Thiemo Voigt

# Architectures for Service Differentiation in Overloaded Internet Servers

UPPSALA
UNIVERSITET

# Architectures for Service Differentiation in Overloaded Internet Servers

Thiemo Voigt

A Dissertation submitted
for the Degree of Doctor of Philosophy
Department of Information Technology
Uppsala University

May 2002

Dept. of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

DoCS 02/119
ISSN 0283-0574

Swedish Institute of Computer Science
Box 1263
SE-164 29 Kista
Sweden

SICS Dissertation Series 30
ISSN 1101-1335
ISRN SICS-D--30--SE

Dissertation for the Degree of Doctor of Philosophy in Computer Systems presented at Uppsala University in 2002.

**ABSTRACT**

Voigt, T. 2002: Architectures for Service Differentiation in Overloaded Internet Servers. *SICS Dissertation Series* 30. Also as *DoCS* 02/119. 153 pp. Uppsala. ISBN 91-506-1559-9.

Web servers become overloaded when one or several server resources such as network interface, CPU and disk become overutilized. Server overload leads to low server throughput and long response times experienced by the clients.

Traditional server design includes only marginal or no support for overload protection. This thesis presents the design, implementation and evaluation of architectures that provide overload protection and service differentiation in web servers. During server overload not all requests can be processed in a timely manner. Therefore, it is desirable to perform service differentiation, i.e., to service requests that are regarded as more important than others. Since requests that are eventually discarded also consume resources, admission control should be performed as early as possible in the lifetime of a web transaction. Depending on the workload, some server resources can be overutilized while the demand on other resources is low because certain types of requests utilize one resource more than others.

The implementation of admission control in the kernel of the operating system shows that this approach is more efficient and scalable than implementing the same scheme in user space. We also present an admission control architecture that performs admission control based on the current server resource utilization combined with knowledge about resource consumption of requests. Experiments demonstrate more than 40% higher throughput during overload compared to a standard server and several magnitudes lower response times.

This thesis also presents novel architectures and implementations of operating system support for predictable service guarantees. The Nemesis operating system provides applications with a guaranteed communication service using the developed TCP/IP implementation and the scheduling of server resources. SILK (Scout in the Linux kernel) is a new networking stack for the Linux operating system that is based on the Scout operating system. Experiments show that SILK enables prioritizing and other forms of service differentiation between network connections while running unmodified Linux applications.

*Thiemo Voigt, Department of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden, E-mail: Thiemo.Voigt@sics.se*



ISSN 0283-0574
ISSN 1101-1335
ISRN SICS-D--30--SE

Printed in Sweden by Elanders Gotab, Stockholm 2002.
Distributor: SICS, Box 1263, SE-164 29 Kista, Sweden.

*To K&K*

# Acknowledgments

First of all I want to thank my supervisor Per Gunningberg. Already as a thesis student in Uppsala many years ago, I decided that if I do a PhD, I will do it for Per. During dark hours in the last years I sometimes regretted having started the PhD studies, but I do not remember (m)any instances where I regretted being Per's student. I have greatly benefitted from Per's technical knowledge, experience in writing and judging the values of papers and his broad contact net.

I am very grateful to Bengt Ahlgren, manager of the Computer and Network Architectures Laboratory (CNA) at SICS, who has also been my secondary advisor and co-authored one of the papers presented in this thesis. During the Pegasus II project, Bengt taught me a lot about writing papers. Probably even more important, Bengt has made this thesis possible by assigning me to projects suitable to pursue my PhD studies.

I am also grateful for the help I have got in the CNA lab. In particular, I am thankful to Laura Feeney and Ian Marsh, who have proofread many of my papers, as well as to Assar Westerlund, Björn Grönvall, Adam Dunkels and Lars Albertsson who answered many questions on UNIX and C programming. Thanks to all my colleagues both in the CNA lab and SICS for making SICS an exciting place to conduct research at. Thanks to my fellow PhD students and the rest of the people at DoCS, in particular the members of the Communication Research group.

One of the greatest experiences during my PhD studies was the internship at the IBM TJ Watson Research Center in Hawthorne, NY. It was an honour and pleasure to work with Renu Tewari, Ashish Mehra and Douglas Freimuth who are also co-authors of one of the papers in my thesis. Thanks also to Erich Nahum and Anees Shaikh for helping me to get the internship.

I also had the pleasure to work with Andy Bavier. Even a buggy TCP could not prevent us from having a great time while working on SILK. Andy provided also valuable comments on this thesis. Thanks also to Larry Peterson and Mike Wawrzoniak who co-authored the SILK paper.

I am grateful for all the help I got from the rest of the Nemesis crowds in Cambridge and Glasgow while working on Nemesis. In particular, without Austin Donnelly's help I might still try to boot Nemesis. Thanks to Steven Hand for coming to Sweden and acting as opponent for my licentiate thesis defense.

Thanks to Jakob Carlström and Jakob Engblom who proofread parts of my licentiate thesis as well as to Ingela Nyström and Arnold Pears for excellent feedback on parts of both my licentiate and PhD thesis.

