LUND UNIVERSITY

# Performance modeling and control of web servers

Andersson, Mikael

2004

Link to publication

*Citation for published version (APA):*
Andersson, M. (2004). *Performance modeling and control of web servers*. [Licentiate Thesis, Department of Electrical and Information Technology]. Lund Institute of Technology.

*Total number of authors:*
1

**General rights**
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

**Take down policy**
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

# Performance Modeling and Control of Web Servers

Mikael Andersson

LUND INSTITUTE OF TECHNOLOGY
Lund University
Department of Communication Systems
Lund Institute of Technology

ISSN 1101-3931
ISRN LUTEDX/TETS–1068–SE+105P
©Mikael Andersson
Printed in Sweden
KFS AB
Lund 2004

To web servers

This thesis is submitted to Research Board FIME - Physics, Informatics, Mathematics and Electrical Engineering - at Lund Institute of Technology, Lund University in partial fulfilment of the requirements for the degree of Licentiate in Engineering.

**Contact information:**
Mikael Andersson
Department of Communication Systems
Lund University
P.O. Box 118
SE-221 00 LUND
Sweden

Phone: +46 46 222 49 62
Fax: +46 46 14 58 23
e-mail: mikael.andersson@telecom.lth.se

## ABSTRACT

This thesis deals with the task of modeling a web server and designing a mechanism that can prevent the web server from being overloaded. Four papers are presented. The first paper gives an M/G/1/K processor sharing model of a single web server. The model is validated against measurements and simulations on the commonly used web server Apache. A description is given on how to calculate the necessary parameters in the model. The second paper introduces an admission control mechanism for the Apache web server based on a combination of queuing theory and control theory. The admission control mechanism is tested in the laboratory, implemented as a stand-alone application in front of the web server. The third paper continues the work from the second paper by discussing stability. This time, the admission control mechanism is implemented as a module within the Apache source code. Experiments show the stability and settling time of the controller. Finally, the fourth paper investigates the concept of service level agreements for a web site. The agreements allow a maximum response time and a minimal throughput to be set. The requests are sorted into classes, where each class is assigned a weight (representing the income for the web site owner). Then an optimization algorithm is applied so that the total profit for the web site during overload is maximized.

## ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Dr. Maria Kihl, for continuously supporting and guiding me through all times in my research and always encouraging me. Gratitude also goes to my supervisor Dr. Christian Nyberg, for always being there with a good solution to any possible problem and clearing out all question marks. Thank you Prof. Ulf Körner for giving me the opportunity to pursue my doctorate in Lund. I owe my colleague Jianhua Cao a great deal of appreciation, for all the joyful cooperation in our papers. Two papers in this thesis have been written in cooperation with the Department of Automatic Control at Lund Institute of Technology; it has been a pleasure working with Anders Robertsson and Björn Wittenmark. I also greatly appreciated a travel grant from the Royal Physiographic Society. And to all my colleagues at the department and friends, it's always great fun to be around you! Finally, thank you so very much for always helping me out, Mom and Dad, and my sister Maria for always being there. I couldn't have done this without you.

Lund, Sweden, September, 2004 Mikael Andersson

# CONTENTS

# 1. INTRODUCTION

During the last years, the use of Internet has increased tremendously. More and more users connect to the Internet. In Sweden, more than 70 percent of the population used the Internet last year (according to Statistics Sweden, [1]). Not only the number of users has increased, the number of services offered on the Internet has exploded the last few years. Companies take their business onto the Internet to a greater extent. 75 percent of the companies in Sweden use Internet to market themselves. The companies are e-commerce ventures that sell records, books, clothes and services, companies that want to present themselves on the Internet, banks, gambling sites, web hotels and so on. The growth in Internet popularity has lead to increasing demands in bandwidth and performance over the Internet and both bandwidth and computer speed *have* increased steadily. However, this is not always enough. Many people still experience the WWW as the World Wide Wait. Instead of being fast and useful, the Internet is at many occasions time-consuming. The long response times do not necessarily have to depend on too little bandwidth or too slow clients. Instead, the bottleneck is often the server systems. Numerous examples can be found when web servers have become too overloaded, leaving all visitors ignored. Situations when this occur is for example when a news site reports events like sport tournaments, crises or political elections. Web shops can be hit with many visitors during sale events on the web, bank sites during pay days, regular companies when they release new products etc.

When a web server gets overloaded, the response time for a web page becomes long which affects the company, as shown in Figure 1.1. If visitors experience long response times, they tend to choose other alternatives on the web, they turn to another web shop or go to another news site. This thesis deals with the task of modeling a web server and designing a mechanism that can prevent the web server from getting overloaded. The basic idea is to reject some requests so that the remaining visitors can have a reasonable response time.

The rest of this chapter is organized as follows; section 1.1 lists the included papers, section 1.2 gives an overview of the research issues, section 1.3 discusses related work and finally a description of further work is found in section 1.4.

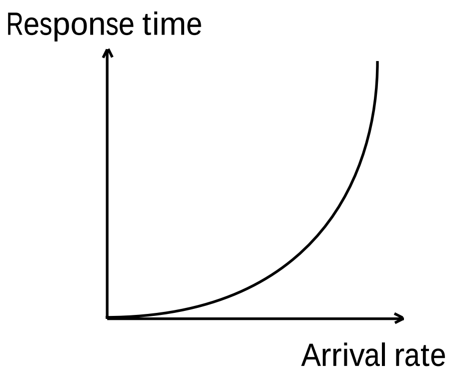

*Fig. 1.1:* The response time goes to infinity when the server load increases.

## 1.1 Summary of papers

This section describes shortly the content of the included papers.

### 1.1.1 Paper I

**Web Server Performance Modeling Using an M/G/1/K*PS Queue**
Jianhua Cao, Mikael Andersson, Christian Nyberg and Maria Kihl
*(Extended version) In Proceedings of the 10th International Conference on Telecommunications, Feb. 2003, Papeete, Tahiti*

The first paper gives a model of the web server. It uses an M/G/1/K queuing model with processor sharing as queuing discipline. The paper also deals with bursty arrival traffic. The model gives closed form expression for several performance metrics. An algorithm is presented for how to identify the parameters used in the model. The theory is validated against real-world measurements and simulations.

### 1.1.2 Paper II

**Modeling and Design of Admission Control Mechanisms for Web Servers using Non-linear Control Theory**
Mikael Andersson, Maria Kihl and Anders Robertsson
*In Proceedings of ITCOM 03, Sep. 2003, Orlando, USA*

Paper II gives an introduction to an admission control mechanism where a queuing model is combined with control theoretic methods to achieve dynamic and robust control. A PI controller is designed for the Apache web server. The goal is to control the CPU load in the web server. The control logic was implemented as a stand-alone Java application where the admission control communicates with the web server via sockets. The controller was tested in the laboratory where transient behaviour was investigated as well as the long term distribution of the CPU load.

### *1.1.3 Paper III*

**Admission Control of the Apache Web Server**
Mikael Andersson, Maria Kihl, Anders Robertsson and Björn Wittenmark
*A shorter version of this paper appears in Proceedings of the 17th Nordic Teletraffic Seminar, Aug. 2004, Fornebu, Norway*

Paper III is a continuation of paper II. In this paper we discuss stability regions for the controller. A crucial design consideration is the controller parameters. If they are chosen unwisely, the result is a controller that behaves worse than many simpler admission control mechanisms. This time, the control logic was implemented as a module in Apache, as described in section 1.2.1. Experiments show settling time and distribution of CPU load.

### *1.1.4 Paper IV*

**Admission Control with Service Level Agreements for a Web Server**
Mikael Andersson, Jianhua Cao, Maria Kihl and Christian Nyberg
*To be submitted*

The last paper investigates service level agreements for a web server. Contracts are introduced where a maximum response time and a minimal throughput are contracted. Each request is sorted into a class, where each class is assigned a weight (representing the income for the web site owner). Then an optimization algorithm is applied so that the total revenue for the web site during overload is maximized. This means that less profitable requests are more likely to be rejected. In this paper, the processing needed for rejecting a request is considered and taken into account when the optimization is performed.

Following papers are not included in this thesis:

### *Paper V*

**Performance Modeling of an Apache Web Server with Bursty Arrival Traffic**
Mikael Andersson, Jianhua Cao, Maria Kihl and Christian Nyberg
*In Proceedings of the International Conference on Internet Computing, June 2003, Las Vegas, USA*

*Paper VI*

**Design and Evaluation of Load Control in Web Server Systems**
Anders Robertsson, Björn Wittenmark, Maria Kihl and Mikael Andersson
*Invited paper, submitted to the Conference on Decision and Control (CDC), Dec. 2003, Atlantis, Bahamas*

*Paper VII*

**Admission Control Web Server Systems - Design and Experimental Evaluation**
Anders Robertsson, Björn Wittenmark, Maria Kihl and Mikael Andersson
*Invited paper, presented at the American Control Conference (ACC), June 2004, Boston, USA*

## 1.2 Research issues

This section gives an overview of the areas of research covered in this thesis. Web servers play a central part, so an explanation of web servers is given, together with a description of the architecture of Apache [2], which is the server used in the papers. Performance modeling of web servers is discussed and the general structure for an admission control mechanism in a communication system is given.

### 1.2.1 Web servers

The web server software offers access to documents stored on the server. Clients can browse the documents in a web server. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Javascript or Perl files. The communication between clients and server is based on HTTP [3]. A HTTP transaction consists of three steps: TCP connection setup, HTTP layer processing and network processing. The TCP connection setup is performed as a so called three-way handshake, where the client and the server exchange TCP SYN, TCP SYN/ACK and TCP ACK messages. Once the connection has been established, a document request can be issued with a HTTP GET message to the server. The server then replies with a HTTP GET REPLY message. Finally, the TCP connection is closed by sending TCP FIN and TCP ACK messages in both directions.

There are many web servers on the market today. Four main types can be identified; process-driven, threaded, event-driven and in-kernel web servers. Threaded and process-driven web servers are the most common, with Apache being the most popular currently. Another popular process-driven web server is Microsoft's IIS [4], covering about 21 percent of the market. Examples of event-driven web servers are Zeus [5] and Flash [6].

A description of event-driven web servers and overload control strategies for such servers is found in [7]. In-kernel web servers are servers that are executed in the operating system kernel, for example Tux [8] and khttpd [9].

### *Apache*

Introduced in 1995 and based on the popular NCSA httpd 1.3, Apache is now the most used web server in the world (Netcraft [10]). It is used in more than 67 percent of all web server systems (more than 52 millions in total, July 2004). One of the reasons to its popularity is that it is free to use. Also, since the source code is free, it is possible to modify the web server. Being threaded (threaded or process-driven depending on the operating system, on Unix, Apache uses processes, while threads are used in Win32 environments) means that Apache maintains a pool of software threads ready to serve incoming requests. Should the number of active threads run out, more can be created. When a request enters the web server, it is assigned one of the free threads, that serves it throughout the requests' lifetime. Apache puts a limit on the number of threads that are allowed to run simultaneously. If that number has been reached, requests are rejected.

### *Modules in Apache*

What makes Apache so attractive is also its architecture. The software is arranged in a kernel part and additional packages called modules. The kernel is responsible for opening up sockets for incoming TCP connections, handling static files and sending back the result. Whenever something else than a static file is to be handled, one of the designated modules takes over. For example, if a CGI page is requested, the mod_cgi module launches the CGI engine, executes the script and then returns the finished page to the kernel. Modules are convenient when new functionality should be added to a web site, because nothing has to be changed in the kernel. A new module can be programmed to respond to a certain type of request. Modules communicate with the kernel with hooks, that are well-defined points in the execution of a request. In Apache, every request goes through a life-cycle, that consists of a number of phases, as shown in Figure 1.2.

The phases are for example **Child Initialization**, **Post Read Request**, **Handlers**, and **Logger**. When a module wishes to receive a request it has to register a hook in the kernel, that is valid for one or more of the phases. For example, mod_cgi is registered to be notified in the Handlers phase, which means that once the request has reached so far in the kernel, it is delivered to the mod_cgi module. mod_cgi then performs its duties and returns the request back to the kernel. The kernel checks whether other modules wants to get a hold of the request in the Handlers phase before continuing to the next phase (Logger). The admission control in paper III was

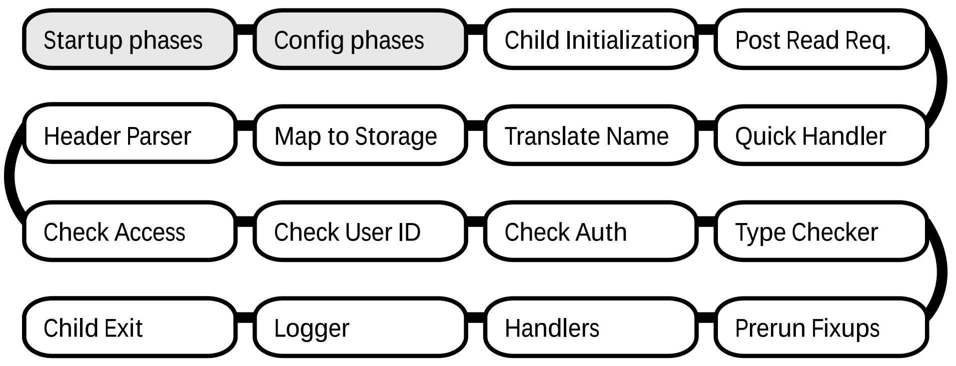

*Fig. 1.2:* The phases in the request life-cycle in Apache as of version 2.0. The shaded phases represent the phases that occur during the startup of the web server. Child initialization and exit phases are only called once in Win32 environments. In a process-driven environment (Unix), they are called for all requests.

implemented this way, by registering a hook in the Handlers phase that was called before any of the other content producing handlers and then letting the admission control module decide whether the request should be allowed to continue in the life-cycle. A more detailed description of the Apache architecture is found in [11] and [12].

### *1.2.2 Performance modeling*

To be able to design an efficient overload control it is important to have a good and reasonable performance model of the web server. It also has to be simple enough to be able to use in practise. Traditional modeling of telecommunication systems means modeling the systems as queuing systems from classical queuing theory. Queuing models are well suited for modeling web servers. A performance model is meant to answer questions like ”What is the average response time at this request rate?”, ”What is the throughput?” and ”What is the rejection probability?”. The M/G/1/K processor sharing model (shown in Figure 1.3) works good for these questions.

Using the processor sharing queuing discipline models the concept of using simultaneously executed threads or processes served in round-robin fashion in the web server well. There are many other models that quite well captures the inner-most details in web servers, for example in [13]. However, these are often complicated and explicit expressions for performance metrics are hard to obtain. Another important issue in performance modeling when it comes to overload control is that the model must capture the performance metrics well in the overload region of the web server. The model also has to be validated for these high arrival rates. Several models have been presented but they have only been validated in the normal operating region. In this

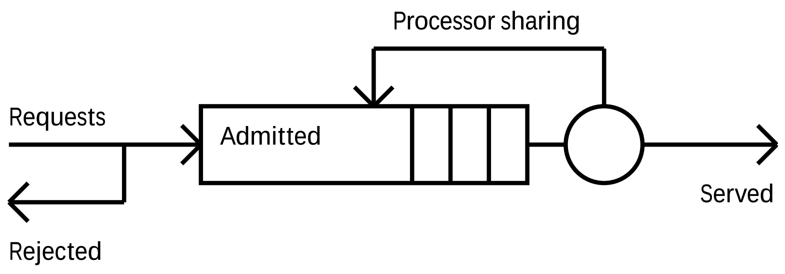

*Fig. 1.3:* The web server model.

thesis, the models have also been validated in the overload region.

### *1.2.3 Admission control*

Admission control mechanisms have since long been designed for telecommunication systems. The admission control mechanism is intended to prevent the system from becoming overloaded by rejecting visitors. Figure 1.4 shows a general structure for admission control. The structure contains three modules; the **Gate**, the **Controller** and the **Monitor**. Since continuous control is not possible in computer systems, time is divided into control intervals. In the beginning of each control interval, the Controller calculates the desired admittance rate for the next interval based on the measurements from the Monitor. The Gate then either admits or rejects visitors depending on the control signal.

#### *The Monitor*

The Monitor monitors the system through measurements on specific performance metrics. Measurements are taken each control interval. The performance metrics that are monitored differ from system to system:

**CPU load**
The Monitor measures the load in the server each control interval. The goal is to keep the load to be lower than a threshold value.

**Queue lengths**
Queue lengths can be measured, for example TCP buffers, HTTP server queues, network card buffers etc. Filled buffers indicate a high load on the server.

**Response times**
The average response time is also an important metric in overload control. If the response time is too high, the server is considered overloaded.

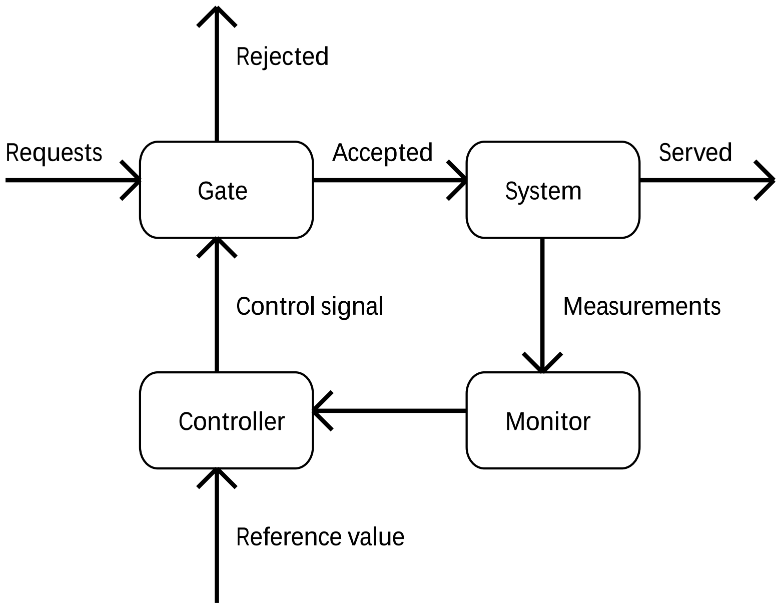

*Fig. 1.4:* An admission control mechanism.

**Call count control**
Here, the arrivals are counted. Only a certain amount of visitors are allowed at one time. This is the case in the original Apache overload control, where a maximum number of threads are set.

*The Controller*

The Controller's task is to decide how many visitors can be admitted into the system, by trying to keep a certain reference value for the desired performance metrics. It compares the actual measurements to the reference value and then reacts according to the deviation. The Controller can be designed in a variety of ways:

**Static controller**
The most simple is when the Controller has a static value that never changes, for example, "Admit 25 visitors every second."

**Step controller**
The step controller has a lower and an upper bound that it allows in the measurements. Whenever the monitored data goes above or beneath these bounds, the output signal to the Gate is increased or decreased with a fixed value per control interval.

**On-off controller**
The on-off controller works in a similar way to the step controller, but instead of increasing/decreasing the admission rate, it admits all or none of the requests in a control interval.

**PI controller**
There are several controllers that can be picked from control theory. A classical one is the PI-controller that has two parts, one proportional to the error and one that is the integral of the error.

*The Gate*

The Gate's task is to admit or reject visitors based on the Controller output. Many different gates can be found in the literature, where the most common ones are:

**Token bucket**
The token bucket algorithm generates tokens ("admission tickets") at a rate set by the Controller. If there are any available tokens upon the arrival of a request, it is admitted. Each admitted request consumes one token. Should there be no tokens, the incoming requests are buffered in a queue.

**Leaky bucket**
The leaky bucket is similar to the token bucket. Both are designed to smoothen out bursty arrival traffic. Arriving requests enter a queue, that has a limited size. If the buffer is full, the requests are rejected. Admitted requests are allowed to leave the queue at a rate set by the Controller.

**Dynamic window**
The dynamic window version works like in TCP, a number of requests are allowed to be inside of the system at the same time. The basic admission control offered in an unmodified Apache works like this, a fixed number of threads is set as an upper bound. In the Apache case, the Controller part can be seen as an on-off controller that reports to the Gate whenever the bound has been reached.

**Call gapping**
A call gapping gate admits a number of requests in the beginning of each control interval. Additional requests are rejected.

**Percent blocking**
When percent blocking is used, a percentage of the requests are admitted each control interval.

### *Content adaptation*

Admission control is one way of preventing a web server from being overloaded. Another technique is content adaption. Content adaption means that content-heavy pages are reduced during heavy load. For example, CGI scripts are very time-consuming for a processor, but nevertheless, modern web sites are often written entirely in some script language. During overload, scripted pages can be dynamically changed to static versions instead. This lowers the load on the server at the cost of lower functionality. Content adaption is not covered in this thesis. More can be read about it in [14, 15]. More on different types of overload control strategies for distributed communication networks can be found in the survey of Kihl, [16].

## *1.3 Related work*

Several attempts have been made to create performance models for web servers. Van der Mei et al. [13] modeled the web server as a tandem queuing network. The model was used to predict web server performance metrics. Wells et al. [17] have made a performance analysis of web servers using colored Petri nets. Their model is divided into three layers, where each layer models a certain aspect of the system. Dilley et al. [18] use layered queuing models in their performance studies. Cherkasova and Phaal [19] use a model that is similar to the one presented in paper I in this thesis but with deterministic service times instead. In their work they use a session-based workload with different classes of work. Beckers et al. [20] proposed a generalized processor sharing performance model for Internet access lines. They established simple relations between access line capacity and the utilization of the access line and download times of Internet objects.

When it comes to admission control, several papers cover different types of mechanisms. Few papers have investigated admission control mechanisms for server systems with control theoretic models though. Abdelzaher [21, 22] modeled the web server as a static gain to find optimal controller parameters for a PI-controller. A scheduling algorithm for an Apache web server was designed using system identification methods and linear control theory by Lu et al. [23]. Bhatti [24] developed a queue length control with priorities. By optimizing a reward function, a static control was found by Carlström [25]. An on-off load control mechanism regulating the admittance of client sessions was developed by Cherkasova and Phaal [19]. Voigt [26] proposed a control mechanism that combines load control for the CPU with a queue length control for the network interface. Bhoj [27] used a PI-controller in an admission control mechanism for a web server. However, no analysis is presented on how to design the controller parameters. Papers analyzing queueing systems with control theoretic methods usually describe the system with linear deterministic models. Stidham Jr [28] argues that deterministic models cannot be used when analyzing queueing systems.

## 1.4 Further work

The next step in my research will be to investigate information systems where high demands are put on availability and stability. The work will be part of a new project dealing with building robust systems during crises. Funded by the Swedish Emergency Management Agency, the project will focus on designing control mechanisms that allow the site to function to some extent by reducing content, rejecting customers or through a combination of both.

# BIBLIOGRAPHY

[1] "Statistics sweden," http://www.scb.se.

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

[3] W. Stallings, *Data & Computer Communications.* Prentice Hall, 2000, sixth Edition.

[4] "Microsoft internet information services," http://www.microsoft.com/WindowsServer2003/iis/default.mspx.

[5] "Zeus web server," http://www.zeus.com.

[6] "Flash web server," http://www.cs.princeton.edu/~vivek/flash/.

[7] T. Voigt, "Overload behaviour and protection of event-driven web servers," in *In proceedings of the International Workshop on Web Engineering*, May 2002, pisa, Italy.

[8] "Tux reference manual," http://www.redhat.com/docs/manuals/tux/TUX-2.1-Manual/.

[9] "khttpd web server," http://www.fenrus.demon.nl/.

[10] "Netcraft," http://www.netcraft.com.

[11] "Apache developer documentation," http://httpd.apache.org/docs-2.0/developer/.

[12] B. Laurie and P. Laurie, *Apache, The Definitive Guide.* O'Reilly, 2003.

[13] R. D. V. D. Mei, R. Hariharan, and P. K. Reeser, "Web server performance modeling," *Telecommunication Systems*, vol. 16, no. 3,4, pp. 361–378, 2001.

