e^{-ct'_{1*}}(1 - e^{-ct'_{2*}})\right] \\ &= E\left[e^{-ct'_{1*}}(1 - e^{-ct'_{2*}})(c_{2*_p} - c_{1*_p})\right] \end{aligned}$$

But the term inside the expected value operator is always positive, so its expected value will also always be positive. Therefore, since the expected time of the busy cycle is the same under policy p and policy p', we conclude that

$$V_{p'} > V_p$$

Since p' yields a higher average revenue per unit time than p, p cannot be an optimal policy.

We have so far shown that the optimal service ordering must be non-idling, piecewise-constant, non-processor-sharing, Markov, and greedy. For the system we have defined here, with identical cost constants and identical service time distributions, the greedy policy defaults to a preemptive-resume last-in-first-out (LIFO-PR) service policy. To see why this is so, we apply the argument from the proof of Theorem 3.4.1. It is clear to see that the newest job has the highest c_i value in the state vector. The server works on this job until either this job departs or a new job arrives. In the former case, the job with the highest c_i value is the next-newest job, and the server chooses to serve this job; in the latter case, the new arrival has the highest c_i value, and the server chooses to process this job.

Chapter 4

EXTENSIONS TO THE OPTIMAL SERVICE ORDERING ANALYSIS OF A WWW SERVER

4.1 Introduction

We now present several extensions to the service ordering analysis outlined in Chapter 3. These extensions represent different scenarios that may exist at a web server. We first present the case in which the file sizes are distributed as described in the previous chapter and the revenue function is exponential, but with a leading constant multiplier. We then explore the case in which file sizes are independent but not identically distributed. Specifically, we assume that request i is for a file whose size is drawn from an exponential distribution with parameter μ_i. For each extension, we assume the same model as used in the previous chapter, unless otherwise noted.

Note that the service time required to complete an HTTP request is directly proportional to the size of the requested file. Thus, for the remainder of this discussion, we replace "file size" with "service time".

4.2 Extension 1: Varying Initial Reward

In this scenario, requests, or jobs, arrive at the web server according to a Poisson process, as in the previous model. The service time of each job is drawn from an exponential distribution with parameter μ, and service times are i.i.d. The difference between this model and the original model is that jobs arrive with an *initial reward*, denoted as C_i for incoming request i. C_i is a positive quantity that represents the amount that job i would be willing to pay for "acceptable" service. Thus, the revenue function for job i under a policy p is $g_{i_p}(t) = C_i e^{-ct_{i_p}}$. The optimal service policy, as before, satisfies Equations 3.1 and 3.2. The policy space under consideration, $\mathcal{S}$, is the same as that presented in Chapter 3.

Using a procedure similar to the original set of proofs, it can be shown that the optimal service policy in this scenario satisfies the following lemmas:

4.2.1 Non-idling Policies

Lemma 4.2.1 *An optimal policy can be found in the class of non-idling policies.*

Proof: The proof here is the same as the proof of Lemma 3.3.1. We "swap" two equally-spaced intervals (of size $dl << 1$), where dl is small enough so that at most one departure can occur in the interval $[0, dl)$. Policies p and p' in $\mathcal{S}$ are defined the same here as in the original proof. The addition of the initial reward factor has the effect of multiplying Equation (3.3) by a constant. Thus,

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, [g_{j*_{p'}}(a_2)] - \mu\, dl\, [g_{j*_p}(a_2 + dl)] \\ &= \mu\, dl\, [C_{j*}e^{-c(a_2 - x_{j*})} - C_{j*}e^{-c(a_2 + dl - x_{j*})}] \\ &= C_{j*}\mu\, dl\, e^{-c(a_2 - x_{j*})}(1 - e^{-c\, dl}) \end{aligned}$$

which is the same equation obtained before, scaled by the initial reward of job $j*$. This last term is still positive, since $C_{j*} > 0$, so we conclude that

$$V_{p'}(\emptyset) > V_p(\emptyset)$$

Again, with some nonzero probability, policy p' will earn more than policy p, so p cannot be an optimal policy according to our definition of optimal.

4.2.2 Piecewise-constant Policies

Lemma 4.2.2 *An optimal service ordering can be found in the class of policies that switch between jobs in service only upon an arrival to or departure from the system.*

Proof: The proof of this lemma is very similar to the proof of Lemma 3.3.2, and the two alternate policies are defined under the same conditions as in the original proof. In this case, the revenue improvement represented by using the first policy p', which involves swapping the intervals $[a_1 - dl, a_1)$ and $[a_1, a_1 + dl)$ when $g_{(j+1)*}(a_1) \geq g_{j*}(a_1)$, is

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, C_{(j+1)*}[g_{(j+1)*}(a_1)] \\ &\quad - \mu\, dl\, C_{(j+1)*}[g_{(j+1)*}(a_1 + dl)] \\ &= \mu\, dl\, C_{(j+1)*} e^{-c(a_1 - x_{(j+1)*})}(1 - e^{-c\,dl}) \end{aligned}$$

This is a positive quantity; thus, for this particular definition of p', we conclude that

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

The difference in the expected revenue between the original policy and the alternate definition of policy p', which involves swapping the intervals $[a_1, a_1 + dl)$ and $[a_2, a_2 + dl)$ when $g_{j*}(a_1) > g_{(j+1)*}(a_1)$, is

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu\, dl\, g_{j*}(a_1 + dl) + \mu\, dl\, g_{(j+1)*}(a_2 + dl) \\ &\quad - \mu\, dl\, g_{(j+1)*}(a_1 + dl) - \mu\, dl\, g_{j*}(a_2 + dl) \\ &= \mu\, dl\, e^{-c\, dl}(e^{-ca_1} - e^{-ca_2}) \\ &\quad (C_{j*}e^{cx_{j*}} - C_{(j+1)*}e^{cx_{(j+1)*}}) \end{aligned}$$

Since $g_{j*}(a_1) > g_{(j+1)*}(a_1)$, we know that the last term in the above equation is positive, and

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

Thus, for each alternate policy p', we have shown that with some nonzero probability,

$$V_{p'}(\emptyset) > V_p(\emptyset)$$

and therefore p cannot be an optimal policy in $\mathcal{S}$.

4.2.3 Non-processor-sharing Policies

Lemma 4.2.3 *There is no policy outside of the class of non-processor-sharing (NPS) policies that generates a strictly higher revenue than the best policy from the NPS class of policies.*

Proof: The proof is similar to the proof of Lemma 3.3.3. Because of the length and complexity of the original proof, the important sections are repeated here.

We assume the existence of a generalized processor sharing (PS) policy that generates a higher revenue than the best NPS policy. As part of this policy, over some interval $[a, b]$ the server splits its resources between at least two jobs. The case where the server splits its resources among n jobs is a natural extension of the simple two-job case. The service times of the two jobs are independent and identically distributed.

Prior to time a, job 1 has spent an amount of time equal to τ_1 in the system, and job 2 has spent a time equal to τ_2 in the system. After time a, the distribution for the total revenue that the server earns from serving the two jobs under policy p is

$$\begin{aligned} g_p(t) &= g_{1_p}(t) + g_{2_p}(t) \\ &= c_f e^{-cT_f} + c_s e^{-cT_s} \end{aligned}$$

where, as in the original proof, $T_f = \min(T_1', T_2')$; T_i' is the remaining service time of job i after time a; $T_f \sim \exp(q\mu + (1-q)\mu) \sim \exp(\mu)$; and $T_s = T_f + Z$, $Z \sim \exp(\mu)$. Also,

$$c_f = \begin{cases} C_1 e^{-c\tau_1} = C_1 c_1, & T_f = T_1' \\ C_2 e^{-c\tau_2} = C_2 c_2, & T_f = T_2' \end{cases}$$

and

$$c_s = \begin{cases} C_1 e^{-c\tau_1} = C_1 c_1, & T_f = T_2' \\ C_2 e^{-c\tau_2} = C_2 c_2, & T_f = T_1' \end{cases}$$

The expected value of the revenue gained over this portion of the sample path is

$$E[g_p(t)] = \int_0^\infty \int_0^\infty (c_f e^{-ct_f} + c_s e^{-ct_s}) f_{T_f,T_s}(t_f, t_s)\, dt_f\, dt_s$$

The joint distribution for the service times is given in Equation (3.4). Making the substitution and simplifying the expression yields

$$E[g_p(t)] = \tfrac{\mu}{c+\mu} c_f + \left(\tfrac{\mu}{c+\mu}\right)^2 c_s$$

If job 1 departs the system first with probability q, the expected revenue from the PS policy is

$$E[g_p(t)] = q\left[\tfrac{\mu}{c+\mu} C_1 c_1 + \left(\tfrac{\mu}{c+\mu}\right)^2 C_2 c_2\right] + (1-q)\left[\tfrac{\mu}{c+\mu} C_2 c_2 + \left(\tfrac{\mu}{c+\mu}\right)^2 C_1 c_1\right] \quad (4.1)$$

If the server chooses to work on one job first and then the other, there are two possible policies. Define policy NPS1 as the policy that serves job 1 and then serves job 2, and policy NPS2 as the policy that serves job 2 first and then serves job 1. Prior to time a, each job has been in the system for a time τ_1 or τ_2, respectively. After time a, the remaining service time required by job 1 is t'_1 and the remaining service time required by job 2 is t'_2.

Under NPS1, the total potential revenue is

$$g_{\text{NPS1}}(t) = C_1 e^{-c(\tau_1 + t'_1)} + C_2 e^{-c(\tau_2 + t'_1 + t'_2)}$$

and under NPS2 it is

$$g_{\text{NPS2}}(t) = C_1 e^{-c(\tau_1 + t'_1 + t'_2)} + C_2 e^{-c(\tau_2 + t'_2)}$$

The expected revenue under policy NPS1 over this interval is

$$E[g_{\text{NPS1}}(t)] = \tfrac{\mu}{c+\mu} C_1 c_1 + \left(\tfrac{\mu}{c+\mu}\right)^2 C_2\, c_2 \tag{4.2}$$

and under policy NPS2, the expected revenue is

$$E[g_{\text{NPS2}}(t)] = \left(\tfrac{\mu}{c+\mu}\right)^2 C_1 c_1 + \tfrac{\mu}{c+\mu} C_2\, c_2 \tag{4.3}$$

But again, (4.1) is just a weighted sum of (4.2) and (4.3). Therefore, (4.1) cannot be greater than both (4.2) and (4.3). Thus, the PS policy cannot earn a strictly higher revenue than both policy NPS1 and policy NPS2.

4.2.4 Markov Policies

Lemma 4.2.4 *An optimal policy can be found among the class of Markov policies.*

Proof: The proof is similar to that of Lemma 3.3.4. In this case G_{t_p}, the state vector of the system at time t under policy p, is given by

$$G_{t_p} = (g_{1_p}(t)\ g_{2_p}(t)\ \ldots\ g_{M_p}(t)) \tag{4.4}$$

M denotes the number of jobs that have arrived at the system since time $t = 0$, and $g_{i_p}(t)$ is given by the equation

$$g_{i_p}(t) = \begin{cases} C_i e^{-c(t_{i_p} - x_i)}, & \text{if job } i \text{ departed prior to } t \\ C_i e^{-c(t - x_i)}\ , & \text{if job } i \text{ is still in the system at } t \end{cases}$$

where t_{i_p} is the departure time of job i under policy p and x_i is the arrival time of job i.

The state vector G_{t_p} contains information about the arrival time of the job at the system, the initial reward of the job, and, if applicable, the departure time of the job from the system. Since each $g_{i_p}(t)$ is an exponential function, each element of the state vector will decay by the same ratio over each time interval, until the job departs the system (at which point its revenue function ceases to decay and retains a constant value).

Because service times are memoryless, the probability of a job i completing its service in the interval $[t, t + dt)$, given that it is still in the system at time t and has already received an amount of service z_i, is

$$\begin{aligned} P(t_i < z_i + dt \mid z_i) &= P(t_i - z_i < dt) \\ &= 1 - e^{-\mu t_i'} \end{aligned}$$

where $t_i' = t_i - z_i$. This again is identical to the distribution of the job's original service time. Thus, knowledge of how much service a job has already attained prior to t, as well as knowledge of the job's initial revenue decay, will not yield any new information as to the chances of that job departing prior to $t + dt$, given that the server chooses to process job i in this interval. We can then replace each $g_{i_p}(t)$ in the state vector with the constant $c_{i_p} \triangleq C_i e^{-c(t-x_i)}$. Doing so removes all knowledge of past history without removing the information necessary to make the decision as to which job to serve. The decisions made in policy p, therefore, are the same decisions that would be made if only the present state of the system is known.

Corollary 4.2.4.1 *An optimal policy over the interval $[t, x_N)$ can be found among the class of policies that are independent of future arrivals to the system.*

Proof: The proof to this corollary is identical to the proof of Corollary 3.3.4.1, with the state vector in that proof replaced with the state vector in Equation (4.4).

4.2.5 An Optimal Policy

We now construct an optimal policy that satisfies the lemmas defined above. In addition, we establish the following definition:

Definition 4.2.1 *The **best job** is the job corresponding to the highest value in the state vector, defined in Equation (4.4), at time t.*

From Lemma (4.2.4), we know that an optimal policy will be a Markov policy. Thus, the state vector at time t is $G_t = (c_1\, c_2\, \ldots\, c_N)$, where each c_i indicates the maximum revenue the server can obtain from that job from time t onward and N indicates the number of requests currently in the system.

The expected revenue gained from serving any one job i beyond time t with remaining service time t'_i is

$$E[g(t'_i)] = E[c_i e^{-ct'_i}] = \tfrac{\mu}{c+\mu}\, c_i$$

where $c_i \overset{\triangle}{=} C_i e^{-c(t-x_i)}$ The job with the highest expected payoff satisfies

$$\max_i\, c_i\ , i \in 1, 2, \ldots, N$$

as before. Thus the job with the highest expected payoff is the job with the largest constant c_i in the state vector G_t, where c_i is as defined in Lemma (4.2.4). We define this job as the "best first job" to serve starting at time t.

Now, we define an optimal policy for this policy space:

Theorem 4.2.1 *An optimal policy will be greedy (i.e., it will choose the "best first job" to serve, regardless of the order chosen for the other jobs in the system).*

Proof: The proof is identical to the proof of Theorem 3.4.1, and the results are the same.

Remarks

We again have shown that the optimal service ordering is non-idling, piecewise-constant, non-processor-sharing, Markov, and greedy. Unlike the original system, the optimal policy for this policy space does not default to a "standard" service policy (e.g., FIFO, LIFO, PS). The optimal policy chooses the job that has the greatest current potential revenue, where potential revenue is a function of both the time spent in the system so far and the request's initial reward.

4.3 Extension 2: Varying Exponential File Size Distributions

We now consider a system in which arrivals are Poisson and the service times vary among entering requests. The service time for job i is an independent exponential random variable with parameter μ_i. As in the original system, each incoming request is willing to pay the same amount for "acceptable" service. We define another quantity of interest for this system, the *revenue rate* of a request, as the expected revenue per unit time for that request.

In this system, the policy space $\mathcal{S}$ contains policies in which the server processes one request at a time and where the server only switches between jobs in service upon an arrival to or departure from the system. (The reasons for these restrictions will be obvious shortly.) In addition, we consider both idling and non-idling policies, and both preemptive and non-preemptive policies, that fit these restrictions.

Given the above assumptions, we can determine an optimal service ordering that satisfies the following criteria.

4.3.1 Non-idling Policies

Lemma 4.3.1 *An optimal policy will be non-idling.*

Proof: Suppose there exists a policy p in $\mathcal{S}$ which is optimal but not non-idling over a sample path in the first busy cycle. Define $[a_1, a_2)$ to be a time interval in this sample path in which the server remains idle under policy p during a busy cycle (while jobs are queued). At time a_2, the server switches from idle to busy, and works on some job $j*$. Define a_3 as the next completion time during this busy period, where $a_3 > a_2$.

As in the original proof, we can construct a policy p' in S that is identical to p except over the interval $[a_1, a_3)$. Define dl to be an infinitesimal time interval ($dl << 1$). We can then define policy p' as the policy that makes a "swap" of two intervals of length dl from policy p. In particular, policy p' serves job $j*$ in the interval $[a_2 - dl, a_2)$, remains idle in the interval $[a_2, a_2 + dl)$, and then returns to the service order defined in policy p at time $a_2 + dl$, and continues following policy p until the end of the busy cycle.

We define the optimal policy as one that maximizes the average revenue per unit time per cycle, as given by (3.1). Thus, under policy p,

$$V_p(\emptyset) = \frac{E\left[\sum_{i=1}^{N} g_{i_p}(t_{i_p})\right]}{E[Z_1]}$$

and under policy p',

$$V_{p'}(\emptyset) = \frac{E\left[\sum_{i=1}^{N} g_{i_{p'}}(t_{i_{p'}})\right]}{E[Z_1]}$$

Since the expected value of the duration of the busy/idle cycle is the same under both policies, we can drop the denominator from our analysis and just consider the numerator in each expression.

The difference in the expected revenue under the two policies is

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] = E[g_{j*_{p'}}(a_2)] - E[g_{j*_p}(a_2+dl)]$$

dl is small enough so that at most one departure can occur in either $[a_2-dl, a_2)$ or $[a_2, a_2+dl)$. The probability of job $j*$ departing within the interval $[0, dl)$ is $\mu_{j*}\, dl + o(dl) \approx \mu_{j*}\, dl > 0$. Thus,

$$\begin{aligned} E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] &= \mu_{j*}\, dl\, [g_{j*_{p'}}(a_2)] - \mu_{j*}\, dl\, [g_{j*_p}(a_2+dl)] \\ &= \mu_{j*}\, dl\, [e^{-c(a_2-x_{j*})} - e^{-c(a_2+dl-x_{j*})}] \\ &= \mu_{j*}\, dl\, e^{-c(a_2-x_{j*})}(1-e^{-c\,dl}) \end{aligned}$$

The last term in the above equation is positive, so we conclude that

$$E\left[\sum_{i=0}^{N} g_{i_{p'}}(t_{i_{p'}})\right] - E\left[\sum_{i=0}^{N} g_{i_p}(t_{i_p})\right] > 0$$

and therefore

$$V_{p'}(\emptyset) > V_p(\emptyset)$$

Thus, with some nonzero probability, policy p' will earn more than policy p, so p cannot be an optimal policy according to our definition of optimal.

4.3.2 Markov Policies

Lemma 4.3.2 *An optimal policy can be found among the class of Markov policies.*

Proof: The proof is similar to the proof of Lemma 3.3.4. We denote the state vector, G_{t_p}, of the system at time t under policy p, where

$$G_{t_p} = \{(g_{1_p}(t), \mu_1)\ (g_{2_p}(t), \mu_2)\ \ldots\ (g_{M_p}(t), \mu_M)\}$$

M denotes the number of jobs that have arrived at the system since time $t = 0$, μ_i is the service rate associated with job i, and $g_{i_p}(t)$ is given by the equation

$$g_{i_p}(t) = \begin{cases} e^{-c(t_{i_p} - x_i)}, & \text{if job } i \text{ departed prior to } t \\ e^{-c(t-x_i)}\ , & \text{if job } i \text{ is still in the system at } t \end{cases}$$

Again, t_{i_p} is the departure time of job i under policy p and x_i is the arrival time of job i.

Since jobs that have already departed do not affect the service selection, we can assume without loss of generality that these jobs depart the state vector as well upon departure from the system. We can then rewrite the state vector by removing the departures and renumbering the remaining jobs based on relative arrival times:

$$G'_{t_p} = \{(c_{1_p}, \mu_1)\ (c_{2_p}, \mu_2)\ \ldots\ (c_{M'_p}, \mu_{M'})\} \tag{4.5}$$

where $M' = M-$ the number of completions before t and $c_{i_p} \overset{\triangle}{=} e^{-c(t-x_i)}$.

Because service times are exponential, the distribution of remaining service time for each job is identical to the distribution of the job's original service time. Thus, knowledge of how much service a job has already attained prior to t will not yield any new information as to the chances of that job departing prior to $t + dt$, given that the server chooses to process job i in this interval. The decisions made in policy p, therefore, are the same decisions that would be made if only the present state of the system is known.

Corollary 4.3.2.1 *An optimal policy over the interval* $[t, x_N)$ *can be found among the class of policies that are independent of future arrivals to the system.*

Proof: The proof of this corollary is identical to the proof of Lemma 3.3.4.1, with the state vector defined by Equation (4.5).

4.3.3 Construction of an Optimal Policy

Since we have shown that an optimal policy for policy space $\mathcal{S}$ satisfies the lemmas presented above, we can construct an optimal service ordering within these limitations. As in the previous cases, before we state what the optimal policy is, we need to specify one further definition:

Definition 4.3.1 *The* ***best job*** *is the job corresponding to the highest $c_i\mu_i$ product at time t, where c_i and μ_i are the state vector values for job i (i.e., (c_i, μ_i)) at time t and the state vector is defined as in Equation (4.5).*

We know from Lemma 4.3.2 that an optimal policy will be a Markov policy. Thus, the state vector at time t is $G_t = \{(c_1, \mu_1)\ (c_2, \mu_2)\ \ldots\ (c_N, \mu_N)\}$. Each c_i indicates the maximum revenue the server can obtain from that job from time t onward. Each μ_i indicates the expected rate of service for job i, or the inverse of the expected remaining service time for job i.

Since the requests are serviced at different rates, it is not sufficient here to define the best job as the request with the highest expected revenue. Rather, we must consider the combination of potential revenue and expected completion time. In particular, we must consider the expected revenue per unit time of each request i in the system at time t, which is equal to the expected revenue divided by the expected completion time, or $c_i\,\mu_i$. Thus, the best job satisfies $\max_i c_i\,\mu_i\ , i \in 1, 2, \ldots, N$.

This definition differs from Definitions 3.4.1 and 4.2.1. In these scenarios, however, the service times are identically distributed. Thus, the expected revenue per unit time for job i is given by $c_i\,\mu$ for all i. Since μ is identical in each term,

$$\max_i c_i\,\mu = \max_i c_i$$

and this is equivalent to maximizing the expected revenue per unit time beyond time t.

With this definition, we can now state the following theorem about the optimal policy.

Theorem 4.3.1 *An optimal policy will be greedy (i.e., it will choose the "best first job" to serve, regardless of the order chosen for the other jobs in the system).*

Proof: Assume there exists an optimal policy, p, that satisfies Lemmas 4.3.1 and 4.3.2, works on one request at a time, and does not switch between jobs in service unless there is an arrival or departure, but is not greedy. That is, at some time t in a sample path during a busy cycle, the server does not choose the best job to serve. Instead, it will choose to serve job $1*_p$ at time t, complete that job, and then serve job $2*_p$ to completion, where $c_{2*_p}\mu_{2*_p} > c_{1*_p}\mu_{1*_p}$.

The state vector at time t is given in the proof of Lemma 4.3.2. We can reorder these values in terms of the order in which the jobs are served under policy p. The state vector then becomes $G_{t_p} = \{(c_{1*_p}, \mu_{1*_p})\ (c_{2*_p}, \mu_{2*_p})\ \ldots\ (c_{N*_p}, \mu_{N*_p})\}$.

The revenue obtained from policy p from t until the end of the current busy cycle is

$$\sum_{j=1}^{N} g_{j_p}(t_{j_p}) \;=\; c_{1*_p}e^{-ct'_{1*p}} + c_{2*_p}e^{-c(t'_{1*p}+t'_{2*p})} + \ldots + c_{N*_p}e^{-c(t'_{1*p}+t'_{2*p}+\ldots+t'_{N*p})}$$

$$= \sum_{j=1}^{N} c_{j*_p} e^{-c \sum_{k=1}^{j} t'_{k*_p}}$$

where t'_{i*_p} is the remaining service time of the ith job served after time t under policy p.

We construct a policy p' that performs an interchange of jobs $1*_p$ and $2*_p$, such that the server processes job $2*_p$ at time t to completion and then processes job $1*_p$ to completion. At all other times, p' is identical to p.

The revenue earned under policy p' from t until the end of the current busy cycle is

$$\begin{aligned}\sum_{j=1}^{N} g_{j_p}(t_{j_p}) &= c_{2*_p} e^{-ct'_{2*p}} + c_{1*_p} e^{-c(t'_{1*p}+t'_{2*p})} + \ldots + c_{N*_p} e^{-c(t'_{1*p}+t'_{2*p}+\ldots+t'_{N*p})} \\ &= c_{2*_p} e^{-ct'_{2*p}} + c_{1*_p} e^{-c(t'_{1*p}+t'_{2*p})} + \sum_{j=3}^{N} c_{j*_p} e^{-c \sum_{k=1}^{j} t'_{k*p}}\end{aligned}$$

The expected revenue under each policy is the expected value of the sum of the revenues collected from each job that departs the system from time t to the end of the current busy cycle.

The difference in revenue between the two policies is

$$\begin{aligned}E\left[\sum_{j=1}^{N} g_{j_{p'}}(t_{j_{p'}})\right] - E\left[\sum_{j=1}^{N} g_{j_p}(t_{j_p})\right] &= E\left[c_{2*_p} e^{-ct'_{2*p}} + c_{1*_p} e^{-c(t'_{1*p}+t'_{2*p})}\right] \\ &\quad - E\left[c_{1*_p} e^{-ct'_{1*p}} + c_{2*_p} e^{-c(t'_{1*p}+t'_{2*p})}\right] \\ &= E\left[c_{2*_p} e^{-ct'_{2*p}} + c_{1*_p} e^{-c(t'_{1*p}+t'_{2*p})}\right.\end{aligned}$$

$$
\begin{aligned}
&\quad - c_{1*_p} e^{-ct'_{1*p}} - c_{2*_p} e^{-c(t'_{1*p}+t'_{2*p})}\Big] \\
&= E\Big[c_{2*_p} e^{-ct'_{2*p}}(1 - e^{-ct'_{1*p}}) \\
&\quad - c_{1*_p} e^{-ct'_{1*p}}(1 - e^{-ct'_{2*p}})\Big] \\
&= E\Big[c_{2*_p} e^{-ct'_{2*p}}(1 - e^{-ct'_{1*p}})\Big] \\
&\quad - E\Big[c_{1*_p} e^{-ct'_{1*p}}(1 - e^{-ct'_{2*p}})\Big] \\
&= \frac{c_{2*_p}\mu_{2*_p}}{c+\mu_{2*_p}}\left(1 - \frac{\mu_{1*_p}}{c+\mu_{1*_p}}\right) \\
&\quad - \frac{c_{1*_p}\mu_{1*_p}}{c+\mu_{1*_p}}\left(1 - \frac{\mu_{2*_p}}{c+\mu_{2*_p}}\right) \\
&= \frac{c_{2*_p}\mu_{2*_p}}{c+\mu_{2*_p}}\left(\frac{c}{c+\mu_{1*_p}}\right) \\
&\quad - \frac{c_{1*_p}\mu_{1*_p}}{c+\mu_{1*_p}}\left(\frac{c}{c+\mu_{2*_p}}\right) \\
&= \frac{c}{(c+\mu_{1*_p})(c+\mu_{2*_p})}(c_{2*_p}\mu_{2*_p} - c_{1*_p}\mu_{1*_p})
\end{aligned}
$$

Both terms in the above equation are always greater than zero. Therefore, since the expected time of the busy cycle is the same under policy p and policy p', we conclude that

$$V_{p'} > V_p$$

Since p' yields a higher average revenue per unit time than p, p cannot be an optimal policy.

4.3.4 Remarks and a Counterexample

We have demonstrated that the optimal policy for the system described above is non-idling, Markov, and greedy. Returning to our definition of the best job, we find that the optimal policy for this system processes requests in decreasing $c_i \mu_i$ order. The job in service only changes upon an arrival to or departure from the system; unlike the original system, however, an arrival only preempts the request currently in service if it has a higher $c_i \mu_i$ product than that of the job in service.

Note that Theorem 4.3.1 only covers the policy space $\mathcal{S}$. A particularly glaring omission from $\mathcal{S}$ is the family of generalized processor-sharing policies. If we define policy space $\mathcal{S}'$ to contain all policies in $\mathcal{S}$ and all generalized non-idling processor-sharing policies, will the same optimal policy still apply? In particular, will the optimal policy still be from the family of non-processor-sharing policies? To answer these questions, we consider the following example:

Conjecture 4.3.1 *A non-processor-sharing (NPS) policy is always at least as good as a processor-sharing (PS) policy over all sample paths in this system.*

Proof: We show that this conjecture does not hold by means of an example.

Suppose there exists a sample path in which the server processes two requests during a busy cycle. We label these jobs "job 1" and "job 2". There are no further arrivals to this system. In this sample path, job 1 arrives at time zero, and job 2 arrives at some time $a > 0$, where a is less than the service time required by job 1. The busy cycle ends at some time b, $b > a$.

We consider three service policies, two of which are non-processor-sharing policies from $\mathcal{S}$ and one that is a processor-sharing policy from $\mathcal{S}'$. We label the policies NPS1, NPS2, and PS, respectively. The three policies behave as follows over the interval $[a, b]$:

- **NPS1**: The server processes job 1 to completion, queueing job 2. Upon job 1's departure, the server processes job 2 to completion.
- **NPS2**: The server processes job 2 to completion, queueing job 1. Upon job 2's departure, the server processes job 1 to completion.
- **PS**: The server processes both job 1 and job 2 simultaneously, until one of the jobs completes service and departs, at which time the server dedicates all of its resources to processing the remaining job to completion.

We are interested in the revenue the server expects to earn over this sample path under each of the three policies. We derive the general expressions for the expected revenue earned under the three policies first, and then describe a specific example where the expected revenue earned by the PS policy is greater than the expected revenues earned by the NPS1 and NPS2 policies separately.

The associated revenue decay for the two jobs at time a is $c_1 = e^{-ca}$ and $c_2 = e^{-c0} = 1$, respectively. We define T_1' as the remaining service time for job 1 beyond time a and T_2 as the service time for job 2 beyond time a. Due to the memoryless property of the exponential distribution, T_1' is distributed identically to T_1, the original service time of job 1. T_1 and T_1' are exponentially distributed with parameter μ_1, and T_2 is exponentially distributed with parameter μ_2.

We consider the two non-processor sharing policies first. Under policy NPS1, the time job 1 spends in the system past time a is T_1'. Job 2 waits for a time period equal to job 1's remaining service time and then departs after its service time; its total time in the system is thus $T_1' + T_2$. The revenue earned by the server during this sample path under this policy is

$$g_{\text{NPS1}} = c_1\, e^{-cT_1'} + c_2\, e^{-c(T_1'+T_2)}$$

The expected revenue is given by the equation

$$E[g_{\text{NPS1}}] = \int_0^\infty \int_0^\infty [e^{-ct_1'} + c_2 e^{-c(t_1'+t_2)}] f_{T_1',T_2}(t_1', t_2) dt_1'\, dt_2$$

Because the service times are independent, $f_{T_1',T_2}(t_1', t_2) = f_{T_1'}(t_1') f_{T_2}(t_2)$, and so

$$\begin{aligned} E[g_{\text{NPS1}}] &= \int_0^\infty \int_0^\infty [e^{-ct_1'} + c_2 e^{-c(t_1'+t_2)}] \mu_1 e^{-\mu_1 t_1'} \mu_2 e^{-\mu_2 t_2} dt_1'\, dt_2 \\ &= c_1 \frac{\mu_1}{c+\mu_1} + c_2 \frac{\mu_1}{c+\mu_1} \frac{\mu_2}{c+\mu_2} \end{aligned} \tag{4.6}$$

Similarly, under NPS2, job 1 spends an amount of time equal to $T_1' + T_2$ beyond time a in the system, and job 2 spends an amount of time equal to T_2 in the system. The revenue earned by the server during this sample path under this policy is

$$g_{\text{NPS2}} = c_1 e^{-c(T_1'+T_2)} + c_2 e^{-cT_2}$$

and, using the same procedure as the one used in evaluating the expected revenue from the NPS1 policy, the expected revenue for policy NPS2 is

$$E[g_{\text{NPS2}}] = c_1 \frac{\mu_1}{c+\mu_1} \frac{\mu_2}{c+\mu_2} + c_2 \frac{\mu_2}{c+\mu_2} \tag{4.7}$$

We now look at the PS policy. Under this policy, the server devotes a fraction q of its total resources to processing job 1 and $1-q$ of its total resources to processing job 2, where $0 < q < 1$. Define T_f as the time at which the first job to complete service departs the system with respect to time a, where $T_f = \min(T_1', T_2)$ and is exponentially distributed with parameter $(q\mu_1 + (1-q)\mu_2) \overset{\triangle}{=} \mu$. Also, define T_s as the time at which the remaining job departs the system. We set $T_s = T_f + Z$, where Z is the remaining processing time for the job remaining in the system past time $T_f + a$. The probability that job 1 completes its service and departs the system first is $(q\mu_1)/\mu$, and the probability that job 2 completes its service and departs the system first is $[(1-q)\mu_2]/\mu$.