Hans Hansson and Per Stenström, the directors of ARTES and PAMP, made my financial support for the last two years of my work possible for which I am grateful. I also want to thank Lars Björnfot, my industrial contact person in that project.

Finally, I want to thank my parents and the rest of my family. My deepest thanks go to my wife Kajsa whose love and support has made it possible for me to both accomplish this thesis and have a wonderful time with our son Kalle.

The work is supported in part by the CEC DG III Esprit LTR project 21917 Pegasus II with additional support from Telia. This work is also partially funded by the national Swedish Real-Time Systems research initiative ARTES (www.artes.uu.se), supported by the Swedish Foundation for Strategic Research.

The Swedish Institute of Computer Science is sponsored by Telia, Ericsson, SaabTech Systems, FMV (Defence Materiel Administration), Green Cargo (Swedish freight railway operator), IBM, Hewlett-Packard and ABB.

This thesis is composed of the following papers. In the summary, the papers will be referred to as papers A through E.

[A] Thiemo Voigt and Bengt Ahlgren. Scheduling TCP in the Nemesis Operating System. *IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks*, Salem, MA, USA, August, 1999.

[B] Thiemo Voigt, Renu Tewari, Douglas Freimuth and Ashish Mehra. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. *Proceedings of Usenix Annual Technical Conference*, pages 189 – 202, Boston, MA, USA, June 2001.

[C] Thiemo Voigt and Per Gunningberg. Kernel-based Control of Persistent Web Server Connections. *ACM Performance Evaluation Review*, 29(2):20–25, September 2001.

[D] Thiemo Voigt and Per Gunningberg. Handling Multiple Bottlenecks in Web Servers Using Adaptive Inbound Controls. *Seventh International Workshop on Protocols for High-Speed Networks (PfHSN 2002)*, Berlin, Germany, April 2002.

[E] Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson and Per Gunningberg. SILK: Scout Paths in the Linux Kernel. Technical Report 2002-009, Department of Information Technology, Uppsala University, Uppsala, Sweden, February 2002.

**Other Papers and Reports**:
These papers constitute part of my thesis work but are not included in the thesis.


[1] Bengt Ahlgren and Thiemo Voigt. IP over ATM. *Pegasus II Project Deliverable*, Technical Report, September 1999.

[2] Bengt Ahlgren, Lars Albertsson and Thiemo Voigt. IPv4 and IP Multicast Functionality. *Pegasus II Project Deliverable*, Technical Report, September 1999.

[3] Bengt Ahlgren and Thiemo Voigt. IP QoS for Nemesis. *Pegasus II Project Deliverable*, Technical Report, September 1999.

[4] Thiemo Voigt. Providing Quality of Service to Networked Applications Using the Nemesis Operating System. *Ph. Lic. thesis*, Technical Report DoCS 99/113, Uppsala University, Sweden, October 1999.

[5] Thiemo Voigt, Renu Tewari and Ashish Mehra. In-Kernel Mechanisms for Adaptive Control of Overloaded Web Servers. *Eunice Open European Summer School*, Twente, The Netherlands, September 2000.

[6] Thiemo Voigt and Per Gunningberg. Dealing with Memory-intensive Web Requests. Technical Report 2001-010, Department of Information Technology, Uppsala University, Sweden, May 2001.

[7] Thiemo Voigt and Per Gunningberg. Handling Persistent Connections in Overloaded Web Servers. *Real-Time in Sweden 2001*, Halmstad, Sweden, August 2001.

[8] Thiemo Voigt. Overload Behaviour and Protection of Event-driven Web Servers. *International Workshop on Web Engineering, Networking 2002*, Pisa, Italy, May 2002.

[9] Thiemo Voigt and Per Gunningberg. Adaptive Resource-based Web Server Admission Control. *7th IEEE Symposium on Computers and Communication 2002*, Taormina/Giardini Naxos, Italy, July 2002.

# Contents

# 1 Introduction

With the advent of the World Wide Web (WWW) the number of Internet users has grown steadily. In some countries more than half of the population uses the Internet regularly. In October 2001, the country with the highest rate of Internet penetration was Sweden with more than 63% of the population having Internet access according to Nua Internet Surveys [44]. At about the same time, the number of people all over the world having Internet access reached 500 million [44].

This growth in popularity, together with the appearance of new services such as e-commerce and on-line banking, has increased the demand made on the capacity of both the Internet infrastructure and web servers. Web servers consist of one or several computers that provide services such as searching and retrieval of documents and information, online banking and electronic commerce. To cope with this increasing demand, huge amounts of bandwidth have been added to the core of the Internet. Technical advances have been made in many areas that have enabled continuous operation of the Internet despite the increasing demands.

However, many users still experience the WWW more as a World Wide Wait than a satisfying medium for business and entertainment. The web response times are not always caused by a congested network, but often by overloaded and non-responding web servers. A web server is overloaded when the demand exceeds the capacity of the server, i.e., when a server receives more user requests than it can handle. In particular, during exciting, often unforeseeable events such as terror attacks or stock market panics, there is a dramatic increase of user requests to the affected web servers which makes it hard if not impossible to retrieve the requested information from these servers. This can be very annoying in case of a stock market panic. It can also lead to serious financial losses if shareholders cannot fulfill their intended financial affairs due to non-responding web servers.