[14] T. Abdelzaher and N. Bhatti, "Web content adaptation to improve server overload behavior," *Computer Networks*, vol. 31, no. 11, 1999.

[15] L. C.-S. R. Mohan, J.R. Smith, "Adapting multimedia internet content for universal access," *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 104–114, 1999.

[16] M. Kihl, “Overload control strategies for distributed communication networks,” Department of Communication Systems, Lund Institute of Technology, Tech. Rep. 131, 2002, ph.D. Thesis.

[17] L. Wells, S. Christensen, L. M. Kristensen, and K. H. Mortensen, “Simulation based performance analysis of web servers,” in *Proceedings of the 9th International Workshop on Petri Nets and Performance Models (PNPM 2001).* IEEE Computer Society, 2001, pp. 59–68.

[18] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, “Web server performance measurement and modeling techniques,” *Performance Evaluation*, vol. 33, pp. 5–26, 1998.

[19] L. Cherkasova and P. Phaal, “Session-based admission control: A mechanism for peak load management of commercial web sites,” *IEEE Transactions on computers*, vol. 51, no. 6, pp. 669–685, June 2002.

[20] J. Beckers, I.Hendrawan, R.E.Kooij, and R. van der Mei, “Generalized processor sharing performance model for internet access lines,” in *9th IFIP Conference on Performance Modelling and Evaluation of ATM and IP Networks*, 2001, budapest.

[21] T. Abdelzaher and C. Lu, “Modeling and performance control of internet servers,” in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000, pp. 2234–2239.

[22] K. S. T.F. Abdelzaher and N. Bhatti, “Performance guarantees for web server end-systems: a control theoretic approach,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, January 2002.

[23] J. S. C. Lu, T.F. Abdelzaher and S. So, “A feedback control approach for guaranteeing relative delays in web servers,” in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001, pp. 51–62.

[24] N. Bhatti and R. Friedrich, “Web server support for tiered services,” *IEEE Network*, pp. 64–71, Sept/Oct 1999.

[25] R. R. J. Carlström, “Application-aware admission control and scheduling in web servers,” in *Proc. Infocom*, 2002.

[26] P. G. T. Voigt, “Adaptive resource-based web server admission control,” in *Proc. 7th International Symposium on Computers and Communications*, 2002.

[27] S. R. P. Bhoj and S. Singhal, “Web2k: Bringing qos to web servers,” *HP Labs Technical report, HPL-2000-61*, 2000.

[28] S. S. Jr., “Optimal control of admission to a queueing system,” *IEEE Transactions on Automatic Control*, vol. 30, no. 8, pp. 705–713, August 1985.

## *Web Server Performance Modeling Using an M/G/1/K*PS Queue*

Jianhua Cao, Mikael Andersson, Christian Nyberg and Maria Kihl
Department of Communication Systems, Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
Email: {jcao, mike, cn, maria}@telecom.lth.se

**Abstract**

Performance modelling is an important topic in capacity planning and overload control for web servers. We present an M/G/1/K*PS queueing model of a web server. The arrival process of HTTP requests is assumed to be Poissonian and the service discipline is processor sharing. The total number of requests that can be processed at one time is limited to K. We obtain closed form expressions for web server performance metrics such as average response time, throughput and blocking probability. Average service time and maximum number of requests being served are model parameters. The model parameters are estimated by maximizing the log-likelihood function of the measured average response time. Compared to other models, our model is conceptually simple and it is easy to estimate model parameters. The model has been validated through measurements in our lab. The performance metrics predicted by the model fit well to the experimental outcome.

## 2.1 Introduction

Performance modelling is an important part of the research area of web servers. Without a correct model of a web server it is difficult to give an accurate prediction of performance metrics. This is the basis of web server capacity planning, where models are used to predict performance in different settings [1,2].

Today web sites can receive millions of hits per day and as a result web servers may become overloaded, i.e. the arrival rate exceeds the server capacity. To cope with this, overload control can be used, which means that some requests are allowed to be served by the web server and some are rejected. In this way the web server can achieve reasonable service times for the accepted requests. In overload control investigations for web servers, performance models predict improvements when using a certain overload control strategy [3,4]. Overload control is a research area of its own, but it is still depending on performance models. It is therefore important to have a model that is valid also in the overloaded work region.

Several attempts have been made to create performance models for web servers. Van der Mei et al. [5] have modeled the web server as a tandem queuing network. The model was used to predict web server performance metrics and was validated through measurements and simulations. Wells et al. [6] have made a performance analysis of web servers using colored Petri nets. Their model is divided into three layers, where each layer models a certain aspect of the system. The model has several parameters, some of which are known. Unknown parameters are determined by e.g. simulations. Dilley et al. [7] use layered queuing models in their performance studies. Cherkasova and Phaal [8] use a model that is similar to the one presented in this paper, but with deterministic service times instead. In their work they use a session-based workload with different classes of work. Beckers et al. [9] proposed generalized processor sharing performance models for Internet access lines. The models are used to describe the flow-level characteristics of the traffic carried by Internet access line. They established simple relations between the access line capacity and the utilization of the access line and download times of Internet objects.

However, several of the previous models are complicated. It lacks a simple model that is still valid in the overloaded work region. A simple model is also able to give accurate predictions of web server performance, but it renders a smaller parameter space compared to a complicated one, i.e. fewer parameters to estimate. Also, in a more complicated model some parameters can be difficult to estimate.

A model like the M/M/1/K or M/D/1/K with a First-Come-First-Served (FCFS) service discipline can predict web server performance quite well. But conceptually it is difficult to assume that the service time distribution is exponential or deterministic and that the service discipline is FCFS.

In this paper we describe a web server model that consists of a processor sharing node with a queue attached to it. The total number of jobs in the system is limited. The arrival process to the server is assumed to be Poissonian, whereas the service time distribution is arbitrary. A system like this is called an M/G/1/K system with processor sharing. The average service time and the maximum number of jobs are model parameters that can be determined by a maximum likelihood estimation. We have derived closed form expressions for web server performance metrics such as throughput, average response time and blocking probability. Compared to others, our model is simple but accurate enough when predicting performance.

We also investigate a slightly modified version of the model, where the arrival traffic is not assumed to be the Poisson process. Instead we let a two-state Markov Modulated Poisson Process (MMPP). MMPP's are commonly used to represent bursty arrival traffic to communication systems, such as web servers (Scott et al. [10]). By simulating the system, we were able to obtain the web server performance metrics mentioned above.

Our validation environment consists of a server and two computers representing the clients connected through a switch. The measurements validate the model. Results show that the model can predict both lighter loaded and overloaded region performance metrics.

The rest of the paper is organized as follows: The next section gives an overview of how a web server works. It also defines what an M/G/1/K system with processor sharing is. In section 2.3 we describe our new web server model and derive expressions for performance metrics of a web server. We explain maximum likelihood estimations of our model parameters in Section 2.4. Section 2.5 shows how we have validated the model through experiments and section 2.6 shows the results and gives a discussion on the results. The last section gives a conclusion of the work.

## 2.2 Preliminaries

This section describes how web servers work and gives a background on the theory of an M/G/1/K queue with processor sharing.

### 2.2.1 Web servers

A web server contains software that offers access to documents stored on the server. Clients can browse the documents in a web browser. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Javascript or Perl files. The communication between clients and server is based on HTTP [11].

A HTTP transaction consists of three steps: TCP connection setup, HTTP layer processing and network processing. The TCP connection setup is performed as a so called three-way handshake, where the client and the server exchange TCP SYN, TCP SYN/ACK and TCP ACK messages. Once the connection has been established, a document request can be issued with a HTTP GET message to the server. The server then replies with a HTTP GET REPLY message. Finally, the TCP connection is closed by sending TCP FIN and TCP ACK messages in both directions.

Apache [12], which is a well-known web server and widely used, is multi-threaded. This means that a request is handled by its own thread or process throughout the life cycle of the request. Other types of web servers e.g. event-driven ones also exist [13]. However, in this paper we consider only the Apache web server. Apache also puts a limit on the number of processes allowed at one time in the server.

### *2.2.2 M/G/1/K*PS queue*

Consider an M/G/1/K queue with processor sharing discipline. The arrival of jobs is according to a Poisson process with rate $\lambda$. The service time requirements have a general distribution with mean $\bar{x}$. An arrival will be blocked if the total number of jobs in the system has reached a predetermined value $K$. A job in the queue receives a small quantum of service and is then suspended until every other job has received an identical quantum of service in a round-robin fashion. When a job has received the amount of service required, it leaves the queue. Such a system can also be viewed as a queueing network with one node [14].

The probability mass function (pmf) of the total number of jobs in the system has the following expression,

$$P[N = n] = \frac{(1-\rho)\rho^n}{(1-\rho^{K+1})}, \tag{2.1}$$

where $\rho$ is the offered traffic and is equal to $\lambda\bar{x}$. We note that the M/M/1/K queue has the same pmf [15, 16]. However in M/M/1/K queue, the service time distribution must be exponential and service discipline must be FCFS.

## *2.3 Web Server Model*

We model the web server using an M/G/1/K queue with processor sharing as Figure 2.1 shows. The requests arrive according to a Poisson process with rate $\lambda$. The average service requirement of each request is $\bar{x}$. The service can handle at most $K$ requests at a time. A request will be blocked if the number has been reached. The probability of blocking is denoted as $P_b$. Therefore the rate of blocked requests is given by $\lambda P_b$.

From (2.1) we can derive the following three performance metrics, average response time, throughput and blocking probability.

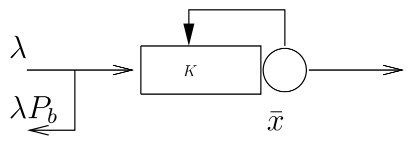

*Fig. 2.1:* An M/G/1/K-PS model of web servers

The blocking probability $P_b$ is equal to the probability that there are $K$ jobs in the system, i.e. the system is full,

$$P_b = P[N = K] = \frac{(1-\rho)\rho^K}{(1-\rho^{K+1})}. \tag{2.2}$$

where $\rho = \lambda\bar{x}$.

The throughput $H$ is the rate of completed requests. When web server reaches equilibrium, $H$ is equal to the rate of accepted requests,

$$H = \lambda(1 - P_b). \tag{2.3}$$

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

$$T = \frac{\bar{N}}{H} = \frac{\rho^{K+1}(K\rho - K - 1) + \rho}{\lambda(1-\rho^K)(1-\rho)} \tag{2.4}$$

### 2.3.1 Bursty Arrival Traffic

When it comes to modeling bursty arrival traffic, we use a different arrival process. Let the requests arrive according to a two-state Markov Modulated Poisson Process (MMPP) with parameters $\lambda_1$, $\lambda_2$, $r_1$, $r_2$. An MMPP is a doubly stochastic Poisson process where the rate process is determined by a continuous-time Markov chain. A two-state MMPP (also known as MMPP-2) means that the Markov chain consists of two different states, $S_1$ and $S_2$. The Markov chain changes state from $S_1$ to $S_2$ with rate $r_1$, and transits back with rate $r_2$. When the MMPP is in state $S_1$, the arrival process is a Poisson process with rate $\lambda_1$, and when the MMPP is in state $S_2$, rate $\lambda_2$ is used, according to Figure 2.2. The mean rate $\bar{\lambda}$ and the variance $v$ in a two-state MMPP are given as follows, see e.g. Heffes [17]:

$$\bar{\lambda} = \frac{\lambda_1 r_2 + \lambda_2 r_1}{r_1 + r_2} \tag{2.5}$$

and

$$\bar{v} = \frac{r_1 r_2(\lambda_1 - \lambda_2)^2}{(r_1 + r_2)^2} \tag{2.6}$$

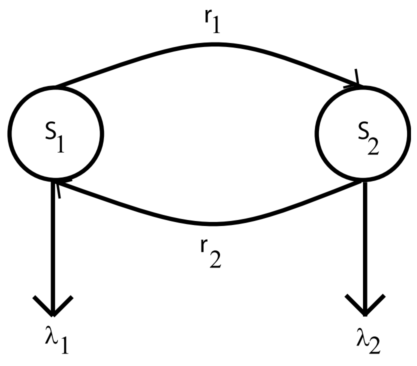

*Fig. 2.2:* The MMPP state model

## *2.4 Parameter Estimation*

There are two parameters, $\bar{x}$ and $K$, in our model. We assume that the average response time for a certain arrival rate can be estimated from measurements. The estimations, $\hat{\bar{x}}$ and $\hat{K}$, are obtained by maximizing the likelihood function of the observed average response time.

Let $T_i$ be the average response time predicted from the model and $\hat{T}_i$ be the average response time estimated from the measurements when the arrival intensity is $\lambda_i, i = 1 \ldots m$. Since the estimated response time $\hat{T}$ is the mean of samples, it is approximately a normal distributed random variable with mean $T$ and variance $\sigma_T^2/n$ when the number of samples $n$ is very large. Hence, the model parameter pair $(\bar{x}, K)$ can be estimated by maximizing the log-likelihood function

$$\log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma_i^2/n_i}} \exp\left[\frac{\left(\hat{T}_i - T_i\right)^2}{2\sigma_i^2/n_i}\right]. \tag{2.7}$$

Maximizing the log-likelihood function above is equivalent to minimize the weighted sum of square errors as follows,

$$\sum_{i=1}^{m} \frac{\left(\hat{T}_i - T_i\right)^2}{\sigma_i^2/n_i}. \tag{2.8}$$

As an approximation, the estimated variance of response time, $\hat{\sigma_i}^2$, can be used instead of $\sigma_i^2$.

Now, the problem of parameter estimation becomes a question of optimization,

$$(\hat{\bar{x}}, \hat{K}) = \underset{(\bar{x},K)}{\arg} \ \min \sum_{i=1}^{m} \frac{\left(\hat{T}_i - T_i\right)^2}{\hat{\sigma_i}^2/n_i}. \tag{2.9}$$

The optimization can be solved in various ways, such as steepest decent, conjugate gradient, truncated Newton and even brute force searching. In this paper, we used a brute force approach. The optimum parameter is selected by examining every point of the discretized parameter space.

### 2.4.1 MMPP Parameters

To be able to use the MMPP in our experiments, its parameters had to be determined. We chose to set the mean arrival rate for the MMPP process, and then determine MMPP parameters from that value. $r_1$ and $r_2$ were set to 0.05 and 0.95 respectively. The low rate, $\lambda_1$, was set to

$$\lambda_1 = 0,75 \cdot \bar{\lambda} \tag{2.10}$$

Equation 2.5 then gives:

$$\lambda_2 = \frac{((r_1 + r_2) \cdot \bar{\lambda} - \lambda_1 r_2)}{r_1} \tag{2.11}$$

This means that $\lambda_2$ is a high rate and that it can be seen as a sudden burst rate. $\lambda_2$ will be used 5 % of the time according to the settings of $r_1$ and $r_2$. The parameters have been set this way in order to simulate bursty traffic with random peaks in the arrival rate in both measurements and simulations.

## 2.5 Experiments

### 2.5.1 Setup

Our validation experiments used one server computer and two client computers connected through a 100 Mbits/s Ethernet switch. The server was a PC Pentium III 1700 MHz with 512 MB RAM. The two clients were both PC Pentium III 700 with 256 MB RAM.

All computers used RedHat Linux 7.3 as operating system. Apache 1.3.9 [12] was installed in the server. We used the default configuration, except for the maximum number of connections. The client computers were installed with a HTTP load generator, which was a modified version of S-Client [18]. The S-Client is able to generate high request rates even with few client computers by aborting TCP connection attempts that take too long time. The original version of S-Client uses deterministic waiting time between requests. We used exponential distributed waiting time instead. This makes the arrival process Poissonian [19].

The clients were programmed to request dynamically generated HTML files from the server. The CGI script was written in Perl. It generates a fix number, $N_\mathrm{r}$, of random numbers, adds them together and returns the summation. By varying $N_\mathrm{r}$, we can simulate different loads on the web server.

*Tab. 2.1:* The configuration of four experiments

| | $N_\mathrm{r} = 1000$ | $N_\mathrm{r} = 2000$ |
|---|---|---|
| $N_\mathrm{conn,max} = 75$ | A1 | B1 |
| $N_\mathrm{conn,max} = 150$ | A2 | B2 |

*Tab. 2.2:* Estimated Parameters of the Model

| | A1 | A2 | B1 | B2 |
|---|---|---|---|---|
| $\hat{\bar{x}}$ | 0.00708 | 0.00708 | 0.00866 | 0.00834 |
| $\hat{K}$ | 208 | 286 | 215 | 298 |

The system was also implemented as a discrete event simulation program in Java to be able to compare the results from the measurements with bursty arrival traffic.

### *2.5.2 Performance metrics*

We were interested in the following performance metrics: average response time, throughput, and blocking probability. The throughput was estimated by taking the ratio between the total number of successful replies and the time span of measurement. The response time is the time difference between when a request is sent and when a successful reply is fully received. The average response time was calculated as the sample mean of the response times after removing transients. An HTTP request sent by a client computer will be blocked either when the maximum number of connections, denoted as $N_\mathrm{conn,max}$, in the server has been reached or the TCP connection is timed out at the client computer. A TCP connection will be timed out by a client computer when it takes too long time for the server to return an ACK of the TCP-SYN. The blocking probability was then estimated as the ratio between the number of blocking events and the number of connection attempts in a measurement period.

For both Poisson and MMPP traffic, we carried out the experiments in four cases by varying $N_\mathrm{r}$ and $N_\mathrm{conn,max}$. Table 2.1 shows the configurations of four experiments: A1, A2, B1 and B2. In each case, the performance metrics were collected while the arrival rate (in number of requests/second) changed from 20 to 300 with step size 20.

The performance metrics can be seen in figures 2.3, 2.4, 2.5 and 2.6. The results from the different measurements are compared in the figures to mathematical expressions and simulations respectively.

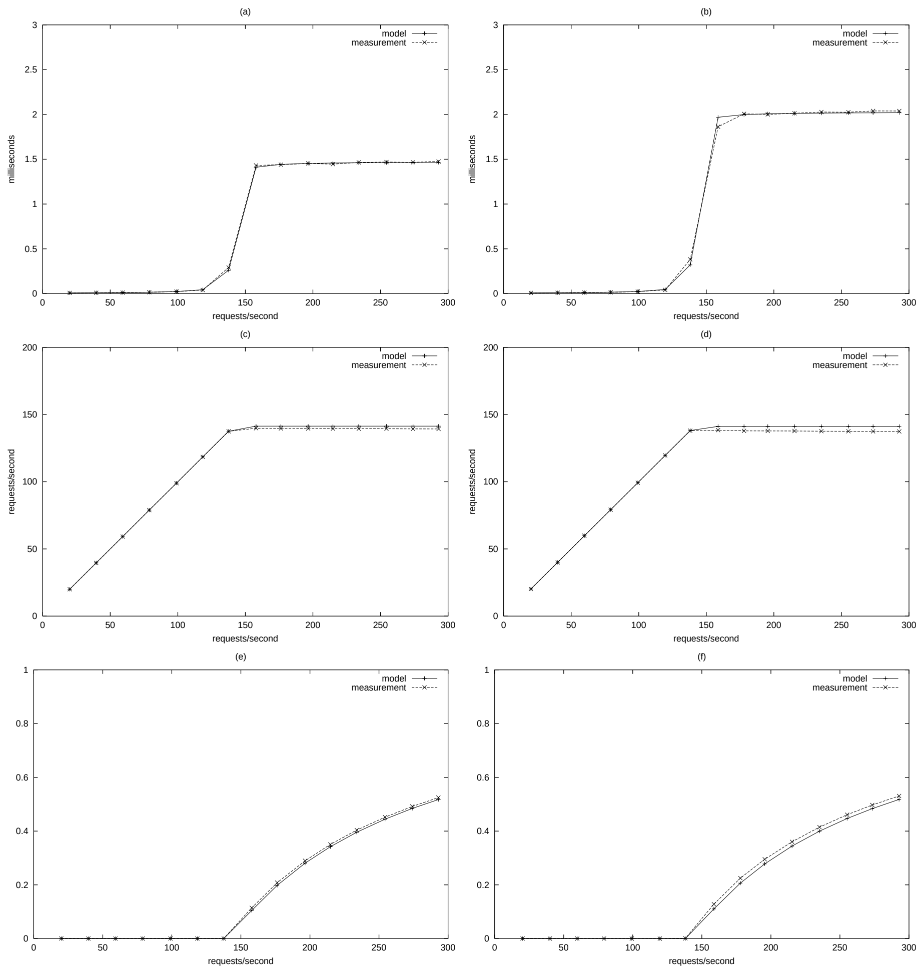

*Fig. 2.3:* Poissonian traffic: (a) Average response time of A1. (b) Average response time of A2. (c) Throughput of A1. (d) Throughput of A2. (e) Blocking probability of A1. (f) Blocking probability of A2.

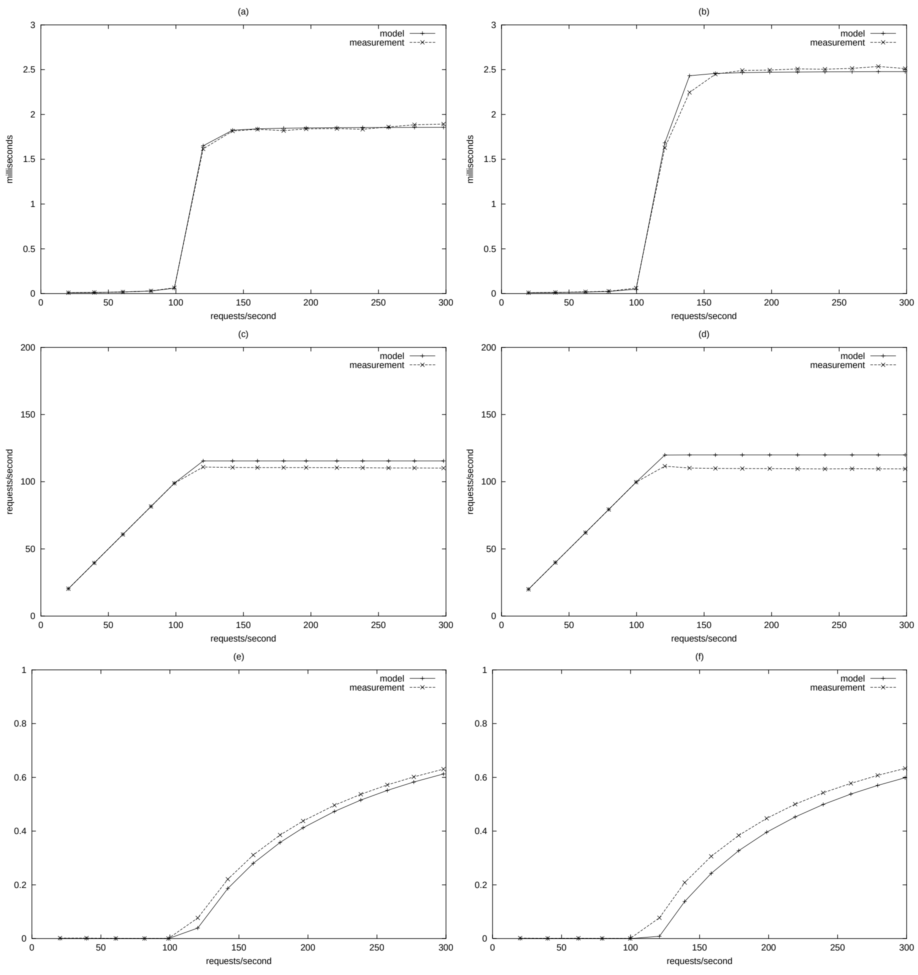

*Fig. 2.4:* Poissonian traffic: (a) Average response time of B1. (b) Average response time of B2. (c) Throughput of B1. (d) Throughput of B2. (e) Blocking probability of B1. (f) Blocking probability of B2.