To define Z, we let $Z = Z_1$ given that job 1 finishes first, and $Z = Z_2$ given that job 2 finishes first. Thus, Z_1 is distributed exponentially with parameter μ_2

and Z_2 is distributed exponentially with parameter μ_1. Then

$$Z = \begin{cases} Z_1, & \text{with probability } (q\mu_1)/\mu \\ Z_2, & \text{with probability } ((1-q)\mu_2)/\mu \end{cases}$$

So the distribution of Z is given by the equation

$$\begin{aligned} f_Z(z) &= \frac{q\mu_1}{\mu} f_{Z_1}(z) + \frac{(1-q)\mu_2}{\mu} f_{Z_2}(z) \\ &= \frac{q\mu_1}{\mu} \mu_2 e^{-\mu_2 z} + \frac{(1-q)\mu_2}{\mu} \mu_1 e^{-\mu_1 z} \\ &= \frac{\mu_1\mu_2}{\mu} [q e^{-\mu_2 z} + (1-q) e^{-\mu_1 z}] \end{aligned}$$

The joint distribution of the service times of the two jobs is

$$\begin{aligned} f_{T_f,T_s}(t_f, t_s) &= f_{T_s|T_f}(t_s|t_f) f_{T_f}(t_f) \\ &= f_{T_s|T_f}(t_f + z|t_f) f_{T_f}(t_f) \\ &= f_{Z|T_f}(z|t_f) f_{T_f}(t_f) \end{aligned}$$

But

$$f_{Z|T_f}(z|t_f) = \frac{q\mu_1}{\mu} f_{Z_1}(z) + \frac{(1-q)\mu_2}{\mu} f_{Z_2}(z) = f_Z(z)$$

and thus T_f and Z are independent. So the joint distribution is

$$\begin{aligned} f_{T_f,T_s}(t_f, t_s) &= f_{T_f}(t_f) f_Z(z) \\ &= \mu_1\mu_2 e^{-\mu t_f} [q e^{-\mu_2 z} + (1-q) e^{-\mu_1 z}] \end{aligned}$$

In addition, we define

$$c_f = \begin{cases} c_1, & T_f = T_1' \\ c_2, & T_f = T_2 \end{cases}$$

and

$$c_s = \begin{cases} c_2, & T_f = T_1' \\ c_1, & T_f = T_2 \end{cases}$$

The revenue collected by the server during the sample path is

$$g_{\text{PS}} = c_f e^{-cT_f} + c_s e^{-cT_s} = c_f e^{-cT_f} + c_s e^{-c(T_f+Z)}$$

The expected value of this revenue is

$$\begin{aligned} E[g_{\text{PS}}] &= \int_0^\infty \int_0^\infty [c_f e^{-cT_f} + c_s e^{-c(T_f+Z)}] f_{T_f}(t_f) f_Z(z) dt_f\, dz \\ &= \int_0^\infty \int_0^\infty [c_f e^{-cT_f} + c_s e^{-c(T_f+Z)}]\, \mu_1\mu_2 e^{-\mu t_f} [q e^{-\mu_2 z} + (1-q) e^{-\mu_1 z}] dt_f\, dz \\ &= c_f \frac{\mu}{c+\mu} + c_s \frac{\mu_1}{c+\mu_1} \frac{\mu_2}{c+\mu_2} \qquad (4.8) \end{aligned}$$

If job 1 completes first with probability $(q\mu_1)/\mu$, then we make the proper substitutions and obtain

$$\begin{aligned} E[g_{\text{PS}}] &= \frac{q\mu_1}{\mu}\left[c_1 \frac{\mu}{c+\mu} + c_2 \frac{\mu_1}{c+\mu_1}\frac{\mu_2}{c+\mu_2}\right] \\ &\quad + \frac{(1-q)\mu_2}{\mu}\left[c_2 \frac{\mu}{c+\mu} + c_1 \frac{\mu_1}{c+\mu_1}\frac{\mu_2}{c+\mu_2}\right] \\ &= \frac{qc_1\mu_1 + (1-q)c_2\mu_2}{c+\mu} + \frac{1}{\mu}\frac{\mu_1}{c+\mu_1}\frac{\mu_2}{c+\mu_2}[q\, c_2\mu_1 + (1-q)c_1\mu_2] \qquad (4.9) \end{aligned}$$

We now show that there exist nonnegative values of $c_1, c_2, \mu_1, \mu_2, c,$ and q for which the expected revenue for the PS policy exceeds the expected revenues of both policies NPS1 and NPS2. Let us assume that the server splits its resources evenly between the two jobs, such that $q = 0.5$. Let us also assume the revenue function is e^{-ct}, where $c = 1$. We have already established that $c_2 = 1$. We set $c_1 = 0.45$, $\mu_1 = 20$, and $\mu_2 = 10$. Plugging these values in to the revenue expressions yields $E[g_{\mathrm{NPS1}}] = 1.2987$, $E[g_{\mathrm{NPS2}}] = 1.2944$ and $E[g_{\mathrm{PS}}] = 1.3008$. Clearly, $E[g_{\mathrm{PS}}] > E[g_{\mathrm{NPS1}}]$ and $E[g_{\mathrm{PS}}] > E[g_{\mathrm{NPS2}}]$. Thus, for this sample path, neither of the NPS revenues is strictly greater than the revenue generated by the PS policy, and the conjecture does not hold.

Discussion: To consider the more general case, recall that in the proofs of Lemmas 3.3.3 and 4.2.3, we were able to claim that the PS policies are not optimal because their expected revenue equations are simply weighted sums of the expected revenue equations for the two NPS policies. In other words, the expected revenue for the PS policy is never strictly greater than *both* expected revenue equations corresponding to the NPS policies. Because we were able to show in this case that there exists at least one sample path where the expected PS revenue is better than both the expected revenues from the NPS policies, we cannot make the same claim here.

To verify this, we take the differences between the expected values of the NPS1 and PS policies, and the expected values of the NPS2 and PS policies. The former yields

$$\begin{aligned} E[g_{\text{NPS1}}] - E[g_{\text{PS}}] &= c_1\frac{\mu_1}{c+\mu_1} + \frac{\mu_1}{c+\mu_1}\frac{\mu_2}{c+\mu_2}\left[c_2 - \frac{q\,c_2\mu_1 + (1-q)c_1\mu_2}{\mu}\right] \\ &\quad - \frac{qc_1\mu_1 + (1-q)c_2\mu_2}{c+\mu} \end{aligned}$$

which reduces to

$$= \frac{(1-q)\{\mu(c+\mu_2)[c_1\mu_1(c+\mu_2) - c_2\mu_2(c+\mu_1)] + \mu_1\mu_2^2(c_2-c_1)(c+\mu)\}}{\mu(c+\mu)(c+\mu_1)(c+\mu_2)} \tag{4.10}$$

The latter yields

$$\begin{aligned} E[g_{\text{NPS2}}] - E[g_{\text{PS}}] &= c_2\frac{\mu_2}{c+\mu_2} + \frac{\mu_1}{c+\mu_1}\frac{\mu_2}{c+\mu_2}\left[c_1 - \frac{q\,c_2\mu_1 + (1-q)c_1\mu_2}{\mu}\right] \\ &\quad - \frac{q\,c_1\mu_1 + (1-q)c_2\mu_2}{c+\mu} \end{aligned}$$

which simplifies to

$$= \frac{q\{\mu_1^2\mu_2(c+\mu)(c_1-c_2) + \mu(c+\mu_1)[c(c_2\mu_2 - c_1\mu_1) + \mu_1\mu_2(c_2-c_1)]\}}{\mu(c+\mu)(c+\mu_1)(c+\mu_2)} \tag{4.11}$$

If both (4.10) and (4.11) are strictly positive for $0 < q < 1$, then we can claim that PS is never strictly better than both of the NPS policies. These equations, though, are inconclusive. In either case, the choice of values for $c_1, c_2, c, \mu_1,$ and μ_2

determines whether these equations are positive or negative, as we have demonstrated in the above example.

Thus, there exists at least one sample path in which a PS policy outperforms a set of NPS policies. We show in Chapter 6, via simulation, that more sample paths like this one exist, and therefore we cannot completely eliminate PS policies from consideration in systems where file size distributions are variable.

Chapter 5

SIMULATION ANALYSIS OF THE ORIGINAL WWW SERVER MODEL

5.1 Introduction

We discuss a simulation model corresponding to the proofs presented in Chapter 3. The model corresponds to a web server system where requests from web users arrive according to a Poisson process with parameter λ. The service times for each incoming request are independent and identically distributed exponential random variables with parameter μ. The server earns an amount of revenue from processing each incoming request, where the amount of revenue earned is an exponentially decreasing function of the total time the request spends at the server, both waiting and in service, which we denote as response time. We are interested in the average reward per unit time earned by the server under an optimal service policy, as defined by Equations (3.1) and (3.2). Because the simulation time is much longer than the time required for the queue to empty, we can look at the total revenue collected per simulation rather than the revenue per busy cycle. The total revenue measure serves as a good approximation, when normalized, to the average total revenue seen per busy cycle.

In this section, we describe a simulation model, constructed using BONeS [12] software, for the system described above. In Chapter 3, we determined that the optimal service ordering policy for this system is a preemptive-resume last-in, first-out (LIFO-PR) policy. We model this service ordering along with two other service orderings commonly found in web server software architecture: first-in, first-out (FIFO) and processor-sharing (PS). We compare the total normalized revenue earned by the server per simulation in each service ordering for a range of service rates and arrival rates. We also look at the statistics related to response time, where response time is defined as the total time a request spends at the server both in queue and in service.

5.2 Simulation Model

Figure 5.1 illustrates the BONeS model for this system. The model consists of four parts: the request generator, the service time generator, the queue/server module, and the statistics modules. The different parts of this model are discussed in the sections that follow.

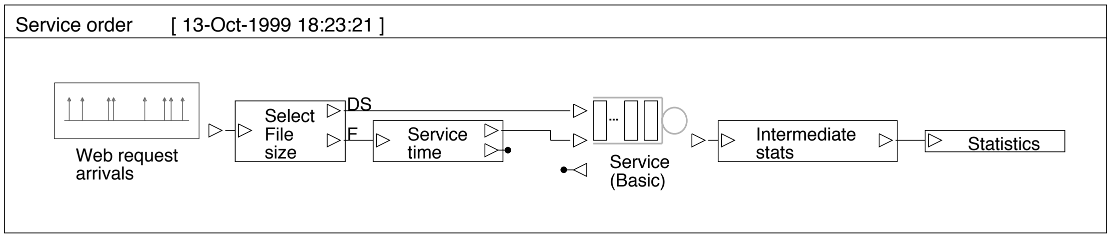

Figure 5.1: The BONeS model of the original system.

5.2.1 Request Generator

The request generator block, labeled as ***Web request arrivals*** in the model, is illustrated in greater detail in Figure 5.2. This module creates the data structures used to represent incoming web requests at the server. Table 5.1 lists the different fields in this data structure and the values assigned to each field.

The first part of this module is the ***Poisson Pulse Train*** block, which generates pulses at exponentially-distributed interarrival times. Each time a pulse is generated, it creates a new web request by triggering the ***Create web request*** block. Each request is assigned a unique *Request ID* number. Next, the module draws a random file size from an exponentially-distributed random number generator block (labeled ***File size***), rounded to the nearest byte. The module then places a timestamp on the web request indicating its arrival time, and sets the current revenue to the initial reward of the entering request.

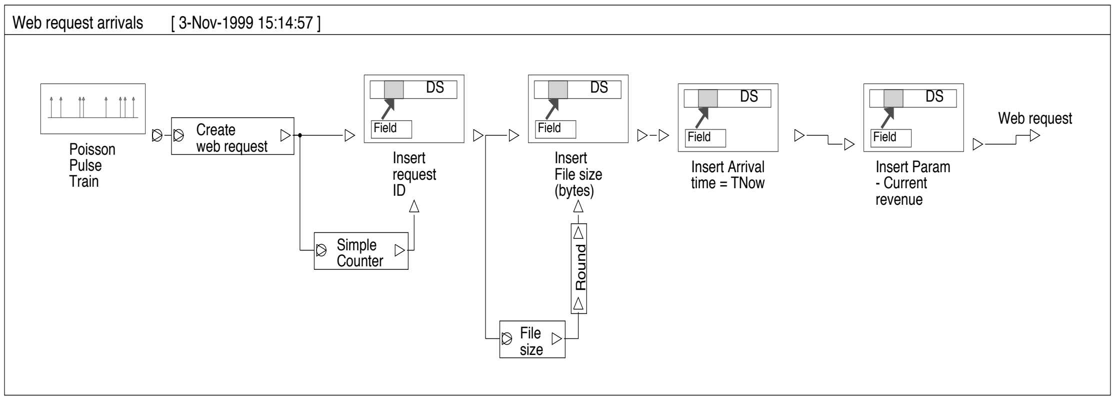

Figure 5.2: The **Web request arrivals** module in the original system.

The ***Poisson Pulse Train*** block has two parameters associated with it, both of which are assigned values at simulation run time. These values are the arrival time of the first request and the mean interarrival time of the pulses. In addition, the values of two other parameters associated with this module are deferred until simulation run time: the mean file size (used to define the file size distribution) and the initial revenue (which is set to 1.0 for this set of simulations).

Table 5.1: The data structure used to represent a web request in the BONeS simulation model.

Field	**Description**	**Value**
Request ID	The identification number of the data structure	Counter value
File size	Size of the requested file in bytes	Mean value is defined at simulation run time
Arrival time	The time at which the request arrives at the server	Current time
Departure time	The time at which the request departs the system	Assigned later
Current revenue	Used here to define the initial revenue multiplier	Defined at simulation run time
Initial revenue factor	Not used	
Service time	Not used	
dummy	Used to calculate server throughput (testing only)	

5.2.2 Service Time Generator

This portion of the model includes the ***Select file size*** and ***Service time*** blocks. The service time is calculated by multiplying the request's file size (assigned in the previous module) by the server's *per-byte processing rate*, another simulation parameter specified at run time. The per-byte processing rate defines the number of bytes per second processed by the server (CPU).

5.2.3 Queue/Server

The queue/server block, labeled ***Service (Basic)*** in the block diagram, is a single server system with no priority levels and no overhead associated with swapping out jobs from service or resuming preempted jobs. The block has two inputs, one for the web request data structure, and one for the request's service time. The type of queue (FIFO, LIFO-PR, PS) is set at simulation run time via the parameters *Queue Discipline*, *Server Mechanism*, and *Preempt Discipline. Queue Discipline* determines whether the queue is first-in, first-out or last-in, first-out. *Server Mechanism* indicates whether the server dedicates all of its resources to one job at a time or splits the resources among jobs (via round robin or processor sharing). *Preempt Discipline* determines whether the server is allowed to preempt jobs in service. The number of servers is also set at run time, although for these simulations this value is always set to one. Incoming requests that encounter a full queue are discarded, although the queue size is set to a large enough number to prevent this from happening under stable conditions (i.e., $\rho < 1$).

This module simulates all the CPU processing inherent in servicing a web request. We assume that memory is large enough so that no swapping or disk read/writes are necessary to service a request. The module does not model any of the TCP or network-associated actions involved in fulfilling a web request, such as writing to the socket queue, waiting for acknowledgement from the client side, or maintaining TCP connections.

5.2.4 Statistics

This portion of the model includes the ***Intermediate stats*** and ***Statistics*** blocks. The former performs calculations for individual requests that are departing the system, while the latter performs computations relevant to the entire collection of requests served during a simulation.

The ***Intermediate stats*** block is shown in Figure 5.3. It performs two calculations. First, it adds a timestamp to the *Departure time* field of the web request data structure and, using this value and the arrival timestamp, calculates the request's response time. Next, it calculates the revenue generated by the server as a result of serving that request via the equation

$$\text{revenue} = \text{initial revenue} * e^{-(\text{cost factor}) \, * \, (\text{response time})}$$

where the *cost factor* parameter indicates how fast the revenue function decays. The revenue is placed in the *Current revenue* field in the data structure.

The ***Statistics*** block, shown in Figure 5.4, calculates the total revenue generated during the simulation run, and outputs this value once the simulation run

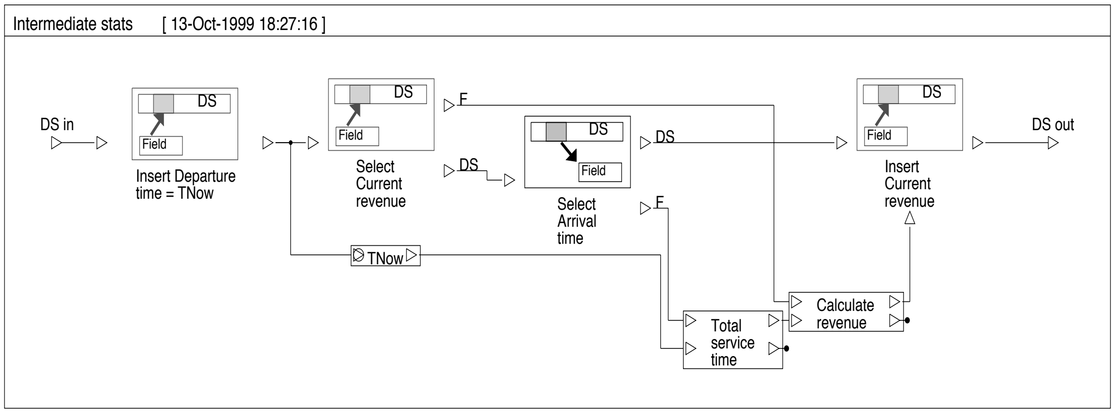

Figure 5.3: The **Intermediate stats** module in the original system.

terminates. It also calculates the total number of requests serviced during a simulation run, which is used to normalize the total revenue measured during the simulation.

5.3 Simulation Parameters

As mentioned in the previous section, some of the parameters associated with the system model are deferred until simulation run time. Table 5.2 lists some of these parameters and the values assigned to them in this set of simulations.

The simulation model also contains probe modules whose purpose is to collect data of interest during each simulation run. The probes and the data they collect are summarized in Table 5.3.

Table 5.2: Simulation parameters assigned at run time

Parameter	**Value**
Service rate (μ)	λ/ρ
Load (ρ)	varied between 0.01 and 0.99
First request arrival time	0.0
Mean interarrival time	$1/\lambda$, $\lambda \in \{1, 5, 10, 50, 100\}$.
Mean file size	2 kB
Per-byte processing rate	Service rate times mean file size
Cost factor	1.0
Initial revenue	1.0
Server mechanism	*Dedicated Server* (LIFO-PR and FIFO simulations) *Processor Sharing* (PS simulation)
Preempt Discipline	*Don't Preempt* (FIFO simulation) *Allow Preemption* (PS and LIFO-PR simulations)
Queue discipline	*First-in, first-out* (FIFO and PS simulations) *Last-in, first-out* (LIFO-PR simulation)
Time slice	1.0 (default for non-round-robin simulations)
TSTOP	the simulation end time; set to 10,000 seconds.

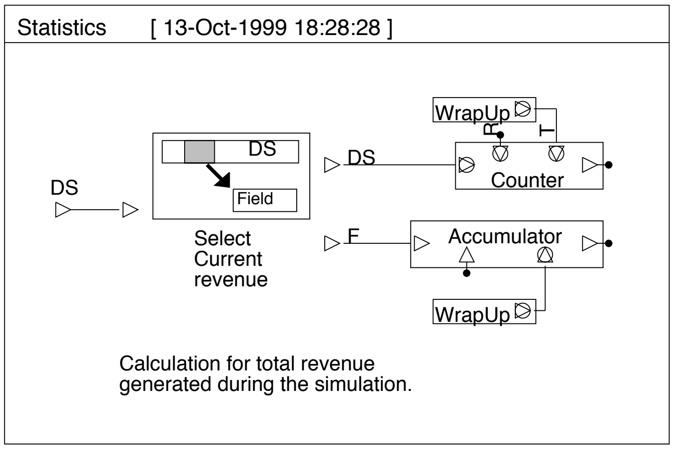
Statistics [13-Oct-1999 18:28:28]
DS
DS
Field
Select Current revenue
DS
F
WrapUp
R
T
Counter
Accumulator
WrapUp
Calculation for total revenue generated during the simulation.

Figure 5.4: The **Statistics** module in the original system.

Table 5.3: Simulation data probes

Probe	**Location**	**Measures**
Service rate	Main module	Time-varying throughput in transactions/sec
Revenue stats	Intermediate stats	Calculates the mean, variance, maximum, and minimum revenues of all departing requests
Response time stats	Intermediate stats	Calculates the mean, variance, maximum, and minimum response times of all departing requests
Job count	Statistics	Counts the number of requests that depart the system
Total revenue	Statistics	Calculates the total revenue collected from all departing requests

5.4 Simulation Results

We ran simulations for three orders of magnitude of arrival rate, λ, and service rate, μ, for each of the three service orders (FIFO, LIFO-PR, and PS). The plots are shown for varying values of system load, ρ, defined as λ/μ. For the fixed λ plots, the load is varied by decreasing μ. For the fixed μ plots, the load is varied by increasing λ. We restrict our system to load values less than 1 so that the simulation system represents a stable queue/server system.

5.4.1 Fixed Arrival Rate Simulations

We first consider the subset of simulations in which the arrival rate is fixed in each simulation run, and the load is iterated during each single simulation run by varying (decreasing) the service rate, μ.

The arrival rates used in the simulations are listed in Table 5.4. These values were chosen to represent one server with fairly low user demand, two with "average" user demand, and two with heavy user demand.

First, we look at the total revenue that the server earns during each of the five simulation runs. For easier comparison, the revenue plots are normalized by the number of requests served in each simulation. When the revenue is normalized in this manner, the plot is the same as the mean per-request revenue plot.

Table 5.4: Arrival rate values used in the fixed arrival rate simulations

λ (requests/sec)	Requests/day
1	86,400
5	432,000
10	864,000
50	4,320,000
100	8,640,000

In addition to the revenue analysis, we also compare the single-request revenue at the expected response time for various values of λ. For an M/M/1 queue, regardless of service discipline, the expected response time is given by the equation $w = 1/(\mu - \lambda)$. We use the equivalent expression $w = \rho/(\lambda(1-\rho))$ since we have defined the simulation system in terms of ρ and λ. The revenue at this response time is given by e^{-w}.

Figure 5.5 shows the normalized revenue plots for each arrival rate. The revenue at the expected M/M/1 response time is illustrated by the dashed line in each of the plots. In the following discussion, we refer to this plot as the "calculated plot".

All of the plots except for the $\lambda = 1$ plot resemble each other; in particular, the $\lambda = 5$ and $\lambda = 10$ plots are very similar in shape, as are the $\lambda = 50$ and $\lambda = 100$ plots. As λ increases, the calculated plot begins to resemble the FIFO plot more closely. The calculated plot is indistinguishable from the FIFO plots for $\lambda = 50$ and $\lambda = 100$, except for the tail at very high loads, which decays at a slightly faster rate than the tails of the FIFO plots for these simulations. For

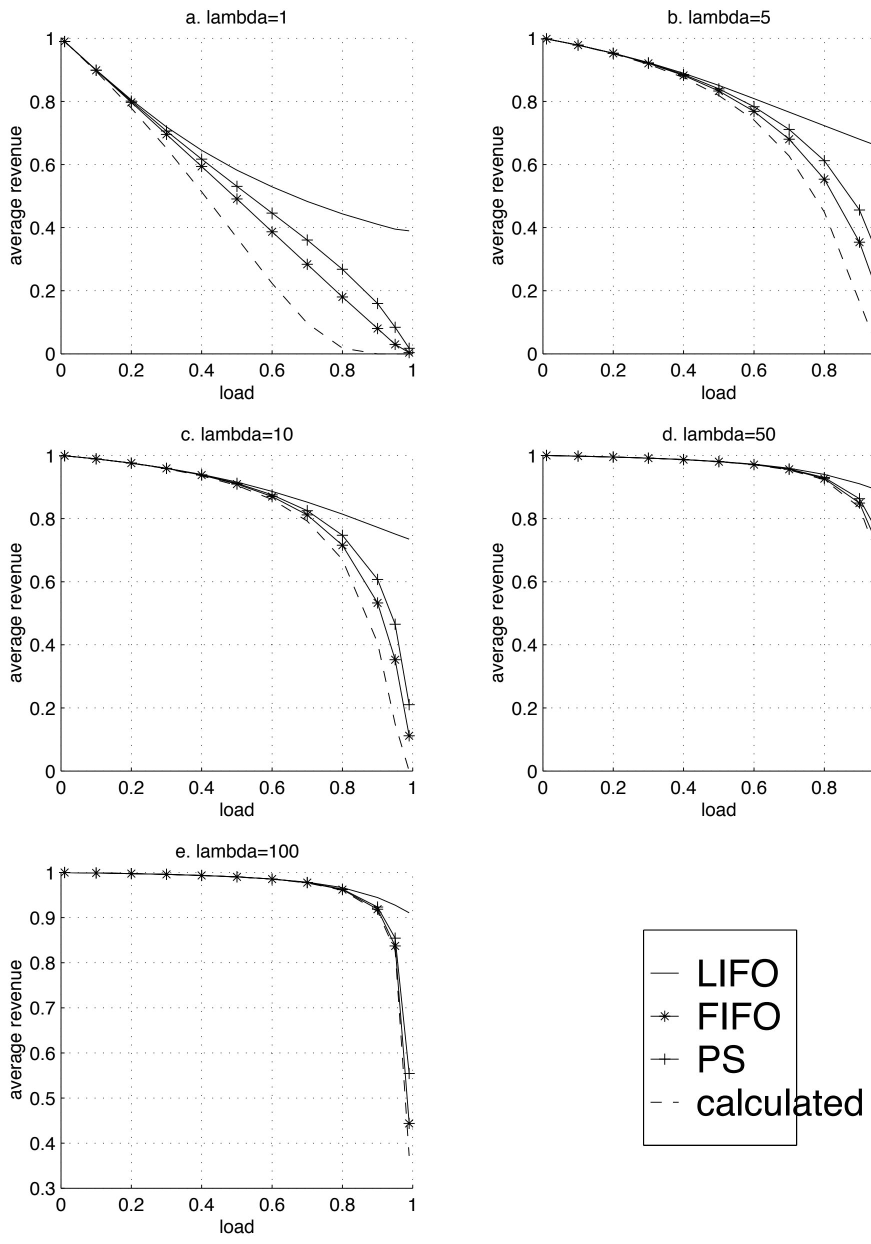

Figure 5.5: The normalized revenue plotted as a function of offered load for each simulation run. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$.

the rest of the plots, the calculated plot performs worse than all of the simulation data plots. In the $\lambda = 1$ and $\lambda = 5$ plots, the calculated plot also shows a tail-off at high values of the load. The tail is very pronounced in the former case and not as pronounced in the latter case. What this tells us is that, particularly at higher values of λ, the expected revenue of the FIFO service order is approximately the revenue evaluated at the expected response time.

At low values of λ, all of the plots decay significantly over the entire range of load values. As the arrival rate increases, we see the plots decaying slowly or holding steady at low loads before they begin to decay more quickly at higher loads. As λ increases, the load at which this period of more rapid decay begins also increases. Higher arrival rates translate into higher service rates at fixed load, which means that jobs are serviced in a shorter amount of time in the simulations with high arrival rates than in those with low arrival rates.

The revenue values are dictated to some extent by the queue length. At low loads, an arrival most likely sees an empty queue; its response time is thus roughly equal to its service time, $1/\mu$. As the load increases, the queue length, which depends on load rather than on the arrival rate and service rate individually, grows proportionally. At these points, the response time depends on the time spent waiting in queue and on the request's service time.

The points at which the queue length begins to grow significantly translate into divergence points on the revenue plots. A divergence point is a load value at which two or more plots start to separate from each other, or diverge, by more than one percent.

For the $\lambda = 1$ plot, there are two divergence points. The first occurs at a load of about 0.12, where the calculated plot begins to decay away from the simulation data plots. The second occurs at a load of 0.2; at this point, the simulation plots begin to separate from each other and the revenue has decayed by approximately 20% from its maximum value. At the maximum load of 0.99, when the service rate is just over 1 request serviced per second, there is a 99% difference between the revenues generated by the LIFO-PR and FIFO policies and 95.5% between the LIFO-PR and PS revenues.

There are also two distinct divergence points in the $\lambda = 5$ plots. The calculated plot diverges from the rest of the plots at a load of 0.3. The LIFO-PR plot then diverges from the FIFO and PS plots at loads of 0.42 and 0.47, respectively. At this point, the revenue has decayed by approximately 10% from the maximum value. At the maximum load of 0.99, the percent difference in revenues generated by LIFO-PR and FIFO is 94%, and the difference between the LIFO-PR and PS revenues is 85%.

When $\lambda = 10$, the divergence points occur when the load equals approximately 0.53 for FIFO and 0.57 for PS, and the revenue has decayed by approximately 9% from the maximum. At the maximum load, there is a 85% difference between the LIFO-PR and FIFO revenues and a 72% difference between LIFO-PR and PS.

At $\lambda = 50$, the FIFO, calculated, and PS plots are practically indistinguishable from each other. (The same is true for the $\lambda = 100$ case.) The LIFO-PR plot diverges from the FIFO and calculated plots at a load of 0.76 and from the PS plot at a load of 0.79. At this point, the revenue has decayed by approximately 4.5% from the maximum value. The maximum percent difference between the LIFO-PR and FIFO revenues is 67%, while between LIFO-PR and PS it is 52%.

The divergence points for the $\lambda = 100$ plot occurs at loads of 0.82 (FIFO, calculated) and 0.83 (PS). At this point, the revenue has only decayed by 4% from its maximum value. The maximum percent difference is 51% between LIFO-PR and FIFO and 39% between LIFO-PR and PS.

The easiest way to explain the differences in the graphs is to understand the differences in response times among the service orders. In all three systems, the expected response time for an individual request is the same; it is the variance that differs. Figure 5.6 shows the calculated and measured variances of the response times for each arrival rate. The variance plots show some discrepancies between the calculated and simulation plots, particularly at low arrival rates. In general, as the arrival rate increases, the differences between the calculated and measured variance plots for all three service orders diminish. The points at which the variance plots diverge away from the x-axis correspond roughly to the points at which the revenue plots diverge.

The measured variance is greatest for LIFO-PR. We would expect, then, the plot of LIFO-PR to show the worst performance. In fact, it shows the best performance, particularly at high loads. Part of this is precisely due to the high variances of the LIFO-PR response times. First of all, since LIFO-PR is preemptive, if a request with a very long response time enters service, the probability is very high that it will be preempted. A shorter job has a better chance of completing quickly, thus registering a short response time with the server. Since the revenue function is exponential, it decays rapidly at first but then slows to a gradual decay within a short time period. Thus, as response times grow, the resulting revenue does not change all that much. The shorter response times seen in LIFO-PR will thus weigh more heavily on the total revenue in the simulation. Compare this with FIFO and PS. FIFO blindly serves all jobs in order of arrival. Thus, longer requests will often cause shorter requests to be delayed, increasing all the response times and thus decreasing the total revenue. In PS, short jobs do often complete before larger jobs, but since they must share the processor with all jobs, their response times are proportional to both their required service times and the number of pending requests (since this dictates what fraction of the processor each request gets).