Under normal load conditions when the rate of incoming requests is below the capacity of a server, the server can service all requests without introducing large delays. During high load, however, one or several of the critical server resources – network interface, disk, physical memory and CPU (Central Processing Unit) – become scarce, which may lead to low server throughput and customers experiencing long delays. In such situations, the server does not have sufficient resources to provide good service to all clients. Instead, the server should perform service differentiation which aims at providing better service, i.e., lower response time and higher throughput to preferred clients as opposed to regular clients. The latter may receive degraded or no service during overload. This thesis deals with architectures that enable service differentiation in overloaded Internet servers, in particular web servers.

Even if one is willing to pay an extra fee to receive guaranteed fast service also during unforeseeable events, banks do not offer such services. The follow on question is why not? One possible explanation is that it is not worth the effort


Figure 1: Interaction between client and web server

for the banks, but there are also technical reasons that do not make it feasible to provide such a service. Most traditional server operating systems, such as UNIX, are designed as time-sharing systems. The aim of such systems is to maximize system utilization, while simultaneously providing users, or processes, with a fair share of the CPU. In an overload situation each process gets a small but fair share of the CPU which means that everyone receives poor service. This is in contrast to the notion of service differentiation.

Service differentiation is also important in other situations, for example, when a service provider wants to give better service to requests associated with electronic purchases or other transactions that provide financial gain as opposed to data requests associated with users merely browsing the provider's site. Furthermore, there is a trend of co-hosting multiple customers' web sites on the same server (in this case a customer can be thought of as a company or an organization). Customers paying a higher fee expect better service for requests to their site than customers paying less for the hosting service.

## 1.1 Background

This section presents how web transactions in general are handled. Readers who are familiar with web transactions and web server operation may wish to proceed directly to Section 1.2.

### 1.1.1 Web Transactions and Web Servers

In the World Wide Web, users request content from web servers. Usually, a browser application such as Internet Explorer or Netscape Navigator running on the user's host[1] (the client), sends a request over the Internet to the web server. The web server responds by sending the requested object back to the client. The client's browser handles the response by displaying the requested web page in the browser's window. Figure 1 shows this interaction between the client and the server called a web transaction.

A web server is typically an application program running on the server host (Figure 2). It receives requests from clients over the network. Before the request is transmitted, a TCP (Transmission Control Protocol) connection is set up (see Figure 3, step 1–4) which requires three messages. The web server application is

[1] A host is a computer that is connected to the network.

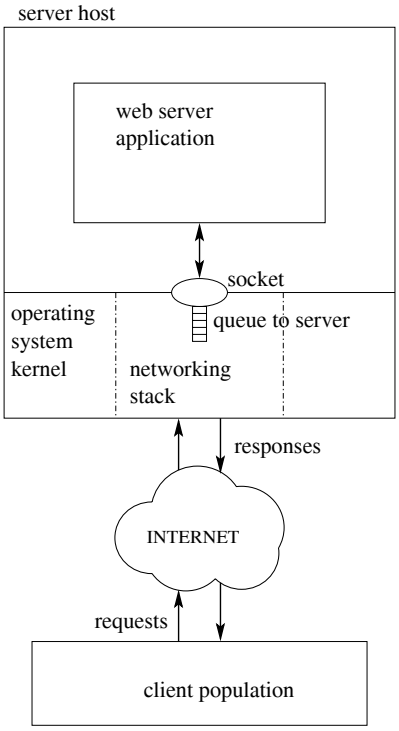


Figure 2: Simplified web server architecture

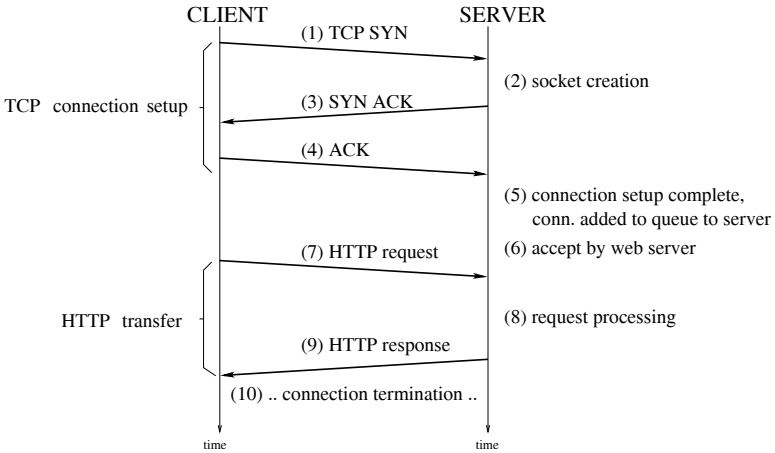


Figure 3: Simple web transaction

not involved in this setup, it is handled by the underlying network stack. After the connection is established (Figure 3, step 5), it is enqueued in the listen queue from which the web server application accepts it (step 6). When the web server application has accepted a connection, it awaits a request from the client, typically a Hypertext Transfer Protocol (HTTP) [11, 26] request (step 7). After processing the request (step 8), the web server returns the requested page to the client (step 9). After the server has transmitted the requested data the connection is closed (step 10).