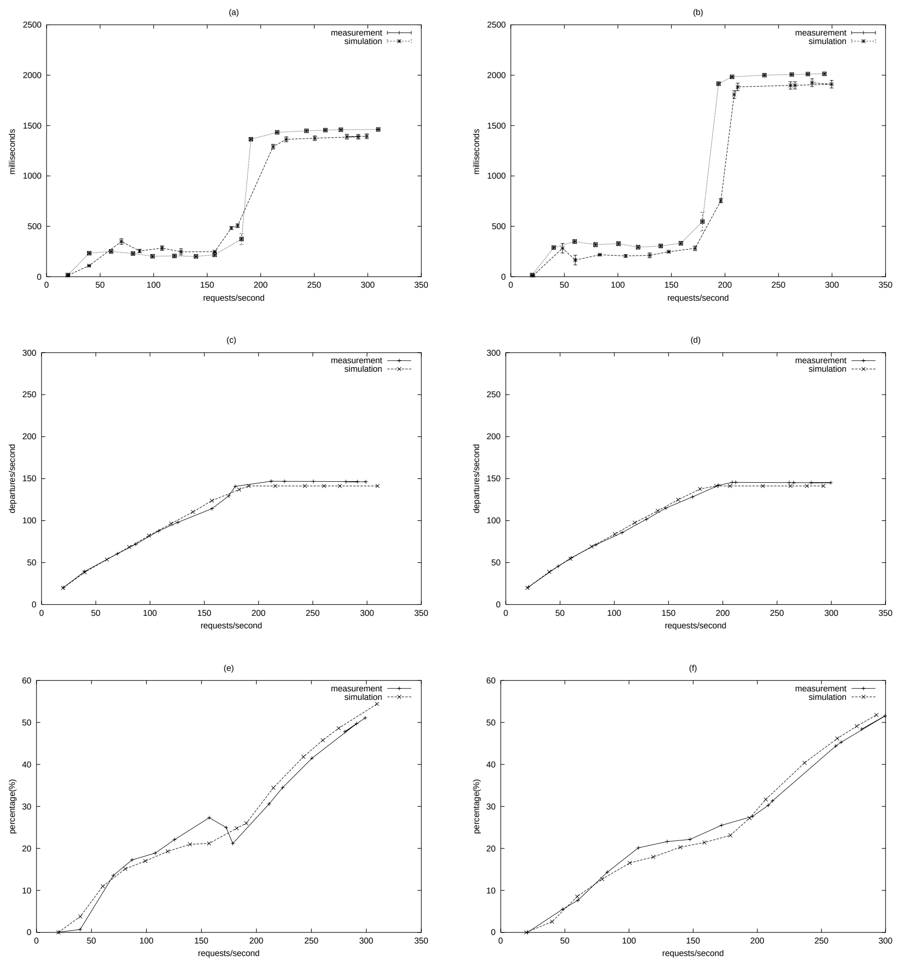

*Fig. 2.5:* MMPP traffic: (a) Average response time of A1. (b) Average response time of A2. (c) Throughput of A1. (d) Throughput of A2. (e) Blocking probability of A1. (f) Blocking probability of A2.

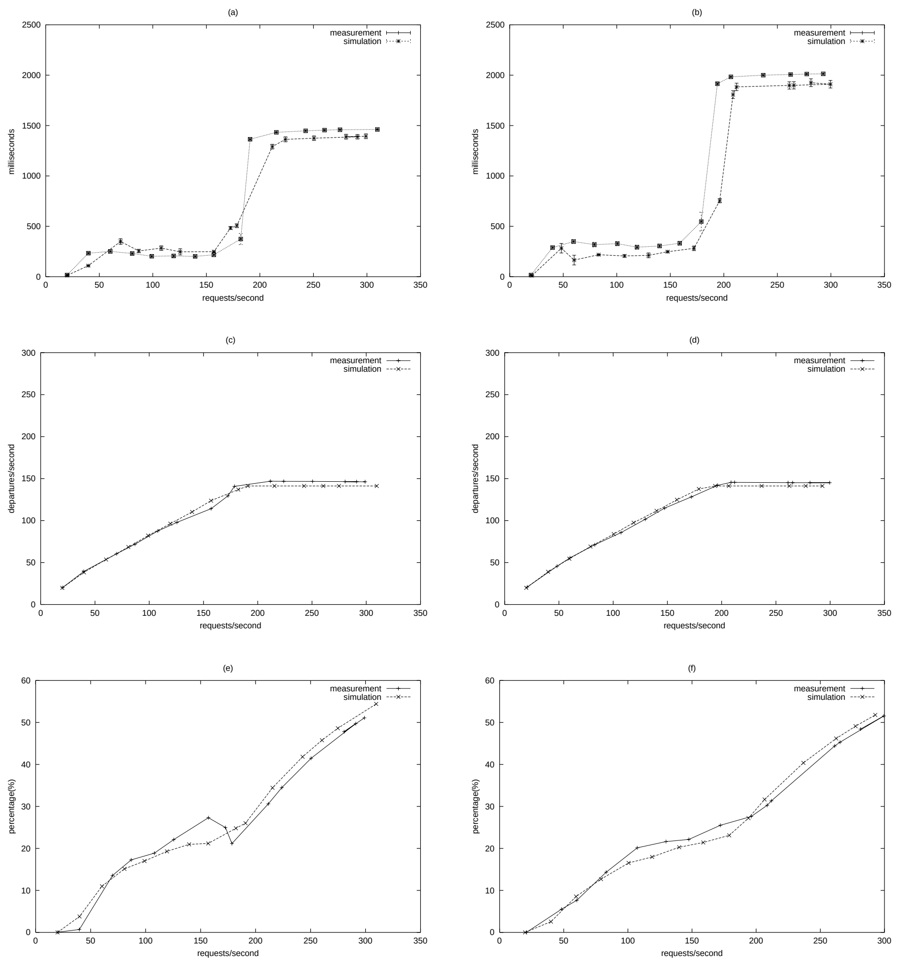

*Fig. 2.6:* MMPP traffic: (a) Average response time of B1. (b) Average response time of B2. (c) Throughput of B1. (d) Throughput of B2. (e) Blocking probability of B1. (f) Blocking probability of B2.

## 2.6 Results and Discussion

The method developed in section 2.4 was used to estimate the parameters from the measurements. The results are presented in Table 2.2.

Using the estimated parameters we can compare measured and predicted web server performance. Figure 2.3 and 2.4 shows average response time, throughput and blocking probability curves. To facilitate the discussion, we divide four experiments into two groups. The first group called $\alpha$ contains experiments A1 and A2 and the second group $\beta$ contains B1 and B2.

We notice the following relations in Table 2.2

$$\hat{\bar{x}}_{\mathrm{A1}} = \hat{\bar{x}}_{\mathrm{A2}} < \hat{\bar{x}}_{\mathrm{B1}} \approx \hat{\bar{x}}_{\mathrm{B2}}.$$

Recall that the same CGI script is used for experiments in the same group. The script for group $\beta$ is more computational intensive than the one for group $\alpha$. The script for group $\alpha$ adds 1000 numbers but the script for group $\beta$ adds 2000 numbers. However $\bar{x}_{\mathrm{B1}}$(or $\bar{x}_{\mathrm{B2}}$) is not twice as large as $\bar{x}_{\mathrm{A1}}$(or $\bar{x}_{\mathrm{A2}}$). This can be understood as that the time spent on the summations is only a fraction of the sojourn time of a job in the system. Other parts of $\bar{x}$ include the connection setup time, the file transferring time, etc., which can be considered as constants in all experiments.

We find that the estimated $K$ in all experiments is much greater than $N_{\mathrm{conn,max}}$ which is a parameter in the configuration of the Apache. One may expect that $K \approx N_{\mathrm{conn,max}}$. However, recognize that in our model $K$ is the limit of the total number of jobs in the system. The jobs can be in the HTTP processing phase as well as in the TCP connection setup phase in which the Apache has no control. On the other hand, $N_{\mathrm{conn,max}}$ is the maximum number of jobs handled by the Apache which runs on top of the TCP layer. Therefore $K$ should be greater than $N_{\mathrm{conn,max}}$.

One can reasonably predict that within the same experiment group, $\alpha$ or $\beta$, the difference of $\hat{K}$ should be approximately equal to the difference of $N_{\mathrm{conn,max}}$ which is 75. In our experiments, $\hat{K}_{\mathrm{A2}} - \hat{K}_{\mathrm{A1}} = 78$, $\hat{K}_{\mathrm{B2}} - \hat{K}_{\mathrm{B1}} = 83$. There is a reason why the differences are close but greater than 75. When $N_{\mathrm{conn,max}}$ is increased, the average load of CPU will increase besides the increase of the total number of jobs in the system. As a result, the TCP listening queue will be visited less frequently by the operating system. This implies that the TCP listening queue size will increase. So the increase of $K$ will be close but greater than the increase of $N_{\mathrm{conn,max}}$. This explanation is also supported by the fact that the increase of $K$ in the experiment group $\beta$ is larger than in group $\alpha$. As we mentioned early, the CGI script of group $\beta$ is more CPU demanding than that of group $\alpha$.

Now we turn our attention from the estimated parameters to the predicted performance metrics. The measured and the predicted average response time in all four experiments fit well. This should be of a little surprise because the measured average response times at various arrival rates are used to estimate the parameters of the model.

The predicted blocking probability is slightly less than the measurements in all four experiments. According to (2.3), the error in the prediction of $P_b$ will also affect the prediction of throughput. Such divergence is expected since we only use the measured average response time in our parameter estimation.

## 2.7 Conclusions

We have presented an M/G/1/K*PS queueing model of a web server. We obtained closed form expressions for web server performance metrics such as average response time, throughput and blocking probability. Model parameters were estimated from the measured average response time. A modified arrival traffic model was also investigated. We validated the two versions of the model through four sets of experiments. The performance metrics predicted by the model fitted well to the experimental outcome.

Future work will include more validation under different types of loads such as network intensive and hard-disk intensive cases. It would also be interesting to see how well the model fits web servers that use an event-driven approach instead of multi-threading.

## 2.8 Acknowledgments

We would like to thank Thiemo Voigt for sharing his code with us and Niklas Widell for useful and interesting discussions. The work has been supported by The Swedish Research Council under contract no. 621-2001-3053.

# BIBLIOGRAPHY

[1] D. A. Menascé and V. A. F. Almeida, *Capacity Planning for Web Services*. Prentice Hall, 2002.

[2] J. Hu, S. Mungee, and D. Schmidt, "Principles for developing and measuring high-performance web servers over ATM," in *Proceedings of INFOCOM '98, March/April 1998*, 1998.

[3] N. Widell, "Performance of distributed information systems," Department of Communication Systems, Lund Institute of Technology, Tech. Rep. 144, 2002, lic. Thesis.

[4] J. Cao and C. Nyberg, "On overload control through queue length for web servers," in *16th Nordic Teletraffic Seminar*, 2002, espoo, Finland.

[5] R. D. V. D. Mei, R. Hariharan, and P. K. Reeser, "Web server performance modeling," *Telecommunication Systems*, vol. 16, no. 3,4, pp. 361–378, 2001.

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

[7] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, "Web server performance measurement and modeling techniques," *Performance Evaluation*, vol. 33, pp. 5–26, 1998.

[8] L. Cherkasova and P. Phaal, "Session-based admission control: A mechanism for peak load management of commercial web sites," *IEEE Transactions on computers*, vol. 51, no. 6, pp. 669–685, June 2002.

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

[10] P. S. S.L. Scott, "The markov modulated poisson process and markvo poisson cascade with applications to web traffic modelling"," *Bayesian Statistics*, 2003.

[11] W. Stallings, *Data & Computer Communications.* Prentice Hall, 2000, sixth Edition.

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

[13] T. Voigt, "Overload behaviour and protection of event-driven web servers," in *In proceedings of the International Workshop on Web Engineering*, May 2002, pisa, Italy.

[14] P. J. B. King, *Computer and Communication Systems Performance Modelling.* Prentice Hall, 1990.

[15] L. Kleinrock, *Queueing Systems, Volume 1: Theory.* John Wiley & Sons, 1975.

[16] S. Lam, "Queueing networks with population size constraints," *IBM Journal of Research and Development*, vol. 21, no. 4, pp. 370–378, July 1977.

[17] H. Heffes, "A class of data traffic processes - covariance function characterization and related queuing results," *The Bell System Technical Journal*, vol. 59, no. 6, 1980.

[18] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *USENIX Symposium on Internet Technologies and Systems*, December 1997, pp. 61–71.

[19] R. Jain, *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

## 3. PAPER II

# *Modeling and Design of Admission Control Mechanisms for Web Servers using Non-linear Control Theory*

Mikael Andersson, Maria Kihl, Anders Robertsson, Björn Wittenmark
Department of Communication Systems, Lund Institute of Technology
Email: {mike, maria}@telecom.lth.se
Department of Automatic Control, Lund Institute of Technology
Email: {andersro,bjorn}@control.lth.se
Box 118, SE-221 00 Lund, Sweden

**Abstract**

Web sites are exposed to high rates of incoming requests. Since web sites are sensitive to overload, admission control mechanisms are often implemented. The purpose of such a mechanism is to prevent requests from entering the web server during high loads. This paper presents how admission control mechanisms can be designed and implemented with a combination of queueing theory and control theory. Since web servers behave non-linear and stochastic, queueing theory can be used for web server modeling. However, there are no mathematical tools in queueing theory to use when designing admission control mechanisms. Instead, control theory contains the needed mathematical tools. By analyzing queueing systems with control theoretic methods, good admission control mechanisms can be designed for web server systems. In this paper we model an Apache web server as a GI/G/1-system. Then, we use control theory to design a PI-controller, commonly used in automatic control, for the web server. In the paper we describe the design of the controller and also how it can be implemented in a real system. The controller has been implemented and tested together with the Apache web server. The server was placed in a laboratory network together with a traffic generator which was used to represent client requests. Measurements in the laboratory setup show how robust the implemented controller is, and how it correspond to the results from the theoretical analysis.

## 3.1 Introduction

Web sites on the Internet can be seen as server systems with one or more web servers processing incoming requests at a certain rate. The web servers have a waiting-queue where requests are queued while waiting for service. Therefore, a web server can be modeled as a queueing system including a server with finite or infinite queues. One problem with web servers is that they are sensitive to overload. The servers may become overloaded during temporary traffic peaks when more requests arrive than the server is designed for. Because overload usually occur rather seldom, it is not economical to overprovision the servers for these traffic peaks, instead admission control mechanisms can be implemented in the servers. The admission control mechanism rejects some requests whenever the arriving traffic is too high and thereby maintains an acceptable load in the system. The mechanism can either be static or dynamic. A static mechanism admits a predefined rate of requests whereas a dynamic mechanism contains a controller that, with periodic time intervals, calculates a new admission rate depending on some control objective. The controller bases its decision from measurements of some control variable, for example the queue length, processor occupancy, or processing delays. The control objective is usually that the value of the control variable should be kept at a reference value. The choice of control variable is an important issue when developing an admission control scheme. First, the control variables must be easy to measure. Second, the control variable must in some way relate to the QoS demands that the users may have on the system. Traditionally, server utilization or queue lengths have been the variables mostly used in admission control schemes. For web servers, the main objective of the control scheme is to protect it from overload. As long as the average server utilization or queue length is below a certain level, the response times are low.

One well-known controller in automatic control is the PID-controller, which enables a stable control for many types of systems (see, for example Åström [1]). The PID-controller uses three actions: one proportional, one integrating, and one derivative. In order to get the system to behave well it is necessary to decide proper control parameters. Therefore, before designing the PID-controller, the system must be analysed so that its dynamics during overload are known. This means that the system must be described with a control theoretic method. If the model is linear, it is easily analysed with linear control theoretic methods. However, a queueing system is both non-linear and stochastic. The main problem is that nonlinear models are much harder to analyse with control theoretic methods. Very few papers have investigated admission control mechanisms for server systems with control theoretic methods. Abdelzaher [2,3] modelled the web server as a static gain to find optimal controller parameters for a PI-controller. A scheduling algorithm for an Apache web server was designed using system identification methods and linear control theory by Lu et al [4]. Bhatti [5] developed a queue length control with priorities. By optimizing a reward function, a static control was found by Carlström [6]. An on-off load control mechanism regulating the admittance of client sessions was developed by Cherkasova [7]. Voigt [8] proposed a control mechanism that combines a load control for the CPU with a queue length control for the network interface. Bhoj [9] used a PI-controller in an admission control mechanism for a web server. However, no analysis is presented on how to design the controller parameters. Papers analyzing queueing systems with control theoretic methods usually describe the system with linear deterministic models. Stidham Jr [10]. argues that deterministic models cannot be used when analyzing queueing systems. Until now, no papers have designed admission control mechanisms for server systems using non-linear control theory. In this paper we implement an admission control mechanism for the Apache [11] web server. Measurements in the laboratory setup show how robust the implemented controller is, and that it corresponds to the results from the theoretical analysis. Section 3.2 describes a general admission control mechanism. Section 3.3 shows how this can be applied on a web server. In section 3.4, we describe a non-linear control theoretic model of an admission control mechanism for a web server. We describe the controller design in section 3.5, where examples of good and bad parameters are given. The control theoretic model is used to design and implement an admission control mechanism for the Apache web server. The measurements are shown in section 3.6, and section 3.7 concludes the work.

## *3.2 Admission Control Mechanism*

A good admission control mechanism improves the performance of a server system during overload by only admitting a certain amount of requests at a time into the system. Admission control mechanisms for server systems usually have the same structure and are based on the same type of rejection mechanisms.

Figure 3.1 shows a general admission control mechanism that consists of three parts: a *gate*, a *controller*, and a *monitor*. The monitor measures a so called control variable, $x$. Using the control variable, the controller decides the rate, $u$, at which requests can be admitted to the system. The objective is to keep the value of the control variable as close as possible to a reference value, $x_{ref}$. The gate rejects those requests that cannot be admitted. The requests that are admitted proceed to the rest of the system. Since the admittance rate may never be larger than the arrival rate, $\lambda$, the actual admittance rate is $\bar{u}$=min[$u$, $\lambda$] requests per second. A survey of different admission control mechanisms for communication systems is given by Kihl [12].

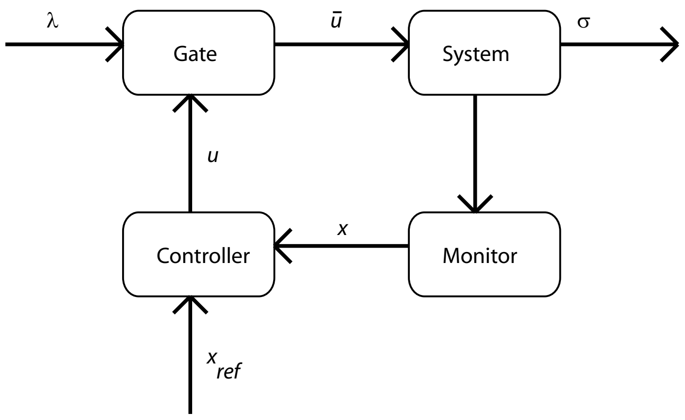

*Fig. 3.1:* An admission control mechanism

### *3.2.1 Gate*

Several gates have been proposed in the literature. One example is *Percent blocking*. In this mechanism, a certain fraction of the requests is admitted. Another example is *Token bucket*. Here, tokens are generated at a certain rate. An arriving request is admitted if there is a token available. The gate can also use a *Dynamic window* mechanism, that sets an upper limit to the number of requests that may be processed or waiting in the system the same time. The window size may be increased or decreased if the traffic conditions change.

### *3.2.2 Controllers*

There are a variety of controllers to choose from when designing an admission control mechanism. Some of the most common controllers are the *Static controller*, the *Step controller*, and the *PID-controller*.

**Static controller.** A static controller uses a fixed acceptance rate, $u_{fix}$, that is set so that the average value of the control variable should be equal to the reference value. In this case, $u_{fix}$ is given by

$$u_{fix} = \frac{\rho_{ref}}{\bar{x}}$$

**Step controller.** The objective of the control law is to keep the control variable between an upper and a lower level. If the value of the variable is higher than the upper level, the admittance rate is decreased linearly. If the value is below the lower level, the admittance rate is increased. This means that the control law is as follows:

$$u(t+1) = \begin{cases} u(t) - s & y(t) > y_{ref} + \varepsilon \\ u(t) + s & y(t) > y_{ref} - \varepsilon \end{cases}$$

where the value of s decides how much the rate is increased/decreased and the value of $\varepsilon$ decides how much the control variable may deviate from the reference value.

**PID-controller.** The PID-controller uses three actions: one proportional, one integrating, and one derivative. The control law in continuous time is as follows:

$$u(t) = K \cdot e(t) + \frac{K}{T_i} \cdot \int_0^t e(v)dv + K \cdot T_d \cdot \frac{d}{dt} \cdot e(t)$$

where e(t) is the error between the control variable and the reference value, that is $e(t) = yref - y(t)$. The gain $K$, the integral time $T_i$, and the derivative time $T_d$ are the controller parameters that are set so that the controlled system behaves as desired. A large value of $K$ makes the controller faster, but weakens the stability. The integrating action eliminates stationary errors, but may also make the system less stable. The derivative action improves the stability, however, in a system with a bursty arrival process the derivative action may cause problems. Therefore, the derivative action is usually either deleted (i.e. $T_d = 0$) or low pass filtered to remove the high frequencies.

## 3.3 Investigated System

The system we have investigated in this work, is a web server with an admission control mechanism. The web server is Apache, described below. The web server is connected to the admission control according to Figure 3.2, where the admission control runs as a stand-alone application, independent of Apache. Not all admission control mechanisms in the literature are implemented like this. It is for example possible to have admission control within the kernel or within the Apache code. This architecture however, makes the system independent of web server.

### 3.3.1 Web servers

A web server like Apache, contains software that offers access to documents stored on the server. Clients can browse the documents in a web browser. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Java script or Perl files. The communication between clients and server is based on HTTP [13]. A HTTP transaction consists of three steps: TCP connection setup, HTTP layer processing and

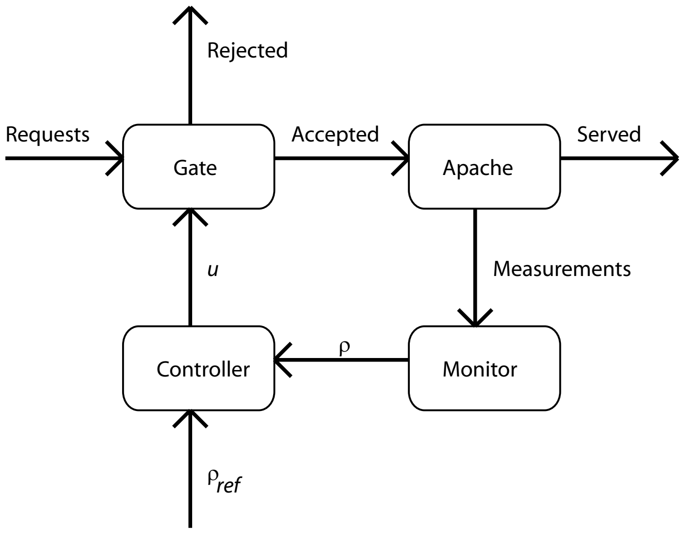

*Fig. 3.2:* The Apache web server with admission control

network processing. The TCP connection setup is performed through a threeway handshake, where the client and the server exchange TCP SYN, TCP SYN/ACK and TCP ACK messages. Once the connection has been established, a document request can be issued with a HTTP GET message to the server. The server then replies with a HTTP GET REPLY message. Finally, the TCP connection is closed by sending TCP FIN and TCP ACK messages in both directions. Apache, which is a well-known web server and widely used, is multi-threaded. This means that a request is handled by its own thread or process throughout the life cycle of the request. Other types of web servers e.g. event-driven ones also exist (Voigt [14]).