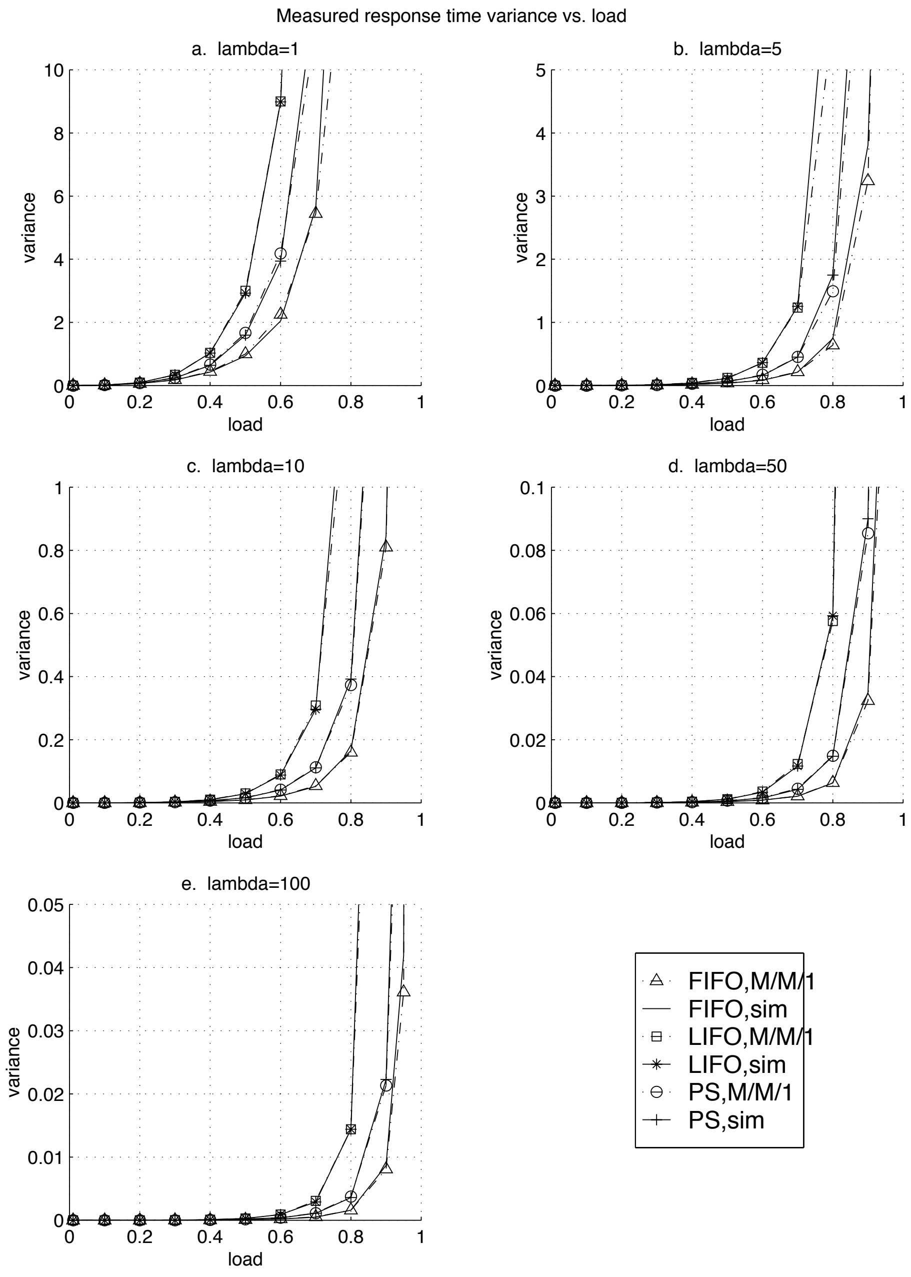

Figure 5.6: Response time variance, calculated and measured. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

At high loads, then, we expect response times to increase (and thus revenues to decrease) when PS is used.

To verify this, we compare the theoretical M/M/1 mean response time as a function of load with the response times corresponding to the points on the expected revenue plots for each service order. The response time is calculated using the equation $w = -\ln(\text{revenue})$. These values give a sense of the "average age" of a request in each of the simulation runs. The theoretical mean response times and "average ages" from the simulation data are plotted in Figure 5.7. The plot is zoomed in to show the behavior of the simulation data, which does not reach as great of a maximum value as the calculated data. The M/M/1 plot is labeled "calc" in the figure.

At lower arrival rates, the calculated response time blows up towards infinity as the load approaches 1. For the higher arrival rates, the maximum calculated response time is fairly small in comparison (several orders of magnitude smaller). This explains why the total revenue plots do not decay as drastically in the higher arrival rate plots.

In all five cases, the FIFO and PS plots show similar shapes. As the arrival rate increases, the two plots become indistinguishable from each other. The M/M/1 response time plot also shows the same behavior, although at the lowest arrival rate it blows up much faster than both the FIFO and PS plots.

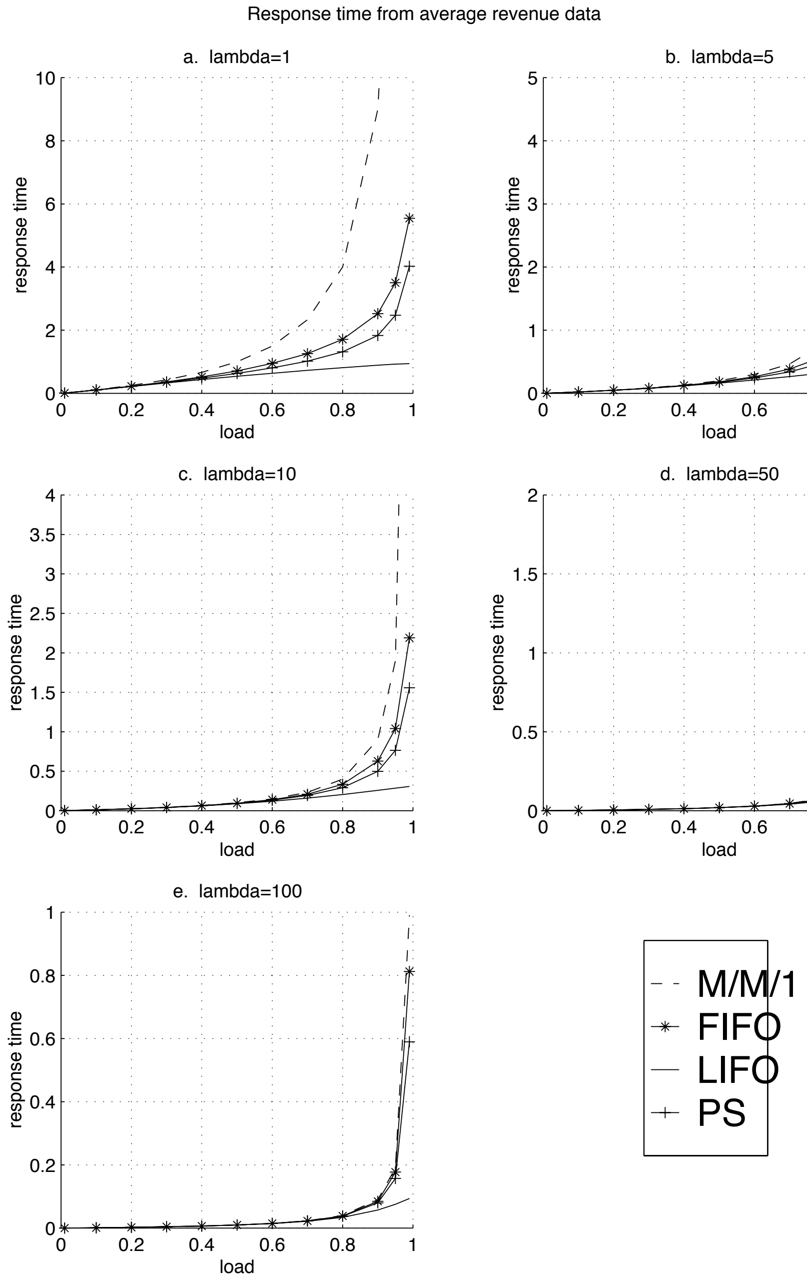

Figure 5.7: Response time calculated at expected revenue for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

The LIFO-PR plot has a much different shape that the other three plots for all values of λ. In particular, it does not blow up as the load increases. In fact, as the load increases, the growth stays fairly small. What this shows is that the average response time measured for a request departing the LIFO-PR system is not only much less than for the same request leaving the FIFO or PS system, but it is also fairly uniform regardless of the offered load at the system.

As a final analysis of this revenue data, we calculate the service rate at which the normalized revenue falls below 0.9. This corresponds to the server obtaining 90 percent of the maximum possible revenue it could earn by serving the given arrival stream.

Note that for LIFO-PR, the revenue is always greater than 0.9 at $\lambda = 100$. The minimum service rate in this set of simulation data is 101.01. If we extrapolate the 0.9 crossing point from this plot, the corresponding value of μ is approximately 91 requests/sec.

Figure 5.8 illustrates the relationship between the arrival rate and the corresponding minimum service rate required for 90% revenue collection. The PS and FIFO plots are nearly linear over the entire range of λ. At the smaller values of λ, the slope of these two lines are each slightly greater than 1 (1.15 and 1.12 for FIFO and PS, respectively). As λ increases towards 100, the slope for FIFO approaches and then falls very slightly below 1. The slope for PS remains around 1 for values of λ from 5 to 100. There is an offset of about 9 in both plots.

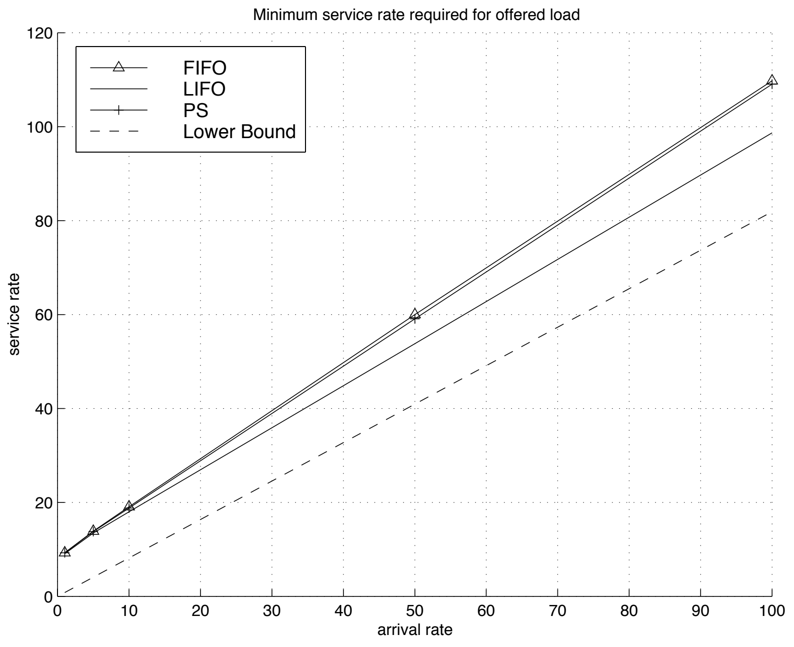

Figure 5.8: Minimum service rate required to guarantee that the server will earn 90% of its expected revenue stream, as a function of arrival rate.

The LIFO-PR plot, however, shows two distinct slopes. In the first interval (λ between 1 and 5), the slope is about 1.1; in the second interval (λ between 5 and 100), the slope decreases to about 0.9. The offset again is about 9.

What these plots tell us is that to maintain a given level of service, the minimum required service rate rises in direct proportion to the arrival rate. The increase depends also on the arrival rate: as the arrival rate gets large, the subsequent increase in service rate is not as dramatic as for lower values of arrival rate. The plots also show that we can support the same amount of traffic at a 9% smaller processing speed if we use LIFO-PR as opposed to either of the other two service orders (for all possible arrival rates).

The lower bound of the 90% revenue plot is also plotted on this graph as the dashed line. The lower bound is the service rate required for the server to process enough customers to earn 90% of the potential revenue. We can see that as the arrival rate increases, the LIFO-PR plot slowly approaches this lower bound; however, the LIFO-PR plot levels off at higher values of λ. None of the simulated data plots ever has a slope close to that of the lower bound, approximately 0.82.

5.4.2 Fixed Service Rate Simulations

We next consider the subset of simulations in which the service rate is varied among simulation runs, and the load is iterated during each single simulation run by varying (increasing) the arrival rate, λ. Three values are used for μ: 10 requests/sec, 100 requests/sec, and 1000 requests/sec. These values were chosen to represent one "slow" server, one "average"-speed server, and one "fast" server.

We look at the total revenue that the server earns during each of the five simulation runs, both normalized and unnormalized. In this section, we are interested in both the normalized and unnormalized revenue plots, because unlike the previous case, the shapes of these plots are not at all similar.

The unnormalized total revenue plots are shown in Figure 5.9. The shape of this set of plots is much different than the fixed λ plots and the normalized fixed μ plots, because in this set of simulations the number of requests serviced per simulation run varies with the load. In the fixed λ plots, the same number of users arrive per simulation regardless of the load—variations in the load reflect variations in the rate of service seen by the incoming requests. In the fixed μ plots, however, variations in the load reflect variations in the arrival rate.

The shapes of these curves are very distinctive. All three plots increase over a portion of the load. Eventually, the rates of increase of the FIFO and PS plots slow; the revenue reaches a peak value and then drops rapidly. However, the LIFO-PR plot continues to increase over the entire range of load, although the growth does slow at higher values of load.

Each μ plot is characterized by one or two points. The first point is the load value at which the three plots diverge; that is, where the FIFO and PS plots start to curve away from the LIFO-PR plot. The second point is the point at which the FIFO and PS plots "peak". (We find that there are some connections between these points and the shape of the normalized revenue plots. These connections are explored later in this section.)

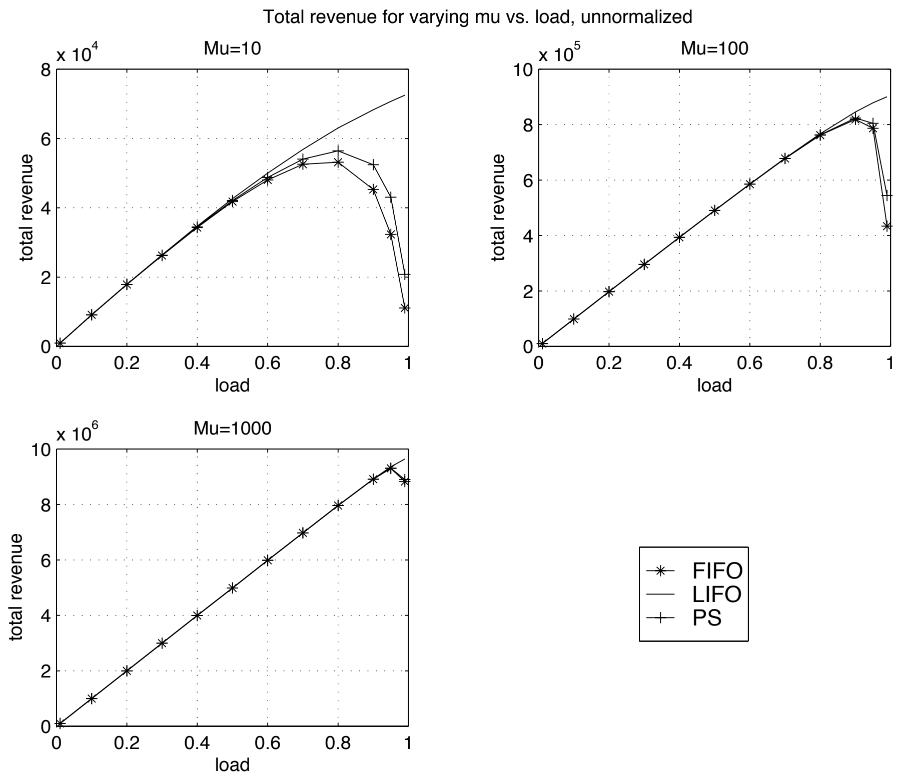

Figure 5.9: The unnormalized total revenue plots for the fixed service rate simulations. (a) $\mu = 10$ (b) $\mu = 100$ (c) $\mu = 1000$

For the $\mu = 10$ plot, the divergence point occurs at a load of about 0.5. The two curves peak at a load of 0.8. Beyond 0.8, these two plots drop sharply, with FIFO falling off slightly more sharply than PS (except at loads of 0.95 and greater, where the rate of decay of the PS plot is actually greater than that of the FIFO plot). The growth of the LIFO-PR plot starts to slow very slightly at a load of about 0.8, but this plot does not decay as the other two do; it is always increasing, even at the highest load values. The maximum percent difference at the maximum load of 0.99 is about 85% between the LIFO-PR and FIFO revenues, and 72% between the LIFO-PR and PS revenues.

For the $\mu = 100$ plot, the divergence point occurs at a load of approximately 0.8. The peak occurs at a load of 0.9, after which the FIFO and PS plots rapidly decline, with the FIFO plot decaying slightly faster than the PS plot. Again, growth of the LIFO-PR plot slows very slightly after the load hits 0.9; as in the previous case, it continues to increase over the entire range of load. There is a 52% difference between the LIFO-PR and FIFO revenues at the maximum load, and a 40% difference between the LIFO-PR and PS revenues.

The $\mu = 1000$ plots diverge at a load of about 0.92 and peak at about 0.95. Instead of the round peak, the FIFO and PS plots show a sharp peak and then a small decline, which may be larger if we extend the range of load out farther. The LIFO-PR plot shows the same increase, with again a very slight slowdown at the same load as the peak value. Again, the revenue decrease for FIFO is slightly greater than that for PS. The percent difference at the maximum load of 0.99 is 8.4% between LIFO-PR and FIFO and 7.6% between LIFO-PR and PS.

The normalized total revenue plots are presented in Figure 5.10. In addition to the plots of the simulation data, we also include a plot of the revenue at the expected response time in an M/M/1 system. This plot is represented by the dashed line. We use the equivalent expression $w = 1/(\mu(1-\rho))$ to calculate the response time of a M/M/1 system.

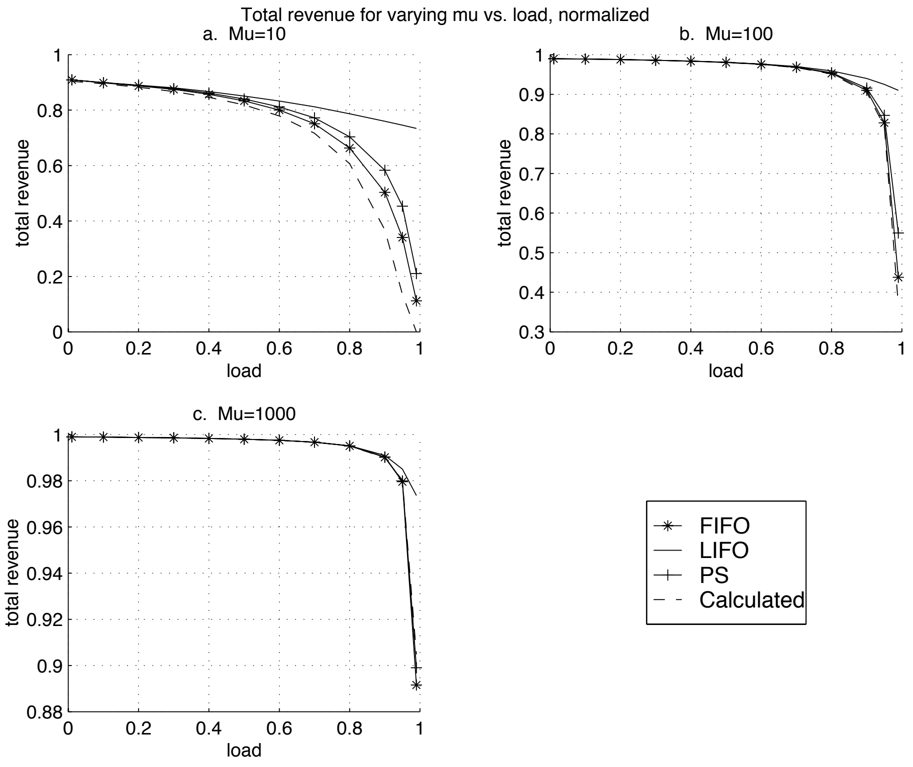

Figure 5.10: The normalized total revenue plots for the service rate simulations. (a) $\mu = 10$ (b) $\mu = 100$ (c) $\mu = 1000$

As in the λ simulations, as μ increases, the closer the calculated plot resembles the simulation data plots, particularly the FIFO plot and to a lesser extent the PS plot.

The $\mu = 100$ and $\mu = 1000$ plots resemble each other very closely. The shape of the $\mu = 10$ plot is slightly different, and has a lower maximum revenue value (0.9). The lower maximum is probably due to the larger minimum average service time; at low loads, the average service time, $1/\mu$, dominates the response time, since there are likely very few if any jobs in queue. As μ increases, the average service time decreases; and thus the average response time decreases.

In the $\mu = 10$ plot, we see two divergence points, similar to the behavior of the λ plots. The first divergence point occurs at a load of about 0.3, when the calculated plot diverges from the simulation data plots. The second point occurs at a load of about 0.45, where the LIFO-PR plot diverges from the FIFO and PS plots. The calculated plot has the same shape as the FIFO and PS plots, but is always less than both of these plots above the first divergence point.

In the $\mu = 100$ and $\mu = 1000$ plots, the calculated plot is indistinguishable from the FIFO plot. Thus, there is only one divergence point on each of the graphs. This point is 0.8 for the former and 0.9 for the latter.

Notice that the peaks on the unnormalized plots occur at the points where the rapid decay starts. The unnormalized plot increases up until this point, even though the normalized plot shows that the growth is flat to slightly negative. As the growth of the normalized plots become negative, the increases on the unnormalized plots slow down. The reason for this behavior is that even though the revenue per-request may be decreasing, the number of requests serviced per simulation is increasing; the large increase offsets the smaller decrease in mean per-request revenue. Eventually, however, the decay catches up to the increase in requests serviced, and results in a decay in the unnormalized plot. The LIFO-PR plot does not exhibit this behavior because the decrease is slow and gradual over the entire range of load, and is thus always offset by the increase in number of requests served.

5.4.3 Concluding Remarks

The simulation data bears out the results obtained mathematically in Chapter 3. More importantly, we find that the LIFO-PR service ordering is very stable, even under the highest offered loads at very heavily-accessed servers. At the highest loads, LIFO-PR outperforms FIFO by up to 99% and PS by up to 95%. LIFO-PR thus appears to be a good design choice for web server architecture.

Chapter 6

SIMULATION ANALYSIS FOR THE EXTENSIONS TO THE ORIGINAL WWW SERVER MODEL

6.1 Introduction

In this chapter, we discuss a series of simulation models created using BONeS software, corresponding to the proofs presented in Chapter 4. The first set of simulations corresponds to the first proof extension, where the service times are exponentially distributed with parameter μ and each incoming request is assigned an initial reward, C_i, that is uniformly distributed between 0 and 1. The second set of simulations corresponds to the second proof extension, where each incoming request is assigned an initial revenue of 1.0 and the service times are drawn from i different exponential distributions, each with parameter μ_i. The third set of simulations is for a hybrid system combining the two proof extensions, where the service times are drawn from various exponential distributions and the initial reward varies among the requests. Finally, we briefly look at a simulation corresponding to the counterexample presented in the "disproof" of Conjecture 4.3.1, where we found that a processor-sharing policy generated a higher expected revenue than a set of non-processor-sharing policies for at least one sample path.

Each of the sets, with the exception of the final simulation, consists of two separate models: one implementing the optimal service ordering policy for that particular system, and one implementing the traditional service ordering policies.[1] The optimal policy for both proof extensions depends not only on the age of the jobs awaiting service, but also on either the initial rewards of each job awaiting service or the expected service time of each job. We refer to these policies as "parameter-based" (PB) policies. Both the traditional and PB models will be presented in each of the sets, along with key simulation results.

[1] "Traditional" in this context refers to FIFO, LIFO-PR, and PS.

We maintain separate models for the PB and traditional policies because the PB policies require a more complex queue/server model that is not necessary in the traditional service order systems. Rather than sending the web requests in the traditional service orders through unnecessary processing, we chose to maintain the original queue/server system for these service orders, with minor modifications. The differences between the two models will be described in detail in the sections below.

6.2 Simulation Set 1: Varying Initial Reward

In this section, we present the models associated with the system where service times are independent and exponentially distributed with parameter μ and where the initial rewards are drawn from a uniform distribution with endpoints [0,1). The optimal policy for this system, as derived in Chapter 4, bases its decision as to which request to serve next in any given time interval on the pending request with the highest c_i value, where c_i is the product of initial revenue and revenue decay so far.

6.2.1 Simulation Model

The two models for this system appear in Figures 6.1 and 6.2. Figure 6.1 shows the simulation model for the "traditional" service orderings (LIFO-PR, FIFO, and PS), while Figure 6.2 illustrates the parameter-based simulation model for this system, which we will refer to as the PB-1 model.

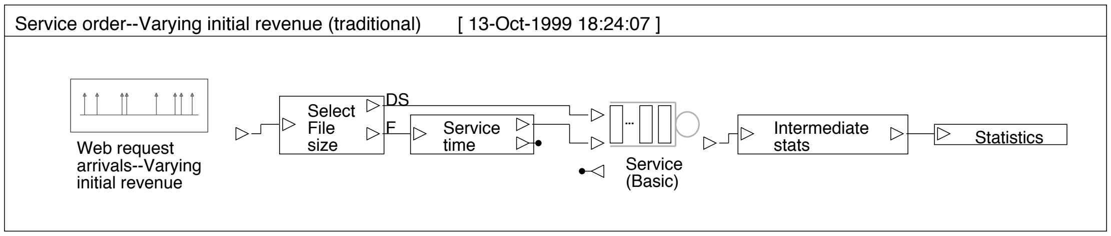
Service order--Varying initial revenue (traditional) [13-Oct-1999 18:24:07]
Web request arrivals--Varying initial revenue
Select File size
DS
F
Service time
Service (Basic)
Intermediate stats
Statistics

Figure 6.1: The BONeS model of the traditional service ordering system for simulation set 1.

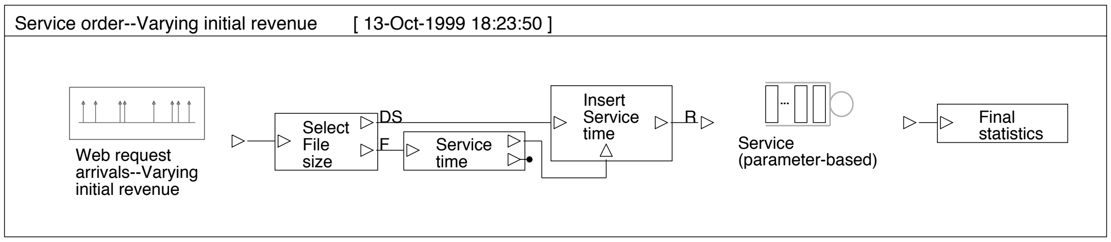
Service order--Varying initial revenue [13-Oct-1999 18:23:50]
Web request arrivals--Varying initial revenue
Select File size
DS
F
Service time
Insert Service time
R
Service (parameter-based)
Final statistics

Figure 6.2: The BONeS model of the PB-1 system.

The traditional model is identical to the original system model except for the request generator portion. The reader is asked to refer to Section 5.2 for descriptions of the unchanged portions of the model. The PB-1 model is similar in structure to the traditional model; however, it contains sufficient variation that the entire model will be discussed in detail.

Request Generator

The request generator consists of the ***Web request arrivals—Varying initial revenue*** block, pictured in Figure 6.3. It is the same in both the traditional and the PB-1 models.

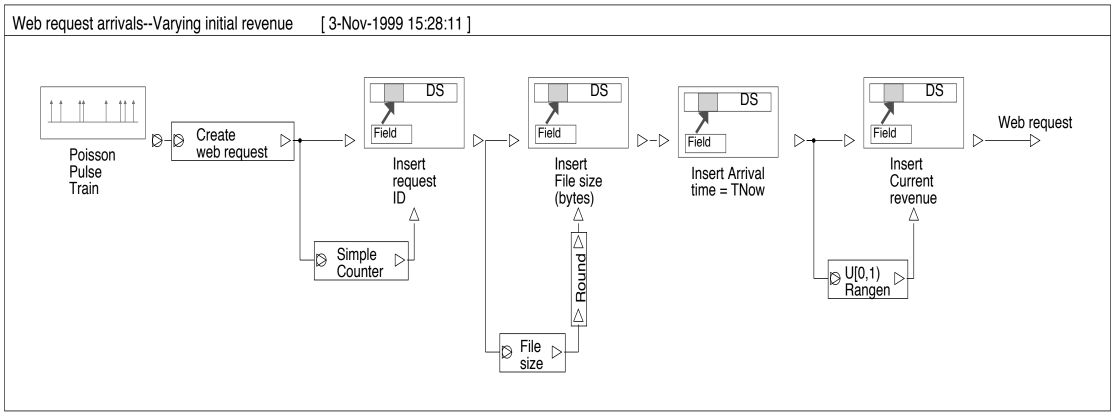

Figure 6.3: The request generator module for the first set of simulations

The request generator is very similar to the request generator in the original model. The main difference is that there is an additional module used to generate the initial reward for each of the generated web requests, denoted by the uniform random number generator block (***U(0,1) Rangen***) in the figure.

The values assigned to the fields in the data structure are also the same as in the original system with a few exceptions. For example, the current revenue is set to the value generated by the initial reward uniform random number generator. Also, while the service time field is unused in the original simulation model and is also unused in the traditional service order model here, it is used by the parameter-based model to store the service time requirement of each request. (The reasons for this modeling difference will be apparent shortly.) The same parameters are deferred until simulation run time as in the original system, with the exception of the *Initial revenue* parameter, which is set via the uniform random number generator here.

Service Time Generator (PB-1 model)

The service time generator includes the ***Insert service time*** block in the PB-1 model. The file size set in the request generator module is used here to calculate the service time. The service time is calculated by dividing the file size by the server's *per-byte processing rate*, another simulation parameter specified at run time. It is inserted manually into the data structure in the PB-1 model because the server keeps track of remaining service time explicitly to determine if a job has completed its service yet. This detail of operation is hidden in the traditional queue and server module, making this block unnecessary there.

Queue/Server

The queue in the PB-1 model, labeled as ***Service (parameter-based)*** in Figure 6.2, is more complex than in the traditional model because it must consider the current state of the system at each arrival or departure event and reevaluate its service decisions at these times. The parameter-based queue/server is illustrated in Figure 6.4. It is composed of several logical parts: an arrival queue, a service queue, a preemptive server, state calculations, and departure/preemption processing. Most of these portions are involved either directly or indirectly with calculating or maintaining the state vector associated with the queue and server.

The state of the system at any time, quantified by its state vector (see Equation (4.4)) is defined by the current revenue value of each request in the system at that time. Each time an arrival or departure event occurs, the state vector must be recalculated.

When an arrival occurs, it is placed in a separate arrival queue until it can be placed in the service queue or directly into service. If there is no job currently in service and the service queue is empty, the new arrival goes straight to the server. Otherwise, the model compares the new request's initial reward to the current revenue of the job in service. If the new request's revenue is greater than that of the request in service, the server preempts the request in service, places it at the head of the service queue, and starts processing the new request. If the

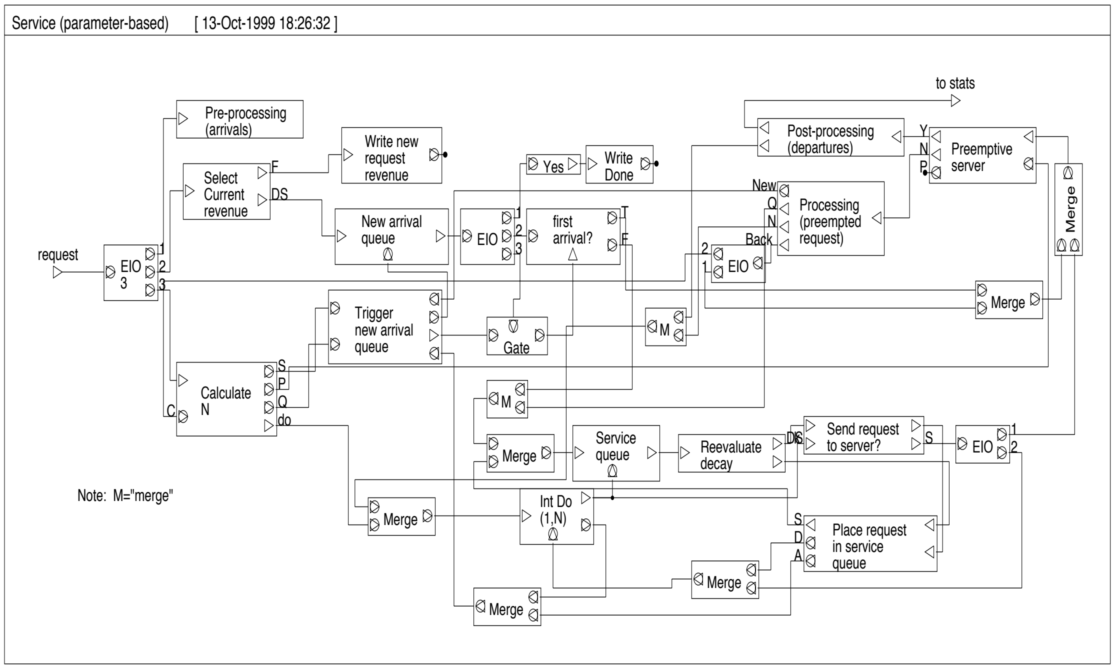

Figure 6.4: The parameter-based queue and server module.

new request's revenue is less than that of the request in service, the module then recalculates the current revenues of all requests waiting in the service queue and determines where to place the new request in this queue. Note that the module does not have to reorder the requests that are already in the service queue, because the decay rates of all of the requests are uniform.