HTTP is a stateless protocol, i.e., HTTP does not require web servers to keep any information about clients and their requests. Cookies are used to keep state in HTTP. A cookie is a small amount of state sent by the server to the client. The server expects the client to include the cookie in subsequent requests. This way, the server is able to maintain information about the client and its state within a session and across different sessions.

The operating system and the web server application use the socket interface to communicate with each other. The content of a message is copied between the operating system and the application, and vice versa. This data movement through the socket interface is a fairly time consuming operation.

Handling and processing requests utilizes web server resources. For example, the processing of messages and parsing of the URL (Uniform Resource Locator) that identifies a web page requires CPU time. Reading the file from the disk consumes disk bandwidth and transmitting the response requires bandwidth on the network interface.

### 1.1.2 The Internet and the World Wide Web

The Internet is the underlying infrastructure of the World Wide Web. The Internet can be regarded as a hierarchy of ISPs (Internet Service Providers) or networks, each having their own administration. As shown in Figure 4, web clients are usually connected to a local network, for example, a university network or a local ISP. Web servers can be placed anywhere in the network. Web clients communicate with servers over the Internet.

Not all requests that clients issue travel all the way to the web server. Some servers have mirrors or replicas, i.e., servers with a duplicate of the content of the original server. The ideal situation is to find an optimal replica for clients where optimal is based on proximity or on other criteria such as load [21]. The simplest scheme, by which a replica can be selected, is to embed replica identities in the URLs of the web pages. The HTTP protocol itself can also redirect a request to a replica by returning a particular response code.

Some requests also get redirected to web caches. A web cache is an intermediary host with storage facilities. A cache stores responses in order to reduce the response time and network bandwidth consumption for equivalent future requests [21]. That means that if another client subsequently requests the same page, the local copy can be provided immediately to satisfy the request. Since web caches primarily aim at reducing the response time, they are often placed

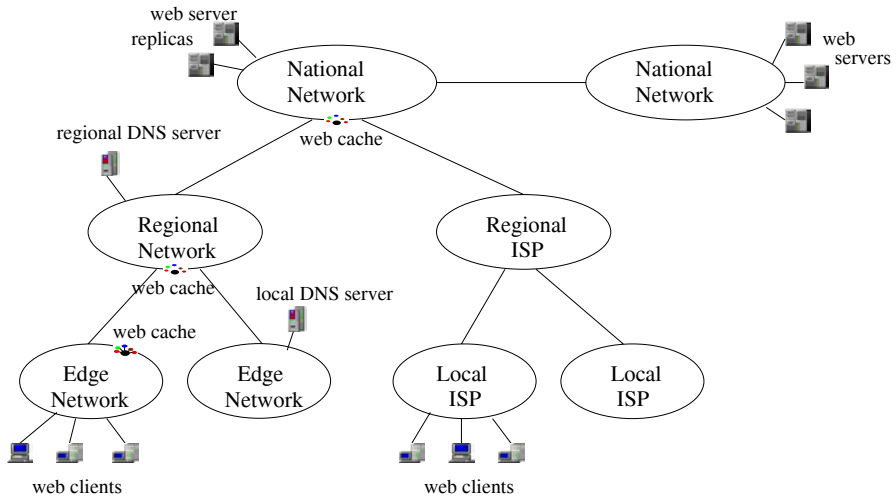


Figure 4: Internet and WWW infrastructure

close to clients. Another placement possibility for web caches are the access points between two different networks to avoid that the packets need to travel through the network [48].

The Domain Name System (DNS) plays an important part in the web infrastructure. Its task is the translation between hostnames, e.g., www.website.com, and IP (Internet Protocol) addresses. IP addresses are the identifiers for hosts in the Internet. The DNS also provides facilities to select server replicas for clients in a user-transparent manner. Instead of having only one possible IP address for each hostname, DNS can select among several servers depending on estimated proximity, load or other properties.

## 1.2 Problem Areas

Servers become overloaded when one or several critical resources become scarce. Server overload affects both the server throughput and the response time experienced by the clients. Figure 5 schematically illustrates the response time and total server throughput as functions of the request rate. The left part of the figure demonstrates how the response time increases with the server load. The response time is low as long as no server resource is overutilized. However, when the server resource bottleneck becomes overutilized, i.e., the bottleneck resource cannot keep up with the arrival rate of requests, the queue length to the resource bottleneck and thus the response time theoretically increases to infinity. This is depicted by the sudden increase of the response time.

The right part of Figure 5 depicts how the server throughput increases with the request rate until the request rate exceeds the capacity of the web server.