### *3.3.2 Admission control*

Since continuous control is not possible in computer systems, time is divided into control intervals of length $h$ seconds. At the end of interval $k$, that is when the time is $kh$, the controller calculates the desired admittance rate for interval $[kh, kh + h]$, denoted $u(kh)$, from the measured average server utilization during the interval, $\rho(kh)$, and the reference value $\rho_{ref}$. There are three main parts in our admission control architecture:

**Gate.** The Gate thread runs a loop that accepts or rejects incoming requests on its own TCP port. It copies the requests to Apache's TCP socket, and then sends back the responses to the clients as the web server replies. In this paper we use the token bucket algorithm to reject those requests that cannot be admitted. New tokens are generated at a rate of $u(kh)$ tokens

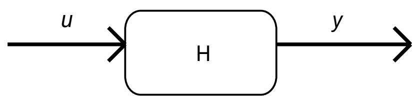

*Fig. 3.3:* A system with transfer function $H$

per second during time interval $[kh, kh + h]$. If there is an available token upon the arrival of a request, the request consumes the token and enters the web server. If there are no available tokens, the request is rejected. When a request is rejected, the TCP socket to the client is closed. Rejected requests are assumed to leave the system without retrials.

**Monitor**. The Monitor thread constantly samples the server utilization every control interval. The server utilization is calculated as one minus the fraction of time an idle process has been able to run during the last control interval. The idle process' priority level is set to the lowest possible, which means that it only runs whenever there is no request requiring CPU work. This way of measuring the load on the CPU results in a quantization effect in server utilization. The reason to this is that the operating system where the admission control mechanism runs has a certain time resolution in function calls regarding process uptimes. This means that the control interval cannot be chosen arbitrary. It has to be long enough not to be affected by the time resolution effects, and short enough so that the controller responds quickly.

**Controller**. The Controller is a PI-controller. The Controller is possible to turn on/off in order to be able to measure on an un-controlled system. The Controller's output is forwarded to the Gate thread. The Controller design is discussed more extensively in section 3.4.

## 3.4 Control Theoretic Model

Control theory is a powerful tool for performance analysis of computer controlled systems. The system must be described in terms of transfer functions or differential (or difference) equations. A transfer function describes the relationship between the z-transforms (or Laplace transforms) of the input and the output of a system, see Figure 3.3. In this case, the input to the system is the actual admittance rate, $\bar{u}$, whereas the output is either the server utilization, denoted $\rho$, or the number of jobs in the server, here denoted $x$.

We use discrete-time control theoretic model of web server developed by Kihl et al. [15]. We assume that the be modeled as a GI/G/1-system with an admission control mechanism. Kihl et al. showed that the queueing model is a good model for admission control purposes. The input to the system is

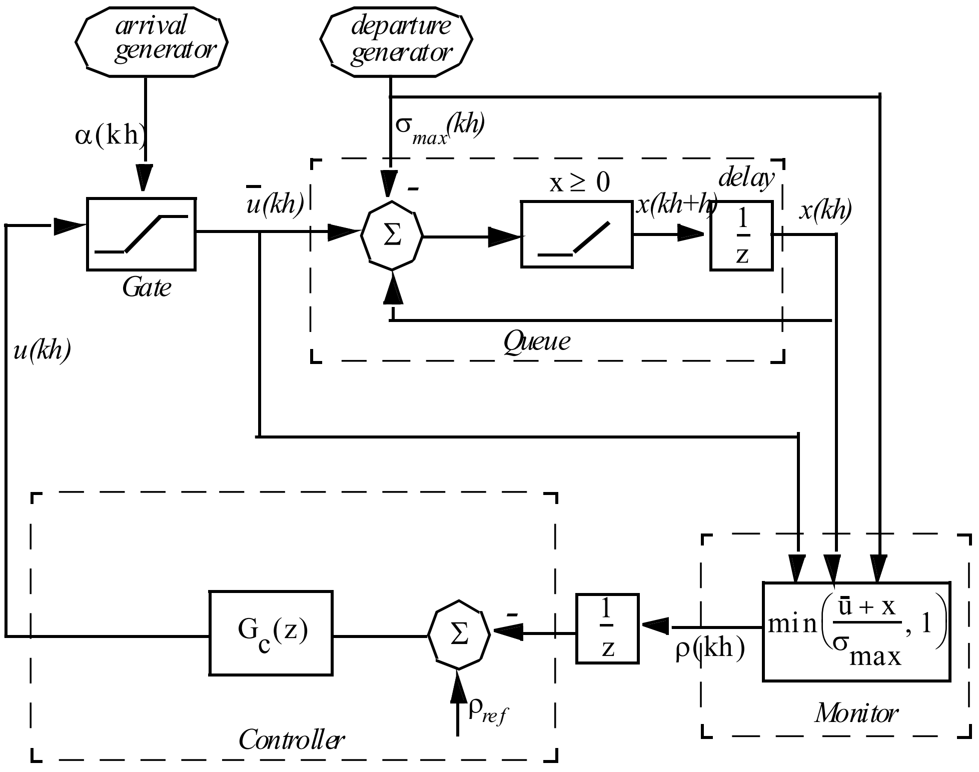

*Fig. 3.4:* A control theoretic model of a GI/G/1-system with admission control.

the actual admittance rate, $\bar{u}$, whereas the output is the server utilization, $\rho$. The model is a flow or liquid model in discrete-time. The model is an averaging model in the sense that we are not considering the specific timing of different events, arrivals, or departures from the queue. We assume that the sampling period, $h$, is sufficiently long to guarantee that the quantization effects around the sampling times are small. The model is shown in Figure 3.4. The system consists of an arrival generator, a departure generator, a controller, a queue and a monitor.

There are two stochastic traffic generators in the model. The *arrival generator* feeds the system with new requests. The number of new requests during interval $kh$ is denoted $\alpha(kh)$. $\alpha(kh)$ is an integrated stochastic process over one sampling period with a distribution obtained from the underlying interarrival time distribution. If, for example, the arrival process is Poisson with mean $\lambda$, then $\alpha(kh)$ is Poisson distributed with mean $\lambda h$. The *departure generator* decides the maximum number of departures during interval $kh$, denoted $\sigma_{max}(kh)$. $\sigma_{max}(kh)$ is also a stochastic process with a distribution given by the underlying service time distribution. If, for example, the service times are exponentially distributed with mean $1/\mu$, then $\sigma_{max}(kh)$ is

Poisson distributed with mean $\mu h$. It is assumed that $\alpha(kh)$ and $\sigma_{max}(kh)$ are independent from between sampling instants and uncorrelated to each other. The *gate* is constructed as a saturation block that limits $u(kh)$ to be

$$\bar{u}(kh) = \begin{cases} 0 & u(kh) < 0 \\ u(kh) & 0 \le u(kh) \le \alpha(kh) \\ \alpha(kh) & u(kh) > \alpha(kh) \end{cases}$$

The *queue* is represented by its state $x(kh)$, which corresponds to the number of requests in the system at the end of interval $kh$. The difference equation for the queue is given by

$$x(kh+h) = f(x(kh) + \bar{u}(kh) - \sigma_{max}(kh))$$

where the limit function, $f(w)$, equals zero if $w < 0$ and $w$ otherwise. The limit function assures that $x(kh+h) \ge 0$. When the limit function is disregarded then the queue is a discrete-time integrator.

The *monitor* must estimate the server utilization since this is not directly measurable in the model. The server utilization during interval $kh$, $\rho(kh)$, is estimated as

$$\rho(kh) = min(\frac{\bar{u}(kh) + x(kh)}{\sigma_{max}(kh)}, 1)$$

The objective of the *controller* is to minimize the difference between the server utilization during interval $kh$, $\rho(kh)$, and the reference value, $\rho_{ref}$. The control law is given by the transfer function, $G_c(z)$.

## 3.5 Controller Design

The PI-controller is commonly used in automatic control. The controller uses two actions: one proportional, and one integrating by using the following discrete-time control law (see Åström1 for more details):

$$u(kh) = Ke(kh) + \sum_{i=0}^{k-1} \frac{K}{T_i} e(ih)$$

where $e(kh) = \rho_{ref} - \rho(kh)$. The gain, $K$, and the integral time, $T_i$, are the controller parameters that are set so that the controlled system behaves as desired. Since the controller is discrete, the controller parameter for the integration action, $T_i$, is given by $T_i = T_{si}/h$ where $Tsi$ would be the integral time in continuous-time. Note that the control signal, $u(kh)$, is allowed to become negative. A negative control signal will be treated as a zero signal in the gate.

**Design and analysis.** The control law for the PI-controller expressed in z-transform is given by

$$G_c(z) = K(1 + \frac{1}{T_i} \cdot \frac{h}{z-1})$$

In this paper we use a linear design method, which means that we during the design analysis consider a deterministic system with no active saturations. However, it is important to note that we can take into account the saturations in the system during the design, since the underlying model is non-linear. This is performed by choosing the controller parameters carefully, since some controller parameters may cause oscillations in a system with saturations. This is the main difference between our work and the previous published work about control theoretic analysis of queueing systems. In those papers, the underlying system models are linear and deterministic, which means that the non-linearitities and stochastic processes in the real systems are ignored.

The linear transfer function from the desired utilization, $\rho_{ref}$, to the (delayed) output, $\rho(kh)$, will be

$$G_{cl} = \frac{G_c(1+G_q)G_m}{1+G_c(1+G_q)G_m} = \frac{(z+1)K(T_iZ - T_i + h)}{z^3\sigma T_i + (KT_i - \sigma T_i z^2 + Kzh + (-KT_i + Kh))}$$

where

$$\begin{cases} G_c = K(1 + \frac{1}{T_i}\frac{h}{z-1}) \\ G_q = \frac{1}{z-1} \\ G_m = \frac{1}{\sigma}\frac{1}{z} \end{cases}$$

are the transfer functions for the controller, for the queue, and for the monitor, respectively. $\sigma$ is the average value of $\sigma_{max}$. The characteristic polynomial for the linear closed loop system will be

$$z \cdot (z^2 + \frac{K - 2\sigma}{\sigma} + \frac{-KT_i + Kh + \sigma T_i}{\sigma T_i}) \tag{3.1}$$

where the pole at $z$=0 is cancelled in the transfer function from the input (the load reference) to the desired output (the load). Assume that the desired characteristic equation is

$$z(z^2 + a_1 z + a_2) = 0$$

The values of the controller parameters that gives this are

$$K = 2\sigma + a_1\sigma T_i = h \cdot \frac{2 + a_1}{1 + a_1 + a_2}$$

The controller parameters $K$ and $T_i$ influence the closed loop response for the system and need to be determined with respect to stability and robustness.

## 3.6 Experiments

The admission control mechanism was implemented on a real web server. We tested the system by running tests on it and collecting performance metrics such as the server utilization distribution and step responses. The admission control mechanism was written in Java and tested on the Windows platform. We also compared the measurements with simulations. The queueing model was represented by a discrete-event simulation program implemented in C, and the control theoretic models were implemented with the Matlab Simulink package. The traffic generators in the discrete-time model were built as Matlab programs. They generate arrivals and departures according to the given statistical distributions.

### *3.6.1 Setup*

Our measurements used one server computer and one computer representing the clients connected through a 100 Mbits/ s Ethernet switch. The server was a PC Pentium III 1700 MHz with 512 MB RAM running Windows 2000 as operating system. The computer representing the clients was a PC Pentium II 400 MHz with 256 MB RAM running RedHat Linux 7.3. Apache 2.0.45 was installed in the server. We used the default configuration of Apache. The client computer was installed with an HTTP load generator, which was a modified version of S-Client [16]. The S-Client is able to generate high request rates even with few client computers by aborting TCP connection attempts that take too long time. The original version of S-Client uses deterministic waiting times between requests. We modified the code to use Poissonian arrivals instead. The client program was programmed to request dynamically generated HTML files from the server. The CGI script was written in Perl. It generates a random number of random numbers, adds them together and returns the summation. The average request rate was set to 100 requests per second in all experiments except for the measurements in Figure 3.5. Apache was installed on the server and set to listen to port 8080. The Gate thread listened to port 80, the normal web server port, and then copied the admitted requests to port 8080. The admission control mechanism and the web server ran on the same computer. In all experiments, the control interval was set to one second.

### *3.6.2 Validation of the Model*

We have validated that the open system, that is without control feedback, is accurate in terms of average server utilization. The average server utilisation for varying arrival rates are shown in Figure 3.5. For a single-server queue, the server utilization is proportional to the arrival rate, and the slope of the server utilization curve is given by the average service time. The measurements in Figure 3.5 gives an estimation of the average service time in the web server, $1/\mu$=0.0255.

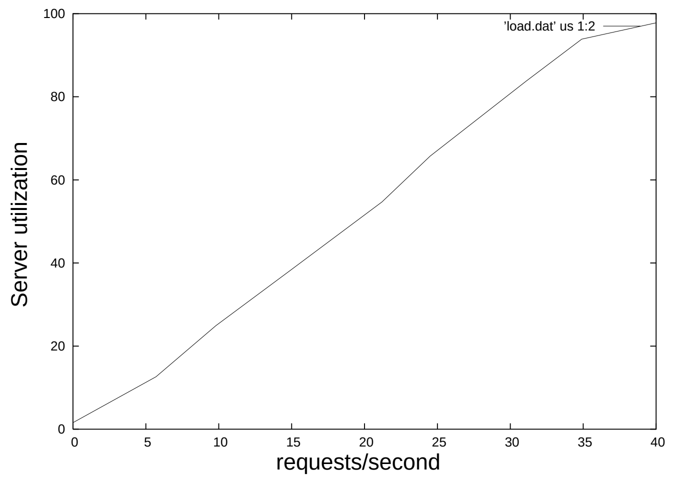

*Fig. 3.5:* Average server utilization for the open system.

### *3.6.3 Controller parameters*

Root locus arguments show that reasonable parameters give a stable closed loop system. Figure 3.6 shows the root locus diagram when $T_i$ =2.8 and the gain $K$ varies between [0,40].

**"Good" controller parameters.** By choosing $\{K, T_i\}$=$\{20, 2.8\}$ the roots of the characteristic polynomial in Equation 3.1 will be rather well damped and the transients from the pole on the real axis will decay fast. This controller design can, therefore, be seen as a good design.

**"Bad" controller parameters.** If the controller parameters are chosen badly, the system will not behave well. One example is $\{K, Ti\}$=$\{20, 0.1\}$, which for the linear systems gives unstable poles placed outside the unit circle.

### *3.6.4 Performance metrics*

An admission control mechanism have two control objectives. First, it should keep the control variable at a reference value, i.e. the error, $e = y_{ref} - y$, should be as small as possible. Second, it should react rapidly to changes in the system, i.e. the so-called settling time should be short. Therefore, we test the mechanism in two ways. First, we show the steady-state distribution of the control variable, by plottting the estimated distribution function. The distribution function is estimated from measurements during 1000 seconds with the specific parameter setting. The distribution function shows how well the control mechanism meets the first control objective. Second, we plot the step response during 60 seconds when starting with an empty system. The step response shows the settling time for the control mechanism.

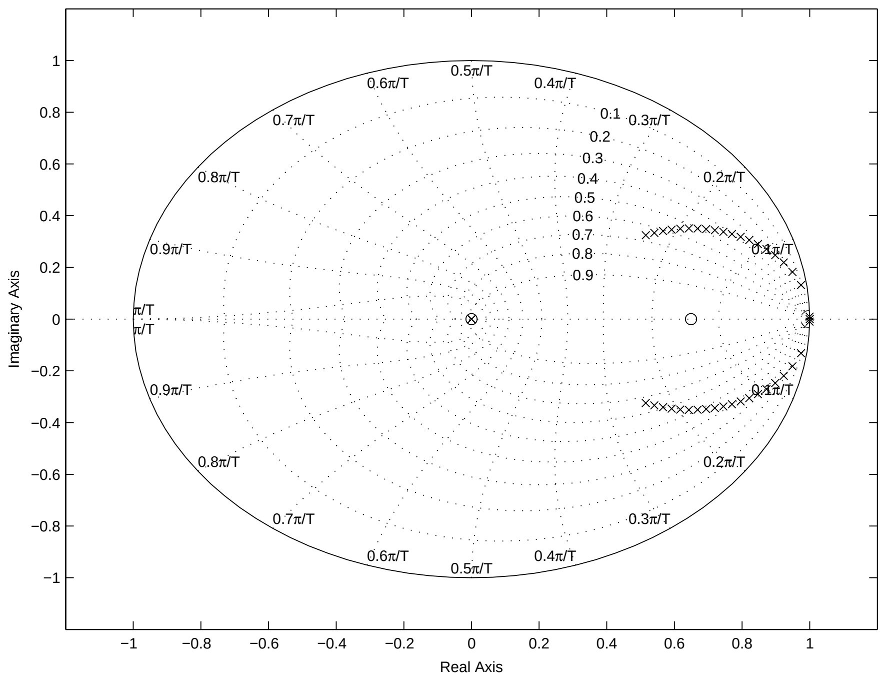

*Fig. 3.6:* Root locus diagram for Ti=2.8. The gain K varies between [0,40].

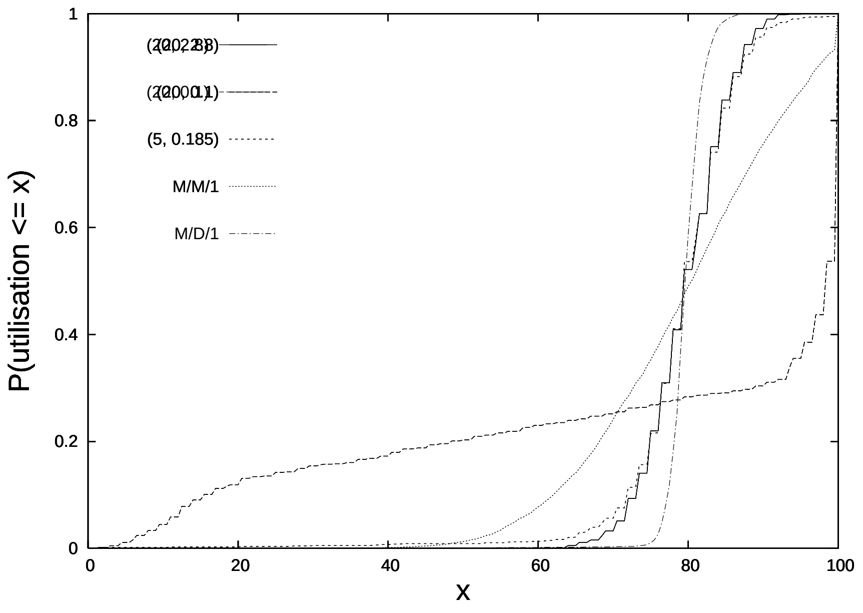

*Fig. 3.7:* Server utilization distribution of measurements from the real system together with simulations of the M/M/1 and the M/D/1 system.

### *3.6.5 Distribution function*

Figure 3.7 shows the estimated distribution function for the PI-controller. Both good and bad parameter settings were used. An ideal admission control mechanism would show a distribution function that is zero until the wanted load, and is one thereafter. In this case, the load was kept at 0.8, and the parameter setting, {K, Ti}={20, 2.8}, results in a controller that behaves very well in this sense. The parameter setting, $\{K, T_i\}$={20, 0.1}, as can be seen, perform worse. Also, as comparison, results from simulations of the M/D/1-system and the M/M/1-system are given in Figure 3.7, when using $\{K, Ti\}$={20, 2.8}. They show that the system behaves as expected.

### *3.6.6 Step response*

Figure 3.8 shows the behaviour of the web server during the transient period. The measurements were made on an empty system that was exposed to 100 requests per second. The good parameter setting, $\{K, Ti\}$={20, 2.8}, exhibits a short settling time with a relatively steady server utilization. The bad parameter setting, $\{K, Ti\}$={20, 0.1} has its poles outside the unit circle and behaves badly, the load oscillates and is never stable. Comparisons to M/D/1 and M/M/1 simulations, also in Figure 3.8, show that the model is accurate.

### 3.6.7 Limitations with linear design

All papers published earlier about control theoretic analysis of queueing systems have only used linear deterministic models that can be analyzed with linear design methods. However, it is important to know that a linear model of a non-linear system is not accurate. The strength with our work is that we can try the controllers in the non-linear model, and thereby see how the real system behaves. For example, the controller parameters $\{5, 0.185\}$ gives unstable poles outside the unit circle. Therefore, they should not be used. However, the non-linear system behaves very well, as shown in Figure 3.7. We have found that this is due to the non-linearity in the queue, since the queue length can never be negative.

## 3.7 Conclusions

Admission control mechanisms have since long been developed for various server systems. Traditionally, queueing theory has been used when investigating server systems, since they usually can be modelled as queueing systems. However, there are no mathematical tools in queueing theory that can be used when designing admission control mechanisms. Therefore, these mechanisms have mostly been developed with empirical methods. Control theory contains many mathematical tools that can be used when designing admission control mechanisms. The main problem here is that queueing systems are non-linear and stochastic, which means that they are difficult to model and analyse with control theoretic methods. The main objective with our work has been to find models and methods from control theory that can be used when designing admission control mechanisms for server systems. In this paper, we have designed admission control mechanisms for this system with control theoretic methods. We have shown that the model is accurate enough for this purpose, and have also shown an example of how to perform the controller design. We have used the PI-controller, commonly used in automatic control. The admission control mechanism has been implemented on a real web server, running the well-known Apache web server software. Measurements were made to examine the performance of the system. The experiments show that the control mechanism behaves as expected. The main conclusion of this paper is that it is possible to use control theoretic methods when designing admission control mechanisms for server systems. However, linear deterministic models, as have been used in previous papers, are not enough for this purpose. A server system is both non-linear and stochastic, and if this fact is ignored during modeling and analysis, the behaviour of the real system may not be as expected. It is obvious that more research is needed in this field. We will, therefore, continue investigating non-linear, stochastic control theoretic models of queueing systems.

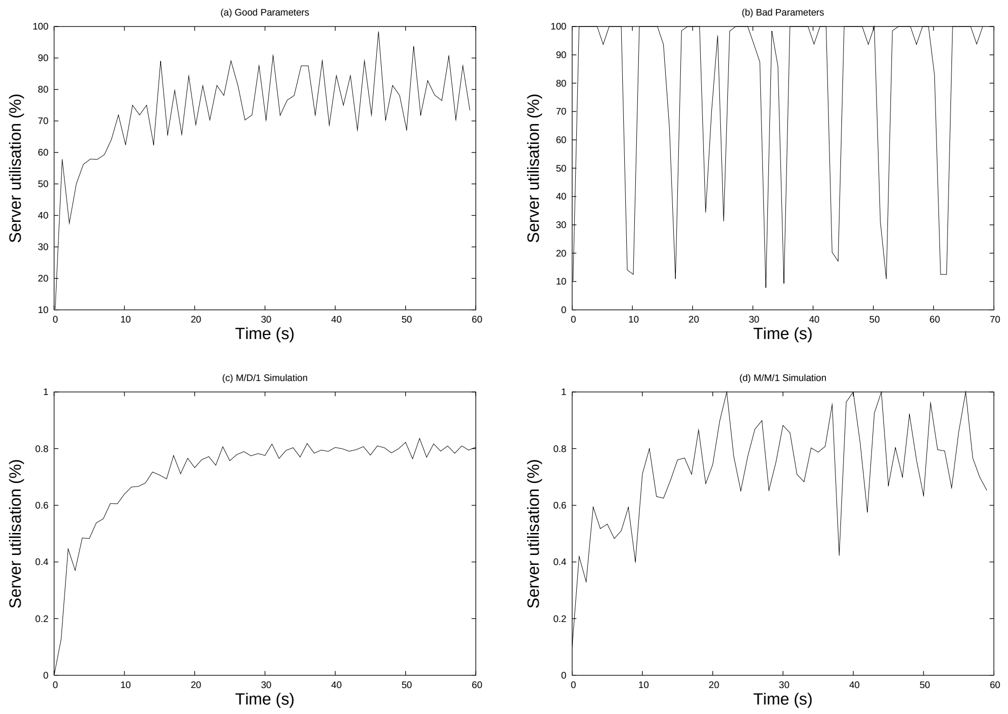

*Fig. 3.8:* (a) Example of a realisation with good parameters. (b) Example of a realisation with bad parameters. (c) Simulation of M/D/1-system with good parameters. (d) Simulation of M/M/1-system with good parameters.