When a departure occurs, the server triggers the service queue to send the request at the head of the queue into service. The module also updates the current revenues of all requests in the system at this point.

The ***Post-processing (departures)*** and ***Pre-processing(arrivals)*** blocks calculate the revenue decay by which to multiply all of the remaining requests in order to update each request's current revenue. They do so by computing the time since the last arrival or departure event and evaluating the revenue function (without the leading multiplier) at this time.

In order to perform these calculations, the module must keep track of certain parameter values in memory. These values include the type of the most recent event (arrival or departure), the time since the last arrival or departure event, the current number of requests awaiting service, and the initial reward of the incoming request. These parameters are stored as internal memory parameters because they are used in multiple calculations at several points in the system. Unfortunately, this has the unwanted side effect of slowing down the simulation run considerably compared to the traditional model, particularly at high arrival rates and high load values.

There are two parameters whose values are deferred until simulation run time. These parameters are the maximum queue size, which indicates the maximum number of elements allowed in the “service queue”; and the decay rate of the revenue function, labeled as the “cost factor”.

Statistics

The PB-1 model combines all of the statistics collection into one module, labeled ***Final Statistics***, shown in Figure 6.5. The main difference between this module and the ***Intermediate Stats/Statistics*** module combination is the absence of the current revenue calculation, which now appears in the ***Post-processing Departures*** block in the ***Service (parameter-based)*** module. The response time is still calculated in this module, as are the total job count and the total simulation revenue values.

6.2.2 Simulation Parameters

There are two sets of parameters used, one for the traditional model and one for the PB-1 model. The simulation parameters used in the traditional model are

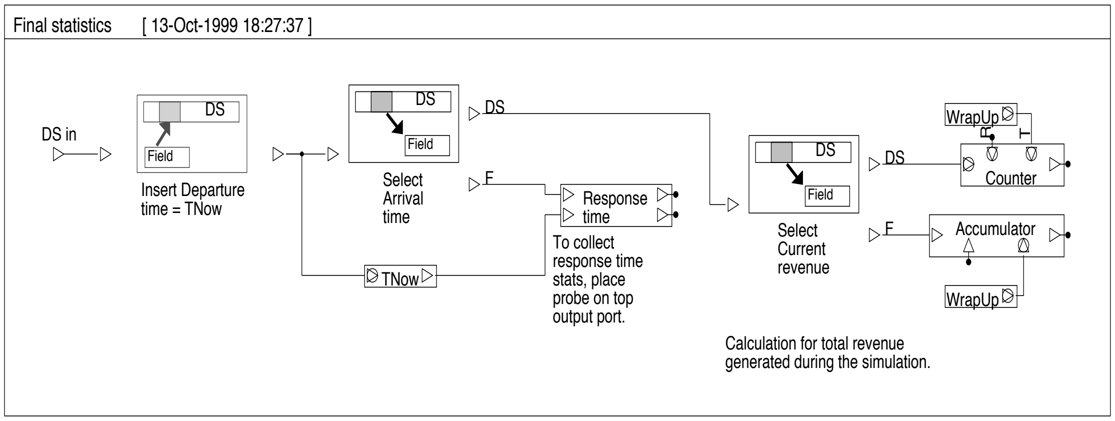
Final statistics [13-Oct-1999 18:27:37]
DS in
DS
Field
Insert Departure time = TNow
DS
Field
Select Arrival time
DS
F
Response time
To collect response time stats, place probe on top output port.
TNow
DS
Field
Select Current revenue
DS
F
WrapUp
Counter
Accumulator
WrapUp
Calculation for total revenue generated during the simulation.

Figure 6.5: The **Final statistics** module from the PB-1 system model.

the same as those listed in Table 5.2. The one exception is the *Initial revenue* parameter, which is not used here since it is set explicitly in the ***Web request arrivals...*** module.

Table 6.1 lists the simulation parameters used in the PB-1 model. Some of these parameters are similar to the parameters used in the traditional model. Notably absent are the queue and service-ordering-based parameters, which were connected to the ***Service (Basic)*** block.

Probes

Both models contain the same probes as each other, and the same probes as those in the original simulation model. Refer to Table 5.3 for a list of these probes. The probes are in the same locations in the traditional model as they

Table 6.1: Simulation parameters, simulation set 1

Parameter	**Value**
Service rate (μ)	λ/ρ
Load (ρ)	varied between 0.01 and 0.99
First request arrival time	0.0
Mean interarrival time	$1/\lambda$, $\lambda \in \{1, 5, 10, 50, 100\}$.
Mean file size	2000 bytes
Per-byte processing rate	Service rate times mean file size
Cost factor	1.0
Maximum queue size [a]	10,000
TSTOP	the simulation end time; set to 10,000 sec.

[a]This value is set explicitly in the Queue/Server model to the same value.

are in the original model. The placement of the probes in the parameter-based model is slightly different. The *Service rate* probe is placed inside the ***Service (parameter-based)*** module. The rest of the probes are placed inside the ***Final Statistics*** module.

6.2.3 Simulation Results

The simulations tested four service orderings: the three "traditional" ones (FIFO, PS, and LIFO-PR), and the optimal service ordering, PB-1. The simulations were run for three orders of magnitude of arrival rate (λ) as in the original set of simulations. The plots are shown for varying values of system load, $\rho \overset{\triangle}{=} \lambda/\mu$. The load is varied by decreasing the service rate (μ).

We first look at the plots of normalized total revenue as a function of load to establish general trends in the data. Next, we illustrate the trends in the revenue plots with plots of the mean and variance of the measured response times, and compare these values to the expected values for an M/M/1 queue. We use the equation $w = \rho/(\lambda(1-\rho))$ to determine the mean M/M/1 response time. Note that we must scale the calculated revenue values by 0.5, the mean of the distribution from which the initial rewards are drawn, in order to compare them to the simulation data values.

Figure 6.6 shows the normalized revenue as a function of load for all of the possible arrival rates. The shapes of these plots again are similar to the normalized revenue plots from the first set of simulation data. The maximum values for each plot are around 0.5, the mean of the initial reward distribution. We also plot $0.5e^{-w}$, the revenue calculated at the expected M/M/1 response time, and denote this plot as the "calculated" plot in the discussion below.

The traditional service order plots and the calculated plots are the same as the plots from the original system scaled by 0.5. The reader is asked to refer to Section 5.4.1 for a discussion of the behavior of these plots. We concentrate here on explaining the relation of the parameter-based plots to the traditional and calculated plots.

The PB-1 plot adds a fourth divergence point to the previous plots. In all cases except for $\lambda = 1$, the "traditional" plots diverge away from the PB plot before they diverge from each other. When $\lambda = 1$, because the maximum revenue is actually slightly greater than 0.5, the calculated plot underperforms the rest of the plots over the entire range of load, and thus does not have a divergence point. If we look at the percent differences between the PB plot and the traditional plots, we see that they are slightly larger than in the original system. Table 6.2 lists the three divergence points (recall these are the points at which the percent difference between PB and the rest hits 1%). In the table, ρ_1 indicates the load at which the

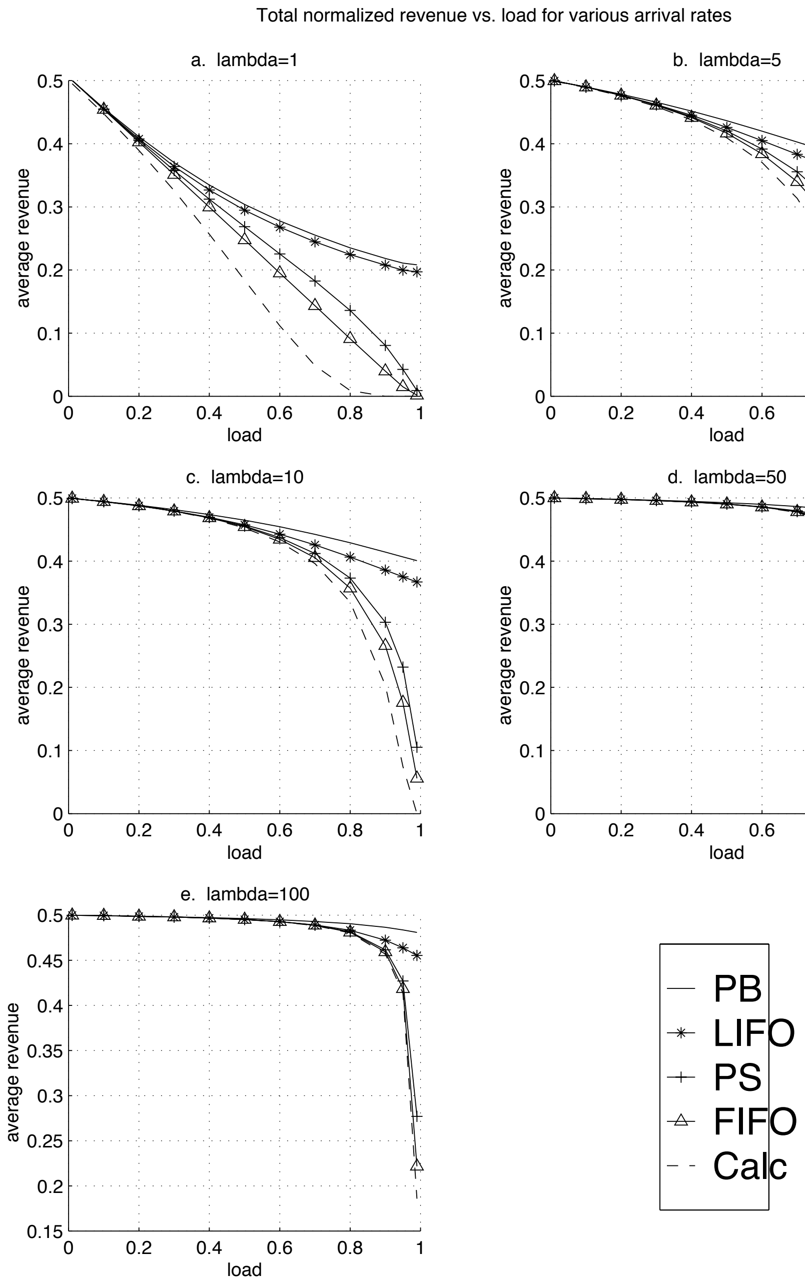

Figure 6.6: Total normalized revenue as a function of load for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

percent difference between the PB-1 plot and the FIFO plot exceeds one percent; ρ_2 and ρ_3 indicate the same load for the PS and LIFO-PR plots, respectively.

Table 6.2: Divergence points for the normalized revenue plots

	Divergence points		
λ	ρ_1	ρ_2	ρ_3
1	0.14	0.16	0.22
5	0.29	0.31	0.33
10	0.38	0.40	0.42
50	0.61	0.62	0.64
100	0.71	0.72	0.73

As the arrival rate increases, the difference between the PB-1 and LIFO-PR plots at high loads increases. At $\lambda = 1$, the difference between the two plots is very small. Once λ increases above five, the differences between the two become more pronounced, although overall the two have similar shapes and behavior.

As in the previous set of simulations, the theoretical mean response time is the same as the measured response times for all values of lambda, so we will not show these plots here.

Figure 6.7 plots the response time variances, both calculated and measured, for each of the arrival rates. The plots have been zoomed in to show regions closer to the load axis. The plots labeled "M/M/1" refer to the calculated variances for an M/M/1 system for the LIFO-PR, FIFO, and PS service orders. In all cases the maximum variance is achieved by the M/M/1 LIFO-PR plot.

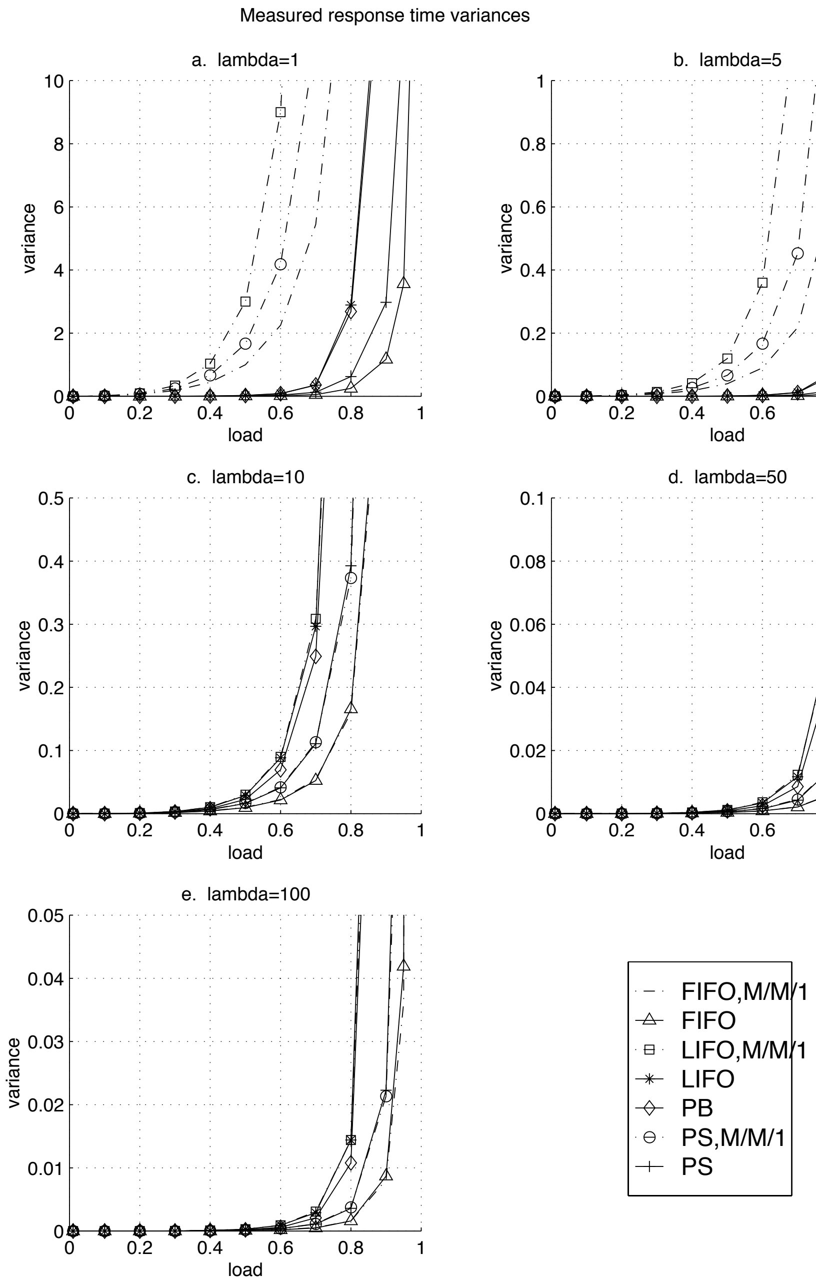

Figure 6.7: Variance of response times for simulation set 1, simulation data and M/M/1. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

Interestingly, the PB-1 plot is nearly identical to the LIFO-PR plot for all values of λ. At $\lambda = 1$ and $\lambda = 5$, the plots are identical except at a load of 0.99. Note that at low revenues, there is a great discrepancy between the calculated variances and the simulation data; this difference disappears as the arrival rate increases. The LIFO-PR and PB-1 plots have the highest variances of the simulated data sets for all values of arrival rate.

To illustrate the effect that variance has on the revenue earned by each service order, we calculate the response times from points on the expected revenue plots via the equation $w = -\ln(\text{Revenue})$. Figure 6.8 shows these plots for all values of arrival rate, zoomed in to the area closest to the x-axis to show the most interesting plot detail. As a point of comparison, we plot the mean response time for an M/M/1 system; it appears as the dashed line in the figure. As expected, the PB-1 plot and the LIFO-PR plot show similar behavior for all values of λ. The differences between these two plots are always very small, with the PB-1 system showing a slightly lower calculated response time than the LIFO-PR system, particularly at high loads. As explained before, this plot gives us a sense of the "average age" of a paying request; we expect the "average ages" seen in the PB-1 and LIFO-PR configurations to be lower due to the higher variability in response times, as explained in Chapter 5.

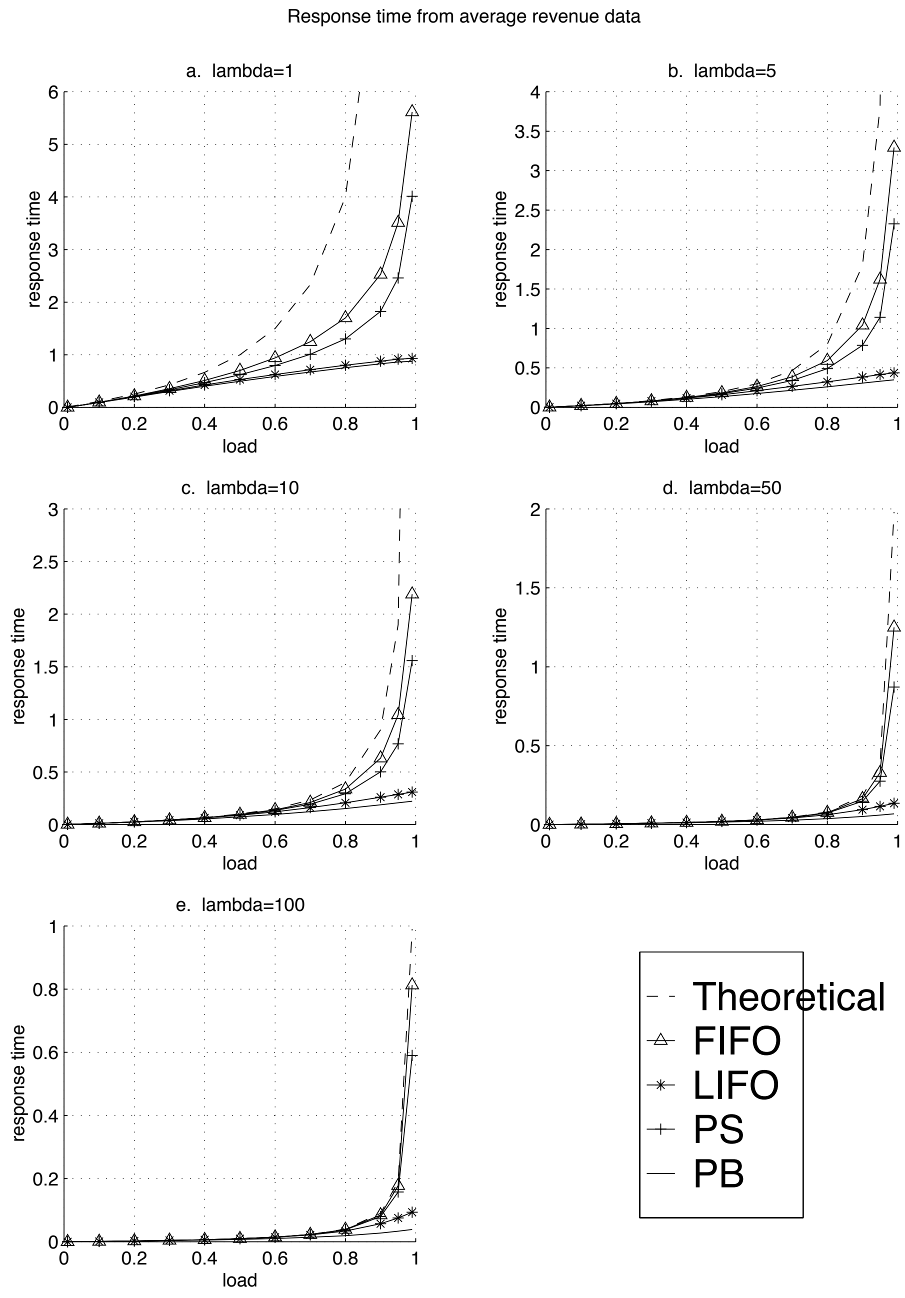

Figure 6.8: Response times calculated at the expected revenue points from the simulation data in simulation set 1. The expected response time for an M/M/1 system is also plotted as a point of comparison.

As in the original set of simulations, we are interested in the minimum service rate needed by the server to ensure that it will earn at least 90% of the possible revenue presented to it at various arrival rates. This data is plotted in Figure 6.9. Note that the values for PB-1 at $\lambda = 50$ and $\lambda = 100$, as well as the value for LIFO-PR at $\lambda = 100$, are extrapolated from the available data, since the server in these cases earns better than 90% of the total possible revenue on average.

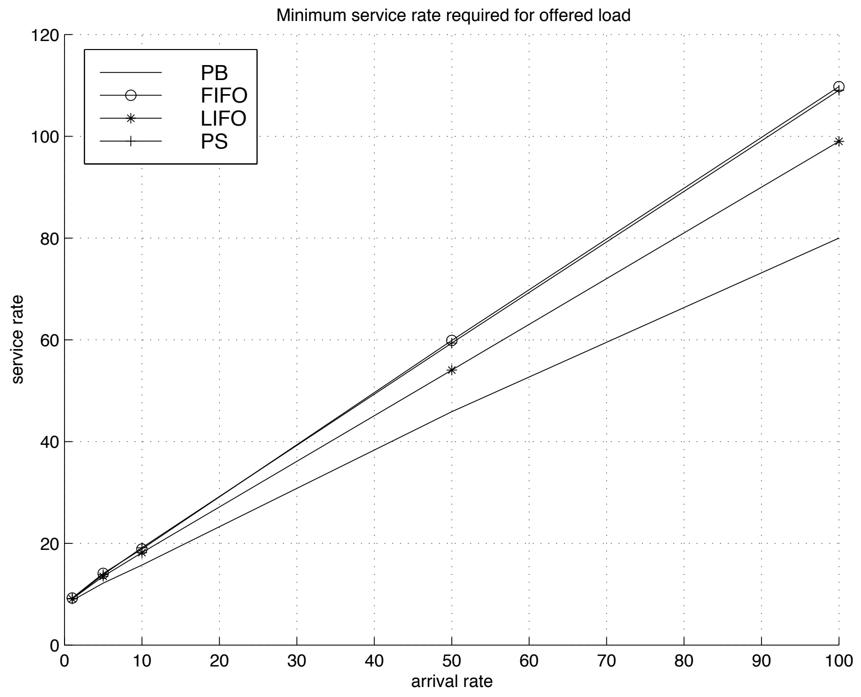

Figure 6.9: Minimum service rate required for the server to collect about 90% of its total possible revenue

All of the simulation data plots are offset by 9 and show linear behavior. The FIFO and PS plots are nearly identical, with minor variations in the slopes. The slope of the LIFO-PR plot is approximately 0.9 at higher arrival rates. The PB plot shows the lowest μ/λ ratio of the four simulation plots, with a slope of about 0.7 at high arrival rates. In fact, the capacity required by the PB-1 configuration to support the highest arrival rates is about 20% lower than the same capacity required by the LIFO-PR configuration and almost 40% lower than the capacity required by the FIFO and PS configurations at the same arrival rate.

6.2.4 Concluding Remarks

We find that the performances of the PB-1 and LIFO-PR configurations for this system are very similar. This result is expected, since PB-1 reduces to LIFO-PR when all incoming requests are assigned the same initial reward. PB-1 service improves upon LIFO-PR service by not only serving the newest job, but weighing this measure by the initial reward offered by a particular job. Both PB-1 and LIFO-PR show excellent performance even at the highest server loads. While the original simulations demonstrated that LIFO-PR is a good choice for a heavily-loaded server, this set of simulations shows that PB-1 represents an even better choice when initial rewards among requests vary.

6.3 Simulation Set 2: Varying File Size Distributions

In this section, we present the models corresponding to the system with service times drawn from varying exponential distributions and with all incoming requests assigned the same initial reward. The optimal policy for this system bases its decision as to which request to serve next in any given time interval on the pending request with the highest $c_i\mu_i$ product, where c_i is the revenue decay so far and $1/\mu_i$ is the expected service time of job i.

To implement these models, we define three unique exponential distributions with parameters μ_1, μ_2, and μ_3, respectively. Each of these distributions corresponds to a different file "type" found at a web server. We define distribution 1 to correspond to HTML files; distribution 2 to correspond to image files; and distribution 3 to correspond to "other" large files, such as CGI scripts, multimedia files, and the like. Incoming requests access each distribution with some probability $p_i, i \in 1, 2, 3$, such that $p_1 + p_2 + p_3 = 1$.

6.3.1 Simulation Model

The traditional model is shown in Figure 6.10 and the proposed optimal model, which we will hereafter refer to as the PB-2 model, is pictured in Figure 6.11.

Both the traditional model and the PB-2 model are the same as the models in the previous section, except for the request generator and the statistics portions. Only these portions of the model will be described here.

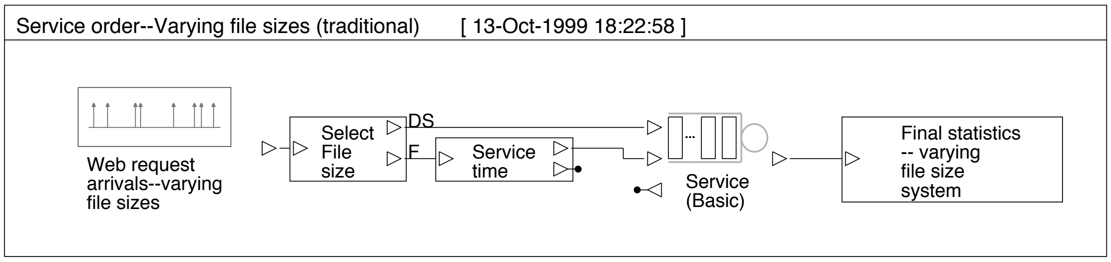
Service order--Varying file sizes (traditional) [13-Oct-1999 18:22:58]
Web request arrivals--varying file sizes
Select File size
DS
F
Service time
Service (Basic)
Final statistics -- varying file size system

Figure 6.10: The BONeS model of the traditional system for simulation set 2.

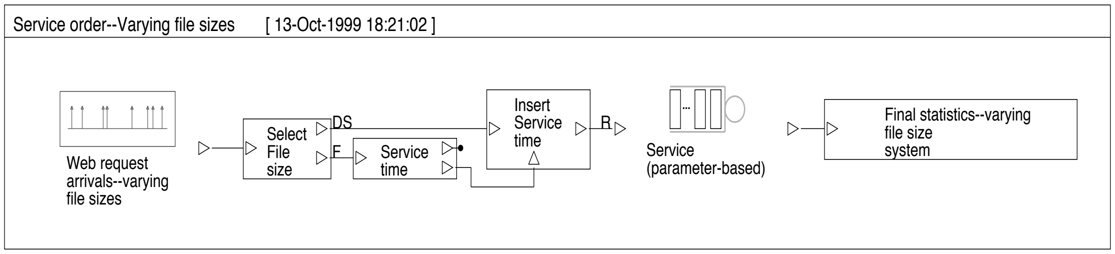
Service order--Varying file sizes [13-Oct-1999 18:21:02]
Web request arrivals--varying file sizes
Select File size
DS
F
Service time
Insert Service time
R
Service (parameter-based)
Final statistics--varying file size system

Figure 6.11: The BONeS model of the PB-2 system, simulation set 2

Request Generator

The request generator consists of the ***Web request arrivals–Varying file sizes*** block, pictured in Figure 6.12. It is the same in both the traditional and the PB-2 system models.

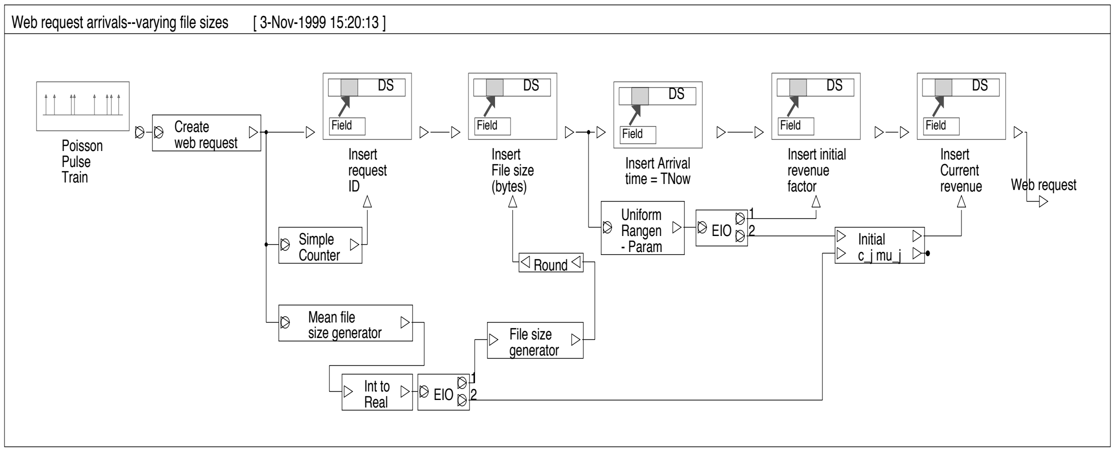

Figure 6.12: The request generator module for simulation set 2

The request generator in this model is similar to the request generator in the original model. The main differences lie in the generation of file sizes and initial reward values.[2]

The ***Mean file size generator*** module, pictured in Figure 6.13, determines which file type is requested by the incoming web request according to the probability of occurrence of that file type. The probability of occurrence is the normalized frequency at which a certain type of file is requested at this server. The mean file sizes and probabilities for each of the three distributions are not set until simulation run time.

The initial reward is generated by a uniform random number generator with

[2]This portion of the model was originally meant to be used for both this extension and the hybrid extension. Since the creation of this module, however, the mechanism for generating initial rewards in the hybrid extension changed, making the uniform number generator unnecessary.

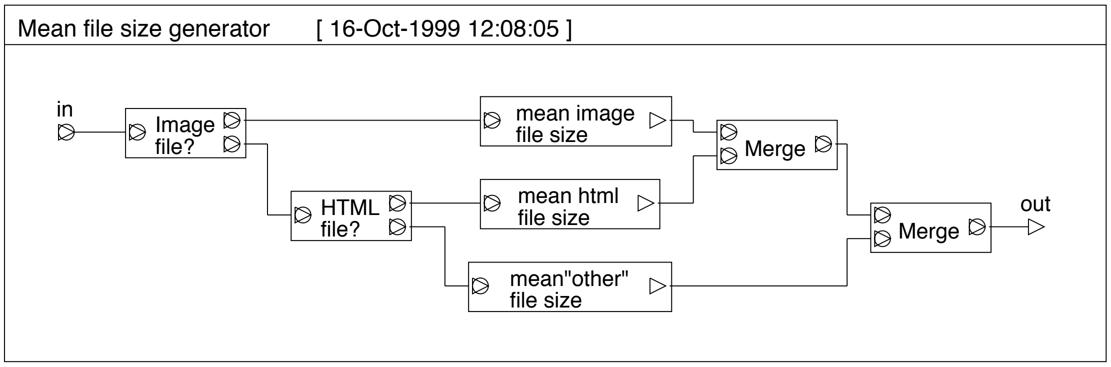

Figure 6.13: The mean file size generator module for simulation set 2

both endpoints set to 1.0. The module then calculates the $c_i\mu_i$ value by converting the mean file size for the request to the mean service time, inverting this measure to get the service rate, and multiplying this value by 1.0. The calculated value is placed in the *Current revenue* field. Note that both the initial reward and current revenue fields are used, unlike in the previous models.

As in the models for simulation set 1, the service time field is unused in the traditional service order model. However, it is used in the parameter-based model.

Statistics

Both models combine all of the statistics collection into one module, labeled ***Final Statistics–varying file size system***, shown in Figure 6.14. This block is a combination of the ***Intermediate stats*** and ***Statistics*** blocks from the original system. It performs the same four calculations as these two modules; namely, the departing request's response time and revenue, the total simulation revenue, and the total number of requests serviced during the simulation run.