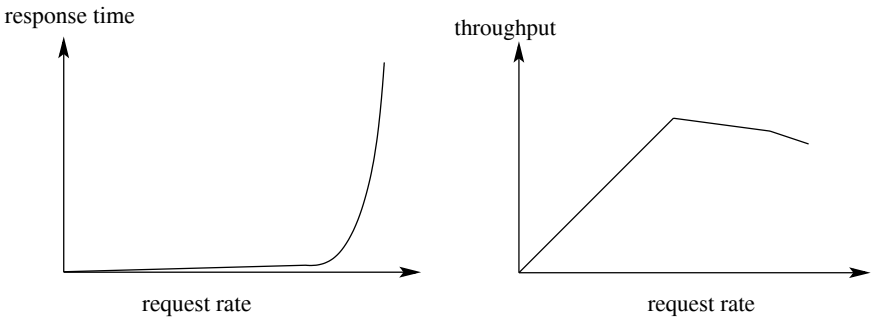


Figure 5: Impact of server load on response time and throughput

At this point, the throughput decreases due to the additional and unproductive time the CPU spends on processing incoming connection requests that are dropped when the listen queue is full. Moreover, the high rate of network interrupts prevents the web server application from making fast progress which contributes to the lower throughput. Lower server throughput leads to loss of revenue, while long delays cause user frustration and decrease task success and efficiency [17]. Users' tolerance for delay is application dependent, but often a threshold of 10 seconds for web interaction is mentioned in the literature [12].

In this thesis I will use the term architecture to define a set of mechanisms and their interaction, designed for a specific task, for example, service differentiation in overloaded web servers. An architecture describes how the mechanisms interact with each other and the environment, i.e., the operating system and the web server.

One way of reducing the load on individual servers is to utilize distributed web server architectures. These architectures distribute client requests to server replicas or caches as described in the previous section. Whereas traditional web cache proxies are placed close to clients to reduce client-perceived latency, reverse proxies are placed close to servers to reduce the load on them [34]. However, not all web data is cacheable, in particular dynamic and personalized data. During recent years, there has been an overall reduction in the fraction of traffic that is cacheable [34].

A current trend is to organize a number of servers in a cluster with front-ends or dispatchers that distribute the incoming requests among the servers [27]. If the capacity of the cluster is not sufficient, more machines can be added to the cluster. However, it is not unusual that the peak demand for web services is 100 times greater than the average demand [31]. Hence, provisioning for peak demand is not economically feasible, since this would imply that most servers are idle most of the time. Neither does it help to improve the efficiency of the web server or the underlying operating system, because also faster servers will become overloaded when the demand is sufficiently high.

Note that it is generally not possible to buffer requests until overload situations have ceased, since the duration of overload situations is unpredictable. Moreover, buffering requests increases the delay experienced by the clients.

In summary, while the approaches above contribute to reduce the load on an individual server, individual servers may still be confronted with a demand exceeding their capacity. In other words, individual servers may always experience overload situations where they do not have enough resources to process all requests in a timely manner. Thus, in an overload situation, we must perform service differentiation and determine which requests to serve and which ones to reject. The main focus of this thesis is to study architectures that enable service differentiation in overloaded individual and clustered web servers.

The content of this thesis is divided in two research areas, described in the next two sections. The first research area deals with mechanisms and architectures that perform overload protection and service differentiation by regulating the access to the server itself. The second area deals with the problem of providing predictable service by regulating the access to the critical server resources.

### 1.2.1 Mechanisms and Architectures for Server Overload Protection and Service Differentiation

Most web servers deploy rather simple schemes for admission control. For example, when the number of new requests enqueued exceeds a predefined threshold, additional incoming requests are dropped. Chen *et al.* state that such a tail-dropping admission control scheme requires careful system capacity planning and works well only in steady workload situations since the approach has problems coping with the highly non-steady and variable demands that web servers experience [19]. Furthermore, tail dropping without considering the identity of the requestor cannot provide any service differentiation. Hence, to provide differentiated access to a web server under high load, we need enhanced mechanisms for request classification and admission control. Request classification is needed to identify and classify the incoming requests to decide which service a request should receive, i.e., to which service class a request belongs to. Admission control mechanisms have to decide about the acceptance of the request, which can be based on several factors such as the request's service class, the expected resource requirements and the current server load.

Given the scarceness of available server resources during high load, the efficiency of these mechanisms is of major importance. Note that even requests that are eventually discarded consume resources. Abdelzaher and Bhatti noticed that under very high load, about 50% of the end-system utilization is wasted on connections that are eventually rejected [1]. Bhoj *et al.* have experienced that under certain conditions classifications can become the server bottleneck [14]. The earlier in the lifetime of a web transaction admission control is performed, the less resources are wasted in case of a rejection. However, the earlier admission control is performed, the less information about both the client and the requested object, including its potential resource consumption, is known.

Paper B presents mechanisms for performing efficient admission control and service differentiation in overloaded web servers.

Mechanisms for classification and admission control of individual requests are important parts of overload protection and service differentiation architectures for web servers. Depending on the admission control strategy all requests, or all requests belonging to a certain service class, may be rejected under high load. Admission control of requests should be triggered when the server starts to experience high load. Different architectures use different indicators of high load, e.g., the length of queues [13], CPU utilization [20] or a variety of other load indicators [31].