## 3.8 Acknowledgments

This work has partially been supported by the Swedish Research Council through the Multi Project Grant 621-2001-3020 and contract 621-2001-3053.

# BIBLIOGRAPHY

[1] K. Åström and B. Wittenmark, *Computer-controlled systems, theory and design.* Prentice Hall Internation Editions, 1997, third Edition.

[2] T. Abdelzaher and C. Lu, "Modeling and performance control of internet servers," in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000, pp. 2234–2239.

[3] K. S. T.F. Abdelzaher and N. Bhatti, "Performance guarantees for web server end-systems: a control theoretic approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, January 2002.

[4] J. S. C. Lu, T.F. Abdelzaher and S. So, "A feedback control approach for guaranteeing relative delays in web servers," in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001, pp. 51–62.

[5] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, pp. 64–71, Sept/Oct 1999.

[6] R. R. J. Carlström, "Application-aware admission control and scheduling in web servers," in *Proc. Infocom*, 2002.

[7] P. P. L. Cherkasova, "Predictive admission control strategy for overloaded commerical web servers," in *Proc. 8th International IEEE Symposium on modeling, analysis and simulation of computer and telecommunication systems*, 2000, pp. 500–507.

[8] P. G. T. Voigt, "Adaptive resource-based web server admission control," in *Proc. 7th International Symposium on Computers and Communications*, 2002.

[9] S. R. P. Bhoj and S. Singhal, "Web2k: Bringing qos to web servers," *HP Labs Technical report, HPL-2000-61*, 2000.

[10] S. S. Jr., "Optimal control of admission to a queueing system," *IEEE Transactions on Automatic Control*, vol. 30, no. 8, pp. 705–713, August 1985.

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

[12] M. Kihl, "Overload control strategies for distributed communication networks," Department of Communication Systems, Lund Institute of Technology, Tech. Rep. 131, 2002, ph.D. Thesis.

[13] W. Stallings, *Data & Computer Communications*. Prentice Hall, 2000, sixth Edition.

[14] T. Voigt, "Overload behaviour and protection of event-driven web servers," in *In proceedings of the International Workshop on Web Engineering*, May 2002, pisa, Italy.

[15] B. W. M. Kihl, A. Robertsson, "Performance modelling and control of server systems using non-linear control theory," in *Proc. 18th International Teletraffic Congress*, 2003.

[16] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *USENIX Symposium on Internet Technologies and Systems*, December 1997, pp. 61–71.

# 4. PAPER III

## *Admission Control of the Apache Web Server*

Mikael Andersson, Maria Kihl, Anders Robertsson, Björn Wittenmark
Department of Communication Systems, Lund Institute of Technology
Email: {mike, maria}@telecom.lth.se
Department of Automatic Control, Lund Institute of Technology
Email: {andersro,bjorn}@control.lth.se
Box 118, SE-221 00 Lund, Sweden

**Abstract**

Web sites are exposed to high rates of incoming requests. The servers may become overloaded during temporary traffic peaks when more requests arrive than the server is designed for. An admission control mechanism rejects some requests whenever the arriving traffic is too high and thereby maintains an acceptable load in the system. This paper presents how admission control mechanisms can be designed with a combination of queueing theory and control theory. By analyzing queueing systems with control theoretic methods, good admission control mechanisms can be designed for web server systems. In this paper we model an Apache web server as a G/G/1-system. Then we design a PI-controller, commonly used in automatic control, for the server. We describe how it can be implemented in a real system. The controller has been implemented as a module inside the Apache source code. Measurements from the laboratory setup show how robust the implemented controller is, and how it correspond to the results from the theoretical analysis.

## 4.1 Introduction

Web sites on the Internet can be seen as server systems with one or more web servers processing incoming requests at a certain rate. The web servers have a queue where requests wait for service. Therefore, a web server can be modeled as a queueing system including a server with finite or infinite queues. One problem with web servers is that they are sensitive to overload. The servers may become overloaded during temporary traffic peaks when more requests arrive than the server is designed for. Because overload usually occurs rather seldom, it is not economical to overprovision the servers for these traffic peaks, instead admission control mechanisms can be implemented in the servers. The admission control mechanism rejects some requests whenever the arriving traffic is too high and thereby maintains an acceptable load in the system. Traditionally, server utilization or queue lengths have been the variables mostly used in admission control schemes. For web servers, the main objective of the control scheme is to protect it from overload. As long as the average server utilization or queue length is below a certain level, the response times are low.

One well-known controller in automatic control is the PID-controller, which enables a stable control for many types of systems (see, for example Åström, [1]). The PID-controller uses three actions: one proportional, one integrating, and one derivative. In order to get the system to behave well it is necessary to decide proper control parameters. Therefore, before designing the PID-controller, the system must be analyzed so that its dynamics during overload are known. This means that the system must be described with a control theoretic method. If the model is linear, it is easily analyzed with linear control theoretic methods. However, a queueing system is both nonlinear and stochastic. The main problem is that nonlinear models are much harder to analyze with control theoretic methods. Very few papers have investigated admission control mechanisms for server systems with control theoretic methods. Abdelzaher ( [2, 3]) modeled the web server as a static gain to find optimal controller parameters for a PI-controller. A scheduling algorithm for an Apache [4] web server was designed using system identification methods and linear control theory by Lu et al [5]. Bhatti [6] developed a queue length control with priorities. By optimizing a reward function, a static control was found by Carlström [7]. An on-off load control mechanism regulating the admittance of client sessions was developed by Cherkasova [8]. Voigt [9] proposed a control mechanism that combines a load control for the CPU with a queue length control for the network interface. Bhoj [10] used a PI-controller in an admission control mechanism for a web server. However, no analysis is presented on how to design the controller parameters. Papers analyzing queueing systems with control theoretic methods usually describe the system with linear deterministic models. Stidham Jr [11] argues that deterministic models cannot be used when analyzing queueing systems. Un-

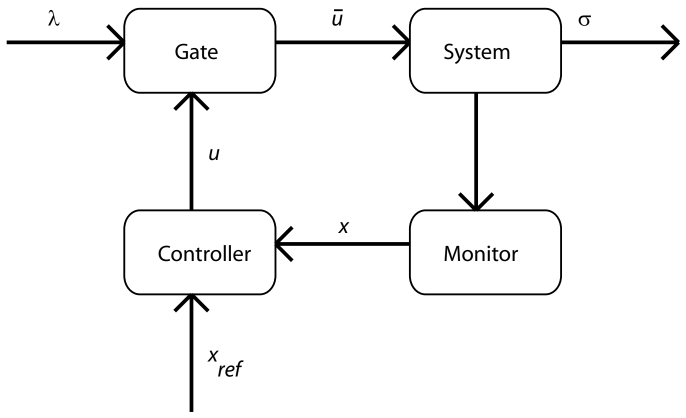

*Fig. 4.1:* An admission control mechanism

til now, no papers have designed admission control mechanisms for server systems using nonlinear control theory. In this paper we implement an admission control mechanism for the Apache web server. Measurements in the laboratory setup show how robust the implemented controller is, and that it corresponds to the results from the theoretical analysis. Section 4.2 describes a general admission control mechanism. Section 4.3 shows how this can be applied on a web server. In section 4.4, we describe a nonlinear control theoretic model of an admission control mechanism for a web server. We give an analysis of the closed loop system in section 4.5. The control theoretic model is used to design and implement an admission control mechanism for the Apache web server. The measurements are shown in section 4.6, section 4.7 discusses the results and section 4.8 concludes the work.

## 4.2 Admission Control Mechanism

Figure 4.1 shows a general admission control mechanism that consists of three parts: a *gate*, a *controller*, and a *monitor*. The monitor measures a so called control variable, $x$. Using the control variable, the controller decides the rate, $u$, at which requests can be admitted into the system. The objective is to keep the value of the control variable as close as possible to a reference value, $x_{ref}$. The gate rejects those requests that cannot be admitted. The requests that are admitted proceed to the rest of the system. Since the admittance rate may never be larger than the arrival rate, $\lambda$, the actual admittance rate is $\bar{u}$=min[$u$, $\lambda$] requests per second. A survey of different admission control mechanisms for communication systems is given by Kihl [12].

### *4.2.1 Gate*

Several gates have been proposed in the literature, for example *Percent blocking*, *Token bucket* and the *Dynamic window* mechanism. In this work, we use the Token bucket, where tokens are generated at a certain rate. An arriving request is admitted if there is a token available.

### *4.2.2 Controllers*

There are a variety of controllers to choose from when designing an admission control mechanism. Some of the most common controllers are the *Static controller*, the *Step controller*, and the *PID-controller*. The PID-controller uses three actions: one proportional, one integrating, and one derivative. The control law in continuous time is as follows:

$$u(t) = K \cdot e(t) + \frac{K}{T_i} \cdot \int_0^t e(v)dv + K \cdot T_d \cdot \frac{d}{dt} \cdot e(t)$$

where e(t) is the error between the control variable and the reference value, that is $e(t) = y_{ref} - y(t)$. The gain $K$, the integral time $T_i$, and the derivative time $T_d$ are the controller parameters that are set so that the controlled system behaves as desired. A large value of $K$ makes the controller faster, but weakens the stability. The integrating action eliminates stationary errors, but may also make the system less stable. The derivative action improves the stability, however, in a system with a bursty arrival process the derivative action may cause problems. Therefore, the derivative action is usually either deleted (i.e. $T_d = 0$) or low pass filtered to remove the high frequencies.

## *4.3 Investigated System*

The system we have investigated in this work, is a web server with an admission control mechanism. The web server is Apache, described below.

### *4.3.1 Web servers*

A web server like Apache, contains software that offers access to documents stored on the server. Clients can browse the documents in a web browser. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Java scripts or Perl files. The communication between clients and server is based on HTTP [13]. An HTTP transaction consists of three steps: TCP connection setup, HTTP layer processing and network processing. The TCP connection setup is performed through a threeway handshake, where the client and the server exchange TCP SYN, TCP SYN/ACK and TCP ACK messages. Once the connection has been established, a document request can be issued with an HTTP GET message to the server. The server then replies with an HTTP GET REPLY message. Finally, the TCP connection is closed by TCP FIN and TCP ACK messages in both directions. Apache, which is a well-known web server and widely used, is multi-threaded. This means that a request is handled by its own thread or process throughout the life cycle of the request. Other types of web servers e.g. event-driven ones and admission control for such also exist (Voigt [14]).

### *4.3.2 Admission control*

Since continuous control is not possible in computer systems, time is divided into control intervals of length $h$ seconds. At the end of interval $k$, that is when the time is $kh$, the controller calculates the desired admittance rate for interval $[kh, kh+h]$, denoted $u(kh)$, from the measured average server utilization during the interval, $\rho(kh)$, and the reference value $\rho_{ref}$. There are three main parts in our admission control architecture:

**Gate.** The Gate module gets notified whenever the web server gets an incoming request. It takes a decision whether to admit the request and then notifies the web server of the result. In this paper we use the token bucket algorithm to reject those requests that cannot be admitted. New tokens are generated at a rate of $u(kh)$ tokens per second during time interval $[kh, kh+h]$. If there is an available token upon the arrival of a request, the request consumes the token and enters the web server. If there are no available tokens, the request is rejected. When a request is rejected, the TCP socket to the client is closed. Rejected requests are assumed to leave the system without retrials.

**Monitor**. The Monitor thread constantly samples the server utilization every control interval. The server utilization is calculated as one minus the fraction of time an idle process has been able to run during the last control interval. The idle process' priority level is set to the lowest possible, which means that it only runs whenever there is no request requiring CPU work. This way of measuring the load on the CPU results in a quantization effect in server utilization. The reason to this is that the operating system where the admission control mechanism runs has a certain time resolution in function calls regarding process uptimes. This means that the control interval cannot be chosen arbitrary. It has to be long enough not to be affected by the time resolution effects, and short enough so that the controller responds quickly.

**Controller**. The Controller is a PI-controller. The Controller's output is forwarded to the Gate module. The Controller design is discussed more extensively in section 4.4.

## 4.4 Control Theoretic Model

Control theory is a powerful tool for performance analysis of computer controlled systems. The system must be described in terms of transfer functions or differential (or difference) equations. A transfer function describes the relationship between the z-transforms (or Laplace transforms) of the input and the output of a system. In this case, the input to the system is the actual admittance rate, $\bar{u}$, whereas the output is either the server utilization, denoted $\rho$, or the number of jobs in the server, here denoted $x$.

We use the discrete-time control theoretic model of web server developed by Kihl et al. [15]. We assume that the system can be modeled as a GI/G/1-system with an admission control mechanism. Kihl et al. showed that the queueing model is a good model for admission control purposes. The input to the system is the actual admittance rate, $\bar{u}$, whereas the output is the server utilization, $\rho$. The model is a flow or liquid model in discrete-time. The model is an averaging model in the sense that we are not considering the specific timing of different events, arrivals, or departures from the queue. We assume that the sampling period, $h$, is sufficiently long to guarantee that the quantization effects around the sampling times are small. The model is shown in Figure 4.2. The system consists of an arrival generator, a departure generator, a controller, a queue and a monitor.

There are two stochastic traffic generators in the model. The *arrival generator* feeds the system with new requests. The number of new requests during interval $kh$ is denoted $\alpha(kh)$. $\alpha(kh)$ is an integrated stochastic process over one sampling period with a distribution obtained from the underlying interarrival time distribution. If, for example, the arrival process is Poisson with mean $\lambda$, then $\alpha(kh)$ is Poisson distributed with mean $\lambda h$. The *departure generator* decides the maximum number of departures during interval $kh$, denoted $\sigma_{max}(kh)$. $\sigma_{max}(kh)$ is also a stochastic process with a distribution given by the underlying service time distribution. If, for example, the service times are exponentially distributed with mean $1/\mu$, then $\sigma_{max}(kh)$ is Poisson distributed with mean $\mu h$. It is assumed that $\alpha(kh)$ and $\sigma_{max}(kh)$ are independent from between sampling instants and uncorrelated to each other. The *gate* is constructed as a saturation block that limits $u(kh)$ to be

$$\bar{u}(kh) = \begin{cases} 0 & u(kh) < 0 \\ u(kh) & 0 \leq u(kh) \leq \alpha(kh) \\ \alpha(kh) & u(kh) > \alpha(kh) \end{cases}$$

The *queue* is represented by its state $x(kh)$, which corresponds to the number of requests in the system at the end of interval $kh$. The difference equation for the queue is given by

$$x(kh + h) = f(x(kh) + \bar{u}(kh) - \sigma_{max}(kh))$$

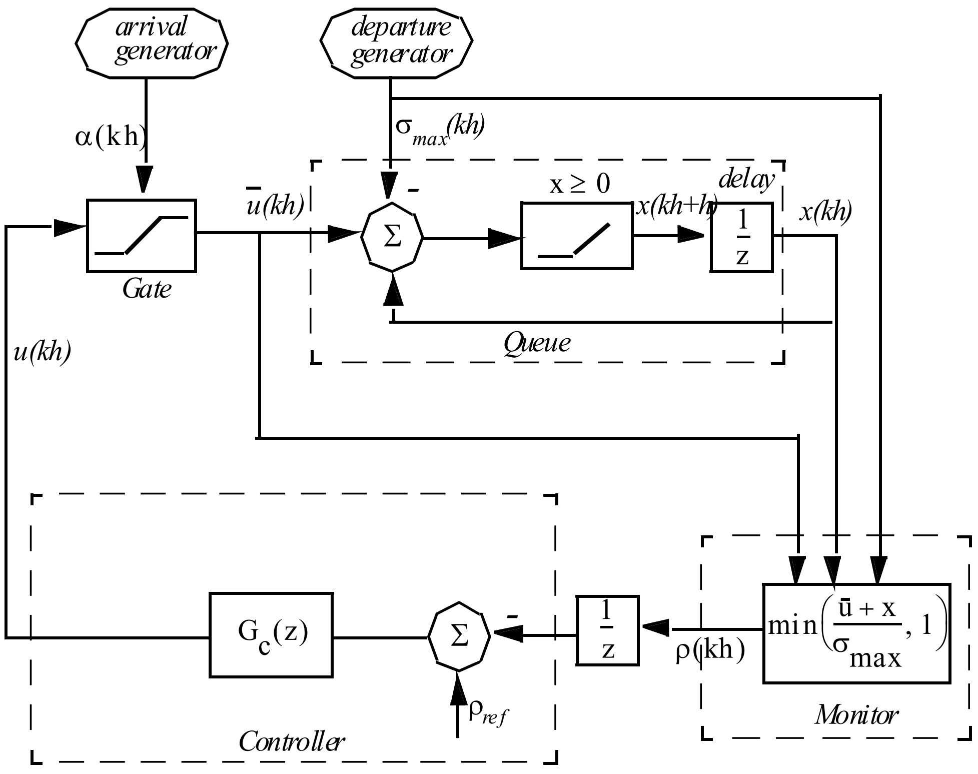

*Fig. 4.2:* Discrete-time model with controller saturation and saturation $\varphi$ for positive queue lengths.

where the limit function, $f(w)$, equals zero if $w < 0$ and $w$ otherwise. The limit function assures that $x(kh+h) \geq 0$. When the limit function is disregarded then the queue is a discrete-time integrator.

The *monitor* must estimate the server utilization since this is not directly measurable in the model. The server utilization during interval $kh$, $\rho(kh)$, is estimated as

$$\rho(kh) = min(\frac{\bar{u}(kh) + x(kh)}{\sigma_{max}(kh)}, 1)$$

The objective of the *controller* is to minimize the difference between the server utilization during interval $kh$, $\rho(kh)$, and the reference value, $\rho_{ref}$. The control law is given by the transfer function, $G_c(z)$.

## 4.5 Stability analysis of closed loop system

In this section we will consider the stability properties of the controlled server node, when using a PI-controller for admission control. First we will consider an approach based on a linear queue model and compare with the admission control parameters derived from nonlinear analysis. The analysis is based on the Tsypkin/Jury-Lee stability criterion (discrete-time versions of the Popov criterion) [16]. In the analysis only the dominating 'queue-limitation' $\varphi$ will be considered. See Section 4.7 for comments on the saturation.

### 4.5.1 Linear design (neglecting saturations)

Neglecting the nonlinearities in Figure 4.2 (assuming $\varphi(z) = z$, i.e., linear and no saturation) and using a standard PI-controller $G_c(z) = K(1+\frac{1}{T_i}\cdot\frac{h}{z-1})$ will result in the closed loop dynamics

$$\begin{aligned} G_c &= \frac{G_c(1+G_q)G_m}{1+G_c(1+G_q)G_m} \\ &= \frac{z \cdot K/\sigma\,(z-1+h/T_i)}{z \cdot (z^2 + (K/\sigma - 2)z + (1 - K/\sigma + Kh/(\sigma T_i))} \end{aligned} \tag{4.1}$$

where $G_q$ and $G_m$ represent the queue and monitor dynamics, respectively. To match the characteristic polynomial

$$z \cdot (z^2 + (K/\sigma - 2)z + (1 - K/\sigma + Kh/(\sigma T_i)) \tag{4.2}$$

with a desired characteristic polynomial

$$z \cdot (z^2 + a_1 z + a_2) \tag{4.3}$$

we get the control parameters

$$K = (2 + a_1)\,\sigma, \quad T_i = h\,(2 + a_1)/(1 + a_1 + a_2)$$

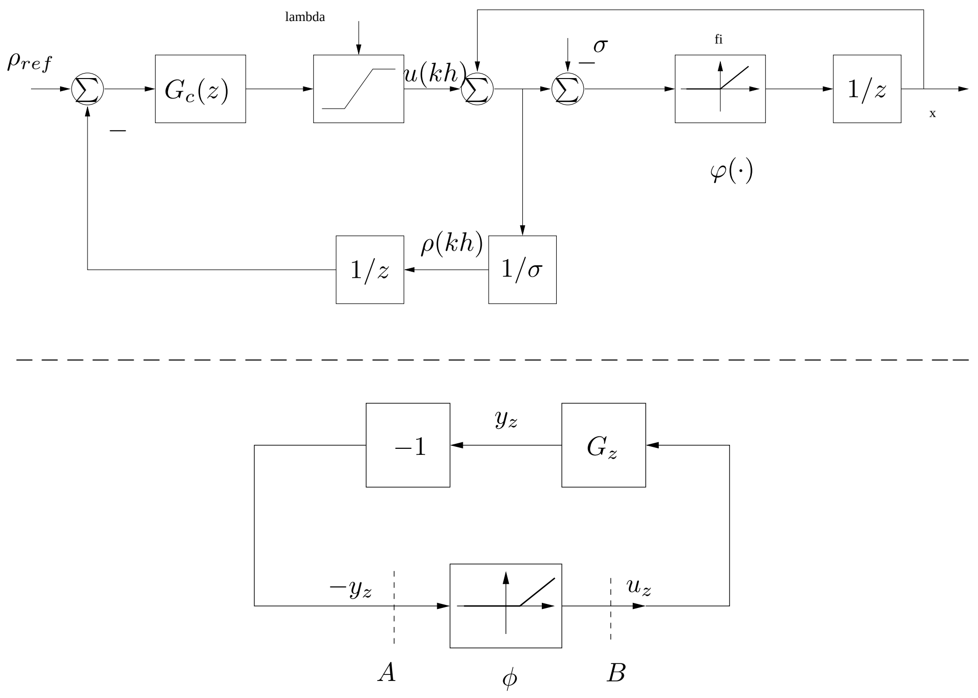

*Fig. 4.3:* Decomposition into a linear block ($G_z$) and a nonlinear block ($\phi$) under negative feedback.

Using the parameters of the PI-controller it is thus possible to make an arbitrary pole-placement, except for the pole $z = 0$, which corresponds to a time delay. A simplified linear analysis will thus predict stability for the closed loop for all coefficients $\{a_1, a_2\}$ belonging to the stability triangle

$$\{\ a_2 < 1, \quad a_2 > -1 + a_1, \quad a_2 > -1 - a_1\ \}, \tag{4.4}$$

see [1].

### 4.5.2 Model with queue limitation

Consider the admission control scheme in Figure 4.3 where we have introduced the states $\{x_1, x_2, x_3\}$ corresponding to the queue length, the (delayed) utilization $\rho$ and the integrator state in the PI-controller, respectively.