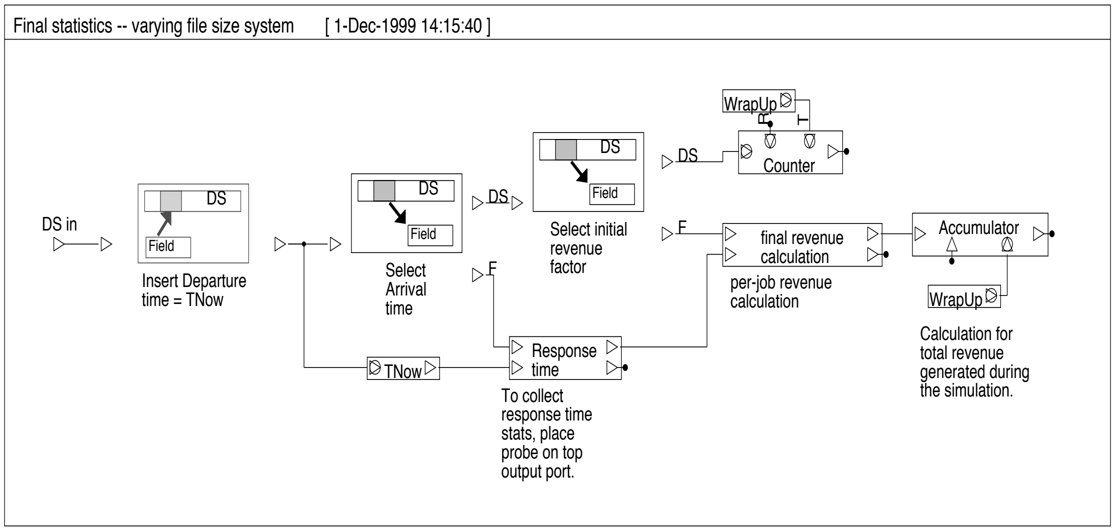
Final statistics -- varying file size system [1-Dec-1999 14:15:40]
DS in
DS
Field
Insert Departure time = TNow
Select Arrival time
DS
Select initial revenue factor
WrapUp
R
T
Counter
F
final revenue calculation
per-job revenue calculation
Accumulator
WrapUp
Calculation for total revenue generated during the simulation.
TNow
Response time
To collect response time stats, place probe on top output port.

Figure 6.14: The **Final statistics** module, simulation set 2

6.3.2 Simulation Parameters

There are two sets of parameters used, one for the traditional model and one for the parameter-based model. Table 6.3 lists the parameters that are common to both the traditional and the PB-2 models. For the traditional model, the following parameters take on the same values as in the first two sets of simulations: first request arrival time, mean interarrival time, cost factor, server mechanism, preempt discipline, queue discipline, and stop time. The parameters for the PB-2 model that have not changed from the PB-1 model are as follows: first request arrival time, mean interarrival time, cost factor, maximum queue size, and stop time.

Table 6.3: List of additional or differing parameters used in the traditional and PB-2 models for simulation set 2

Parameter	**Value**
HTML mean file size	2kB
Image mean file size	14kB
"Other" mean file size	100kB
P(HTML file)	0.2
P(Image file)	0.7
P("Other" file)	1 - (P(HTML)+P(image))
Mean file size (combined)	P(image)*14kB + P(HTML)*2kB + P(other)*100kB
Load	0.01 to 1.0
Per-byte processing rate	Mean file size (combined) * λ/ρ

Both models contain the same probes as the original simulation model. The probes are in the same locations in the traditional model as they are in the original model, except that any probes previously placed in the ***Intermediate stats*** or ***Statistics*** modules are now found in the ***Final Statistics...*** module. Refer to Table 5.3 for a list of these probes.

6.3.3 Simulation Results

The simulation results presented here are all for fixed arrival rates at three orders of magnitude. As in the previous simulation sets, we are interested in the total revenue as a function of load normalized by the number of requests served as well as the response time statistics for this system. We vary load by decreasing the per-byte processing rate of the server.

Figure 6.15 plots the normalized revenue as a function of load for each value of arrival rate and for each of the service orders. In addition, the revenue at the mean response time for the M/M/1 system is plotted for each of these cases as a point of comparison. Note that the simulation models represent an M/G/1 system, because the service rate is a mixture of exponential distributions, so the M/M/1 calculation is less accurate here.

Once again, the PB-2 plot outperforms the traditional service orders, although it is very similar to the LIFO-PR plot. The percent difference between the PB-2 and LIFO-PR revenues is always less than 4%. At the highest loads, particularly under high values of λ, there is almost no decay in the total revenue plots for either PB-2 or LIFO-PR. Again, these two configurations perform remarkably well under high server loads.

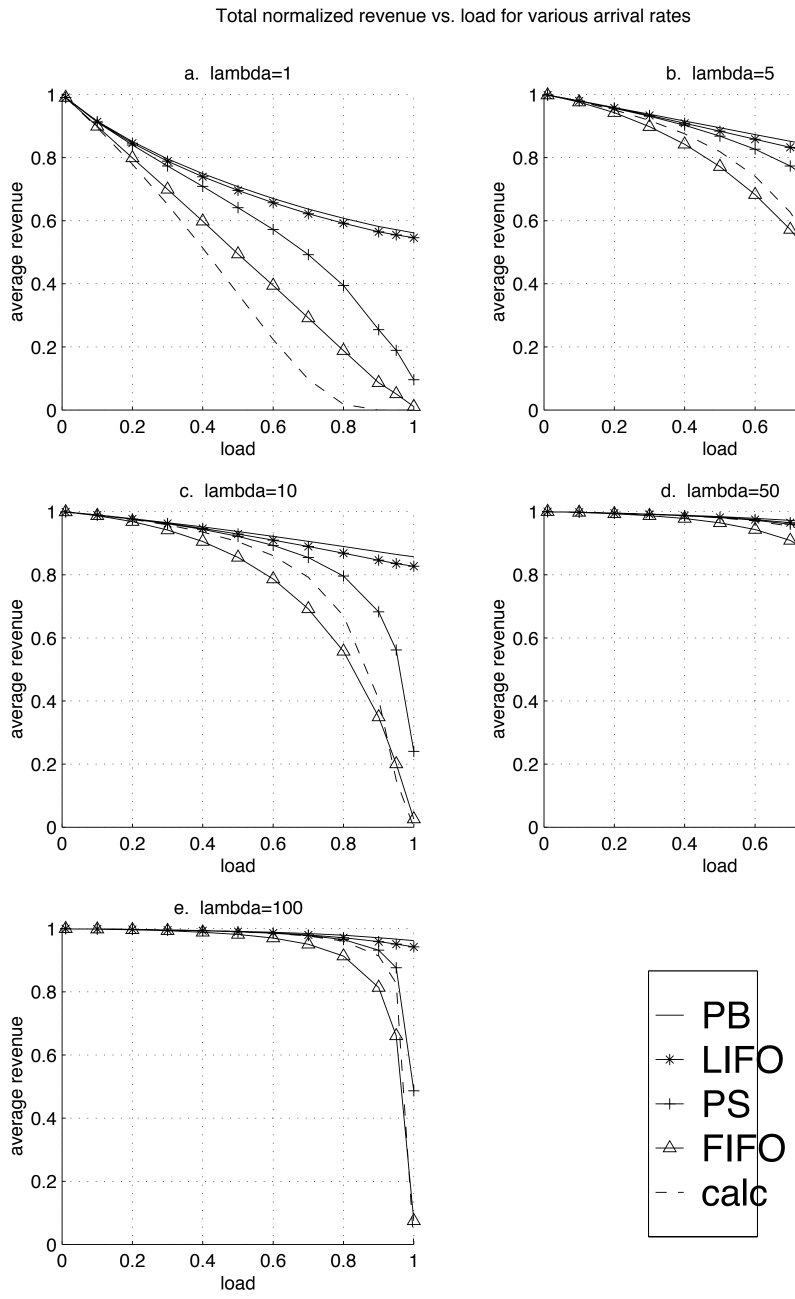

Figure 6.15: Total normalized revenue for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

The calculated plot is a cross between the FIFO and PS plots. Again, the calculated plot shows a very slight "tail-off" at the end (at loads of 0.9 and greater) which is more noticeable at the lower arrival rates. When $\lambda = 1$, the calculated plot has a lower revenue than all of the measured data plots. As the arrival rate increases, the calculated plot goes from resembling the FIFO plot to becoming nearly identical to the PS plot.

Unlike the previous simulations, the performance of FIFO, in terms of the percent difference in earned revenue between it and the optimal policy at the highest load values, does not improve significantly with increasing arrival rate. The maximum percent difference, at a load of 1.0, between the FIFO and PB-2 revenues is always greater than 90%. The performance of PS, however, in the same respect, does improve with increasing arrival rate. At all arrival rates, the FIFO plot decays faster than any of the other plots.

Each of the plots show four divergence points, similar to the previous sets of simulations. The first divergence point, ρ_1, occurs at the load where the FIFO plot breaks off from the rest of the plots. The second divergence point, ρ_2, occurs when the calculated plot breaks off from the remaining plots. At the third divergence point, ρ_3, the PS plot breaks away from the LIFO-PR and PB-2 plots; while the fourth divergence point, ρ_4, is where the LIFO-PR and PB-2 plots separate. Table 6.4 lists these divergence points.

Table 6.4: Divergence points for the normalized revenue plots

	Divergence points			
λ	ρ_1	ρ_2	ρ_3	ρ_4
1	0.06	0.06	0.17	0.32
5	0.14	0.21	0.34	0.45
10	0.20	0.34	0.42	0.53
50	0.39	0.60	0.66	0.76
100	0.50	0.73	0.75	0.85

Next, we examine the response time data collected from the simulations. We first look at the simulation response time data and compare this to the mean M/M/1 response time (see the previous sections for an explanation of how this quantity is obtained). We then do the same for the variance plots. We expect the M/M/1 plots to underestimate the response time statistics, particularly the variances, because the simulated systems represent a more highly variable M/G/1 system consisting of a mixture of exponentials.

Figure 6.16 contains the plots of measured mean response time for each simulation as well as the M/M/1 mean response time (labeled "calc" in the figure). The simulation data show some slight variation from the M/M/1 plot (and from each other). FIFO always has the highest measured mean response time, particularly at high loads. As λ increases, the LIFO-PR response time goes from being closer to the mean PB-2 response time to being closer to the PS mean response time, with a slight deviation at $\lambda = 10$.

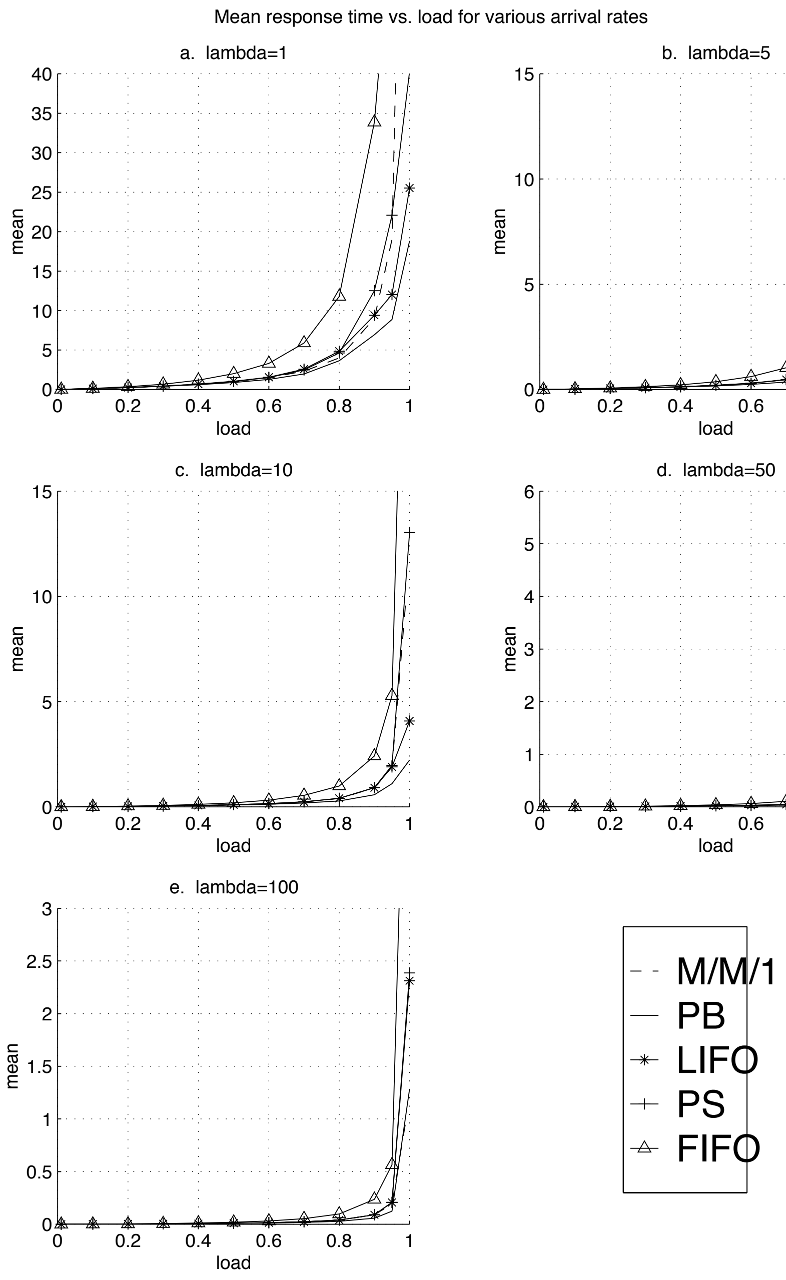

Figure 6.16: Measured mean response time for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

The response time variance plots (calculated and measured) are shown in Figure 6.17. For $\lambda = 1$, all of the measured data plots show higher variances than the M/M/1 plots. For all arrival rates above 1 request/sec, the measured FIFO and PS variance plots are the same. At arrival rates above 5 requests/sec, fewer differences are seen in the measured variances. Here, the variances are even larger than in the previous simulations, particularly at high loads, because now the service time distribution is a mixture of exponentials. This may explain why the LIFO-PR and PB-2 plots perform even better (in terms of average revenue generated at high server loads) for this set of simulations than the previous sets of simulations.

Figure 6.18 shows the plots of the response times calculated from the points on the expected revenue plots (i.e., $w = -\ln(\text{revenue})$). We compare these plots to the calculated plot of mean response time for an M/M/1 system. Note that the maximum load used in the theoretical calculations is 0.99 and not 1.0 as used in the simulations to avoid divide by zero errors. Note also that the plot of the average response time blows up as the load increases; therefore, the view has been zoomed in to capture the behavior seen in the simulation data.

The shapes of these plots are similar to the ones from the previous simulation. There is almost no difference between the PB-2 and LIFO-PR plots. As the arrival rate increases, the response times drop slightly.

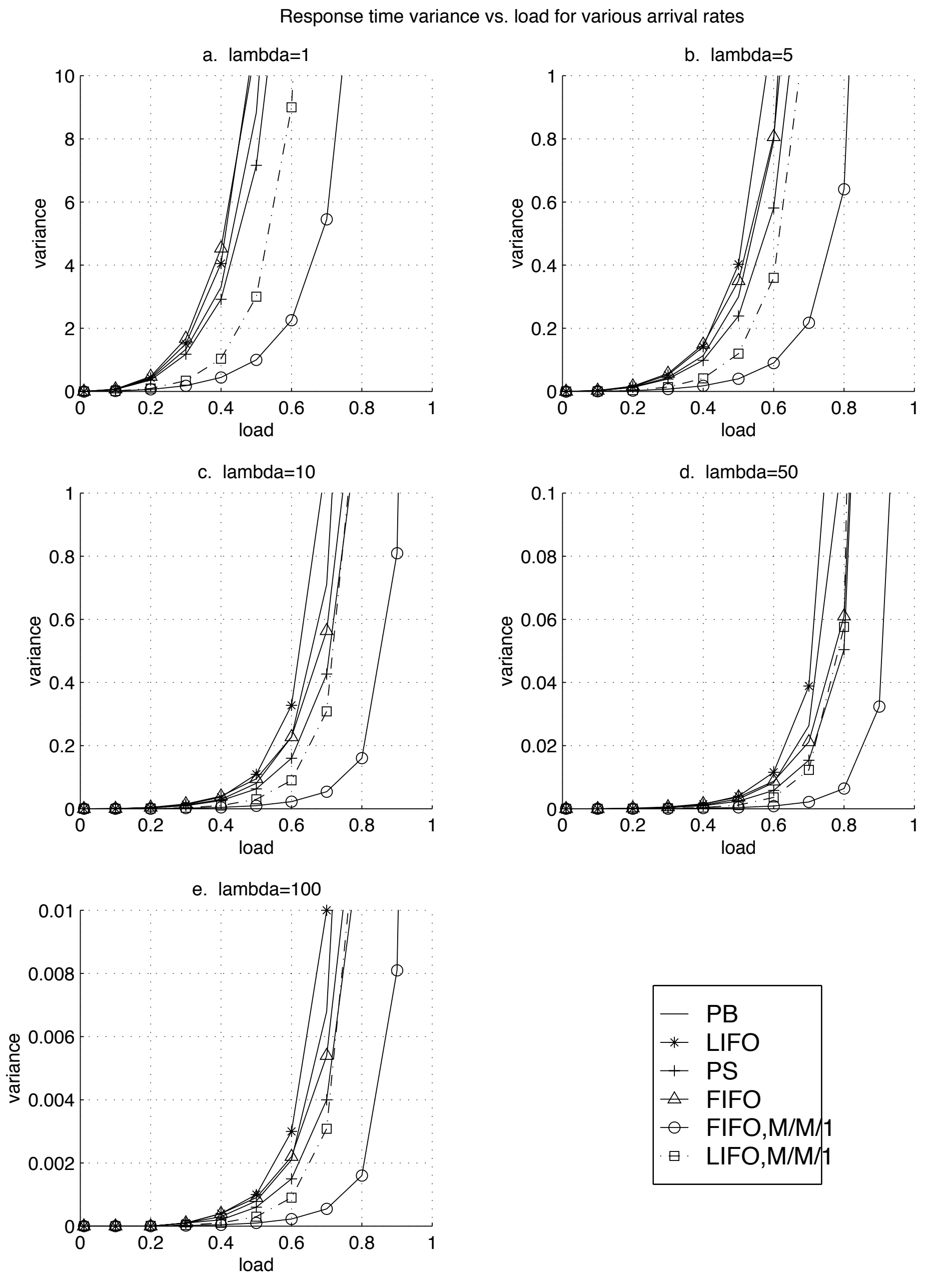

Figure 6.17: Response time variance at various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

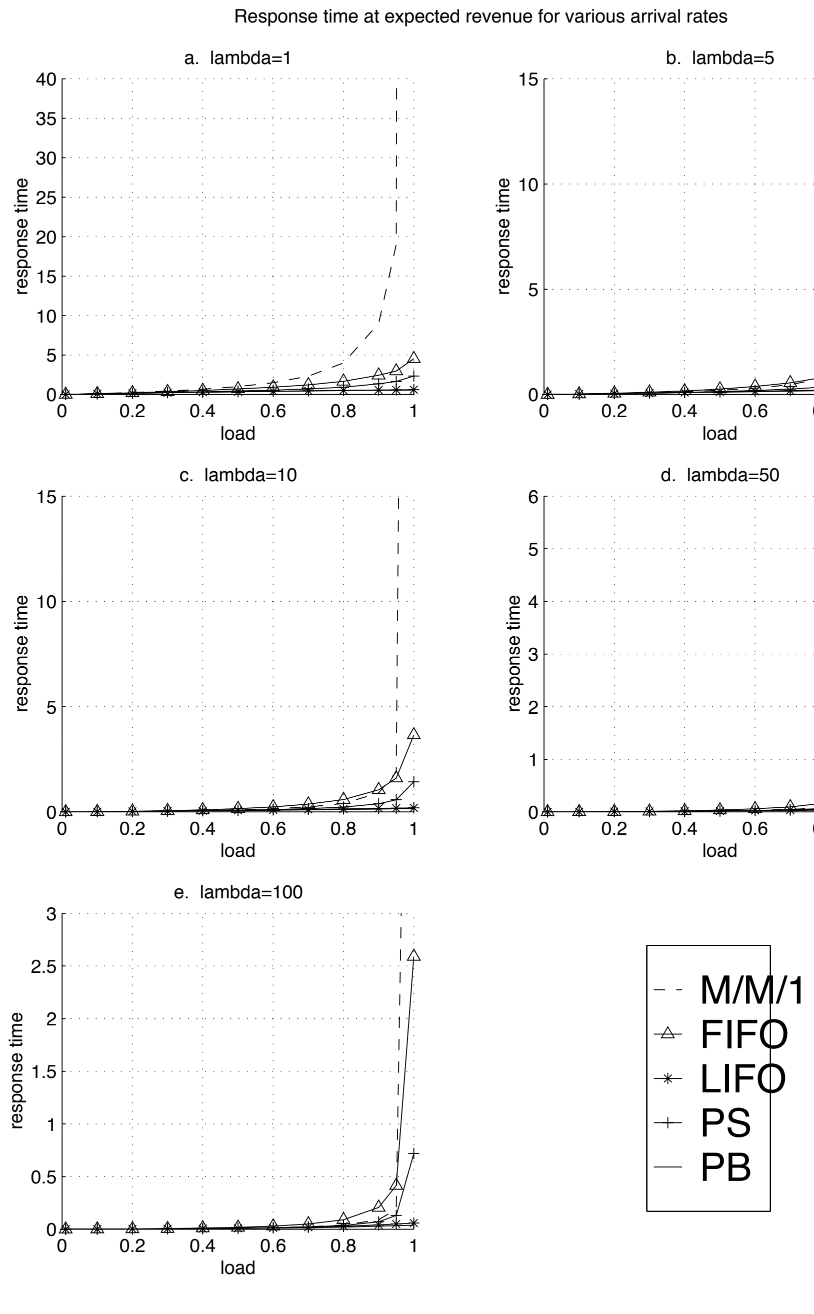

Figure 6.18: Response times calculated from the expected revenue plots for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

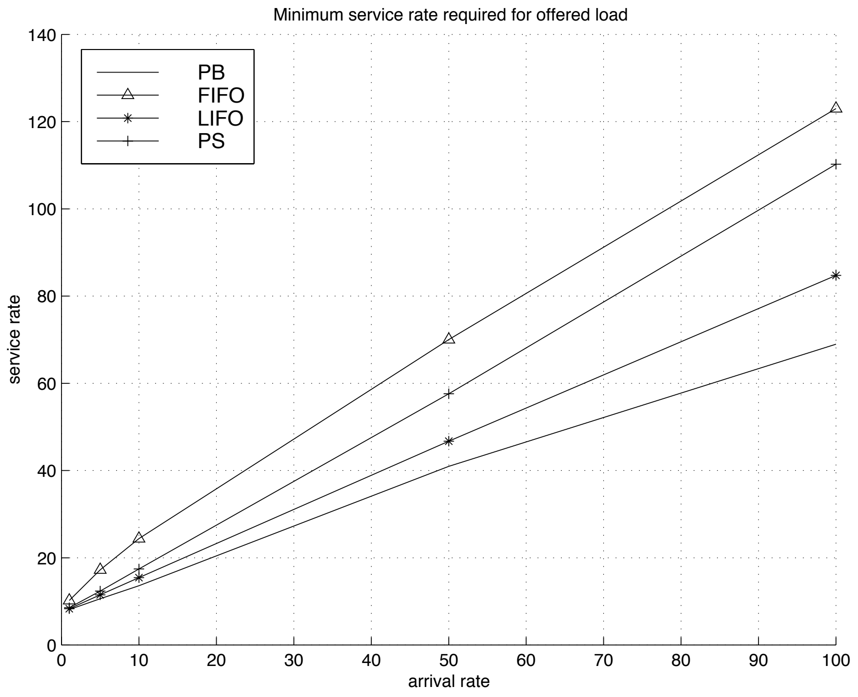

Figure 6.19: Minimum service rate needed to earn 90 percent of the possible revenue, simulation set 2

Figure 6.19 shows the plot of minimum service rates needed for the server to earn 90% of the total possible revenue offered to it as a function of arrival rate. The service rate values for the PB-2 and LIFO-PR plots at arrival rates above 50 requests per second are extrapolated from the available data, since neither of these plots ever drop below 0.9. All four plots show linear behavior. The slope for LIFO-PR remains fairly constant (around 0.78) for all values of λ. The slope of the PS plot increases from 0.95 to 1.05 over the range of λ. The slope of the FIFO plot actually decreases from a high of 1.76 at low values of λ to a low of 1.06 at the higher λ values. The slope of the PB-2 plot fluctuates between 0.68 and 0.56, ending at this low value for the higher values of λ.

Again, the PB-2 system has the lowest service rate requirements of the four configurations. At the highest arrival rates, the capacity of the PB-2 system is roughly 20% lower than the capacity required by the LIFO-PR system, and significantly lower than the capacity required by either the PS or FIFO systems.

6.3.4 Concluding Remarks

For this set of simulations, the differences between the performance of the LIFO-PR algorithm and the parameter-based algorithm are even smaller than in the previous set of simulations. Taking the expected remaining service time into consideration when making service decisions, which is basically what the PB-2 policy does, improves the server's performance slightly, at least for this combination of file types and sizes; although the improvement in capacity to support an arrival stream that generates 90% of the total possible revenue is about the same here as in the previous set of simulations.

6.4 Simulation Set 3: The Hybrid System

This section discusses the simulation models for the system with service times drawn from varying exponential distributions and with incoming requests assigned varying initial rewards. The optimal policy for this system is the same as the optimal policy for the system corresponding to simulation set 2.

To implement these models, we again define three unique exponential distributions with parameters μ_1, μ_2, and μ_3, respectively. Incoming requests access each distribution with some probability $p_i, i \in 1, 2, 3$, such that $p_1 + p_2 + p_3 = 1$. The initial rewards are then assigned based on the requested file type. Thus, the initial reward can take on one of three values: C_1, C_2, or C_3.

6.4.1 Simulation Model

The simulation models for the traditional and parameter-based systems are shown in Figures 6.20 and 6.21, respectively. These models are the same as those for simulation set 2, with the exception of the web request generator module. This module is the same in both the traditional system and in the parameter-based system, which we refer to as PB-3 in this discussion.

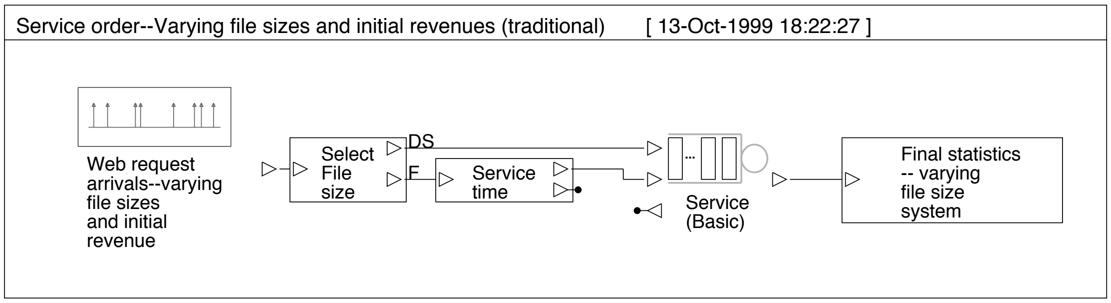
Service order--Varying file sizes and initial revenues (traditional) [13-Oct-1999 18:22:27]
Web request arrivals--varying file sizes and initial revenue
Select File size
DS
F
Service time
Service (Basic)
Final statistics -- varying file size system

Figure 6.20: The traditional model for the hybrid system

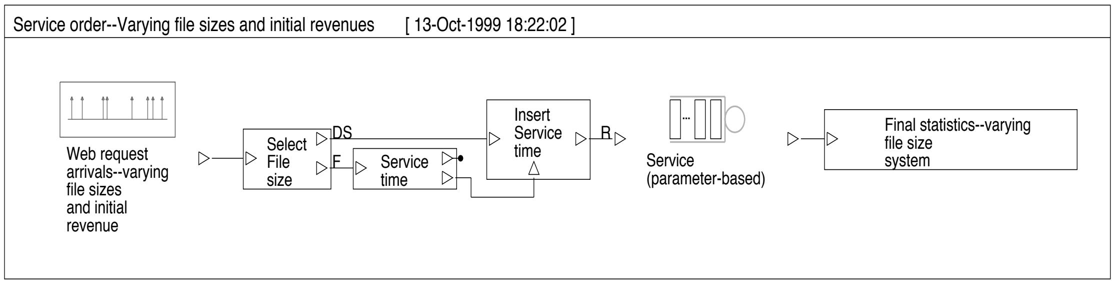
Service order--Varying file sizes and initial revenues [13-Oct-1999 18:22:02]
Web request arrivals--varying file sizes and initial revenue
Select File size
DS
F
Service time
Insert Service time
R
Service (parameter-based)
Final statistics--varying file size system

Figure 6.21: The PB-3 system model

Request Generator

The request generator module for this system, labeled ***Web request arrivals—Varying file sizes and initial revenues***, is pictured in Figure 6.22. It is very similar to the web request generator from the previous set of simulations. The main difference is in the mean file size generator module, which is labeled ***Initial revenue/mean file size generator*** and is pictured in Figure 6.23. This module operates in the same way as its predecessor, but in addition assigns an initial reward to each request based on the file size distribution chosen. This initial reward value is placed in the *Initial revenue factor* field in the data structure. The web request generator module also calculates the $c_i\mu_i$ product and places this value in the *Current revenue* field in the data structure.

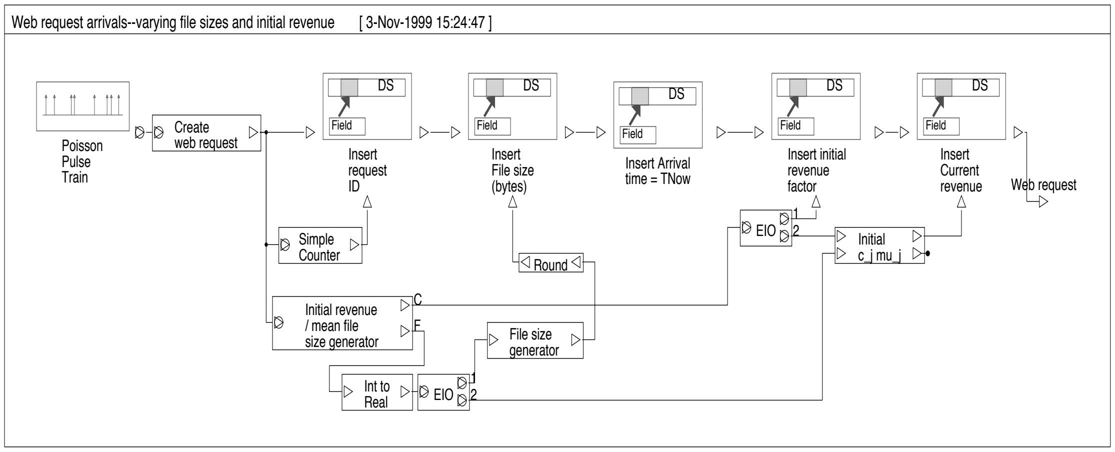

Figure 6.22: The request generator model for the hybrid system

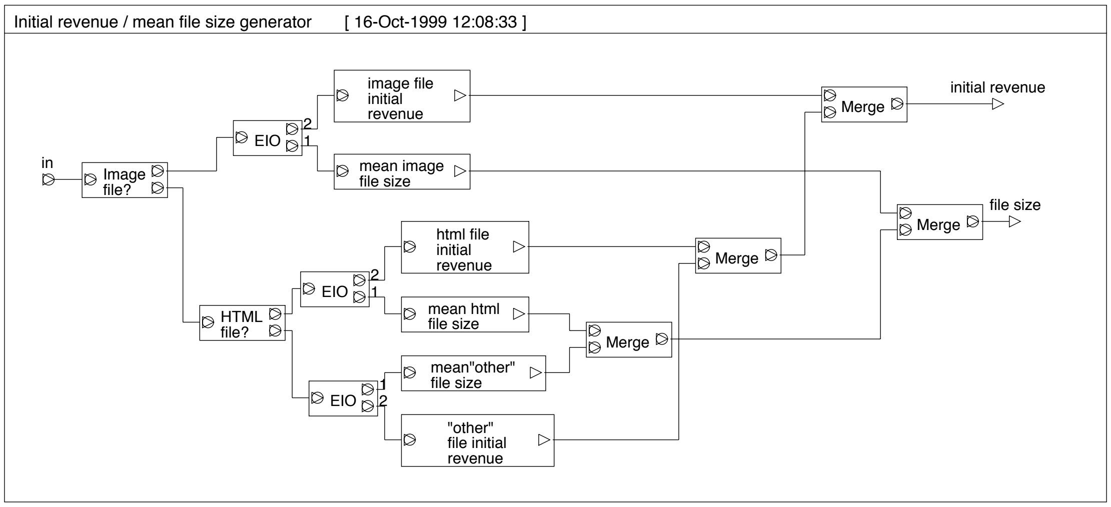
Initial revenue / mean file size generator [16-Oct-1999 12:08:33]
in
Image file?
EIO
2
1
image file initial revenue
mean image file size
HTML file?
EIO
html file initial revenue
mean html file size
EIO
mean"other" file size
"other" file initial revenue
Merge
Merge
Merge
Merge
initial revenue
file size

Figure 6.23: The file size and initial revenue generator for the hybrid system

6.4.2 Simulation Parameters

The parameter values for this set of simulations are the same as those for the previous set of simulations. There are three additional parameters added to both the PB-3 and the traditional models; these are listed in Table 6.5. The probes are the same here as in all the other models.