Many architectures try to avoid server overload by limiting the number of requests that are allowed to enter the system during a certain time unit [14, 31] or that are in the system concurrently [2]. Many of them use static thresholds as indicators of high load and limit the number of accepted requests when the threshold is exceeded. However, web server workloads change frequently, for example with the popularity of documents or services. If one chooses low thresholds, it is possible to guarantee low response times since no server resources are fully utilized. On the other hand, choosing low thresholds also leads to lower server throughput and thus loss of potential revenue. If one chooses high thresholds, it is possible to achieve higher utilization and throughput, but there is a risk of overload and high response times. Hence, in order to maximize throughput while keeping response times low, adaptation of the threshold values to the current workload can be advantageous.

Depending on the current workload, some server resources can be overutilized, while the demand on other resources is not very high because certain types of requests utilize one resource more than others. Paper D describes an architecture that sets the maximum number of requests admitted per time unit dynamically based on the current server resource utilization in combination with acquired knowledge about resource consumption of requests.

An overload protection architecture also needs to deal with persistent connections. Persistent connections allow clients to send several requests on the same TCP connection to reduce client latency and server overhead [41]. Persistent connections represent a challenging problem for web server admission control, since the HTTP header of the first request does not reveal any information about the resource consumption of the requests that may follow on the same connection. This problem is addressed in Paper C.

### 1.2.2 Operating System Support for Predictable Service

The goal of Quality of Service (QoS) is to provide *predictable* service to users or applications independent of the demand of other applications competing for the same resources. In order to provide QoS guarantees to applications, it is necessary that all resources used by, or on behalf of, an application are accounted for correctly. Without proper resource accounting, resources cannot be provided

to applications in a predictable way because one application may exceed its share which prevents other applications from receiving their shares. Traditional operating systems have problems with correct accounting of resources. For example, in UNIX systems the CPU time spent in the context of a network device interrupt, triggered by an arriving packet, is accounted to the interrupted application instead of the application the packet is destined for [23]. In microkernel environments work performed by shared servers is often not accounted to the right application [37]. This coarse-grained resource control is inappropriate for multimedia applications that are sensitive to variations of the delay. These applications need more fine-grained resource controls to, for example, avoid flicker in video displays.

One approach to providing fine-grained QoS guarantees is to design and build operating systems from scratch with the goal of fine-grained QoS in mind. Nemesis is such an operating system [37]. In Nemesis, applications can reserve CPU time and bandwidth on network interfaces. Paper A shows how the Nemesis operating system can provide applications a guaranteed communication service by scheduling CPU time and transmit bandwidth. A guaranteed communication service enables the transmission of data at a specified rate, provided that the bandwidth is not limited by the network.

One of the problems with operating systems developed from scratch is that their distribution is often limited. People do not want to invest a lot of time to get acquainted with and to learn a new system. To be able to fully exploit the features of new operating systems, applications must often be modified which people are hesitant to do. An alternative approach is to change the internals of an existing operating system while maintaining the user API (application programming interface). Paper E presents SILK (Scout in the Linux Kernel) which is a port of the Scout operating system [43] to run as a kernel module in the popular Linux operating system. SILK is a modular, configurable, communication-oriented operating system developed from scratch for small network appliances. By running in the Linux kernel, SILK can take advantage of existing Linux applications with small or no modifications at all.

### 1.2.3 Combining QoS and Admission Control Architectures

The QoS architectures described in the previous section are able to provide fine-grained QoS guarantees even during server overload by controlling access to server resources. However, low priority requests that have entered such a system and consumed resources, might be starved or must be preempted when high priority requests enter the system and consume the available resources. Combining the QoS architectures with the admission control architectures described in Section 1.2.1 avoids this problem by not admitting such low priority requests when the server is becoming overloaded. On the other hand, the admission control architectures are much more lightweight, meaning they only require small changes or rather additions to existing operating systems. Hence, when the architectural goal is merely to protect important customers from the

consequences of server overload, these admission control architectures are more appropriate. Comparing these two types of architectures with the service classes for quality of service in IP internetworks, the admission control architectures are comparable to the Controlled-Load Service [53], while the QoS architectures are comparable to the Guaranteed Quality of Service [50].

## 1.3 Method

The research method for the work presented in this thesis is mainly experimental. Experimental research often starts with either a potential or concrete problem.

The first step towards solving the problem is to find and formulate a hypothesis, i.e., an idea or statement that can be validated or invalidated. As an example, a hypothesis in Paper B is formulated as: "Kernel mechanisms for overload protection of web servers are more efficient than mechanisms implemented in user space".

The next step is to design experiments that validate or invalidate the hypothesis. In the case of my work, this phase also includes the design and implementation of a prototype such as admission control mechanisms in an existing operating system. Having a prototype that is complete enough, experiments need to be designed and conducted. Here, the experiments have been conducted in isolated, controlled environments. The advantage of conducting experiments in an isolated network is the possibility to obtain consistent and repeatable results. The disadvantage is that disturbances that may occur in real-world scenarios or in a complete system may not appear in a controlled testbed and, thus, not be taken into account properly.