The state space model will be

$$\begin{aligned} x_1(kh+h) &= \varphi\left(u + x_1(kh) - \sigma\right) \\ x_2(kh+h) &= \frac{1}{\sigma}(u + x_1(kh)) \\ x_3(kh+h) &= Kh/T_i(\rho_{ref} - x_2(kh)) + x_3(kh) \end{aligned} \tag{4.5}$$

where $u = K(\rho_{ref} - x_2) + x_3$ and $\varphi(\cdot)$ is the saturation function in Figure 4.3. By introducing the *forward shift operator* and leaving out the time arguments, we get

$$\begin{aligned} q\,x_1 &= \varphi\left(K(\rho_{ref} - x_2) + x_3 + x_1 - \sigma\right) && (4.6)\\ q\,x_2 &= \frac{1}{\sigma}\left(K(\rho_{ref} - x_2) + x_3 + x_1\right) && (4.7)\\ q\,x_3 &= Kh/T_i\,(\rho_{ref} - x_2) + x_3 && (4.8) \end{aligned}$$

The equilibrium for the system (4.6–4.8) satisfies $qx = x$. From (4.8) we get

$$x_3 = Kh/T_i\,(\rho_{ref} - x_2) + x_3 \quad \Rightarrow \quad x_2^o = \rho_{ref}$$

Inserting this in (4.6) and (4.7) we get

$$\begin{aligned} &x_1^o = \varphi(x_3^o + x_1^o - \sigma)\\ &x_2^o = \rho_{ref} = \frac{1}{\sigma}\,(x_3^o + x_1^o)\\ &\Rightarrow\\ &x_1^o = \varphi\left(\sigma(\rho_{ref} - 1)\right) \end{aligned} \qquad (4.9)$$

As $\rho_{ref} \in [0,1]$ and using the fact that $\varphi(z) = 0$, $\forall z \leq 0$ we get

$$\begin{cases} x_1^o = 0\\ x_2^o = \rho_{ref}\\ x_3^o = \sigma x_2^o = \sigma\rho_{ref} \end{cases} \qquad (4.10)$$

By introducing the change of variables

$$\begin{cases} z_1 = x_1 - 0\\ z_2 = x_2 - \rho_{ref}\\ z_3 = x_3 - \sigma\rho_{ref} \end{cases} \qquad \text{or} \qquad \begin{aligned} x_1 &= z_1\\ x_2 &= z_2 + \rho_{ref}\\ x_3 &= z_3 + \sigma\rho_{ref} \end{aligned}$$

we get

$$\begin{aligned} q\,z_1 = q\,x_1 - 0 \quad &= \varphi\left(-Kz_2 + z_3 + \sigma\rho_{ref} + z_1 - \sigma\right)\\ q\,z_2 = q\,x_2 - \rho_{ref} \quad &= \frac{1}{\sigma}\left(-Kz_2 + z_3 + \sigma\rho_{ref} + z_1\right) - \rho_{ref}\\ q\,z_3 = q\,x_3 - \sigma\rho_{ref} \quad &= -\,Kh/T_i\,z_2 + z_3 + \sigma\rho_{ref} - \sigma\rho_{ref} \end{aligned}$$

Rewriting this as a linear system in negative feedback with the nonlinear

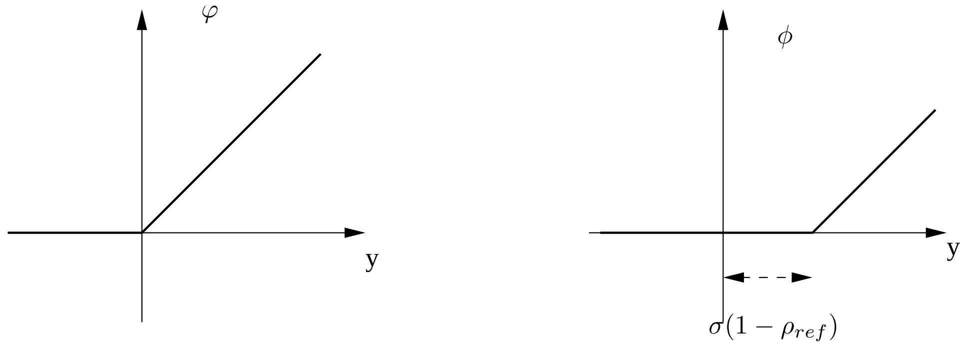

*Fig. 4.4:* $\phi(y) = \varphi(y - \sigma(1-\rho_{ref}))$ where $\sigma > 0$ and $\rho_{ref} \in [0,1]$.

function $\phi : y \to \varphi(y - \sigma(1-\rho_{ref}))$, we get

$$\begin{aligned} qz &= A_z z + B_z u_z = A_z z + B_z \phi(-y) \\ y &= C_z z \end{aligned}$$

$$q\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1/\sigma & -K/\sigma & 1/\sigma \\ 0 & -Kh/T_i & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \phi(-y)$$

$$y = \begin{bmatrix} -1 & K & -1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Note that for $\rho_{ref} \in [0,1]$ the function $\phi(\cdot)$ will belong to the same cone as $\varphi(\cdot)$, namely $[\alpha, \beta] = [0,1]$, see Figure 4.4. The incremental variation will also have the same maximal value (=1).

The transfer function $G_z = G_{u_z \to y_z}(z)$ from cut $B$ to cut $A$ in Figure 4.3 will be

$$\begin{aligned} G_z &= C_z(zI - A_z)^{-1} B_z \\ &= \frac{-z \cdot (z-1)}{z \cdot \left(z^2 + (-1 + K/\sigma)\, z + K\,(h - T_i)\,/(\sigma T_i)\right)} \end{aligned} \tag{4.11}$$

For the forthcoming stability analysis we determine for which control parameters the linear subsystem $G_z$ is stable.

The poles of (4.11) are stable for the area depicted in Figure 4.5 for the normalized parameters $K/\sigma$ and $h/T_i$.

### 4.5.3 Stability analysis for discrete-time nonlinear system

To determine the stability for the nonlinear system in (4.11) we can use the Tsypkin criterion or the Jury-Lee criterion which are the discrete-time counterparts of the Popov criterion for continuous time systems [17].

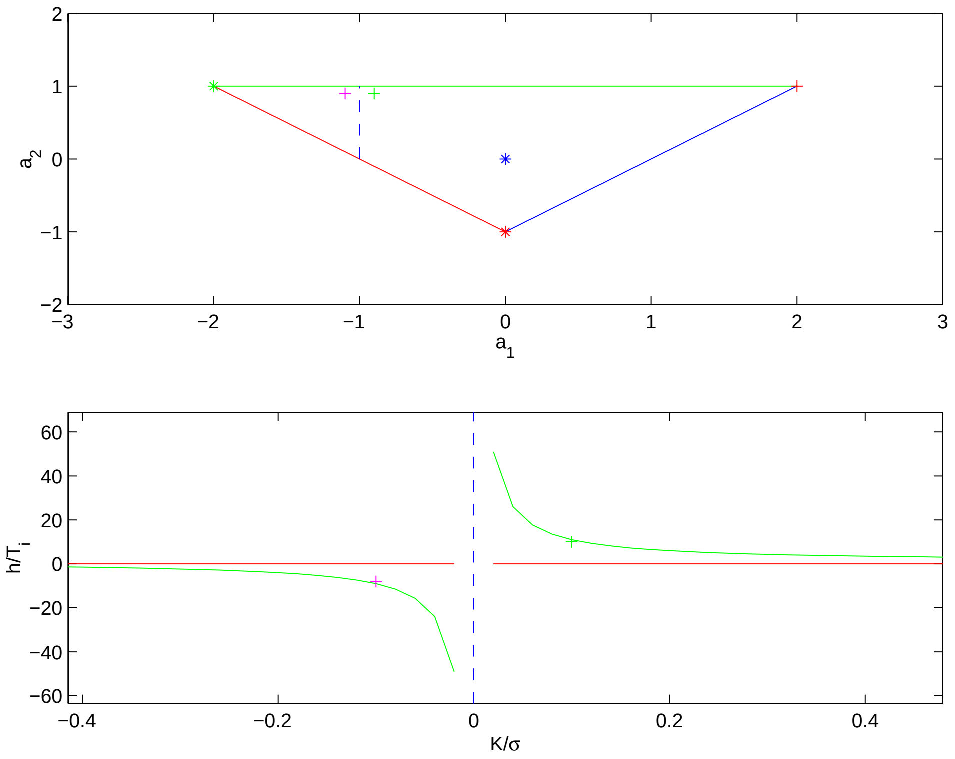

*Fig. 4.5:* *(Upper:)* The internal of the triangle corresponds to the stability area for the characteristic polynomial $z \cdot (z^2 + a_1 z + a_2)$ of $G_z$. *(Lower:)* Corresponding stabilizing control parameters $\{K/\sigma,\, h/T_i\}$.

Sufficient conditions for stability are that $G_z$ has all its poles within the unit circle $|z| < 1$ and that there exists a (positive) constant $\eta$ such that

$$\mathrm{Re}[(1 + \eta(1 - z^{-1}))G_z(z)] + \frac{1}{k} \geq 0 \text{ for } z = e^{i\omega},\ \omega \geq 0 \qquad (4.12)$$

where the nonlinearity $\phi$ belongs to the cone $[0, k = 1]$.

In the upper plot of Figure 4.6 we have *the stability triangle* for the characteristic polynomial of Eq.(4.2). By choosing coefficients for the characteristic polynomial (4.2) in the upper left triangle (A1) we will get controller parameters $\{K,\, T_i\}$ which also will give a stable transfer function $G_z$. The corresponding poles are plotted in the lower diagram of Figure 4.6. Figure 4.7 shows a graphical representation of the Tsypkin condition (4.12) for this set of control parameters. The dashed non-intersecting line in Figure 4.7 corresponds to the existence of a positive parameter $\eta$ satisfying Eq.(4.12). Thus, absolute stability for the nonlinear system also is guaranteed for this choice of parameters.

Remark: The Tsypkin criterion guarantees stability for any cone bounded

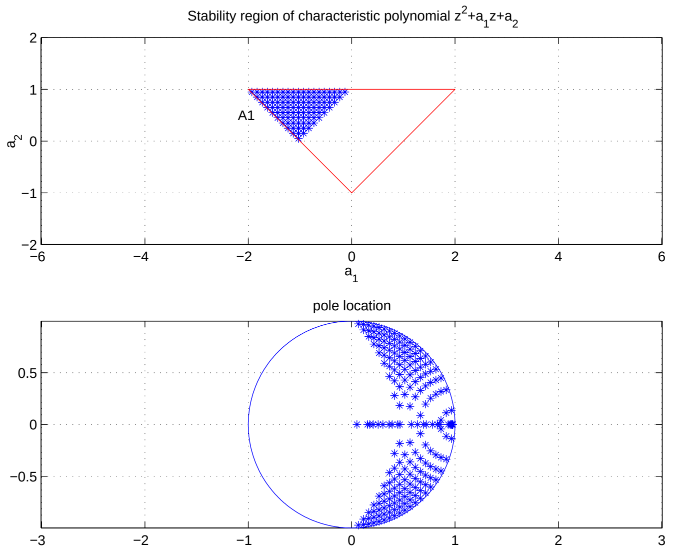

*Fig. 4.6: (Upper:)* The large triangle is the stability area a linear model would predict. However, from this parameter set, nonlinear analysis guarantees only stability for parameters $\{a_1,\, a_2\}$ in the upper left triangle ($A_1$, $'*'$). *(Lower:)* Pole location corresponding to $\{a_1,\, a_2\}\in A1$.

nonlinearity in $[0, 1]$ and we can thus expect to have some robustness in addition to stability in our case.

## *4.6 Experiments*

The admission control mechanism was implemented in the Apache web server. Apache is made up of a core package and several modules that handle different operations, such as Common Gateway Interface (CGI) execution, logging, caching etc. A new module was created that contains the admission control mechanisms. The new module was then hooked into the core of Apache, so that it was called every time a request was made to the web server. The module could then either reject or admit the request according to the control mechanism. The admission control mechanism was written in C and tested on a Windows platform. We tested the system by running tests on it and collecting performance metrics such as the server utilization distribution and step responses. We also compared the measurements with simulations. The queueing model was represented by a discrete-event simulation program implemented in C, and the control theoretic models were implemented with the Matlab Simulink package. The traffic generators in the discrete-time model were built as Matlab programs. They generate arrivals and departures according to the given statistical distributions.

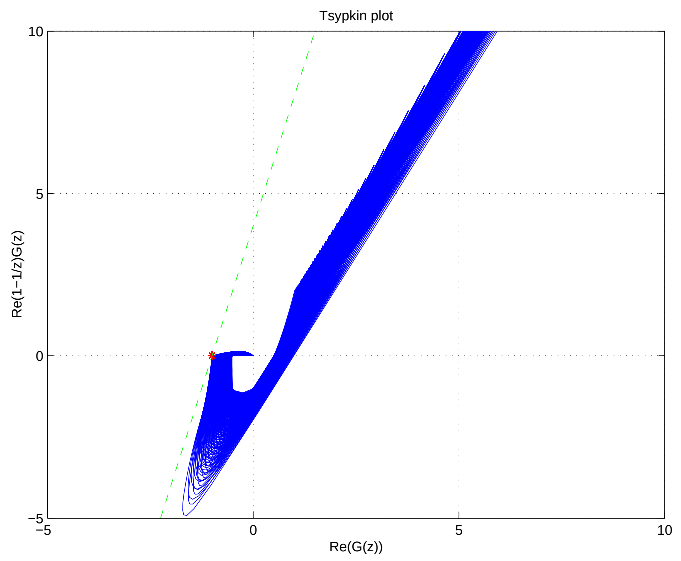

*Fig. 4.7:* Set of Tsypkin plots which all satisfy the frequency condition (4.12) for $G_z = G_z(K,\, T_i)$, where $(K,\, T_i)$ correspond to pole locations in Figure 4.6

### *4.6.1 Setup*

Our measurements used one server computer and one computer representing the clients connected through a 100 Mbits/s Ethernet switch. The server was a PC Pentium III 1700 MHz with 512 MB RAM running Windows 2000 as operating system. The computer representing the clients was a PC Pentium II 400 MHz with 256 MB RAM running RedHat Linux 7.3. Apache 2.0.45 was installed in the server. We used the default configuration of Apache. The client computer was installed with an HTTP load generator, which was a modified version of S-Client [18]. S-Client is able to generate high request rates even with few client computers by aborting TCP connection attempts that take too long time. The original version of S-Client uses deterministic waiting times between requests. We modified the code to use Poissonian arrivals instead. The client program was programmed to request dynamically generated HTML files from the server. The CGI script was written in Perl. It generates a number of random numbers, adds them together and returns the summation. The average request rate was set to 100 requests per second in all experiments except for the measurements in Figure 4.8. In all experiments, the control interval was set to one second.

### *4.6.2 Validation of the Model*

We have validated that the open system, that is without control feedback, is accurate in terms of average server utilization. The average server uti-

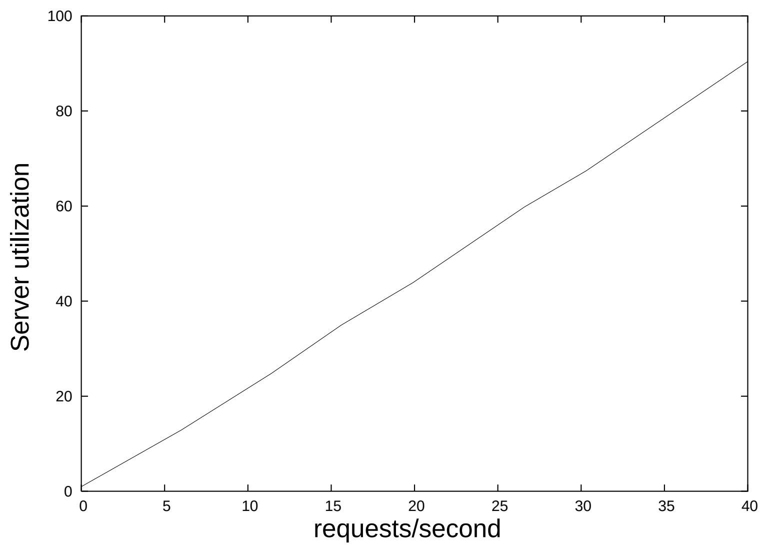

*Fig. 4.8:* Average server utilization for the open system.

lization for varying arrival rates are shown in Figure 4.8. For a single-server queue, the server utilization is proportional to the arrival rate, and the slope of the server utilization curve is given by the average service time. The measurements in Figure 4.8 gives an estimation of the average service time in the web server, $1/\mu$=0.0225.

### 4.6.3 Controller parameters

Control parameters for the PI-controller are chosen from the stability area A1 in Figure 4.6. In the simulations and experiments below we use $\{K, T_i\}$=$\{$20, 2.8$\}$. The parameter setting is also compared to $\{K, T_i\}$=$\{$20, 0.1$\}$, found outside the stability area.

### 4.6.4 Performance metrics

An admission control mechanism has two control objectives. First, it should keep the control variable at a reference value, i.e. the error, $e = y_{ref} - y$, should be as small as possible. Second, it should react rapidly to changes in the system, i.e. the so-called settling time should be short. Therefore, we test the mechanism in two ways. First, we show the steady-state distribution of the control variable, by plottting the estimated distribution function. The distribution function is estimated from measurements during 1000 seconds with the specific parameter setting. The distribution function shows how well the control mechanism meets the first control objective. Second, we plot the step response during 60 seconds when starting with an empty system. The step response shows the settling time for the control mechanism.

### 4.6.5 Distribution function

Figure 4.9 shows the estimated distribution function for the PI-controller. Both good and bad parameter settings were used. An ideal admission control mechanism would show a distribution function that is zero until the wanted load, and is one thereafter. In this case, the load was kept at 0.8, and the parameter setting, {K, Ti}={20, 2.8}, chosen from results in a controller that behaves very well in this sense. The parameter setting, $\{K, T_i\}$={20, 0.1}, as can be seen, perform worse. Also, as comparison, results from simulations of the M/D/1 system and the M/M/1 system are given in Figure 4.9, when using $\{K, Ti\}$={20, 2.8}. They show that the system behaves as expected.

### 4.6.6 Step response

Figure 4.10 shows the behaviour of the web server during the transient period. The measurements were made on an empty system that was exposed to 100 requests per second. The good parameter setting, $\{K, Ti\}$={20, 2.8}, exhibits a short settling time with a relatively steady server utilization. The bad parameter setting, $\{K, Ti\}$={20, 0.1} has its poles outside the unit circle and behaves badly, the load oscillates and is never stable. Comparisons to M/D/1 and M/M/1 simulations, also in Figure 4.10, show that the model is accurate.

## 4.7 Discussion

The analysis in Section 4.5.3 gives sufficient conditions and a region for control parameters which guarantee stability of the nonlinear closed loop as well as for the simplified linear model. We are of course not restricted to choose parameters from only this region as the main objective is that the nonlinear system should be stable. However, we can conclude that

- Pole-placement based on a linear model is OK in a restricted area (region $A1$ in Figure 4.6).
- There are choices of parameters that gives stable closed loop poles, but where the linear analysis would indicate an unstable closed loop systems.

During simulation studies the dominant nonlinear effect has come from the queue nonlinearity $\varphi$. The saturation due to limited arrival rate can be handled with a standard implementation of an anti-reset windup scheme, see [19].

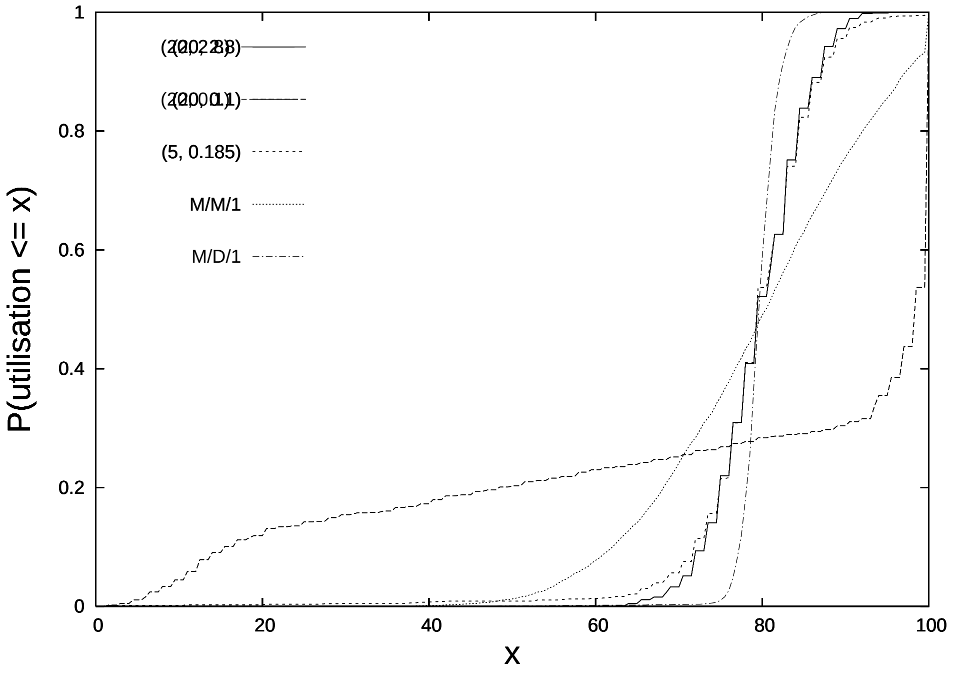

*Fig. 4.9:* Server utilization distribution of measurements from the real system together with simulations of the M/M/1 and the M/D/1 system.

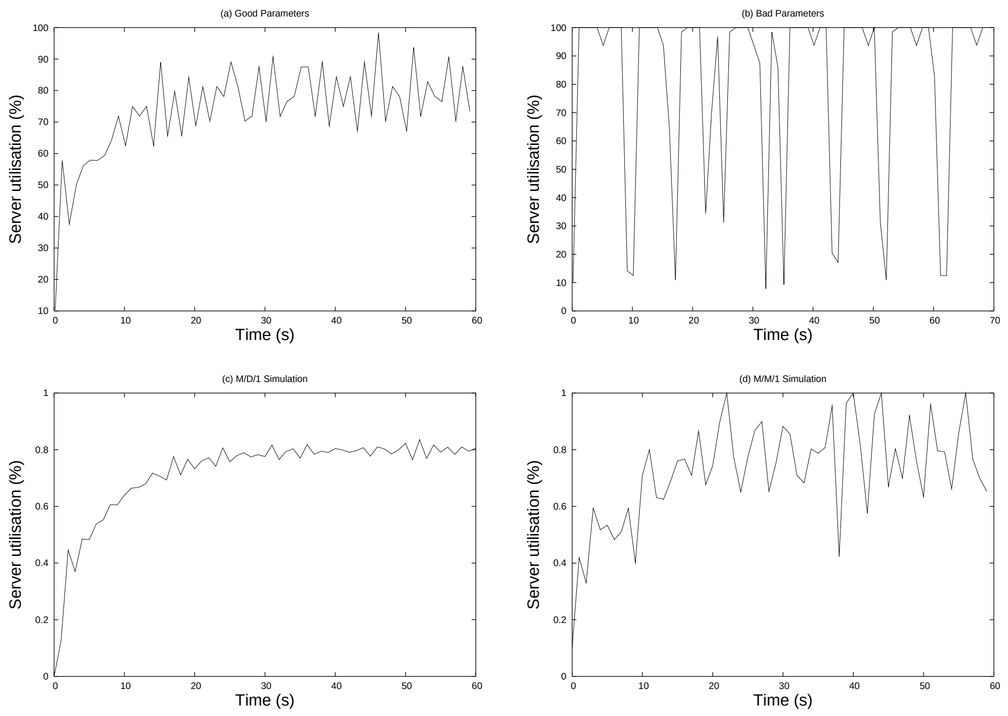

*Fig. 4.10:* (a) Example of a realisation with good parameters. (b) Example of a realisation with bad parameters. (c) Simulation of M/D/1-system with good parameters. (d) Simulation of M/M/1-system with good parameters.

## *4.8 Conclusion*

Traditionally, queuing theory has been used when investigating server systems. However, within queuing theory there are few mathematical tools for design and stability analysis of, for instance, admission control mechanisms. Therefore, these mechanisms have mostly been developed with empirical methods. In this paper, we have designed load control mechanisms for a web server system with control theoretic methods and analyzed its stability properties. The controller structure considered is a PI-controller and a region for stabilizing control parameters is presented.

The designs have been experimentally verified with simulations and experiments on an Apache web server system.

## *4.9 Acknowledgments*

This work has partially been supported by the Swedish Research Council through the Multi Project Grant 621-2001-3020 and contract 621-2001-3053.

# BIBLIOGRAPHY

[1] K. Åström and B. Wittenmark, *Computer-controlled systems, theory and design.* Prentice Hall Internation Editions, 1997, third Edition.