Table 6.5: List of additional or differing parameters used in the hybrid models

Parameter	**Value**
HTML initial revenue	0.95
Image initial revenue	0.65
“Other” initial revenue	0.85

6.4.3 Simulation Results

The simulation results presented here are all for fixed arrival rates at two orders of magnitude. In general, all of the plots for this set of simulations are identical to or similar to the plots from simulation set 2. In particular, the response time plots are identical in both sets of simulations and will not be repeated here. The other plots (total normalized revenue and 90% revenue) will be briefly discussed here.

Figure 6.24 plots the normalized revenue as a function of load for each value of arrival rate and for each of the service orders. In addition, the revenue at the mean response time for the M/M/1 system is plotted for each of these cases as a point of comparison. Again, note that we actually have an M/G/1 system here, so we expect the M/M/1 plot to underestimate the performance of this system.

These plots are identical to the normalized revenue plots from simulation set 2, scaled by a factor of about 0.73, the weighted mean of the initial rewards. The divergence points are nearly the same as those in the previous simulation set; in fact, the FIFO and PS divergence points are all greater by at most 0.1, and the LIFO-PR divergence points are all greater by 0.3. (Refer to Table 6.4 to obtain the divergence points from the previous simulation set.) The maximum percent difference between PB-3 and LIFO-PR is always between two and four percent.

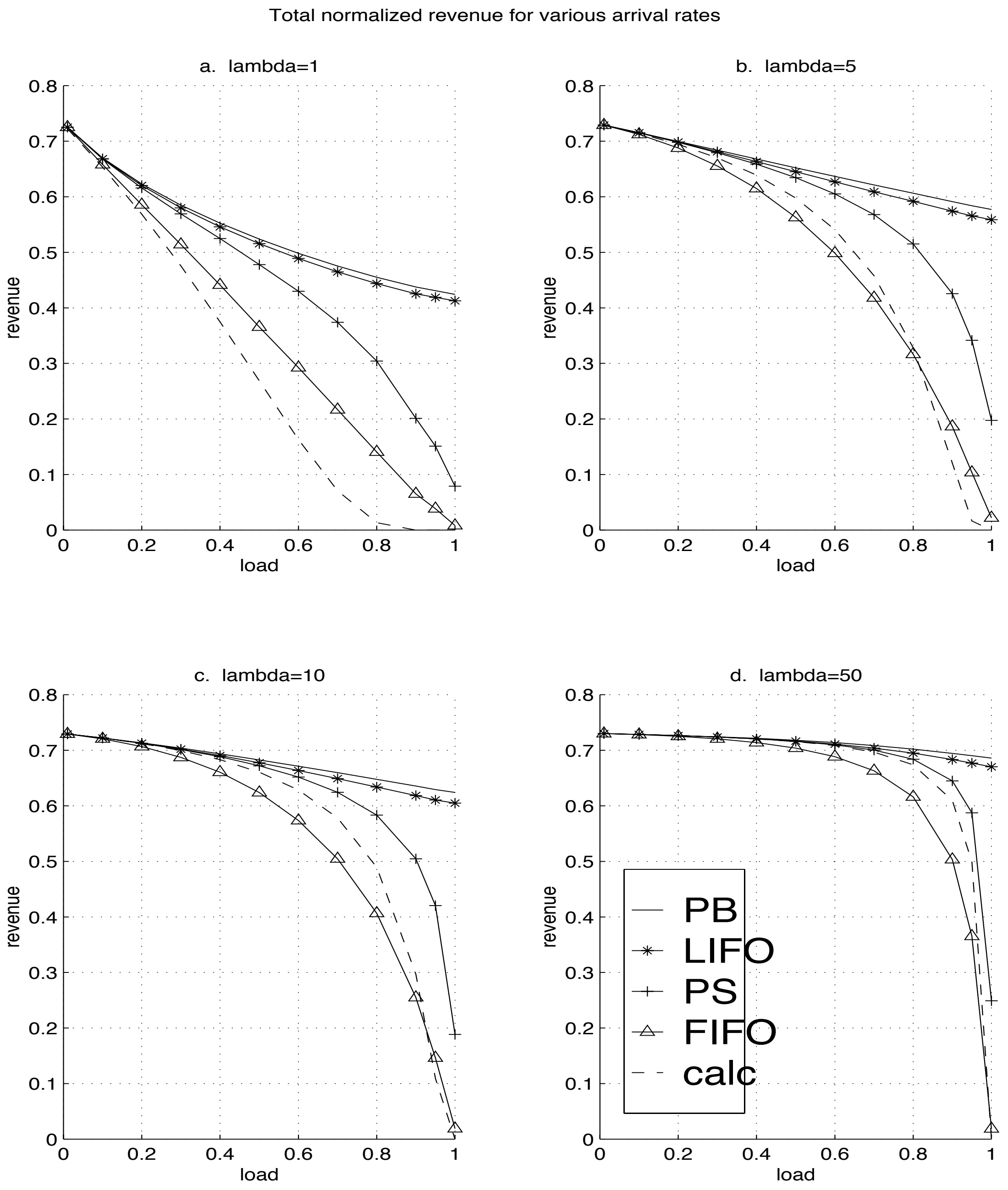

Figure 6.24: Total normalized revenue for various arrival rates. (a) $\lambda = 1$ (b) $\lambda = 5$ (c) $\lambda = 10$ (d) $\lambda = 50$ (e) $\lambda = 100$

Again, we see that the parameter-based policy represents a slight improvement in performance over the LIFO-PR policy.

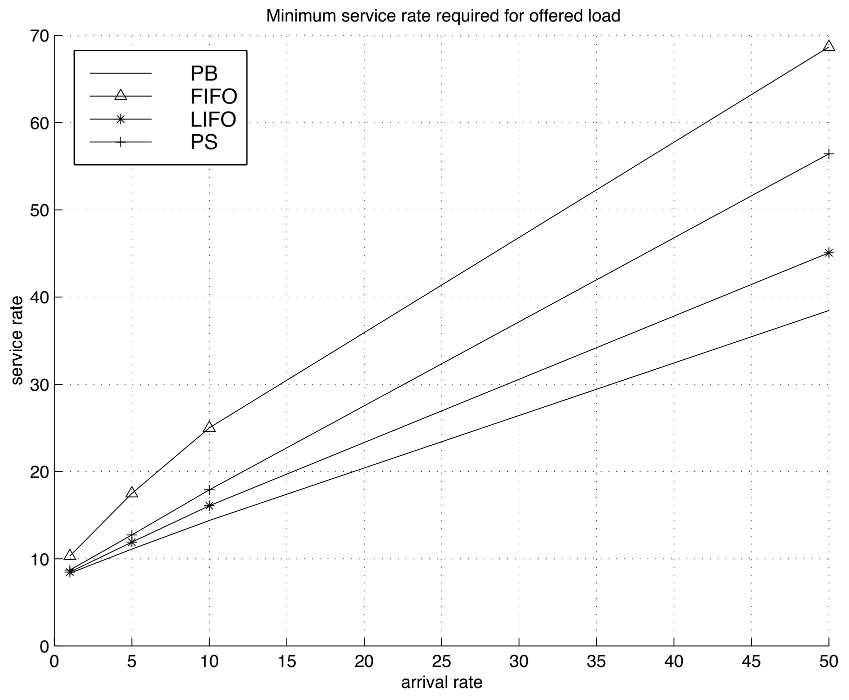

Figure 6.25: Minimum service rate required per arrival rate for the server to earn 90 percent of the possible revenue, hybrid system

Figure 6.25 shows the plot of minimum service rates required for the server to earn 90% of its possible revenue for this system. Once again, we extrapolate the data points for LIFO-PR and PB-3 at $\lambda = 50$ from the available data. All four plots show linear behavior with an offset of about 9 (10 for the FIFO plot). In general, the slopes of these lines decrease with increasing arrival rate. Again, we see that at the highest arrival rates, the capacity required for the PB-3 system is about 16% lower than the capacity required for the same system using LIFO-PR ordering, and much lower than the capacity required for the FIFO and PS systems.

6.4.4 Concluding Remarks

We see few differences between the performance of this system compared to the performance of a similar system with identical initial rewards. The performance difference between LIFO-PR and PB-3 is again very small. The important thing to note is that the performances of both the LIFO-PR and PB-3 policies degrade very little with increasing load, which is true in all the simulation sets studied so far.

6.5 Simulation Analysis of Conjecture 4.3.1

We now present simulation results for the case where a processor-sharing (PS) policy outperforms two non-processor-sharing (NPS) policies, as presented in the counterexample in the proof of Conjecture 4.3.1. In this system, the mean file sizes vary among the incoming requests, as in the systems for simulation sets 2 and 3. In this section, we briefly present the BONeS system model and the simulation results for a two-job sample path similar to the one presented analytically. For this model, we assume an even sharing of processor resources between the two requests when the PS policy is used.

In the following discussion, the “initial reward” of a job represents either the revenue decay of a job prior to the time of interest or the initial reward associated with that request. In the counterexample, we considered a two-job sample path in which one job arrives before the second job and has not departed before the second job arrives. Thus, the first job will have some revenue decay associated with it and the second job may or may not have some initial reward.

Figure 6.26 illustrates the BONeS model for this system. It consists of the traditional queue/server module, a modified statistics block, and two request generators. The two request generators artificially create a two-job sample path throughout the simulation run.

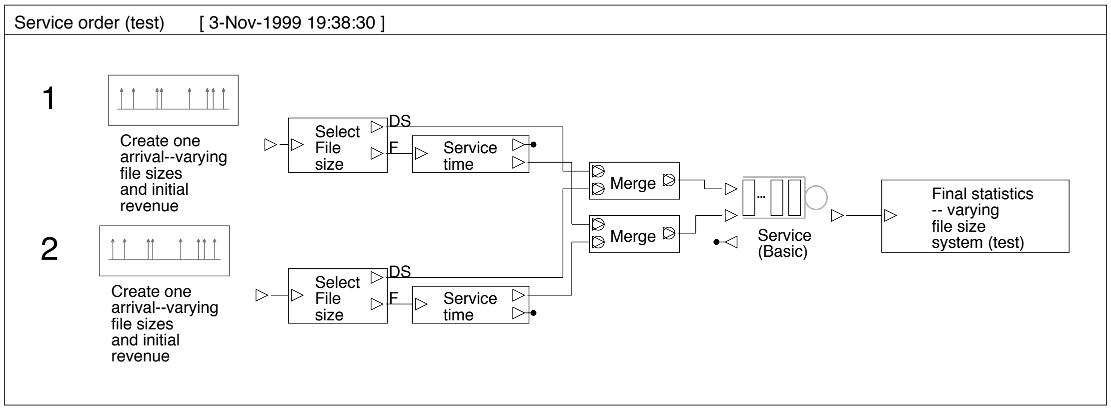
Service order (test)
[3-Nov-1999 19:38:30]
1
Create one arrival--varying file sizes and initial revenue
2
Create one arrival--varying file sizes and initial revenue
Select File size
DS
F
Service time
Select File size
DS
F
Service time
Merge
Merge
Service (Basic)
Final statistics -- varying file size system (test)

Figure 6.26: The BONeS model of the system in which the PS policy is sometimes optimal.

Figure 6.27 shows the detail of the request generator block. Note that many of the features are similar to the request generator blocks in simulation sets 2 and 3. The main difference is the uniform pulse generator; again, this is used to artificially create the two-job sample paths. The mean file sizes and initial rewards/revenue decays associated with each of the jobs are set at simulation run time.

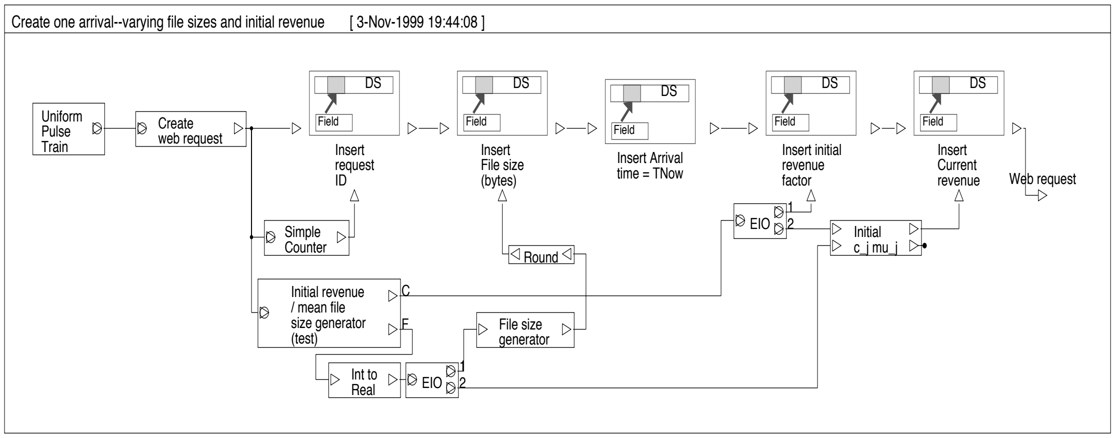

Figure 6.27: The request generator block for the system in which the PS policy is sometimes optimal.

The statistics module is pictured in Figure 6.28. Again, much of this block is similar to the ***Final statistics...*** modules in the models for simulation sets 2 and 3. The main difference is the additional accumulator block located just before the final accumulator block. This additional accumulator module calculates the total revenues earned over each two-job sample path. These revenues are then averaged throughout the simulation run to obtain the values that correspond to the equations derived in the counterexample.

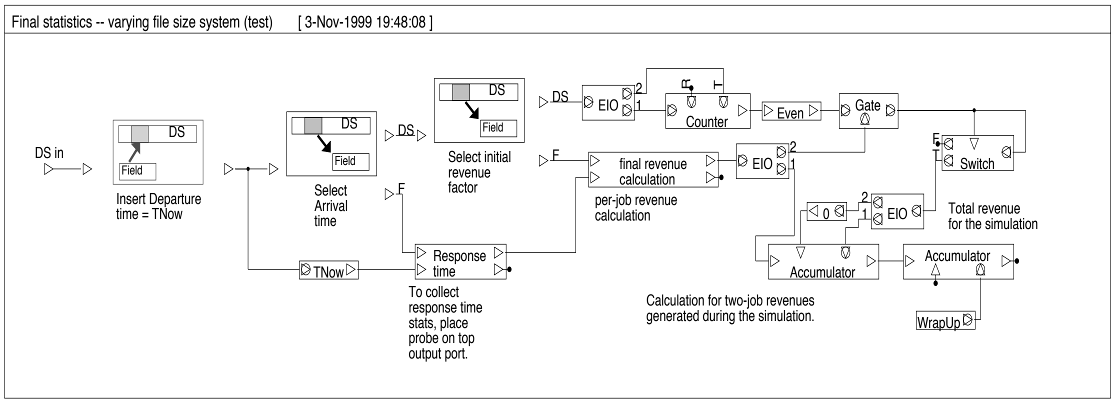
Final statistics -- varying file size system (test) [3-Nov-1999 19:48:08]
DS in
DS
Field
Insert Departure time = TNow
DS
Field
Select Arrival time
DS
F
DS
Field
Select initial revenue factor
DS
F
EIO
2
1
Counter
Even
Gate
final revenue calculation
per-job revenue calculation
EIO
2
1
Switch
Total revenue for the simulation
0
2
1
EIO
Response time
To collect response time stats, place probe on top output port.
TNow
Accumulator
Accumulator
Calculation for two-job revenues generated during the simulation.
WrapUp

Figure 6.28: The statistics block for the system in which the PS policy is sometimes optimal.

Table 6.6 lists the parameter values for this simulation, and Table 6.7 lists the probes used in this simulation. All of the probes are located in the ***Final statistics...*** module. In this set of simulations, we explicitly set the values of the two mean file sizes and the initial revenue of request two. The simulation parameter is the initial revenue of request one.

Figure 6.29 shows the expected revenues per sample path for each of the three policies: NPS1, NPS2, and PS. The revenues are plotted as a function of c_1, the revenue decay of request one. In this simulation, the processor evenly splits its resources between the two requests in the PS policy. We can see that, at least for values of c_1 below 0.55, the PS policy is never strictly better than both of the NPS policies. The PS policy performs better than both of the NPS policies at values of c_1 above 0.782.

Table 6.6: Simulation parameters assigned at run time, PS simulation

Parameter	**Value**
First request arrival time	0.0
Mean interarrival time	100; set so that the interarrival time is much greater than the service times of the two jobs (to force the two-job sample paths)
Per-byte processing rate	20000 bytes per second
Cost factor	1.0
Mean file size 1	Mean file size for request 1; set to 2kB
Mean file size 2	Mean file size for request 2; set to 1kB
initial revenue 1	The initial reward or revenue decay for request 1; varied between 0.1 and 1
Initial revenue 2	The initial reward for request 2; set to 0.5
Server mechanism	*Dedicated Server* for NPS policies; *Processor sharing* for PS policy
Preempt discipline	*Don't Preempt* for NPS policies; *Allow Preemption* for PS policy
Queue discipline	FIFO or LIFO-PR; PS uses FIFO ordering.
TSTOP	Simulation end time; set to 10,000.

Table 6.7: Data probes for the PS simulation

Probe	**Measures**
Mean total (two-job) revenue	Calculates the mean revenue per sample path
Total revenue	Calculates the total revenue collected from all departing requests

Figure 6.30 shows the same plot zoomed in to part of the area where PS performs better than both of the NPS policies. The line styles have been changed in this plot so that the differences are more noticeable. Notice that even though the PS policy clearly has a higher expected revenue in this region, the difference between this revenue and the revenue of the closest NPS policy is very small.

It is interesting to note that we have so far been unable to duplicate these results for sample paths that contain more than two requests, when arrivals form a Poisson process. We speculate that the periods where PS performs better than an NPS policy occur less frequently than periods where an NPS policy produces a higher revenue; thus, over a long time period, the effects of these PS policies on the total revenue lessens.

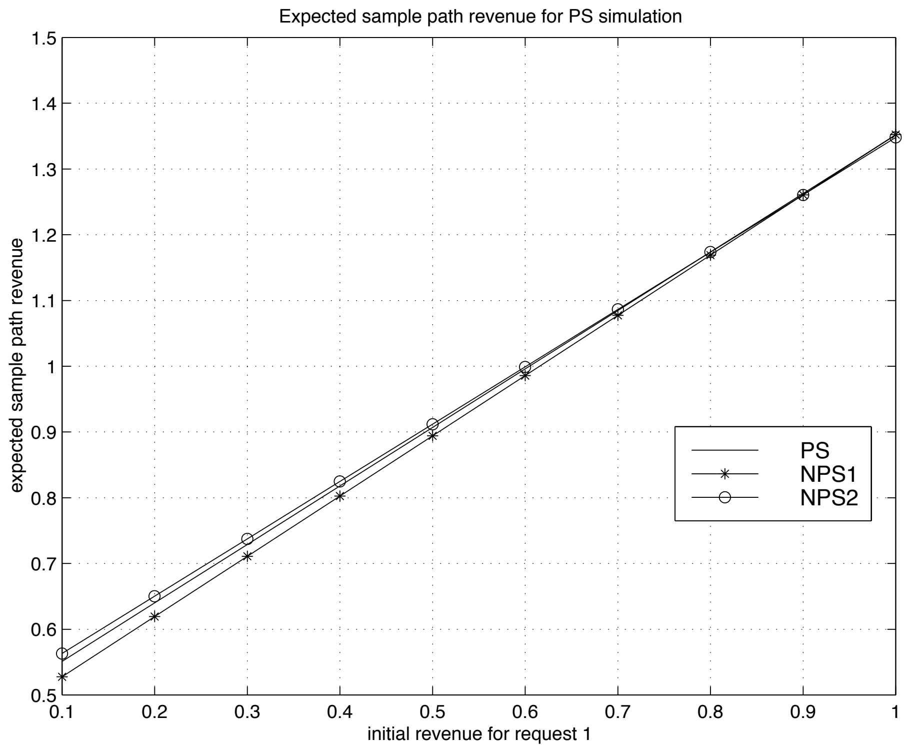

Figure 6.29: The expected revenue per sample path for the NPS1, NPS2, and PS policies.

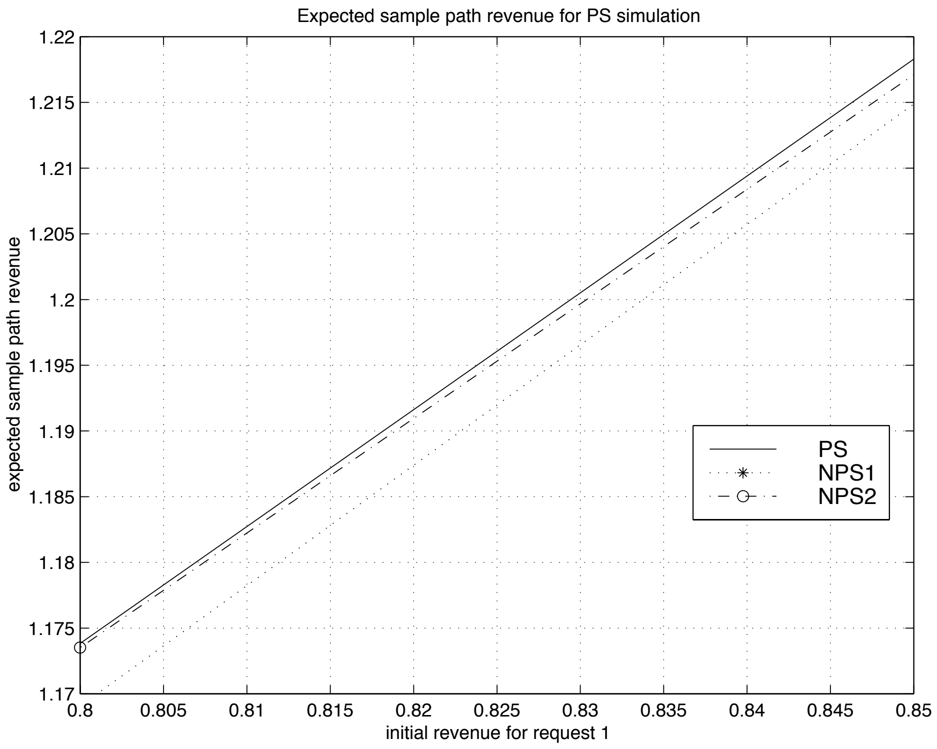

Figure 6.30: The expected revenue per sample path for the NPS1, NPS2, and PS policies, zoomed in to part of the region where PS outperforms the other two policies.

Chapter 7

SERVER DIMENSIONING

In this chapter, we discuss issues related to server dimensioning. The previous two chapters presented simulations for systems with one processor at various arrival rates and service rates. This chapter explores situations in which either the speed of the processor or the number of processors is varied. The goal is to determine the optimal operating points of the server for specific arrival rates and specific performance levels, where the performance levels are denoted as a percentage of the total revenue earned.

For each configuration studied, we present both a revenue analysis and a cost analysis. The revenue analysis compares the expected revenue per unit time earned by the server at various arrival rates for each configuration. The cost analysis determines which of these configurations represent the best tradeoff of cost and performance. Here, we define "cost" as the cost of the processor(s) used in each configuration.

In the first section, we analyze the effects of increasing the speed of the server's processor. In the second section, we vary the number of processors while keeping the service rate of each processor constant. Finally, we compare two configurations for two separate systems, where each configuration in a system has the same overall service rate. One configuration contains two processors, each at half the speed of the overall service rate; the other configuration contains one processor at the full service rate. The two systems represent two different overall service rates. In each of these scenarios, the simulations are iterated over the offered load by varying the arrival rate.

7.1 System Model

The model used in this chapter is the same as the simulation model presented in Chapter 5. Arrivals are Poisson with rate λ, service times are exponentially distributed with parameter μ, and all incoming requests are assigned the same initial reward.

For the dimensioning simulations, we only consider the optimal service policy for this system, preemptive-resume last-in, first-out (LIFO-PR). The parameters and probes are identical to those presented in Chapter 5, except for the cost factor parameter, which is set to 4.0. The corresponding revenue function, e^{-4w}, decays more quickly than the original revenue function. By increasing the decay rate of the revenue function, we attempt to reduce the performance of the system, so that the gains due to dimensioning are more noticeable.

7.2 Varying the Speed of the Processor

Figure 7.1 plots the unnormalized total revenue as a function of arrival rate for various values of service rate. The service rate is fixed at three values ranging from 100 requests per second to 400 requests per second.

For all three service rates, the plot is linear when the arrival rate is much less than the service rate. When the arrival rate is about half of the service rate, the increase in revenue begins to slow down, with a slight levelling off of the curve when the arrival rate is nearly equal to the service rate.

To explain the shapes of these plots, we look at the behavior of a typical M/M/1 system. Recall that the response time, or the total time a request spends waiting and in service, is given by the equation $w = 1/(\mu(1 - \rho))$. At low loads, the response time is on average the service time, or $1/\mu$. Since the service rate is held constant over the entire simulation run, as long as the response time is dominated

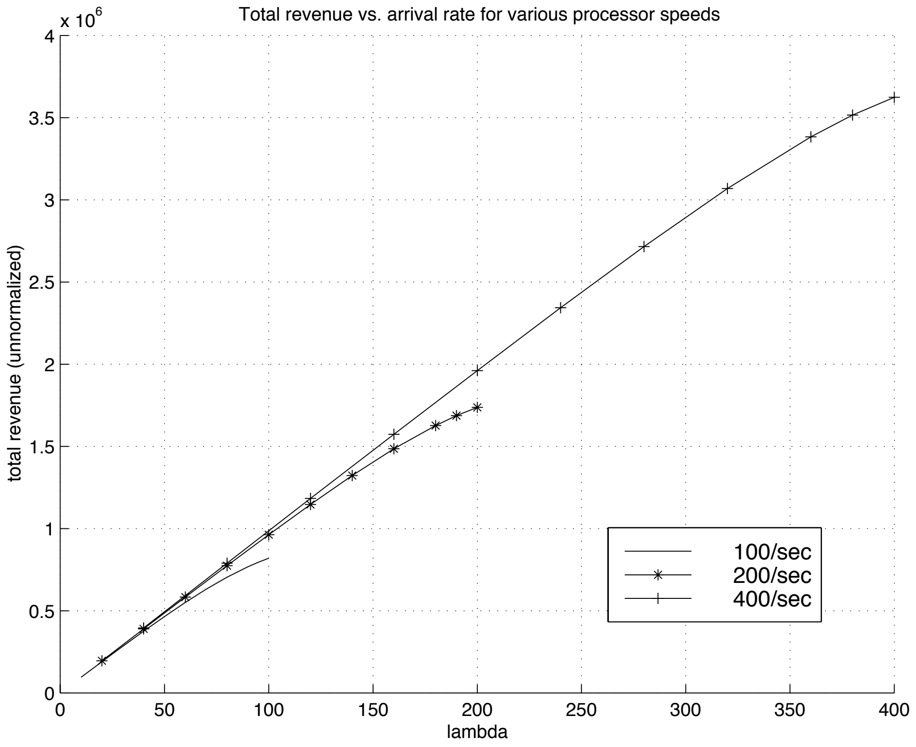

Figure 7.1: Total unnormalized revenue as a function of arrival rate for various processor speeds

by the $1/\mu$ term we will see linear growth of the revenue because the number of jobs serviced increases linearly with the arrival rate in this range. As the load increases, the queue size increases, and now the response time depends on both the mean service time and the number in queue, which is a function of the load. In terms of the response time equation, the μ and $(1-\rho)$ values both influence the response time at high loads, and cause the plot to deviate from linear.

Figure 7.2 shows the same data plotted against the load presented to the server, normalized by the processor speed. Here, we see the same basic plot shape as in Figure 7.1. The difference is that now all three plots are on the same scale. Again, at very low loads the three plots are linear or nearly linear. As the load increases towards one, the increase in revenue slows and then levels off. The point at which this slowing of revenue increase occurs is related to the speed of the processor in the configuration. For example, the configuration with the slowest processor (100 requests per second) has the lowest slowdown point, at a load of 0.2. The configuration with the next slowest processor (200 requests per second) remains linear until the load hits 0.4; the system with the fastest processor (400 requests per second) continues to increase linearly until the load reaches 0.6.

The above plot demonstrates the regions over which it is beneficial to use a higher-speed processor, in terms of the revenue generated per unit of processing speed. At low loads, there is not much difference in the revenue generated per

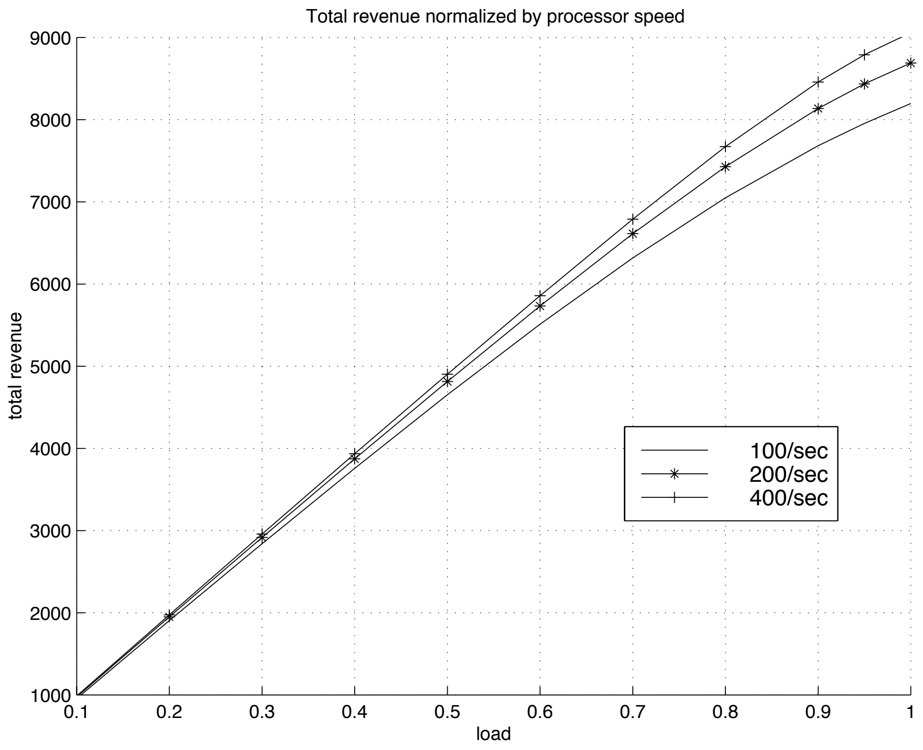

Figure 7.2: Total revenue as a function of load for various processor speeds, normalized by the processor speed

unit of processor speed among the three processors. As the offered load increases towards one, we see that the highest-speed processor offers the highest revenue per unit of processing speed of the three processors. This could mean that the higher-speed processor maintains a higher level of revenue growth over a longer portion of the offered load than the lower-speed processors.

7.3 Cost Analysis for the Processor Speed Simulations

In this section we present a simple cost analysis related to the processor speed simulations discussed in the previous section. In order to compare processor speeds, we use CPU speed benchmarks. We assign benchmark values to each of the processors in the simulation. We then match the benchmark values to actual processors and determine the price of each processor. Finally, we calculate the cost per second by spreading out the cost of the processor over its lifetime, and compare this to the revenue per second earned by the server in each simulation.

The benchmark values used are SPECint95 ratings, which are standard benchmark values from the Standard Performance Evaluation Corporation [57], an independent benchmark organization. SPECint gives a relative rating of the performance (speed) of a CPU in performing integer-intensive calculations.[1] The value expresses the ratio of the completion time of the series of tasks in the benchmark test on a reference machine (a Sun SPARCstation 10/40) to the completion time of the same tasks on the test machine. The benchmark value is the geometric mean of the scores from each of the eight programs that comprise the benchmark test.

We use the SPECint benchmark scores to determine the relative number of web requests that different CPUs can handle per second. The benchmark value tells us approximately how much faster one processor is than another in servicing one web request; from this measure, we extrapolate to calculate how many more requests per second can be serviced when using a certain processor than by using a different processor. For this analysis, we consider the Pentium family of processors manufactured by Intel.