When not working with real users generating requests, the choice of the workload and the workload generator is very important. The workload should both be realistic, i.e., it should conform with empirical measurements or use representative values such as typical requested file sizes from web servers, and at the same time give the desired effects. For example, many request generators use simple methods that cannot generate requests at a rate that exceeds the capacity of the web server and, thus, fail to evaluate web server behaviour during overload [8].

In the third step, the results of the experiments need to be collected and analyzed to see if they are conclusive and whether they validate the hypothesis or not. If the hypothesis can neither be validated nor invalidated, the experiment has to be redesigned.

The described process is iterative in the sense that the validation of a hypothesis often leads to a more fine-grained or completely new hypothesis. In particular, unexpected behaviour discovered in the experiments often leads to new insights and the formulation of new hypotheses.

## 1.4 Results and Scientific Contributions

The scientific contributions presented in this thesis are:

- Design of, and evaluation of, efficient in-kernel mechanisms for service differentiation and overload protection of web servers, and demonstrating the improved efficiency and scalability of the in-kernel mechanisms compared to the same mechanisms implemented in user space.
- A demonstration of the problem persistent connections cause for web server admission control and a kernel-based architecture that solves the problem. The architecture provides service differentiation judging the importance of persistent connections based on cookies.
- An adaptive admission control architecture that supervises multiple resource bottlenecks in server systems. The architecture uses TCP SYN policing and HTTP header-based connection control in a combined way to perform efficient and yet informed web server admission control.
- Demonstrating that the TCP/IP implementation in Nemesis can utilize the scheduling of CPU time and transmit bandwidth to provide applications with a guaranteed communication service.
- A new networking subsystem for Linux based on the Scout path architecture that is QoS-capable. The idea and evaluation of the concept of extended paths, which enables coscheduling of application and network processing.

Additional results in the form of prototypes that have been an outcome during the course of my work are:

- An IP version 4 implementation for Nemesis, including the transport protocols UDP and TCP as well as end host support for RSVP.
- An implementation of IP version 4 on top of ATM, running under Nemesis.
- Traffic control schemes to provide IP QoS to Nemesis applications.
- Prototype implementations of the proposed architectures including enhanced versions of the *sclient* [8] traffic generator.
- The IBM Linux Technology Center has ported one of the proposed mechanisms for service differentiation in web servers to Linux and distributes it as a Linux patch.

# 2 Summary of the Papers

## 2.1 Paper A

### Scheduling TCP in the Nemesis Operating System

This paper was written within the context of the EU Esprit Pegasus II project. The aim of the project was to explore an operating system design that provides guaranteed quality of service to applications, in particular to multimedia applications. High-quality multimedia applications not only demand a specific amount of resources but also timely, usually periodic, access to resources.

In the Nemesis operating system [37], designed and implemented during the project, applications use shared library code to perform functionality usually provided by the operating system kernel. This feature enables correct accounting of all resources, which is a necessary prerequisite for enforcing and providing guaranteed access to resources. In Nemesis, CPU time, memory, disk I/O bandwidth as well as transmit bandwidth on network interfaces are resources that can be reserved.

The paper reports on the TCP/IP implementation for Nemesis. We study CPU scheduling of TCP/IP, the scheduling of network interface transmit bandwidth, and their interdependence in the context of the Nemesis operating system. We present a set of experiments which demonstrate the ability of Nemesis to provide appropriate end-system communication guarantees for the application. First, we show that the scheduling of transmit bandwidth can both be used as a rate limiter and to provide guaranteed transmit bandwidth. We measure the amount of CPU time an application needs to be able to run the TCP/IP protocol stack and to send data at a particular speed. Our experiments show that the CPU time needed to run the protocol stack increases linearly with the amount of data sent for a given packet size. We also show that the measured values hold, even when several applications strive for CPU time and transmit bandwidth.

The schedulers in Nemesis are primarily designed for networked multimedia applications and periodic access to resources. It is not obvious that TCP, which is designed for reliable data transfer, should work well in such an environment. Nevertheless, we were able to demonstrate that TCP/IP can utilize CPU and transmit bandwidth scheduling of this type to provide applications with a guaranteed communication service, provided that the bandwidth is not limited by the network.

#### Comments

I presented this paper at the sixth International Workshop on Protocols for High-Speed Networks, held in Salem, MA, USA in August 1999.

Most of the architecture design work was done by me, partly with help of Bengt Ahlgren. I designed and conducted the experiments. The paper was

written by me except for parts of the introduction which were written by Bengt Ahlgren.

## 2.2 Paper B

### Kernel Mechanisms for Service Differentiation in Overloaded Web Servers

Web servers need to be protected from overload since overload can lead to high response times, low throughput and even loss of service. Also, it is highly desirable that web servers provide continuous service during overload, at least to preferred customers.