[2] T. Abdelzaher and C. Lu, "Modeling and performance control of internet servers," in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000, pp. 2234–2239.

[3] K. S. T.F. Abdelzaher and N. Bhatti, "Performance guarantees for web server end-systems: a control theoretic approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, January 2002.

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

[5] J. S. C. Lu, T.F. Abdelzaher and S. So, "A feedback control approach for guaranteeing relative delays in web servers," in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001, pp. 51–62.

[6] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, pp. 64–71, Sept/Oct 1999.

[7] R. R. J. Carlström, "Application-aware admission control and scheduling in web servers," in *Proc. Infocom*, 2002.

[8] P. P. L. Cherkasova, "Predictive admission control strategy for overloaded commerical web servers," in *Proc. 8th International IEEE Symposium on modeling, analysis and simulation of computer and telecommunication systems*, 2000, pp. 500–507.

[9] P. G. T. Voigt, "Adaptive resource-based web server admission control," in *Proc. 7th International Symposium on Computers and Communications*, 2002.

[10] S. R. P. Bhoj and S. Singhal, "Web2k: Bringing qos to web servers," *HP Labs Technical report, HPL-2000-61*, 2000.

[11] S. S. Jr., "Optimal control of admission to a queueing system," *IEEE Transactions on Automatic Control*, vol. 30, no. 8, pp. 705–713, August 1985.

[12] M. Kihl, “Overload control strategies for distributed communication networks,” Department of Communication Systems, Lund Institute of Technology, Tech. Rep. 131, 2002, ph.D. Thesis.

[13] W. Stallings, *Data & Computer Communications.* Prentice Hall, 2000, sixth Edition.

[14] T. Voigt, “Overload behaviour and protection of event-driven web servers,” in *In proceedings of the International Workshop on Web Engineering*, May 2002, pisa, Italy.

[15] B. W. M. Kihl, A. Robertsson, “Performance modelling and control of server systems using non-linear control theory,” in *Proc. 18th International Teletraffic Congress*, 2003.

[16] Y. Z. Tsypkin, “Frequency criteria for the absolute stability of nonlinear sampled-data systems,” *Automation and Remote Control*, vol. 25, no. 3, pp. 261–267, 1964.

[17] M. Larsen and P. V. Kokotovic, “A brief look at the tsypkin criterion: from analysis to design,” *Int. J. of Adaptive Control and Signal Processing*, vol. 15, no. 2, pp. 121–128, 2001.

[18] G. Banga and P. Druschel, “Measuring the capacity of a web server,” in *USENIX Symposium on Internet Technologies and Systems*, December 1997, pp. 61–71.

[19] B. W. A. Robertsson and M. Kihl, “Analysis and design of admission control in web server systems,” in *Proceedings of ACC03*, 2003.

# 5. PAPER IV

## ***Admission Control with Service Level Agreements for a Web Server***

Mikael Andersson, Jianhua Cao, Maria Kihl and Christian Nyberg
Department of Communication Systems, Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
Email: {mike, jcao, maria, cn}@telecom.lth.se

**Abstract**

One problem with web servers is that they are sensitive to overload. The servers may become overloaded during temporary traffic peaks when more requests arrive than the server is designed for. Because overload usually occurs rather seldom, it is not economical to overprovision the servers for these traffic peaks, instead admission control mechanisms can be implemented in the servers. This paper investigates two overload control strategies with performance bounds for a web server. In service level agreements, we bound average response times and throughputs for all service classes. Each request is sorted into a class, where each class is assigned a weight representing the income for the web site owner. Then a linear optimization algorithm is applied so that the total revenue for the web site during overload is maximized.

## 5.1 Introduction

Web sites on the Internet can be seen as server systems with one or more web servers processing incoming requests at a certain rate. The web servers have a waiting-queue where requests are queued while waiting for service. Therefore, a web server can be modelled as a queueing system including a server with finite or infinite queue. One problem with web servers is that they are sensitive to overload. The servers may become overloaded during temporary traffic peaks when more requests arrive than the server is designed for. Because overload usually occurs rather seldom, it is not economical to overprovision the servers for these traffic peaks, instead admission control mechanisms can be implemented in the servers. The admission control mechanism rejects some requests whenever the arriving traffic is too high and thereby maintains an acceptable load in the system. In this paper we study two admission control strategies based on percent blocking where we assume that the traffic rate are known.

Other papers have been presented in this area of research. Chen et al. describes in [1] an admission control scheme that divides requests into classes and then tries to guarantee a maximum response time for prioritized classes. Lee et al. describe a similar admission control scheme in [2]. Zhang et al. [3] develop a profit-aware QoS policy for web servers, where each request generates a certain profit to the site owner depending on the response time. Kanodia and Knightly propose an admission control scheme without profit optimization where requests are given priorities and response time limits called Latency-Targeted Multiclass Admission Control (LMAC) [4].

Also, control theoretic methods have been applied to web servers, Abdelzaher [5, 6] modelled the web server as a static gain to find optimal controller parameters for a PI-controller. A scheduling algorithm for an Apache web server was designed using system identification methods and linear control theory by Lu et al [7]. Bhatti [8] developed a queue length control with priorities. By optimizing a reward function, a static control was found by Carlström [9]. An on-off load control mechanism regulating the admittance of client sessions was developed by Cherkasova [10]. Voigt [11] proposed a control mechanism that combines a load control for the CPU with a queue length control for the network interface. Bhoj [12] used a PI-controller in an admission control mechanism for a web server.

In this paper we use a web server model that consists of a processor sharing node with a queue attached to it. A more thorough investigation of the model can be found in previous works, [13] and [14], by Cao et al. From the studies by Kihl and Widell [15] and Menasce et al. [16], we have introduced a set of classes that each request to a typical E-commerce site can be sorted into. The classes have different attributes such as revenue, throughput, service time requirements. Together with the class definition, we set up a service level agreement that the E-commerce site should uphold.

The service level agreement regulate the throughput for each class as well as the maximum allowed average response time.

Requests can be rejected in numerous ways, but since the requests generate different revenues for the site owner, it is a good idea to optimize the total profit during overload. Two different control strategies are therefore investigated that optimizes the total profit. One controller acknowledges the request's class attribute whereas the other one disregards the class attribute. The latter one is introduced as a comparison to the class dependent controller. Their performance regarding the ability to hold the service level agreement and the generated profit during overload is studied. One important thing to consider is that a rejection requires processing. It is not enough to simply disconnect the client, instead some "rejection page" should be sent. The rejection action therefore costs about the same amount of work as a small static web page. This is included in our model and the simulations show that this affects the total profit in an overloaded server. Also, the connection setup processing is considered. Before any rejections can be performed, the web server must set up a connection to see what kind of request that is coming. This connection setup processing has to be performed for all requests, not only the admitted ones.

The rest of the paper is organized as follows; Section 5.2 gives an introduction to web servers. It describes the web server model we use and explains the concept of classes and the service level agreement. Section 5.4 defines the admission control problem that can be formulated as two alternate linear programming problems. Section 5.5 shows simulations that compare the two methods investigated in this paper. Section 5.6 discusses the results while the last section concludes the work.

## *5.2 Preliminaries*

This section describes how a web server works. We describe the web server model that we use and we also define classes in a commercial web site context.

### *5.2.1 Web servers*

A web server contains software that offers access to documents stored on the server. Clients can browse the documents in a web browser. The documents can be for example static Hypertext Markup Language (HTML) files, image files or various script files, such as Common Gateway Interface (CGI), Javascript or Perl files. The communication between clients and server is based on HTTP [17].

An HTTP transaction consists of three steps: TCP connection setup, HTTP layer processing and network processing. The TCP connection setup is performed as a so called three-way handshake, where the client and the server exchange TCP SYN, TCP SYN/ACK and TCP ACK messages. Once the connection has been established, a document request can be issued with a HTTP GET message to the server. The server then replies with a HTTP GET REPLY message. Finally, the TCP connection is closed by sending TCP FIN and TCP ACK messages in both directions.

### *5.2.2 Classes*

It is natural to define a set of request classes when it comes to a server system like the web server. The type of classes that are considered depends on the web site. In this work, as will be shown, we have chosen to adopt and extend the request types found in the works of Kihl and Widell [15] and Menasce et al. [16]. In [15] they investigate admission control strategies for commercial web sites using different types of requests, for example Buy, Browse and Pay requests. The request types correspond to the different stages that a visitor to the site goes through in a typical session.

### *5.2.3 Admission Control*

A good admission control mechanism improves the site performance during overload by only admitting a certain amount of customers at a time into the site. The fundamental observation is that it is sometimes better to reject some customers so that other customers may finish their tasks and thereby generate some revenue for the site. Several kinds of blocking mechanisms to use in admission control have been proposed in the literature, a survey is given by Kihl in [18]. In this paper, we use percent blocking in the admission control. In this mechanism, a certain fraction of the requests is admitted.

### *5.2.4 Model description*

To be able to predict the web server's performance it is important to have a good performance model.

In [13] and [14] we show how a web server can be modeled as a single server queue with a processor sharing discipline. The queue length is restricted to a certain number of jobs. The model used here is similar (Figure 5.1).

The difference is that in this work, there is no maximum number of threads in the web server. As will be seen, the admission control is handled with another technique based on percent blocking. The server serves N classes of requests. The arrival processes of all classes are assumed to be Poisson. The arrival rate for the customers of class $i$ is $\lambda_i$. The mean service requirement for the customers of class $i$ is $v_i$ besides the connection setup time, denoted as $v_{init}$. The connection setup time is the same for all classes. For each received request a TCP connection has to be set up. To be able

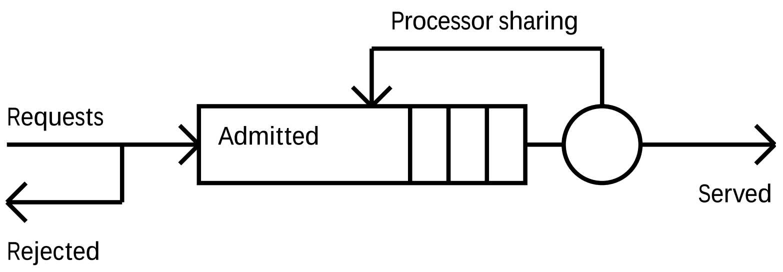

*Fig. 5.1:* The web server model.

to determine what class the request belongs to, the HTTP header must be parsed in the HTTP layer. The total arrival rate of all classes is therefore

$$\Lambda = \sum_{i=1}^{N} \lambda_i \tag{5.1}$$

Since the service discipline is processor sharing, the actual service time distribution can be neglected. Let the customer of class $(N+1)$ represent the "rejection service". In some papers dealing with admission control, the rejection service is neglected. If an admission controller should work in a realistic way, it is not suitable to just drop connections. Some sort of message should be sent to the rejected visitor that notifies about the rejection. The rejection service required must be less than the originally requested service to be of any practical use. We assume that a rejection requires $v_{rej}$ amount of service and

$$v_{rej} \leq min_i(v_i), (i = 1..N) \tag{5.2}$$

For requests of class $i$, the probability that the request will be served normally, that is without rejection is denoted $x_i$. If the request is rejected, it will become a request of class $(N+1)$. This gives

$$\lambda_{N+1} = \sum_{i=1}^{N} (1 - x_i)\lambda_i \tag{5.3}$$

## 5.3 Admission Control

We will investigate two admission controllers based on percent blocking:

**CAC-CI**, Contract-based Admission Control - Class Independent: The customers are accepted with probability $x_i = x$, disregarding their class identity.

**CAC-CD**, Contract-based Admission Control - Class Dependent: The customers of class $i$ are accepted based on their class identity, with probability

$x_i$. For the CAC-CI controller the server utilization is

$$\rho = \sum_{i=1}^{N} \lambda_i \cdot (v_{init} + x \cdot v_i + (1-x) \cdot v_{rej}) \tag{5.4}$$

and for the CAC-CD controller

$$\rho = \sum_{i=1}^{N} \lambda_i \cdot (v_{init} + x \cdot v_i + (1-x_i) \cdot v_{rej}) \tag{5.5}$$

It follows from the model that the average response time for served customers of class $i$ is

$$w_i = \frac{v_{init} + v_i}{1-\rho} \tag{5.6}$$

where $\rho$ is the server utilization and thus depends on the type of admission control in question.

The purpose of admission control is to guarantee that the served customers enjoy reasonable service times. Let $\tau_i$ be the upper bound of the average response time for customers of class $i$. We want

$$w_i \leq \tau_i,\ \forall i = 1,..,N. \tag{5.7}$$

We require that for customers of class $i$, the minimum acceptance probability must be $\alpha_i$. For the CAC-CI controller, this means

$$max\ \alpha_i \leq x \leq 1 \tag{5.8}$$

and for type CAC-CD admission control

$$\alpha_i \leq x_i \leq 1 \tag{5.9}$$

Now, given $w_1, w_2, ...w_N$ and $\alpha_1, \alpha_2, ...\alpha_N$ we define the so called *service level agreement* to be

$$S = (\{w_i\}_{i=1}^{N}, \{\alpha_i\}_{i=1}^{N}). \tag{5.10}$$

The service level agreement is considered broken if one or more of its constraints are violated. For example, if the response time for a certain class $i$ exceeds $\tau_i$ or less than $\alpha_i$ requests gets served, the service level agreement is broken.

Usually there are infinitely many admission control policies that satisfies the service level agreement.

Let $\gamma_i$ be the revenue (potential or real) for serving a class $i$ customer. We can therefore restrict our attention to those policies that maximize the reward. Since the CAC-CI controller accepts customers regardless of class identity, this is equivalent to maximize the throughput i.e.

$$\max x \cdot \sum_{i=1}^{N} \gamma_i \cdot \lambda_i \tag{5.11}$$

*Tab. 5.1:* Parameter list

| Variable | Description |
|---|---|
| $N$ | number of customer classes |
| $i$,$j$=1..N | indices of the customer class |
| $\lambda_i$ | arrival rate for class $i$ |
| $v_i$ | average service requirement for class $i$ |
| $\gamma_i$ | revenue for request of class $i$ |
| $v_{init,rej}$ | service requirements for connection setup and rejection |
| $\alpha_i$ | acceptance rate guarantee |
| $\tau_i$ | mean service time guarantee |
| $\rho$ | server utilization |

For type CAC-CD, it is slightly more complicated but still, the objective function is linear,

$$\max \sum_{i=1}^{N} \gamma_i \cdot \lambda_i \cdot x_i \tag{5.12}$$

To summarize we give a list of parameters and variables in Table 5.1.

## 5.4 Linear programming formulations

Since both of the objective functions for the controllers are linear, it is now feasible to set up linear programming formulations. For the CAC-CI and CAC-CD controllers they can be formulated as follows:

**CAC-CI**: maximize

$$x \cdot \sum_{i=1}^{N} \gamma_i \cdot \lambda_i$$

subject to (for all $i$ = 1..N)

$$v_{init} + v_i \leq \tau_i(1 - \sum_{j=1}^{N} \lambda_j(v_{init} + xv_j + (1-x)v_{rej}) \tag{5.13}$$

$$\max_i \ \alpha_i \leq x \leq 1 \tag{5.14}$$

**CAC-CD**: maximize

$$\sum_{i=1}^{N} \gamma_i \cdot \lambda_i \cdot x_i$$

subject to (for all $i$ = 1..N)

$$v_{init} + v_i \leq \tau_i(1 - \sum_{j=1}^{N} \lambda_j(v_{init} + x_j v_j + (1-x)v_{rej}) \quad (5.15)$$

$$\max_i \ \alpha_i \leq x \leq 1 \quad (5.16)$$

**CAC-CI.** The CAC-CI problem can be solved explicitly as follows:

From 5.13, we have

$$x \leq \frac{1 - (v_{init} + v_{rej}) \sum_{j=1}^{N} \lambda_j - \frac{v_{init}+v_i}{\tau_i}}{\sum_{j=1}^{N} \lambda_j (v_j - v_{rej})} = l_i$$

Let K:= arg $max_j \frac{v_{init}+v_j}{\tau_j}$. Clearly the CAC-CI problem has a solution

$$\max x_i \ \leq l_K.$$

If the condition above is satisfied the optimal solution for the CAC-CI problem is

$$x = l_K.$$

**CAC-CD.** The CAC-CD problem can be solved by any linear programming solver, e.g. CPLEX [19] quite easily.

We will first examine the necessary and sufficient conditions for the existence of a solution and then study the special case when ($v_{init} = v_{rej} =$ 0).

By the definition of $w_i$ and 5.7, we have

$$\rho \leq 1 - \frac{v_{init} + v_i}{\tau_i} \ \forall i = 1, .., N$$

On the other hand, we can achieve the smallest server utilization when we reject all customers and still fulfill the service level agreement:

Let K:= arg $max_j \frac{v_{init}+v_j}{\tau_j}$. Hence the problem has a solution

$$\sum_{j=1}^{N} \lambda_j (v_{init} + \alpha_j + (1-\alpha_j)v_{rej}) \leq 1 - \frac{v_{init} + v_K}{\tau_K}$$

Let $\rho_i := \lambda_i/\Lambda$ where $\Lambda = \sum_{j=1}^{N} \lambda_j$. The condition above implies

$$\Lambda \leq \frac{1 - \frac{v_{init}+v_j}{\tau_j}}{\sum_{j=1}^{N} \rho_j (v_{init} + \alpha_j + (1-\alpha_j)v_{rej}) \leq 1 - \frac{v_{init}+v_K}{\tau_K}}$$

The similar condition for the CAC-CI controller is

$$\Lambda \leq \frac{1 - \frac{v_{init}+v_j}{\tau_j}}{\sum_{j=1}^{N} \rho_j (v_{init} + \alpha_j + (1-\alpha_j)v_{rej}) \leq 1 - \frac{v_{init}+v_K}{\tau_K}}$$

where $\alpha_s = max_i \; \alpha_i$.

## 5.5 Experiments

In all simulations we set a total arrival rate, $\Lambda$. The arrival rates for the different classes were then determined by the request type distribution derived from [15] where the ratio of "leaving customers" were ignored. The distribution originally comes from the work of Menasce et al. [16], where the occasional buyer on a web site is studied. The buyer's requests are categorized into the classes shown in Table 5.2 and then the rates of each class are determined. The service times for each class has been taken from the simulation values in [15].

In all experiments, the work required in the connection setup phase was set to 0.005 seconds. The rejection work was also set to 0.005 seconds. The queueing model and the admission control algorithms were implemented as a discrete event simulation program in Java. Two sets of simulations were performed; one set where the CAC-CI controller was used, and one where the CAC-CD controller was used. Both methods were evaluated for their ability to enforce the service level agreement. In all simulations, the total arrival rate was increased from 10 to 60 requests per second, in steps of 5 requests per second. Table 5.2 shows the simulation configuration for both controllers with required service times $v_i$, distribution $d_i$, request revenue $\gamma_i$, acceptance rate guarantee $\alpha_i$ and mean service time guarantee $\tau_i$ for classes 1 to 5. For the CAC-CD controller optimization of the linear programming formulation was performed by using the Java version of lpsolver [20].

The simulations evaluated the controllers in terms of response times, throughput counted as admitted requests and the total profit generated at each arrival rate.

## 5.6 Results and Discussion

Figure 5.2 and 5.3 shows the response time for each class as a function of the total arrival rate. The solid lines in the diagrams represent the agreed

*Tab. 5.2:* Class Parameters

| | Description | $v_i$ | $d_i$ | $\gamma_i$ | $\alpha_i$ | $\tau_i$ |
|---|---|---|---|---|---|---|
| 1 | Browse | 0.015 | 0.41 | 1 | 0.2 | 1.5 |
| 2 | Search | 0.030 | 0.40 | 1 | 0.4 | 3.0 |
| 3 | Select | 0.015 | 0.17 | 1 | 0.6 | 1.5 |
| 4 | Add | 0.015 | 0.014 | 5 | 0.8 | 1.5 |
| 5 | Pay | 0.035 | 0.006 | 10 | 1.0 | 3.0 |

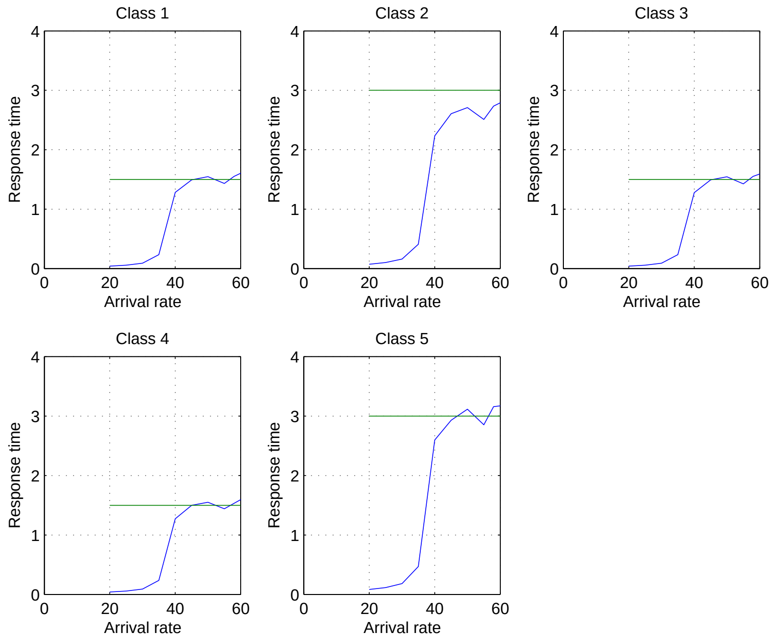

*Fig. 5.2:* Response times per class (CAC-CD)

response time limits. Figure 5.2 shows that the CAC-CD is capable of keeping the agreed response time limits whereas the CAC-CI controller cannot at higher arrival rates.

Figure 5.4 and 5.5 shows the throughput as a percentage, of completed requests for each class. The solid lines in the diagrams represent the agreed minimum service level. When it comes to throughput, each class receives the contracted amount of throughput with the CAC-CD controller. The CAC-CI breaks the contract at higher rates for classes 3, 4 and 5.