Figure 7.3 illustrates the relation between price and performance for Intel processors.[2] The plot contains two data sets, one for the Pentium processor family and one corresponding to the Xeon processor family. The Xeon is the "high-end" version of the Pentium processor, manufactured for use specifically in servers. While there is a considerable price discrepancy between the Pentium II and III

[1]This type of calculation is a good approximation of the tasks a CPU performs in processing web requests.

[2]The reason for the unusual shape of the plots, which we expect to be more linear, is that we were unable to locate a comprehensive list of Intel CPU price data. We therefore looked at a sample of low and high prices for one week in October [58] and took the average of the two prices.

processors and the Xeon versions of these same processors, the plot shows that the performance difference between these types, with respect to the SPECint scores, is small. Table 7.3 lists the processors that correspond to the benchmark values from Figure 7.3.

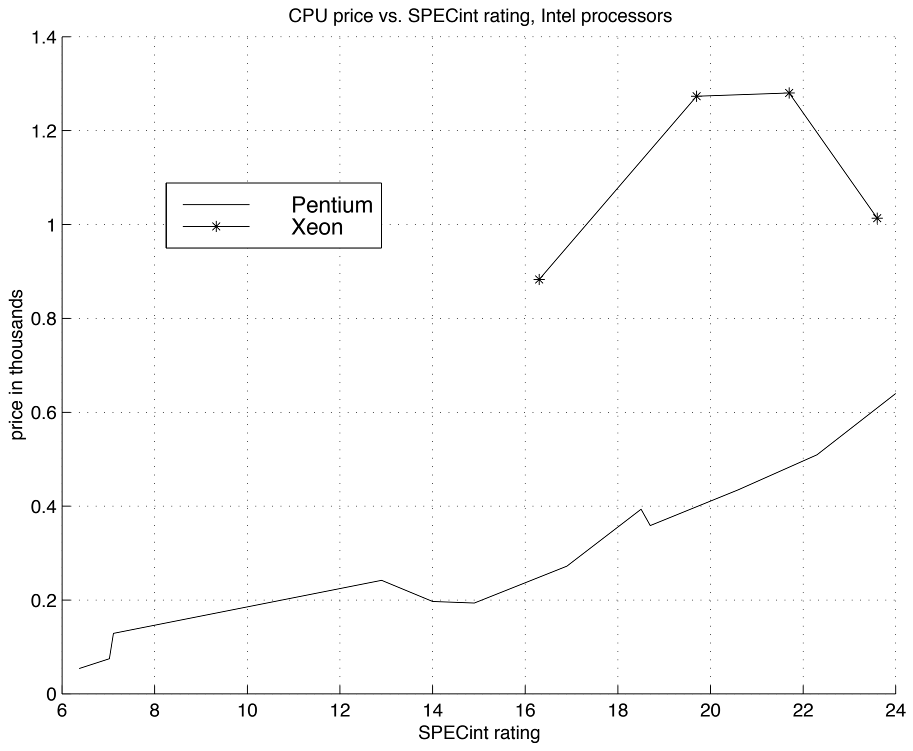

Figure 7.3: Relation between price and processor speed for several Intel processors.

We have enough cost data to almost quadruple the processor speed. We thus assign the processor with a benchmark of 6.37 to the processor in the simulation that processes an average of 100 requests per second. If we double the processor speed from 100 requests per second to 200 requests per second, or from a SPECint

Table 7.1: List of SPECint benchmark values and the processors to which they correspond

SPECint rating	Processor
6.37	Pentium 200 MHz MMX
7.02	Pentium 233 MHz MMX
7.11	Pentium Pro 166 MHz
12.9	Pentium II 300 MHz
14.0	Pentium II 333 MHz
14.9	Pentium II 350 MHz
16.3	Pentium II 400 MHz Xeon
16.9	Pentium II 400 MHz
18.5	Pentium II 450 MHz
18.7	Pentium III 450 MHz
19.7	Pentium II 450 MHz Xeon
20.6	Pentium III 500 MHz
21.7	Pentium III 500 MHz Xeon
22.3	Pentium III 550 MHz
23.6	Pentium III 550 MHz Xeon
24.0	Pentium III 600 MHz

rating of 6.37 to a SPECint rating of 12.9, the price increases by 300%, from 0.06 to 0.24. Doubling the speed again (roughly) from 200 requests per second to 400 requests per second, or from a SPECint rating of 12.9 to a SPECint rating of 24, results in a price increase of 167%, from 0.24 to 0.64. Note that none of the processor speed changes here ever require the use of one of the more expensive Xeon processors. The upgrade from 6.37 to 12.9 means upgrading from a Pentium 200 MHZ to a Pentium II 300 MHz; the processor associated with the 24 rating is a Pentium III 600 MHz.

Figure 7.4 plots the net revenues for the three processor speeds, where net revenue is the total revenue per second minus the cost per second[3] of the processor. The total revenue per second is normalized so that it is roughly the same magnitude as the cost per second of the processors.

When there are overlaps in arrival rates among the plots, the lower-speed processor generates a higher net revenue than the higher-speed processor. For example, when the arrival rate is between zero and 100 requests per second, the 100 requests per second processor generates the highest net revenue of the three processors. When the arrival rate is between 100 and 200 requests per second, the 200 requests per second processor generates a higher net revenue than the 400 requests

[3]The cost per second is calculated as the cost of the processor divided by the lifetime of the processor. In this analysis, we assume the processor has a lifetime of about one year. During its lifetime, we assume that the processor operates at peak load for four hours out of the day; thus, the processor's lifetime in seconds is $365 * 4 * 3600$.

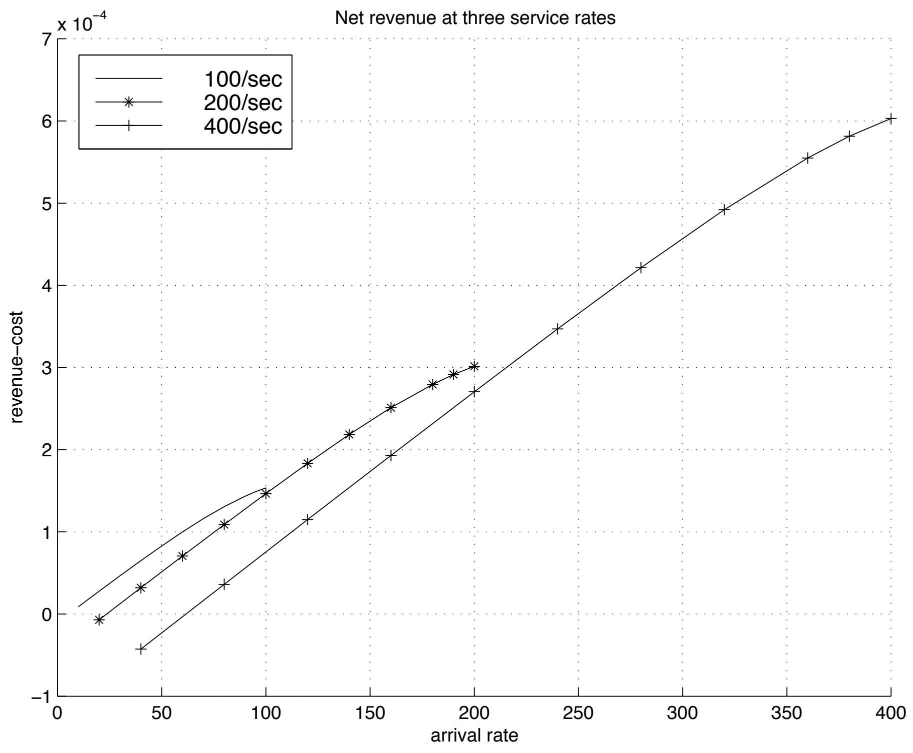

Figure 7.4: The net revenue for three Intel Pentium processors: one that can serve 100 web requests per second, one that can serve 200 web requests per second, and one that can serve 400 web requests per second.

per second processor. This is the opposite of what Figure 7.2 illustrates. The plot tells us, almost counterintuitively, that it is always beneficial from a net revenue standpoint to go with a slower processor wherever possible. In fact, we see that at very low arrival rates (below 25 requests per second for the 200 requests per second processor and below 60 requests per second for the 400 requests per second processor), the net revenue is actually negative! These arrival rates for these processors represent low offered loads. Thus, if it is known that the server will always operate at a low offered load, it is more beneficial from a cost standpoint to use a slower processor.

The reason for this apparent discrepancy is the non-uniform increase in price as processor speed increases. As described in the previous section, doubling the processor speed always results in an increase in cost that is more than double (up to four times as much as) the cost of the previous CPU.

7.3.1 Limitations of the Benchmark Analysis

A few notes related to the above analysis deserve mention here. First, it is important to remember that processor speed is only one factor relating to web server performance. Of equal, if not greater, importance is the amount and type of memory in the web server. Proper allocation of memory (specifically, static RAM, also referred to as "cache") limits the amount of paging required by the server to serve a particular set of documents. Paging refers to swapping documents in and out of memory. Ideally, a web server should have enough cache to store all of the documents it serves so that it never has to do disk accesses, which can greatly slow down its performance, while servicing web requests.

For this analysis, we ignore the memory constraints and assume that we have enough memory to serve all web documents out of the cache. We also ignore the networking components associated with transmitting the documents over TCP/IP; this is beyond the scope of our analysis, since it was not addressed in either the analytical analysis or simulation analysis of the web server.

7.4 Varying the Number of Processors

In this section, we present simulation data for a system in which the number of processors is varied between one and eight. The speed of each processor is fixed at 200 requests per second. In each simulation, the load is varied between 0.1 and 1.0 by varying the arrival rate.

Figure 7.5 illustrates the total revenue per simulation as a function of the arrival rate for one, two, four, and eight processors. In these plots, we see behavior that is similar to the varying speed simulation data plots. For low values of arrival rate, all four plots increase linearly at the same rate. For each set of data, as the arrival rate approaches the service rate (defined as the number of processors times the speed per processor), the increase in revenue slows. For each of these plots, the point at which the revenue increase slows corresponds to an offered load of approximately 0.6.

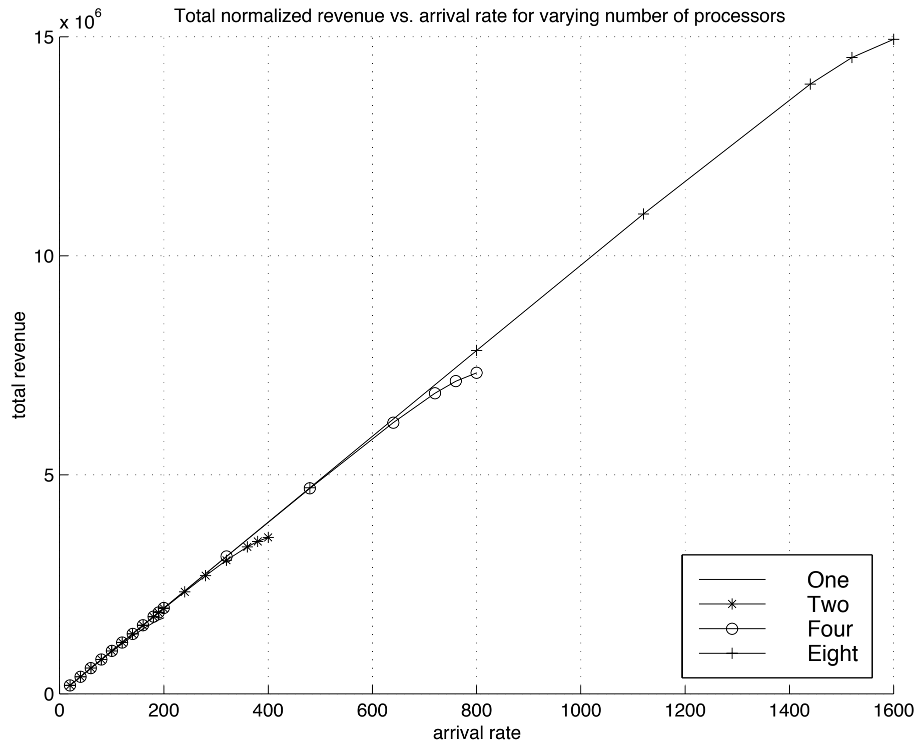

Figure 7.5: Total revenue for the varying number of processors simulations

Figure 7.6 shows the same data plotted as a function of offered load, normalized by the overall processor speed of each configuration. Again, we see that at low offered loads, each data set shows linear growth, with a slowing of the increase in revenue as the offered load approaches one. For the one-processor configuration, this slowdown occurs at a load of 0.5; the same slowdown occurs at a load of 0.7 for the two-processor configuration and 0.9 for the four-processor configuration. Note however that there is little difference in the revenues, particularly between the four- and eight-processor data sets.

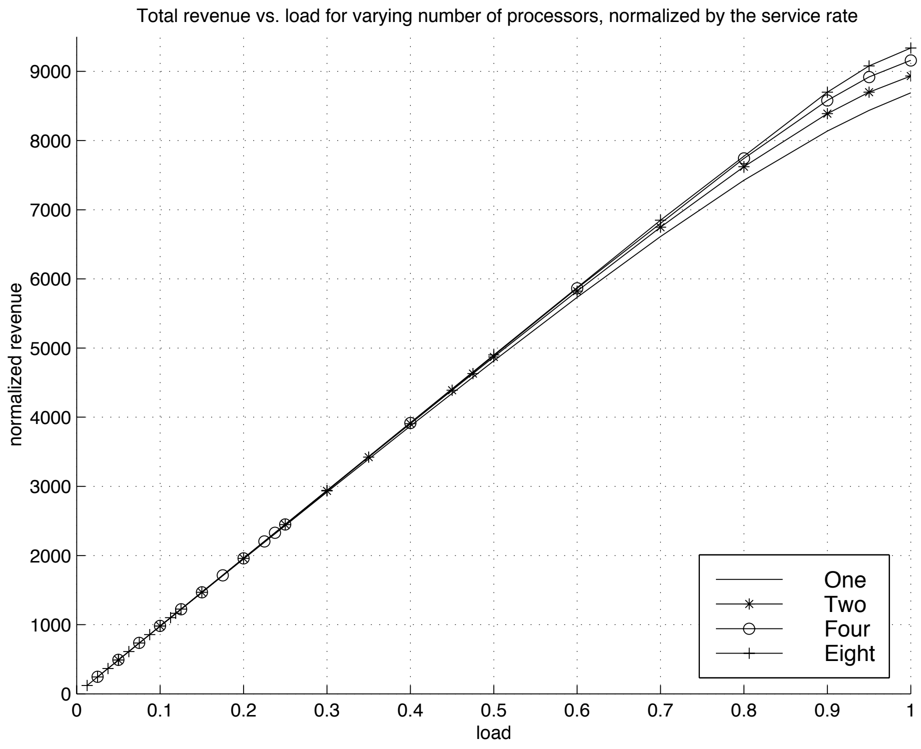

Figure 7.6: Total normalized revenue for the varying number of processors simulations

This normalized revenue plot is similar to the normalized revenue plot from the varying processor speed simulations. At low loads, there is not much difference in the revenue per unit of processing speed among the four configurations. As the offered load increases towards one, the configurations containing more processors generate more revenue per unit of processing speed than the configurations with fewer processors. The server configuration with more processors maintains a higher level of revenue growth over a longer portion of the offered load than server configurations with fewer processors. This is comparable to the behavior seen in the varying processor speed simulations, because in essence we are increasing the overall processor speed of the system as we increase the number of processors.

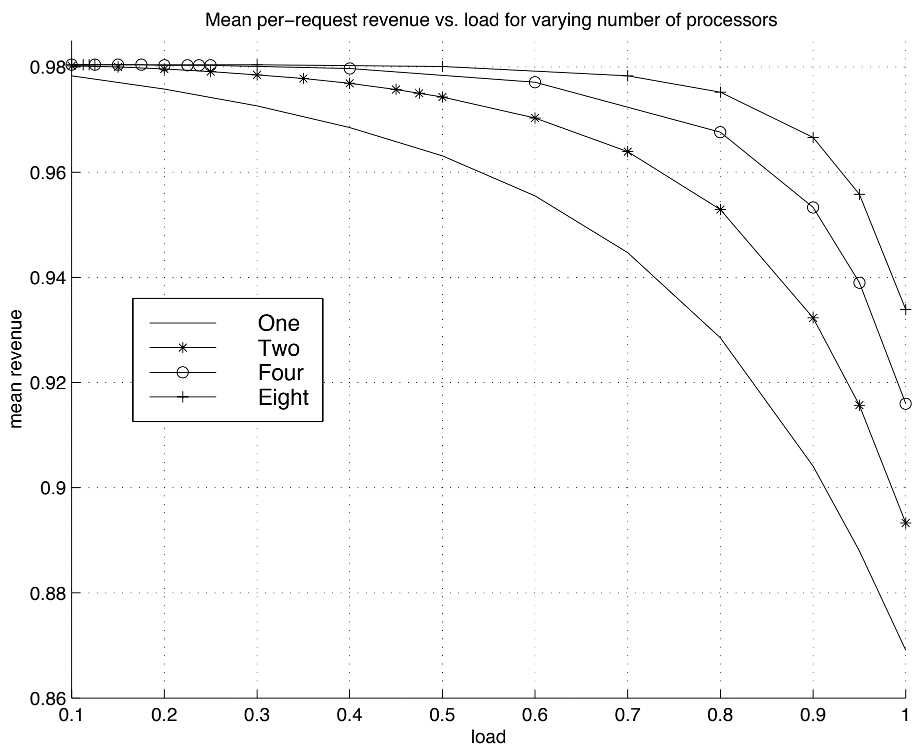

Figure 7.7: Total revenue for the varying number of processors simulations, normalized by the number of requests serviced.

Figure 7.7 shows the total revenue plot as a function of offered load, normalized this time by the number of requests processed during each simulation run. There is a noticeable difference between the one-processor data and the multiprocessor data in these plots. For instance, the maximum value reached by the one-processor configuration is slightly less than the maximum value reached by the multiprocessor configurations. The one-processor plot also decays right away, whereas the other three plots show a period of slight or no decay before they begin to drop off. The multiprocessor plots are identical over a portion of the range of the load; as the load increases, the plots one by one begin to diverge from each other. The plots hit minimum values of 0.87, 0.89, 0.92, and 0.93, respectively. Even under extremely high loads and using a revenue function that decays very quickly, we see that all of the configurations tested here hold up well in terms of the percentage of total revenue earned.

7.5 Cost Analysis for the Varying Number of Processors Simulations

We now present a cost analysis corresponding to the varying number of processors simulations. Since we have assumed a per-processor rate of 200 requests/sec, we use the Pentium II 300 MHz processor in the following analysis.

Figure 7.8 plots the net revenue associated with the varying number of processor simulations as a function of arrival rate. Figure 7.9 shows the same data, zoomed in to show the detail of the one- and two-processor configurations.

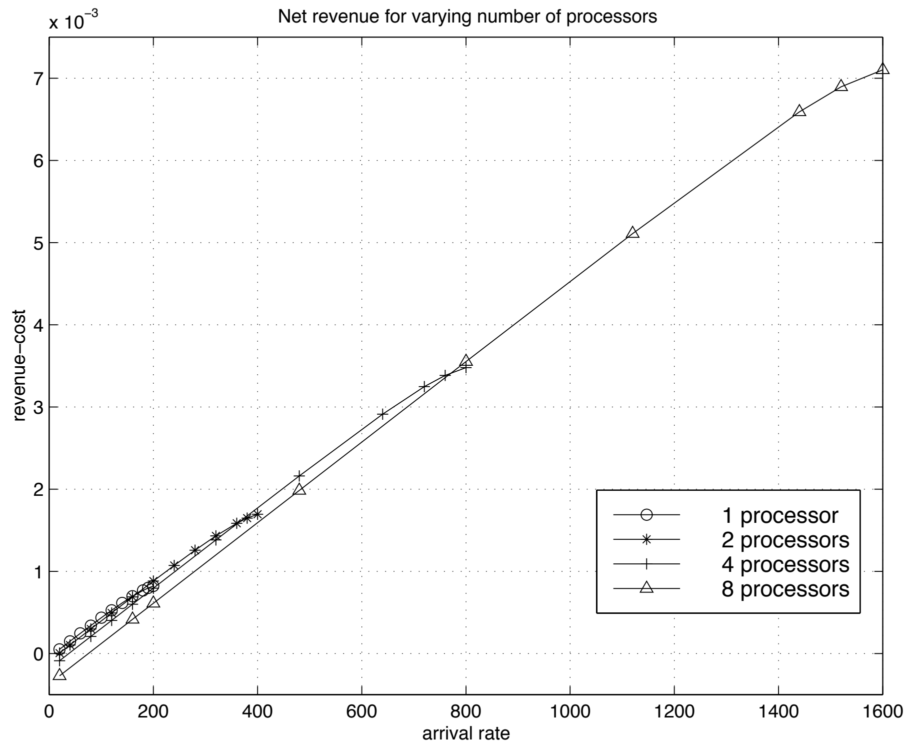

Figure 7.8: The net revenue versus arrival rate for a varying number of Intel processors.

The behavior of these plots is slightly different from the varying processor speed net revenue plots. Again, we see parallel linear growth among the four data sets over a portion of the arrival rate range. At some point on each plot, the increase in revenue begins to slow and eventually "flatten" out.

At low offered loads, the configurations containing fewer processors generate higher net revenues than the configurations with more processors. As the number of processors increases, the net revenue decreases; at the lowest offered loads

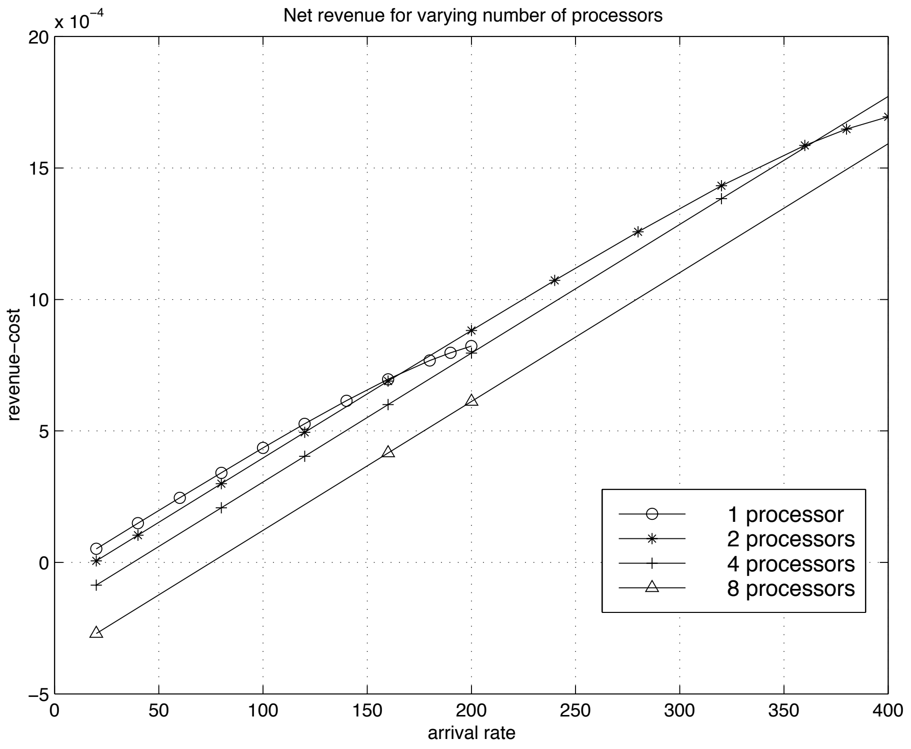

Figure 7.9: The net revenue versus arrival rate for a varying number of Intel processors, zoomed in to show the one- and two-processor plot data in more detail.

(corresponding to arrival rates between zero and 150 requests per second), the net revenue is highest for the one-processor configuration and lowest for the eight-processor configuration. We also see that at very low offered loads, the net revenues of the four- and eight-processor configurations are negative.

As the offered load increases towards one for each configuration, we see some interesting behavior that was not present in the varying processor speed simulations. At some high value of offered load, the plots actually cross, and the configuration with fewer processors now has a lower net revenue than the configuration with the next greatest number of processors. For instance, the two-processor configuration generates a higher net revenue than the one-processor configuration for arrival rates between 170 and 200 requests per second. In fact, when the offered load for the one-processor system is one (at an arrival rate of 200 requests per second), its net revenue is only very slightly higher than that of the four-processor system! Similarly, the four-processor configuration generates a higher net revenue than the two-processor configuration for arrival rates between 365 and 400 requests per second, and the eight-processor plot generates a higher net revenue than the four-processor plot for arrival rates between 770 and 800 requests per second.

The main difference between this plot and the net revenue plot from the varying processor speed simulation set lies in the cost of the processors for each configuration. In the previous system, we doubled the processor speed by changing the processor. The resulting increase in the price of the new processor was always more than double the original price. Here, we alter the overall processor speed by adding identical processors to the system; thus, the cost varies directly with the increase in speed. That is, doubling the overall speed results in a cost that is twice that of the original system.

7.6 Combination Simulations

This section presents the results from two sets of simulations. Both sets compare a one-processor configuration to a two-processor configuration. The processor in the one-processor configuration is twice the speed of each of the processors in the two-processor configuration, but the overall processor speed in each configuration is the same. The configurations in the first set of data consist of a processor with a service rate of 200 requests per second compared against two processors each with a service rate of 100 requests per second. The second set compares a one-processor configuration with a service rate of 400 requests per second against a two-processor configuration where each processor has a service rate of 200 requests per second. The simulations are iterated over various arrival rates representing offered loads between 0.1 and 1.0.

Figure 7.10 shows the total unnormalized revenue plotted against arrival rate for all four configurations. Note that the one-processor configuration in each of the simulation sets generates a slightly higher revenue than the corresponding two-processor configuration. The reasons for this will be explained shortly.

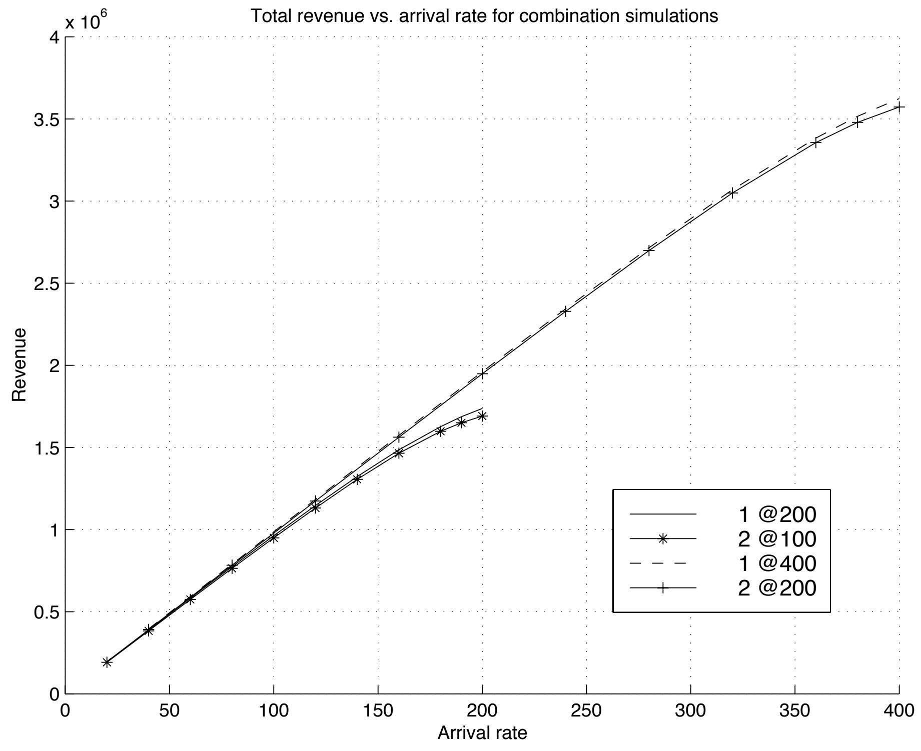

Figure 7.10: Total revenue for the one- and two-processor systems for the combination simulations

The four plots are identical for arrival rates between 20 and 90 requests per second. Above 90 requests per second, the plots begin to separate, with the plots corresponding to the system with an overall speed of 200 requests per second starting to slow in terms of the increase in revenue. At an arrival rate of about

140 requests per second, the difference in revenue between the one-processor and two-processor plot for the slower system becomes more pronounced; this difference is greatest at an arrival rate of 200 requests per second. We see similar behavior in the higher speed system; the plots start to separate when the arrival rate reaches about 340 requests per second.

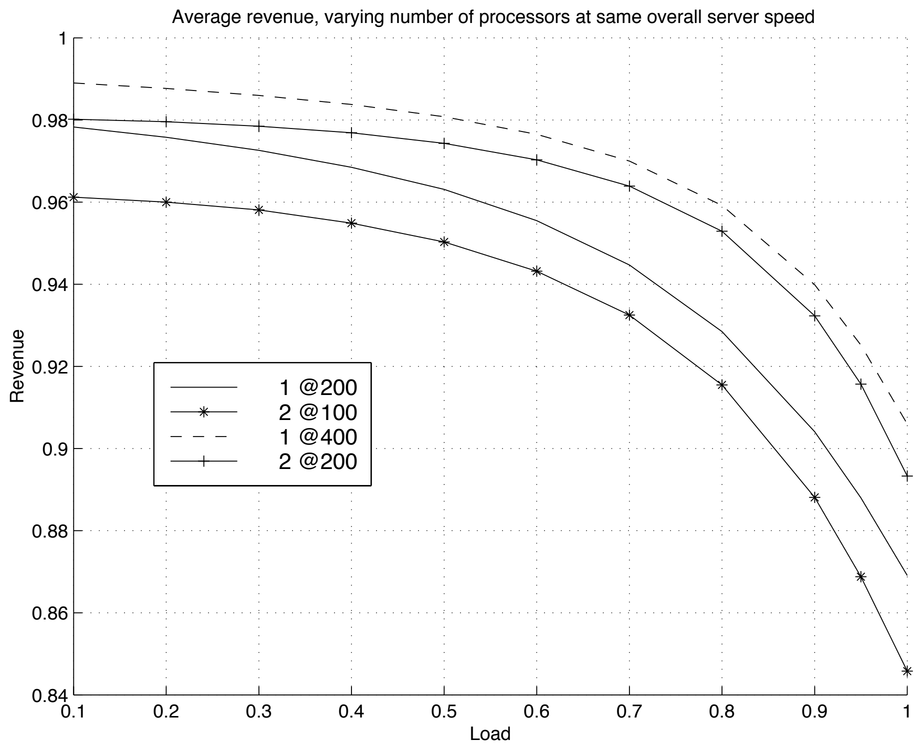

Figure 7.11: Mean per-request revenue for the combination simulation

To explain why the one-processor systems generate a slightly higher revenue than the two-processor configurations, we look at the per-request revenue, which again is identical to the total revenue normalized by the number of requests processed during each simulation. Figure 7.11 illustrates the average per-request revenue as a function of the offered load. The revenue is higher for the one-processor system than for the two-processor system in each case, even though the overall shape of the two plots is identical. The difference between the two systems is never more than 0.017 for the first set and not more than 0.009 for the second set.

The reason for the discrepancy between the one- and two-processor plots can be described in traditional queueing theory terms. We are essentially comparing an M/M/1 system to an M/M/2 system, where both systems have the same arrival rate and same *overall* processing rate. The one-processor system has a smaller total delay on average than the two-processor system (although it does have a larger average waiting time). The reason for this is that the M/M/1 system arrives at the null state faster than the M/M/2 system does. The Markov chains associated with each system have identical transitions for all states where there are at least two jobs in the system. The two chains differ in the transition from state 1 (one job in the system) to state 0 (no jobs in the system). In the M/M/1 Markov chain, the transition from state 1 to state 0 occurs at rate 2μ. By contrast, when the M/M/2 Markov chain is in state 1, the transition down to state 0 occurs at rate μ, because only one of the servers can process the one request. Thus, on average, the M/M/1 system will spend less time in state 1 and more time in state 0 than the M/M/2 system.

From the normalized revenue plots, we see that both of the plots associated with the higher overall service rate are greater over all values of the load than those for the system with the lower overall service rate. Again, this can be explained using the Markov chain illustration. At low loads, the average response time is approximately equal to the average service time, because jobs are more likely to arrive at an empty system. For the M/M/1 system, the average service time when one job is in the system is $1/2\mu$; for the M/M/2 system, the average service time when one job is in the system is $1/\mu$. We expect then the order (from least revenue to most revenue) of the four simulations to be as follows: M/M/2, $\mu = 200$; M/M/1, $\mu = 200$; M/M/2, $\mu = 400$; M/M/1, $\mu = 400$. This is in fact the order that is seen in the plots.

To further illustrate this point, figure 7.12 plots the measured mean response time for the two sets, zoomed in to the range [0,0.1]. There is a slight difference in mean response times between the two systems, and in fact the one-processor system does have a slightly lower mean response time than the two-processor system in both cases.

Figure 7.13 shows the total revenue, normalized by the total service rate of each configuration, plotted as a function of load. Again, the one-processor plots outperform their two-processor counterparts slightly. At low offered loads, the differences in revenue per unit of processor speed for each of the plots is negligible.

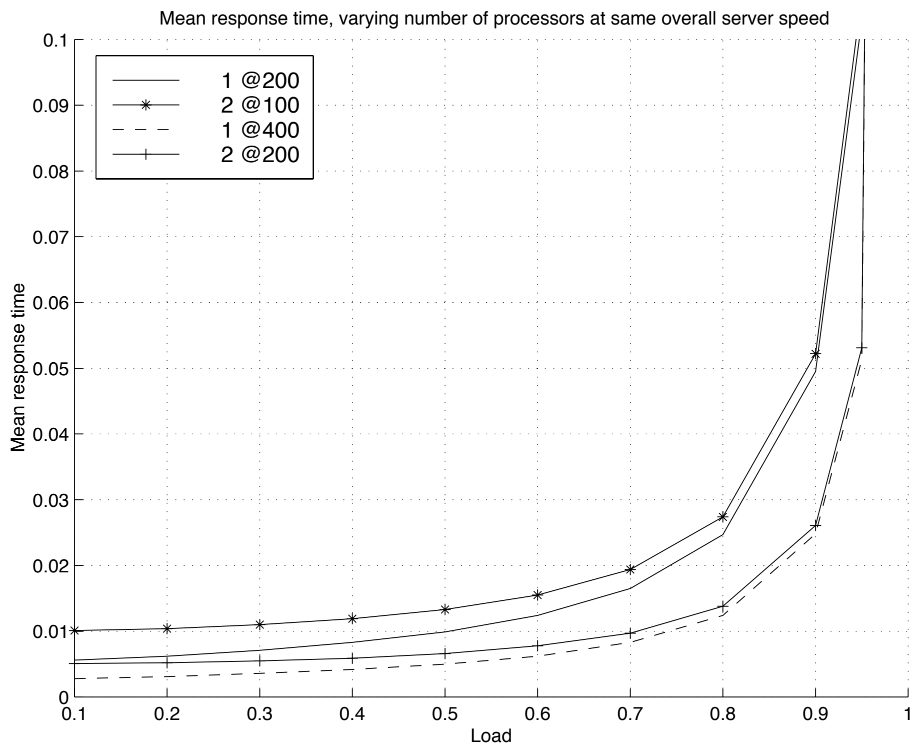

Figure 7.12: The mean and variance of the response time for the one- and two-processor systems

As the load increases towards one, the plots begin to separate from each other, and we find that both configurations in the faster system generate higher revenues per unit of processor speed than the slower system configurations. The one- and two-processor configurations do not show significant differences until the offered load is very high (0.8 for the slower system and 0.9 for the faster system). This behavior is similar to that seen in Figures 7.2 and 7.6, in that increasing the service rate of the system results in a higher revenue per unit of processing speed that continues over a larger range of the offered load.

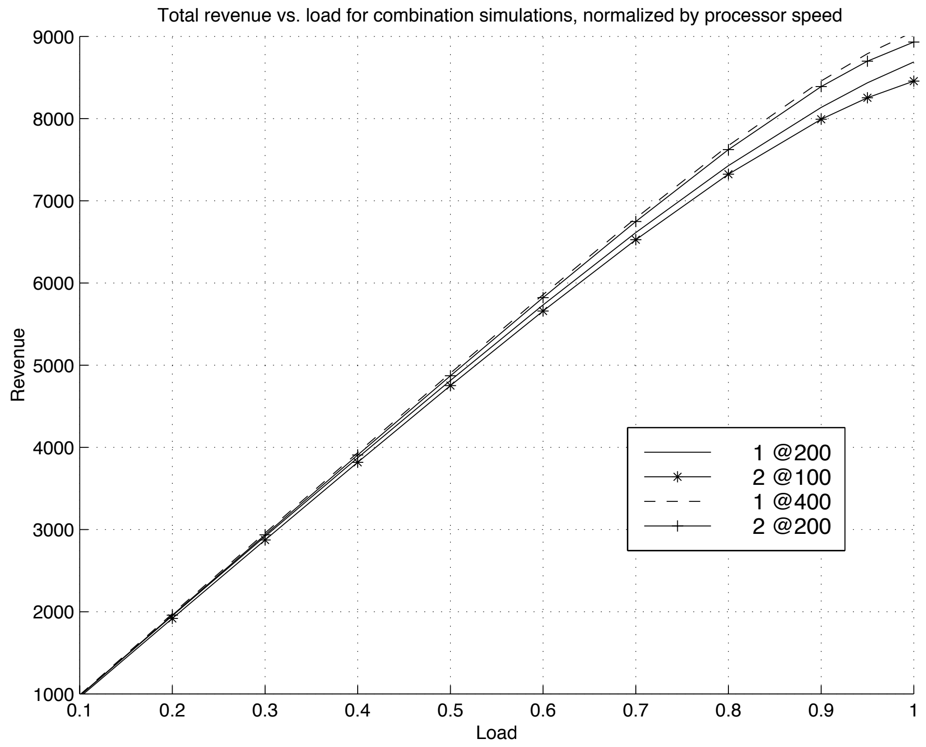

Figure 7.13: Total revenue for the one- and two-processor systems, normalized by the total processor speed

7.7 Cost Analysis–Combination Simulations

In this section, we explore the cost/revenue tradeoffs of the two systems presented in the previous section. We use the same three processors as in the cost analysis of the varying processor speed simulations: the Pentium 200 MHz, the Pentium II 300 MHz, and the Pentium III 600 MHz.

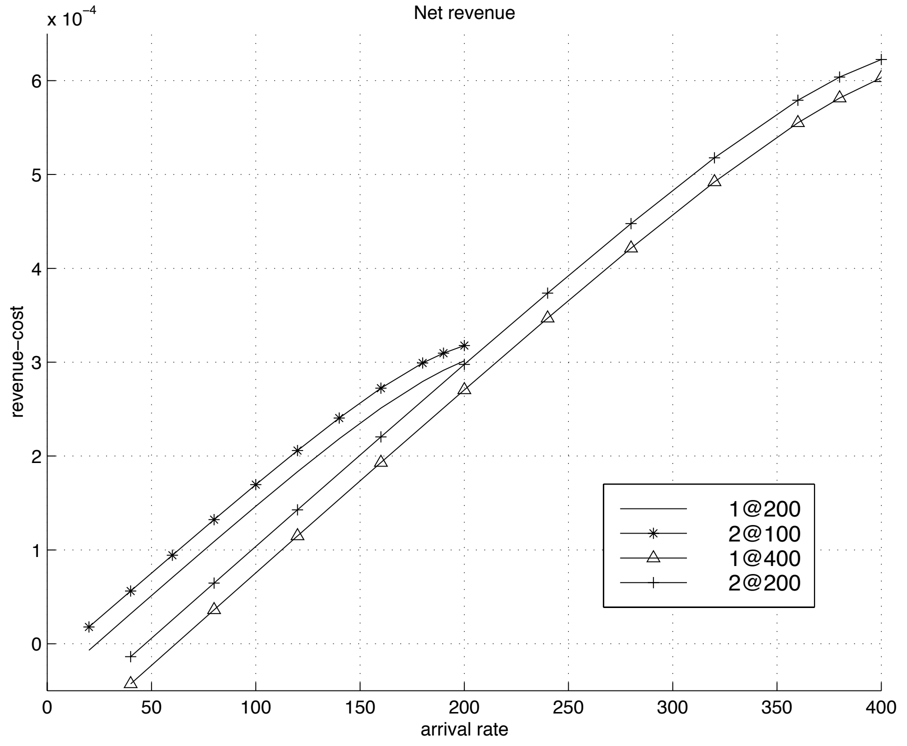

Figure 7.14: Net revenue plots associated with the combination simulations.

Figure 7.14 illustrates the net revenue as a function of arrival rate for each of the four configurations. Here, the two-processor plots actually outperform the one-processor plots for each individual system. The reason for this is that the slower processor is often less than half the price of the faster processor; therefore, doubling the cost of the lower processor yields a cost that is still less than the faster processor. As expected, the shapes of these plots resemble the shapes of the total revenue plots in Figure 7.10.

Again, similar to the varying speed net revenue plots, for the higher-speed system at the lowest arrival rates, the net revenue is actually negative. Therefore, we conclude that it is not beneficial to use a faster processor or set of processors for a system that does not generate a lot of traffic. The net revenue for the two-processor configuration for the higher-speed system does become positive at a lower arrival rate than the one-processor configuration (at 48 requests per second compared to 61 requests per second).

In the range of arrival rates between 20 and 200 requests per second, the slower system with two processors yields the highest net revenue, followed by the slower system with one processor, followed by the faster system with two processors, followed by the faster system with one processor. Note that there is very little difference (about one percent) in net revenue between the slower system with one processor and the faster system with two processors when the arrival rate is 200 requests per second; this is due to the slowing of the increase in revenue of the lower-speed system plots. Between 200 and 400 requests per second, the two plots corresponding to the higher-speed system look identical to the total revenue plots.

For these two systems, then, it is more cost-effective to use two processors at a lower speed than to use one faster processor. The gains in revenue from using one processor are lost here because the processor costs are not proportional to the increases in processor speed. The degree to which the net revenue for the two-processor system is higher than that of the one-processor system is dependent on the increase in cost between the slower and faster processor. In this set of data, the faster processor is between 2.67 and four times more expensive than the half-speed processor. If we had chosen processors whose costs were more in line with the comparative speeds of the processors, the behavior seen in Figure 7.14 would not be as pronounced.

Chapter 8

CONCLUSIONS

In this chapter, we summarize the key points of this doctoral dissertation research, and point out areas which remain unresolved and/or are left for future researchers to complete.

Chapter 1 framed the goals of this research as a two-part problem. The first part involved the triple characterization of a web session from the user's perspective, the web server's perspective, and the browser's (client's) perspective. The second part of the problem involved identifying methods, through queueing analysis, of improving web server performance, and verifying these findings via simulation. The overall objective of this two-sided approach was to increase our understanding of how the World Wide Web works and the possible implications on Internet traffic as a whole.

In Chapter 2, we described the problem of web characterization. We illustrated a typical web session, where HTTP/1.1 is used by the browser and the server, via a state diagram, where each state corresponds to a portion of the web session. We developed three separate diagrams, one each corresponding to the user, web browser (client), and web server. Transitions from one state to another occur within the individual state diagrams and also between state diagrams, and are either event-driven (meaning that the transition is caused by a particular action) or timer-driven (meaning that the transition occurs after some time interval regardless of the actions of any of the entities). We identified the transitions and their types for all three state diagrams, and we also identified the times associated with each state. We then defined the time spent in each state and gave general descriptions of the distributions of those times. The chapter concluded with a pseudocode description of the state diagrams, which can be used to construct a computer model of a web session for simulation purposes.

In Chapter 3, we derived an optimal policy for a web server. The web server under consideration contains files with sizes drawn from a single exponential distribution. The performance of the web server is defined in terms of a *revenue function*, an exponentially decreasing function of response time. Each user pays an amount of revenue relative to the response time it experiences at a server. The objective of the server is to maximize the amount of revenue it collects on average per unit time. We determined that, in such a case, the optimal policy serves incoming requests in preemptive last-in, first-out order. Chapter 5 verified these analytical results via a simulation model, implemented in BONeS. The simulation results confirmed the superiority of the LIFO-PR algorithm as compared to two traditional service orderings, FIFO and PS. In some cases, the performance improved by several orders of magnitude, particularly under high server loads.

In Chapter 4, we extended the optimal policy analysis from Chapter 3 to include two permutations of the original system. In the first permutation, the initial reward varies among the incoming requests, meaning that different users have different service expectations. We find in this case that the optimal policy chooses to serve the incoming requests in order of decreasing potential revenue, where potential revenue is defined as the product of initial reward and the decay in the revenue function so far. This policy is similar to the last-in, first-out policy, but additionally takes the potential payoff of each request under consideration. In the second permutation, the file sizes are drawn from several exponential distributions, each with a different mean. This system mimics the case where different types of content reside on the same server and are best described by separate distributions rather than a single distribution. We found that if we exclude processor-sharing policies from the space of policies under consideration, the optimal policy chooses to serve the incoming requests in order of decreasing rate of revenue, where revenue rate is defined as the ratio of expected payoff to expected service time. However, we presented an example in which a processor-sharing policy outperforms the optimal policy just derived. Chapter 6 verified these analytical results via BONeS simulations, similarly to Chapter 5. The optimal policies, which we refer to as *parameter-based policies* since the service decision depends on the state of the system, outperformed the FIFO and PS policies by up to several orders of magnitude and presented a slight improvement over the LIFO-PR policy. Again, the differences are most pronounced at high server load. We also showed in Chapter 6 that in cases where the PS policy outperforms other policies, the improvement in performance is very small and not sustained over large sample paths.

In Chapter 7, we extended the server analysis to include alternate server configurations: servers with multiple processors and servers with faster processors. The simulation analysis presented in this chapter assumes a more aggressive measure of performance, in the form of a revenue function that decays much faster than the revenue function used in the previous chapters. We determined tradeoffs in the cost of each configuration as compared to the revenue generated per configuration, and commented on the validity of each configuration.

Several questions remain unanswered in this thesis. We leave these problems for future researchers to address. The key unanswered questions are described briefly below.

First, we have not specified the exact values and/or distributions of the times identified in the web session state diagrams in Chapter 2. The nature of this work is more suited to a Master's level project, and is left as such.

Second, we have not explicitly identified the circumstances under which processor-sharing policies outperform non-processor sharing policies when the mean file sizes are variable. We believe that in these cases, the revenue function is influenced by a second-order or higher term that did not affect the revenue function previously. However, attempts to quantify this behavior quickly grew beyond the scope of this thesis. Thus, we leave this problem for future researchers to resolve.

Finally, we have not explored alternate models for the revenue function and file size distributions. In particular, it may be instructive to apply file size models similar to those described by other researchers (i.e., the heavy-tailed distribution), or to explore file size distributions in which the server knows or can deduce the file size of an incoming request.

Bibliography

[1] Jussara Almeida, Virgilio Almeida, and David Yates. "Measuring the behavior of a World Wide Web server". Technical Report BUCS-TR-96-025, Universidade Federal de Minas Gerais, Brazil, 1996.

[2] The Apache web server. http://www.apache.org.

[3] Martin F. Arlitt and Carey L. Williamson. "Web server workload characterization: The search for invariants". In *Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 126–137, 1996.

[4] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. "TCP behavior of a busy Internet server: Analysis and improvements". In *Proceedings of IEEE INFOCOM 1998*, San Francisco, March 1998.

[5] Gaurav Banga and Peter Druschel. "Measuring the capacity of a web server". In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 61–71, Monterrey, CA, December 1997.

[6] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. "Changes in web client access patterns: Characteristics and caching implications". *World Wide Web Special Issue on Characterization and Performance Evaluation*, 1999.

[7] Paul Barford and Mark Crovella. "A performance evaluation of hyper text transfer protocols". In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 188–197, May 1999.

[8] Paul Barford and Mark Crovella. “Measuring web performance in the wide area”. *Performance Evaluation Review: Special Issue on Network Traffic Measurement and Workload Characterization*, August 1999.

[9] Paul Barford and Mark E. Crovella. “Generating representative web workloads for network and server performance evaluation”. Technical Report BUCS-TR-97-006, Boston University Department of Computer Science, November 1997.

[10] Michael Bender, Soumen Chakrabarti, and S. Muthukrishnan. “Flow and stretch metrics for scheduling continuous job streams”. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[11] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*, May 1996. RFC 1945.

[12] BONeS DESIGNER Version 3.6, revision 0.5. The Alta Group of Cadence Design Systems, Inc., Foster City, CA.

[13] Kaare Christian and Susan Richter. *The UNIX Operating System*. John Wiley and Sons, Inc., New York, third edition, 1994.

[14] Kimberly A. Claffy, Hans-Werner Braun, and George C. Polyzos. “A parameterizable methodology for Internet traffic flow profiling”. *IEEE Journal on Selected Areas in Communications*, 13(8):1481–1494, October 1995.

[15] Mark E. Crovella and Azer Bestavros. “Explaining World Wide Web traffic self-similarity”. Technical Report TR-95-015, Boston University Department of Computer Science, October 1995.

[16] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. “Connection scheduling in web servers”. Technical Report BUCS-TR-99-003, Boston University Department of Computer Science, April 1999.

[17] Mark E. Crovella, Mor Harchol-Balter, and Cristina D. Murta. “Task assignment in a distributed system: Improving performance by unbalancing load”. Technical Report BUCS-TR-97-018, Boston University Department of Computer Science, October 1997.

[18] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. “Characteristics of WWW client-based traces”. Technical Report BUCS-TR-95-010, Boston University Department of Computer Science, April 1995.

[19] Helen Custer. *Inside Windows NT*. Microsoft Press, Redmond, WA, 1993.

[20] Sidnie Feit. *TCP/IP.* McGraw-Hill, Inc., New York, second edition, 1995.

[21] Anja Feldmann, Jennifer Rexford, and Ramon Caceres. "Efficient policies for carrying web traffic over flow-switched networks". *IEEE/ACM transactions on networking*, 6(6):673–685, December 1998.

[22] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999. RFC 2616.

[23] Steven Gribble and Eric Brewer. "System design issues for Internet middleware services: deductions from a large client trace". In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[24] Mor Harchol-Balter, Mark E. Crovella, and Cristina D. Murta. "To queue or not to queue: When queueing is better than timesharing in a distributed system". Technical Report BUCS-TR-1997-017, Boston University Department of Computer Science, October 1997.

[25] Mor Harchol-Balter, Mark E. Crovella, and Cristina D. Murta. "Task assignment in a distributed server". In *10th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Lecture Notes in Computer Science*, number 1469, pages 231–242, September 1998.

[26] Mor Harchol-Balter, Mark E. Crovella, and Sung Sim Park. "The case for SRPT scheduling in web servers". Technical Report MIT-LCS-TR-767, MIT Laboratory for Computer Science, October 1998.

[27] Ronald Howard. *Dynamic Programming and Markov Processes.* MIT, Cambridge, MA, 1960.

[28] James C. Hu, Sumedh Mungee, and Douglas C. Schmidt. "Principles for developing and measuring high-performance web servers over ATM". Technical Report WUCS-97-09, Washington University Department of Computer Science, February 1997.

[29] Bernardo A. Huberman, Peter L.T. Pirolli, James E. Pitkow, and Rajan M. Lukose. "Strong regularities in World Wide Web surfing". *Science*, 280:95–97, April 1998.

[30] Internet Software Consortium. Internet domain survey, July 1999. http://www.isc.org/ds/.

[31] Van Jacobson. "Congestion avoidance and control". In *Proceedings of SIGCOMM 1988*, Stanford, CA, August 1988.

[32] Leonard Kleinrock. *Queueing Systems, Volume I: Theory.* John Wiley and Sons, Inc., New York, 1975.

[33] Leonard Kleinrock. *Queueing Systems, Volume II: Computer Applications.* John Wiley and Sons, Inc., New York, 1976.

[34] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. "User access patterns to NCSA's World Wide Web server". Technical report, University of Illinois at Urbana-Champaign Department of Computer Science, February 1995.

[35] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. "On the self-similar nature of Ethernet traffic". *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[36] Bruce A. Mah. "An empirical model of HTTP network traffic". In *Proceedings of IEEE INFOCOM 1997*, 1997.

[37] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. "Performance issues of enterprise level web proxies". In *Proceedings of ACM SIGMETRICS '97*, Seattle, WA, June 1997.

[38] Stephen Manley, Michael Courage, and Margo Seltzer. "A self-scaling and self-configuring benchmark for web servers". Technical report, Harvard University, 1997.

[39] Stephen Manley and Margo Seltzer. "Web facts and fantasy". In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[40] Microsoft Internet Information Server. http://www.microsoft.com.

[41] Jeffrey C. Mogul. "Network behavior of a busy web server and its clients". Research Report 95/5, Digital Western Research Laboratory, October 1995.

[42] Jeffrey C. Mogul. "The case for persistent connection HTTP". Research Report 95/4, Digital Western Research Laboratory, May 1995.

[43] Dan Mosedale, William Foss, and Rob McCool. "Administering very high volume Internet services". In *Proceedings of LISA IX*, pages 901–908, September 1995.

[44] NCSA Mosaic. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/.

[45] Netscape Enterprise Server programmer's bookshelf. http://developer.netscape.com/docs/manuals/enterprise/bookshelf/index.htm.

[46] Netscape Navigator. http://home.netscape.com.

[47] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, and Chris Lilley. "Network performance effects of HTTP/1.1, CSSI, and PNG". Technical Note NOTE-pipelining-970624, World Wide Web Consortium, June 1997. Available at http://www.w3.org/TR/NOTE-pipelining-970624.

[48] Venkata N. Padmanabhan and Jeffrey C. Mogul. "Improving HTTP latency". *Computer Networks and ISDN Systems*, 28:25–35, December 1995.

[49] Vern Paxson and Sally Floyd. "Wide area traffic: The failure of Poisson modeling". *IEEE/ACM transactions on networking*, 3(3):226–244, June 1995.

[50] Vern Paxson and Sally Floyd. "Why we don't know how to simulate the Internet". In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, December 1997.

[51] PHTTPD - a multithreaded web server. http://sf.www.lysator.liu.se/phttpd.

[52] James E. Pitkow. "Summary of WWW characterizations". Technical report, HTTP-NG WCG, Xerox Palo Alto Research Center, 1998.

[53] Jon B. Postel. *Transmission Control Protocol*, September 1981. RFC 793.

[54] Sheldon M. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, New York, 1983.

[55] Louis P. Slothouber. "A model of web server performance". StarNine Technologies, Inc.

[56] Simon Spero. "Analysis of HTTP performance problems". Technical report, World Wide Web Consortium, July 1994.

[57] Standard Performance Evaluation Corporation. The SPEC95 benchmarks. http://www.spec.org/.

[58] Streetprices.com. http://www.streetprices.com.

[59] Nua Internet Surveys. How many online worldwide, 1999. http://www.nua.ie/surveys/analysis/graphs_charts/comparisons/how_many_online.html.

[60] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1989.

[61] Nancy J. Yeager and Robert E. McGrath. *Web Server Technology.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[62] The Zeus web server. http://www.zeus.co.uk/products/zeus3.

APPENDIX: STATE DIAGRAM SUMMARIES AND PSEUDOCODE

Table A.1: State names

Format: *diagramState*

	STATE NAME	**DESCRIPTION**
User diagram	uClosed	Closed
	uOpen	Open browser
	uConnect	Connect to server
	uLoad	Page loading
	uStop	Stop (abort prematurely)
	uView	View page
	uDisconnect	Close connection
	uCloseApp	Close application
Client diagram	cIdle	Idle
	cSetup1	Setup TCP connection at client
	cSetup2	Setup TCP connection with server
	cRequest	Request file
	cWait1	Wait for response from server
	cReceive	Receive response from server
	cParse	Parse response/file from server
	cWait2	Wait for user action
	cTerminate	Terminate TCP connection with server
Server diagram	sIdle	Idle
	sSetup	Setup TCP connection with client
	sWait	Wait for message from client
	sProcess	Process request from client
	sResponse	Send response to client
	sTerminate	Terminate TCP connection with client

Table A.2: Event names

Format: *event.origin_state.destination_state*

	EVENT	**DESCRIPTION**
From user states	TCPConnect.uConnect.cIdle	Open TCP connection to server
	getPage.uLoad.cRequest	Retrieve web page or document
	abort.uStop.cWait1	Abort file transfer
	abort.uStop.cReceive	Abort file transfer
	abort.uStop.cParse	Abort file transfer
	getNextPage.uView.cWait2	Get next document from same server as the previous document retrieval
	newServer.uView.cWait2	Get document from different server
	endSession.uView.cWait2	Session is over; close any open TCP connection(s) to server
From client states	TCPConnect.cSetup1.sIdle	Open TCP connection
	TCPEstablished.cSetup2.uConnect	TCP connection to server established
	getFile.cRequest.sWait	Get next file in web page
	pageDone.cParse.uLoad	All files in web page have been retrieved
	TCPDisconnected.cTerminate.uDisconnect	TCP connection to server closed
	TCPDisconnected.cTerminate.uStop	TCP connection closed due to aborted file transfer
	TCPDisconnect.cTerminate.sWait	Close TCP connection
From server states	TCPEstablished.sSetup.cSetup2	TCP connection established
	startSend.sSend.cWait1	Begin file transfer
	doneSend.sSend.cReceive	End of file transfer
	TCPDisconnect.sTerminate.cWait2	Close TCP connection
	TCPDisconnected.sTerminate.cTerminate	TCP connection closed

Table A.3: Timer names

Format: *time.state*

	TIMER	**DESCRIPTION**
User diagram	intersession.uClosed	Time between web sessions
	initialize.uOpen	Time required to initialize browser software
	timeout.uLoad	User's "patience" timer
	think.uView	User "think time"
	closeApp.uCloseApp	Time required to close browser software
Client diagram	setup.cSetup1	Time required to set up client half of a TCP connection
	sendRequest.cRequest	Time required to send a file request
	parse.cParse	Time required to parse a server response
	timeout.cWait1	Client's "patience" timer
Server diagram	TCPEstablish.sSetup	Time required to set up server half of a TCP connection
	timeout.sWait	Server timeout for idle connection
	process.sProcess	Time required to process a file request
	sendResponse.sResponse	Time required to send a file or error message
	terminateTCP.sTerminate	Time required to tear down the server half of a TCP connection

Table A.4: Pseudocode description of the user state diagram actions

STATE	**ACTION**	**EVENT/TIMER/NEXT STATE**
uClosed:	wait	intersession.uClosed
	goto	uOpen
uOpen:	wait	initialize.uOpen
	goto	uConnect
uConnect:	send	TCPConnect.uConnect.cIdle
	wait	TCPEstablished.cSetup2.uConnect
	goto	uLoad
uLoad:	send	getPage.uLoad.cRequest
	wait	timeout.uLoad
	goto	uStop
		OR
	wait	pageDone.cParse.uLoad
	goto	uView
uStop:	send	abort.uStop.cWait1
		OR
	send	abort.uStop.cReceive
		OR
	send	abort.uStop.cParse
	wait	TCPDisconnected.cTerminate.uStop
	goto	uView
		OR
	goto	uConnect
uView:	wait	think.uView
	send	endSession.uView.cWait2
	goto	uClose
		OR
	send	newPage.uView.cWait2
	goto	uLoad
		OR
	send	newServer.uView.cWait2
	goto	uConnect
uClose:	wait	TCPDisconnected.cTerminate.uClose
	goto	uCloseApp
uCloseApp:	wait	closeApp.uCloseApp
	goto	uClosed

Table A.5: Pseudocode description of the client state diagram actions

STATE	**ACTION**	**EVENT/TIMER/NEXT STATE**
cIdle:	wait	TCPConnect.uConnect.cIdle
	goto	cSetup1
cSetup1:	wait	setup.cSetup1
	send	TCPConnect.cSetup1.sSetup
	goto	cSetup2
cSetup2:	wait	TCPEstablished.sSetup.cSetup2
	send	TCPEstablished.cSetup2.uConnect
	goto	cRequest
cRequest:	wait	getPage.uLoad.cRequest
	send	getFile.cRequest.sWait
	wait	sendRequest.cRequest
	goto	cWait1
cWait1:	wait	abort.uStop.cWait1
	goto	cTerminate
		OR
	wait	startSend.sSend.cWait1
	goto	cReceive
		OR
	wait	timeout.cWait1
	goto	cSetup1
cReceive:	wait	abort.uStop.cResponse
	goto	cTerminate
		OR
	wait	endSend.sSend
	goto	cParse
		OR
	goto	cWait1
cParse:	wait	abort.uStop.cResponse
	goto	cTerminate
		OR
	wait	parse.cParse
	goto	cRequest
		OR
	wait	parse.cParse
	send	pageDone.cParse.uLoad
	goto	cWait2

(Continued on the next page)

cWait2:	wait	TCPDisconnect.sTerminate.cWait2
	goto	cTerminate
		OR
	wait	newPage.uView.cWait2
	goto	cRequest
		OR
	wait	newServer.uView.cWait2
		OR
	wait	endSession.uView.cWait2
	goto	cTerminate
cTerminate:	send	TCPDisconnect.cTerminate.sWait
	wait	TCPDisconnected.sTerminate.cTerminate
	goto	cIdle

Table A.6: Pseudocode description of the server state diagram actions

STATE	**ACTION**	**EVENT/TIMER/NEXT STATE**
sIdle:	wait	TCPConnect.cSetup1.sIdle
	goto	sSetup
sSetup:	wait	TCPEstablish.sSetup
	send	TCPEstablished.sSetup.cSetup2
	goto	sWait
sWait:	wait	getFile.cRequest.sWait
	goto	sProcess
		OR
	wait	timeout.sWait
	goto	sTerminate
sProcess:	wait	process.sProcess
	goto	sSend
sSend:	send	startSend.sSend.cWait1
	wait	sendResponse.sSend
	send	endSend.sSend.cReceive
	goto	sWait
		OR
	goto	sTerminate
sTerminate:	send	TCPDisconnect.sTerminate.cWait2
	wait	terminateTCP.sTerminate
	send	TCPDisconnected.sTerminate.cTerminate
	goto	sIdle

VITA

Amy Csizmar Dalal

Place of birth
Buffalo, New York USA

Education

- **Ph.D. - Electrical Engineering**
  Department of Electrical and Computer Engineering
  Northwestern University, Evanston, IL, USA - 1999
- **M.S. - Electrical Engineering**
  Department of Electrical and Computer Engineering
  Northwestern University, Evanston, IL, USA - 1997
- **B.S. - Electrical Engineering**
  Department of Electrical Engineering
  University of Notre Dame, Notre Dame, IN, USA - 1994

Relevant Professional Experience

- Gooitech
  June 1998 - July 1998
  Authored technical specifications for a proprietary transport layer protocol. Developed network experiments for said protocol.

- 3Com
  June 1997 - September 1997
  Performed a source characterization of an H.323 (videoconferencing) protocol, and coauthored a technical report on the results. Developed and implemented a model of a client/server network using BONeS software.

- Ameritech Cellular Services
  June 1995 - September 1995
  January 1996 - September 1996
  Developed and implemented a model of a Cellular Digital Packet Data (CDPD) network using BONeS software. Authored a white paper on CDPD security issues. Developed a program to perform network management duties on the CDPD network.

Teaching Experience

- Instructor, Engineering Analysis 1
  Winter Quarter, 1997-98 and 1998-99 Academic Years
  Taught an introductory level engineering course covering linear algebra and basic programming concepts using Matlab. Revised portions of the existing syllabus. Created original homework assignments, computer lab exercises, and examinations. Responsible for assigning final grades to each student. Supervised one teaching assistant.

- Teaching Assistant Coordinator
  Fall Quarter, 1997-98 and 1998-99 Academic Years
  Coordinated the activities of seven teaching assistants, and served as a liaison between the teaching assistants and five professors. Prepared and taught weekly recitation sections and occasional review sessions. Assigned final numeric grades to all 300+ students in the course.

- Teaching Assistant
  Fall Quarter, 1995-96 Academic Year
  1996-97, 1997-98, 1998-99 Academic Years
  Assigned to a variety of courses, from introductory to senior-level. Presided over laboratory sections in electronic circuits and digital logic design, as well as office hours and computer consultation hours in other courses. Taught weekly or biweekly review sessions. Graded homeworks, quizzes, and midterm and final exams. Developed and maintained several course web pages.

Honors and Awards

- Teaching Assistant Fellow, Searle Center for Teaching Excellence, Northwestern University, 1998-99 Academic Year
- 1998 Best Teaching Assistant Award, Department of Electrical and Computer Engineering, Northwestern University
- 1999 Best Teaching Assistant Award Honorable Mention, Department of Electrical and Computer Engineering, Northwestern University
- Tau Beta Pi member
- Eta Kappa Nu member
- Walter P. Murphy Fellow, 1994-95 Academic Year