The two methods were compared from a profit perspective in Figures 5.6 and 5.7. The figure shows the profit per second versus arrival total rate. As can be seen for the CAC-CD controller, requests from classes 4 and 5 are more likely to be admitted at the expense of requests in class 1,

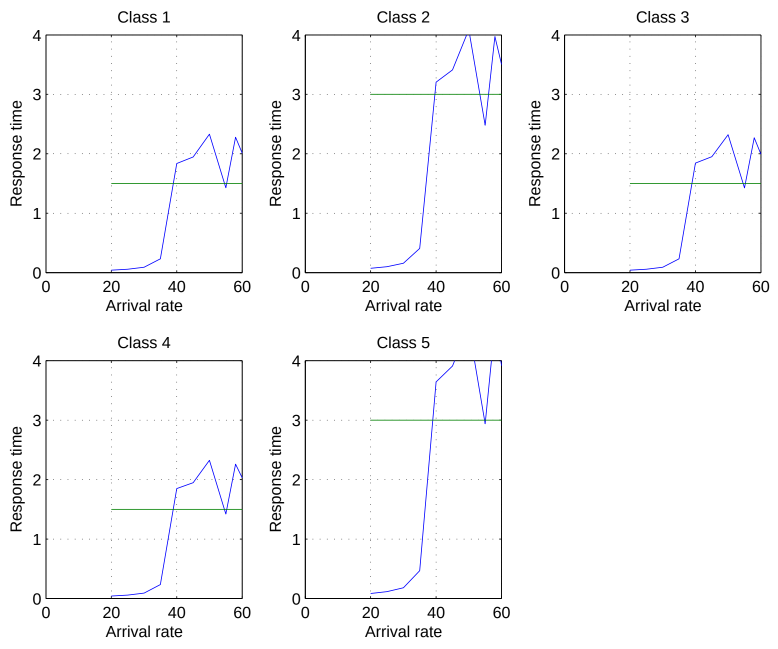

*Fig. 5.3:* Response times per class (CAC-CI)

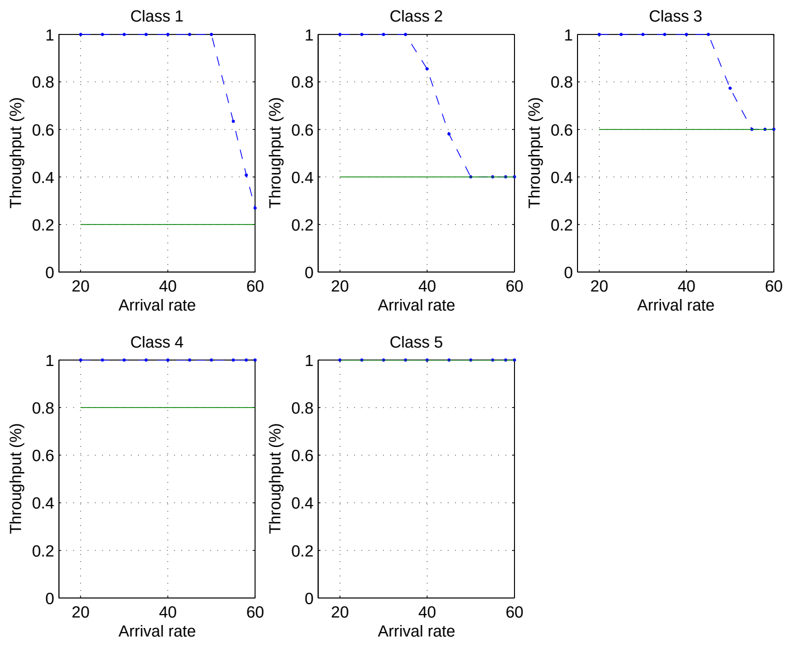

*Fig. 5.4:* Throughput per class (CAC-CD)

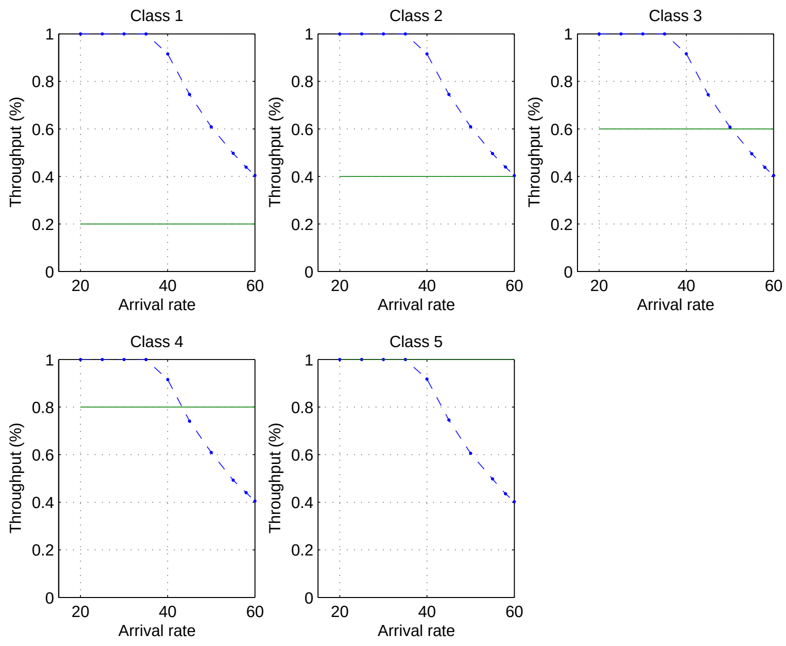

*Fig. 5.5:* Throughput per class (CAC-CI)

2 and 3 in higher arrival rates. The reason is that higher individual request revenue is generated in class 4 and 5. It may seem strange that the total profit decreases after its peak at $\lambda = 40$. The decline of total profit when the aggregated traffic rate reaches a certain limit is mainly due to that the server is busy with rejection most of the time in that traffic rate region. This implies that the server should be properly dimensioned in order to achieve the best performance, i.e. maximum profit, when the service of rejection cannot be neglected. The same trend can be found in Figure 5.7. For CAC-CI however, the total profit is lower at higher arrival rates. This is shown in the comparison in Figure 5.8. The controllers behave the same, profit-wise, up until $\lambda = 35$, after which the CAC-CD yields more total profit.

## 5.7 Conclusions

In this paper we have presented and compared two admission control strategies for a web server. The CAC-CI controller disregards the request's class property resulting in inferior performance compared to the CAC-CD controller that does regard class property. Both controllers optimize the total profit given the constraints given in the service level agreement concerning response times and throughput. The fact that each request is associated with an initialization work (even for rejected requests) and that rejections

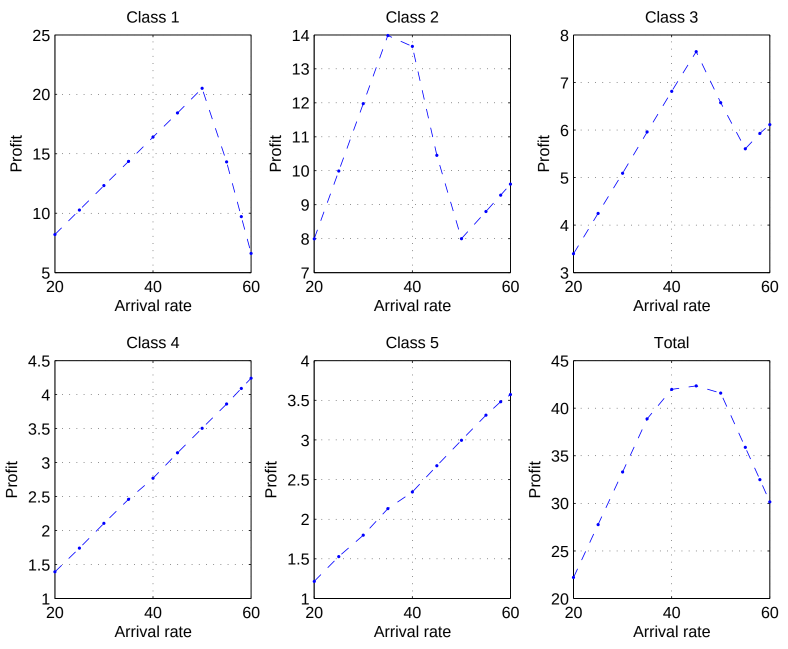

*Fig. 5.6:* Profit per class (CAC-CD)

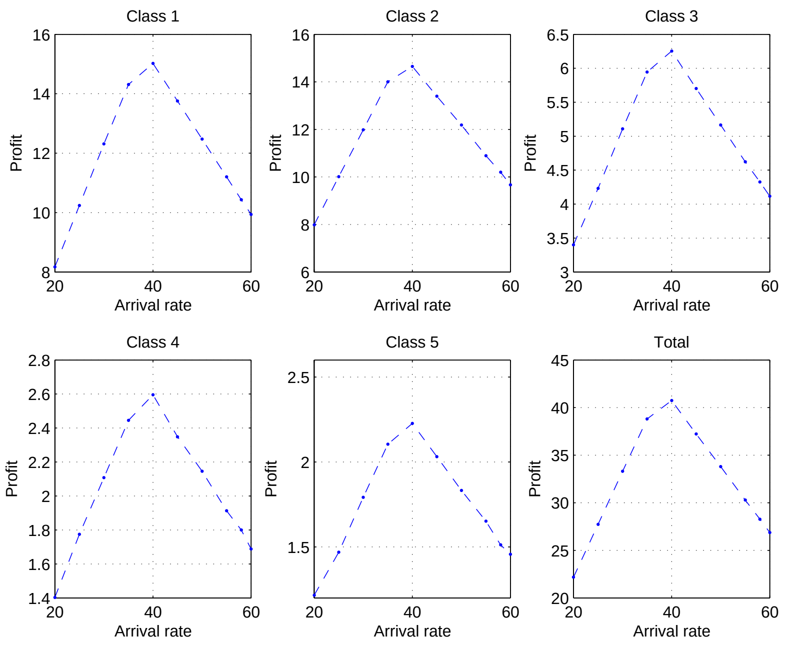

*Fig. 5.7:* Profit per class (CAC-CI)

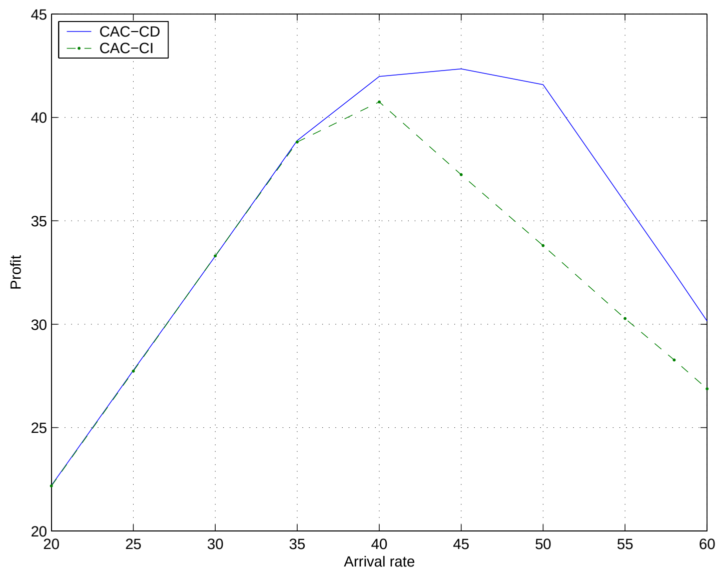

*Fig. 5.8:* Total profit

also cost in terms of processing power is considered. It results in declining total profit at higher arrival rates.

## *Acknowledgments*

The work has been supported by The Swedish Research Council under contract no. 621-2001-3053.

# BIBLIOGRAPHY

[1] H. C. Xiangping Chen and P. Mohapatra, "Aces: An efficient admission control scheme for qos-aware web servers," *Computer Communications*, no. 26, p. 1581, 2003.

[2] J. C. L. Sam C.M. Lee and D. K. Yau, "A proportional-delay diffserv-enabled web server: Admission control and dynamic adaptation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, 2004.

[3] E. S. Qi Zhang and G. Ciardo, "Profit-driven service differentiation in transient environments," in *In Proceedings of MASCOTS*, 2003, p. 230.

[4] V. Kanodia and E. W. Knightly, "Ensuring latency targets in multiclass web servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 1, 2003.

[5] T. Abdelzaher and C. Lu, "Modeling and performance control of internet servers," in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000, pp. 2234–2239.

[6] K. S. T.F. Abdelzaher and N. Bhatti, "Performance guarantees for web server end-systems: a control theoretic approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, January 2002.

[7] J. S. C. Lu, T.F. Abdelzaher and S. So, "A feedback control approach for guaranteeing relative delays in web servers," in *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001, pp. 51–62.

[8] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, pp. 64–71, Sept/Oct 1999.

[9] R. R. J. Carlström, "Application-aware admission control and scheduling in web servers," in *Proc. Infocom*, 2002.

[10] P. P. L. Cherkasova, "Predictive admission control strategy for overloaded commerical web servers," in *Proc. 8th International IEEE Symposium on modeling, analysis and simulation of computer and telecommunication systems*, 2000, pp. 500–507.

[11] P. G. T. Voigt, "Adaptive resource-based web server admission control," in *Proc. 7th International Symposium on Computers and Communications*, 2002.

[12] S. R. P. Bhoj and S. Singhal, "Web2k: Bringing qos to web servers," *HP Labs Technical report, HPL-2000-61*, 2000.

[13] C. N. M. K. Jianhua Cao, Mikael Andersson, "Web server performance modeling using an m/g/1/k*ps queue," in *In Proceedings of International Conference on Telecommunication (ICT)*, 2003.

[14] M. K. C. N. Mikael Andersson, Jianhua Cao, "Performance modeling of an apache web server with bursty arrival traffic," in *In Proceedings of International Conference on Internet Computing (IC)*, 2003.

[15] M. K. Niklas Widell, "Admission control schemes guaranteeing customer qos in commercial web sites," in *In Proceedings of IFIP and IEEE Conference on Network Control and Engineering (NETCON)*, 2002.

[16] R. F. D. Menasce, V. Almeida and M. Mendes, "Business-oriented resource management policies for e-commerce servers," *Performance Evaluation*, vol. 42, 2000.

[17] W. Stallings, *Data & Computer Communications.* Prentice Hall, 2000, sixth Edition.

[18] M. Kihl, "Overload control strategies for distributed communication networks," Department of Communication Systems, Lund Institute of Technology, Tech. Rep. 131, 2002, ph.D. Thesis.

[19] "Cplex," http://www.ilog.com/products/cplex/.

[20] "Linear programming solver, washington university in saint louis," http://www.cs.wustl.edu/~javagrp/help/LinearProgramming.html.

## Reports on Communication Systems

101. **On Overload Control of SPC-systems**
Ulf Körner, Bengt Wallström, and Christian Nyberg, 1989.
CODEN: LUTEDX/TETS- -7133- -SE+80P

102. **Two Short Papers on Overload Control of Switching Nodes**
Christian Nyberg, Ulf Körner, and Bengt Wallström, 1990.
ISRN LUTEDX/TETS- -1010- -SE+32P

103. **Priorities in Circuit Switched Networks**
Åke Arvidsson, Ph.D. thesis, 1990.
ISRN LUTEDX/TETS- -1011- -SE+282P

104. **Estimations of Software Fault Content for Telecommunication Systems**
Bo Lennselius, Lic. thesis, 1990.
ISRN LUTEDX/TETS- -1012- -SE+76P

105. **Reusability of Software in Telecommunication Systems**
Anders Sixtensson, Lic. thesis, 1990.
ISRN LUTEDX/TETS- -1013- -SE+90P

106. **Software Reliability and Performance Modelling for Telecommunication Systems**
Claes Wohlin, Ph.D. thesis, 1991.
ISRN LUTEDX/TETS- -1014- -SE+288P

107. **Service Protection and Overflow in Circuit Switched Networks**
Lars Reneby, Ph.D. thesis, 1991.
ISRN LUTEDX/TETS- -1015- -SE+200P

108. **Queueing Models of the Window Flow Control Mechanism**
Lars Falk, Lic. thesis, 1991.
ISRN LUTEDX/TETS- -1016- -SE+78P

109. **On Efficiency and Optimality in Overload Control of SPC Systems**
Tobias Rydén, Lic. thesis, 1991.
ISRN LUTEDX/TETS- -1017- -SE+48P

110. **Enhancements of Communication Resources**
Johan M Karlsson, Ph.D. thesis, 1992.
ISRN LUTEDX/TETS- -1018- -SE+132P

111. **On Overload Control in Telecommunication Systems**
Christian Nyberg, Ph.D. thesis, 1992.
ISRN LUTEDX/TETS- -1019- -SE+140P

112. **Black Box Specification Language for Software Systems**
Henrik Cosmo, Lic. thesis, 1994.
ISRN LUTEDX/TETS- -1020- -SE+104P

113. **Queueing Models of Window Flow Control and DQDB Analysis**
Lars Falk, Ph.D. thesis, 1995.
ISRN LUTEDX/TETS- -1021- -SE+145P

114. **End to End Transport Protocols over ATM**
Thomas Holmström, Lic. thesis, 1995.
ISRN LUTEDX/TETS- -1022- -SE+76P

115. **An Efficient Analysis of Service Interactions in Telecommunications**
Kristoffer Kimbler, Lic. thesis, 1995.
ISRN LUTEDX/TETS- -1023- -SE+90P

116. **Usage Specifications for Certification of Software Reliability**
Per Runeson, Lic. thesis, May 1996.
ISRN LUTEDX/TETS- -1024- -SE+136P

117. **Achieving an Early Software Reliability Estimate**
Anders Wesslén, Lic. thesis, May 1996.
ISRN LUTEDX/TETS- -1025- -SE+142P

118. **On Overload Control in Intelligent Networks**
Maria Kihl, Lic. thesis, June 1996.
ISRN LUTEDX/TETS- -1026- -SE+80P

119. **Overload Control in Distributed-Memory Systems**
Ulf Ahlfors, Lic. thesis, June 1996.
ISRN LUTEDX/TETS- -1027- -SE+120P

120. **Hierarchical Use Case Modelling for Requirements Engineering**
Björn Regnell, Lic. thesis, September 1996.
ISRN LUTEDX/TETS- -1028- -SE+178P

121. **Performance Analysis and Optimization via Simulation**
Anders Svensson, Ph.D. thesis, September 1996.
ISRN LUTEDX/TETS- -1029- -SE+96P

122. **On Network Oriented Overload Control in Intelligent Networks**
Lars Angelin, Lic. thesis, October 1996.
ISRN LUTEDX/TETS- -1030- -SE+130P

123. **Network Oriented Load Control in Intelligent Networks Based on Optimal Decisions**
Stefan Pettersson, Lic. thesis, October 1996.
ISRN LUTEDX/TETS- -1031- -SE+128P

124. **Impact Analysis in Software Process Improvement**
Martin Höst, Lic. thesis, December 1996.
ISRN LUTEDX/TETS- -1032- -SE+140P

125. **Towards Local Certifiability in Software Design**
Peter Molin, Lic. thesis, February 1997.
ISRN LUTEDX/TETS- -1033- -SE+132P

126. **Models for Estimation of Software Faults and Failures in Inspection and Test**
Per Runeson, Ph.D. thesis, January 1998.
ISRN LUTEDX/TETS- -1034- -SE+222P

127. **Reactive Congestion Control in ATM Networks**
Per Johansson, Lic. thesis, January 1998.
ISRN LUTEDX/TETS- -1035- -SE+138P

128. **Switch Performance and Mobility Aspects in ATM Networks**
Daniel Søbirk, Lic. thesis, June 1998.
ISRN LUTEDX/TETS- -1036- -SE+91P

129. **VPC Management in ATM Networks**
Sven-Olof Larsson, Lic. thesis, June 1998.
ISRN LUTEDX/TETS- -1037- -SE+65P

130. **On TCP/IP Traffic Modeling**
Pär Karlsson, Lic. thesis, February 1999.
ISRN LUTEDX/TETS- -1038- -SE+94P

131. **Overload Control Strategies for Distributed Communication Networks**
Maria Kihl, Ph.D. thesis, March 1999.
ISRN LUTEDX/TETS- -1039- -SE+158P

132. **Requirements Engineering with Use Cases - a Basis for Software Development**
Björn Regnell, Ph.D. thesis, April 1999.
ISRN LUTEDX/TETS- -1040- -SE+225P

133. **Utilisation of Historical Data for Controlling and Improving Software Development**
Magnus C. Ohlsson, Lic. thesis, May 1999.
ISRN LUTEDX/TETS- -1041- -SE+146P

134. **Early Evaluation of Software Process Change Proposals**
Martin Höst, Ph.D. thesis, June 1999.
ISRN LUTEDX/TETS- -1042- -SE+193P

135. **Improving Software Quality through Understanding and Early Estimations**
Anders Wesslén, Ph.D. thesis, June 1999.
ISRN LUTEDX/TETS- -1043- -SE+242P

136. **Performance Analysis of Bluetooth**
Niklas Johansson, Lic. thesis, March 2000.
ISRN LUTEDX/TETS- -1044- -SE+76P

137. **Controlling Software Quality through Inspections and Fault Content Estimations**
Thomas Thelin, Lic. thesis, May 2000
ISRN LUTEDX/TETS- -1045- -SE+146P

138. **On Fault Content Estimations Applied to Software Inspections and Testing**
Håkan Petersson, Lic. thesis, May 2000.
ISRN LUTEDX/TETS- -1046- -SE+144P

139. **Modeling and Evaluation of Internet Applications**
Ajit K. Jena, Lic. thesis, June 2000.
ISRN LUTEDX/TETS- -1047- -SE+121P

140. **Dynamic traffic Control in Multiservice Networks - Applications of Decision Models**
Ulf Ahlfors, Ph.D. thesis, October 2000.
ISRN LUTEDX/TETS- -1048- -SE+183P

141. **ATM Networks Performance - Charging and Wireless Protocols**
Torgny Holmberg, Lic. thesis, October 2000.
ISRN LUTEDX/TETS- -1049- -SE+104P

142. **Improving Product Quality through Effective Validation Methods**
Tomas Berling, Lic. thesis, December 2000.
ISRN LUTEDX/TETS- -1050- -SE+136P

143. **Controlling Fault-Prone Components for Software Evolution**
Magnus C. Ohlsson, Ph.D. thesis, June 2001.
ISRN LUTEDX/TETS- -1051- -SE+218P

144. **Performance of Distributed Information Systems**
Niklas Widell, Lic. thesis, February 2002.
ISRN LUTEDX/TETS- -1052- -SE+78P

145. **Quality Improvement in Software Platform Development**
Enrico Johansson, Lic. thesis, April 2002.
ISRN LUTEDX/TETS- -1053- -SE+112P

146. **Elicitation and Management of User Requirements in Market-Driven Software Development**
Johan Natt och Dag, Lic. thesis, June 2002.
ISRN LUTEDX/TETS- -1054- -SE+158P

147. **Supporting Software Inspections through Fault Content Estimation and Effectiveness Analysis**
Håkan Petersson, Ph.D. thesis, September 2002.
ISRN LUTEDX/TETS- -1055- -SE+237P

148. **Empirical Evaluations of Usage-Based Reading and Fault Content Estimation for Software Inspections**
Thomas Thelin, Ph.D. thesis, September 2002.
ISRN LUTEDX/TETS- -1056- -SE+210P

149. **Software Information Management in Requirements and Test Documentation**
Thomas Olsson, Lic. thesis, October 2002.
ISRN LUTEDX/TETS- -1057- -SE+122P

150. **Increasing Involvement and Acceptance in Software Process Improvement**
Daniel Karlström, Lic. thesis, November 2002.
ISRN LUTEDX/TETS- -1058- -SE+125P

151. **Changes to Processes and Architectures; Suggested, Implemented and Analyzed from a Project viewpoint**
Josef Nedstam, Lic. thesis, November 2002.
ISRN LUTEDX/TETS- -1059- -SE+124P

152. **Resource Management in Cellular Networks -Handover Prioritization and Load Balancing Procedures**
Roland Zander, Lic. thesis, March 2003.
ISRN LUTEDX/TETS- -1060- -SE+120P

153. **On Optimisation of Fair and Robust Backbone Networks**
Pål Nilsson, Lic. thesis, October 2003.
ISRN LUTEDX/TETS- -1061- -SE+116P

154. **Exploring the Software Verification and Validation Process with Focus on Efficient Fault Detection**
Carina Andersson, Lic. thesis, November 2003.
ISRN LUTEDX/TETS- -1062- -SE+134P

155. **Improving Requirements Selection Quality in Market-Driven Software Development**
Lena Karlsson, Lic. thesis, November 2003.
ISRN LUTEDX/TETS- -1063- -SE+132P

156. **Fair Scheduling and Resource Allocation in Packet Based Radio Access Networks**
Torgny Holmberg, Ph.D. thesis, November 2003.
ISRN LUTEDX/TETS- -1064- -SE+187P

157. **Increasing Product Quality by Verification and Validation Improvements in an Industrial Setting**
Tomas Berling, Ph.D. thesis, December 2003.
ISRN LUTEDX/TETS- -1065- -SE+208P

158. **Some Topics in Web Performance Analysis**
Jianhua Cao, Lic. thesis, June 2004.
ISRN LUTEDX/TETS- -1066- -SE+99P

159. **Overload Control and Performance Evaluation in a Parlay/OSA Environment**
Jens K. Andersson, Lic. thesis, August 2004.
ISRN LUTEDX/TETS- -1067- -SE+100P

160. **Performance Modeling and Control of Web Servers**
Mikael Andersson, Lic. thesis, September 2004.
ISRN LUTEDX/TETS- -1068- -SE+105P