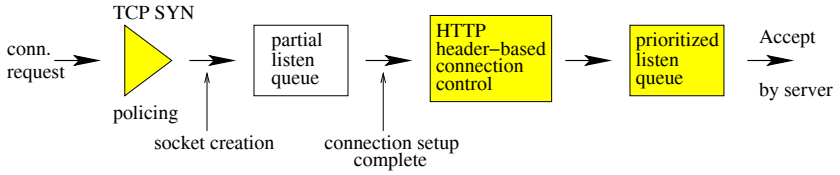


Figure 6: Kernel mechanisms

Most existing web server admission control architectures are implemented in user space. However, performing admission control in user space implies that requests that are later discarded consume a non-negligible amount of resources. Mogul and Ramakrishan have demonstrated the benefit of dropping packets early [42]. Following the principle of "early discard", we have designed and implemented kernel mechanisms that protect web servers against overload by providing admission control and service differentiation based on customer site, client and application layer information. Figure 6 shows the placement of the mechanisms:

- The first mechanism, *TCP SYN policing*, is located at the bottom of the protocol stack. It limits the acceptance of new SYN packets based on compliance with a token bucket policer. A token bucket policer is a token bucket used for admission control. It has a rate, denoting the average number of requests accepted per second, and a burst, denoting the maximum number of requests admitted at one time.

- The next mechanism, *HTTP header-based connection control* is located higher up in the stack. It is activated when the HTTP header is received. It enables admission control and priority based on application-layer information contained in the header, for example, URLs and cookies.

- The third mechanism, *prioritized listen queue*, is located at the end of the TCP 3-way handshake, i.e., when the connection is established. This

mechanism supports different priority levels among established connections by inserting connections into the listen queue according to their priority.

The first mechanism is the least costly but the most coarse-grained. The third is the most costly but provides the most fine-grained admission control.

We have implemented these controls in the AIX 5.0 operating system kernel as a loadable module. We present experimental results to demonstrate that these mechanisms effectively provide selective connection discard and service differentiation in an overloaded server. We also compare the performance of our mechanisms against the same application layer mechanisms added in the Apache 1.3.12 server.

The contribution of this paper is the design and evaluation of the kernel-based mechanisms. In particular, we show that the implementation of the mechanisms in the kernel is much more efficient and scalable compared to user space implementations.

**Comments**

Most of the work described in this paper was done during my internship at IBM TJ Watson Research Center in Spring/Summer 2000. I presented the paper at the Usenix 2001 Annual Technical Conference held in Boston, MA, USA in June 2001.

The prioritized listen queue mechanism presented in the paper has been ported to Linux at the IBM Linux Technology Center and is commercially distributed as a kernel patch. Discussions about integrating the mechanism into the standard Linux kernel are underway.

Most of the architecture design work was done by me together with Ashish Mehra and Renu Tewari. I implemented the mechanisms and designed and conducted the experiments. Renu Tewari and I wrote the paper together. Renu Tewari focused more on the introduction while I did most of the work on the experimental sections.

## 2.3 Paper C

### Kernel-based Control of Persistent Web Server Connections

This paper builds on the work described in Paper B. Paper C extends that work by presenting a solution for handling persistent web server connections. This problem is ignored by most of the web server architectures described in the literature.

Web servers use admission control for overload protection. Some web servers base their admission decision on information found in the HTTP header. Persistent connections allow HTTP clients to send several requests on the same TCP connection to reduce client latency and server overhead [41]. Using the

same TCP connection for several requests makes admission control more difficult, since the admission control decision should be performed when the first request is received. However, the HTTP header of the first request does not reveal any information about the resource consumption of the following requests on the same connection. Thus, persistent connections make admission control a trade-off. If one is too conservative, and sets low acceptance rates, potential customers might be rejected unnecessarily, resulting in loss of revenue. If one is too optimistic the server may become overloaded, with long response times and low throughput as a possible consequence. Our solution avoids uncontrollable overload while maximizing access.

If there is an overload situation caused by resource consumption of persistent connections we abort persistent connections. But we do not abort connections blindly. Instead, we preserve connections regarded as important and abort connections considered less important. For example, a connection can be regarded as important when the client has placed some items in a shopping bag.

The admission control mechanism judges the importance of persistent connections based on cookies in the HTTP header. The web application decides when a cookie denoting the importance of the connection should be sent to the client. Using cookies has several advantages: Cookies are a widely used technique; they can contain long-lasting information such as customer identification; and they are easy to remove or update. We present experiments demonstrating that our approach prevents server overload and provides service differentiation between important and less important connections under high load.

The key contribution of this paper is the kernel-based architecture that prevents overload in web servers caused by persistent connections. To our knowledge, this paper is the first to provide a solution for the challenges of persistent connections.

#### Comments

I presented an extended version of this paper at the workshop PAWS 2001, Performance and Architecture of Web Servers. This workshop was held in conjunction with the ACM SIGMETRICS conference in Boston, MA, June 2001. The version presented in this thesis appeared in the ACM Performance Evaluation Review.

The work described in the paper was done by me with discussions with Per Gunningberg.

## 2.4 Paper D

### Handling Multiple Bottlenecks in Web Servers Using Adaptive Inbound Controls

This paper describes an adaptive admission control architecture that uses mechanisms presented in Paper B. Servers become overloaded when one or several

critical resources, such as network interface, CPU or disk, are overutilized and become the bottleneck of the server system. The key idea of Paper D is to avoid server overload by preventing overutilization of specific server resources using adaptive inbound controls.

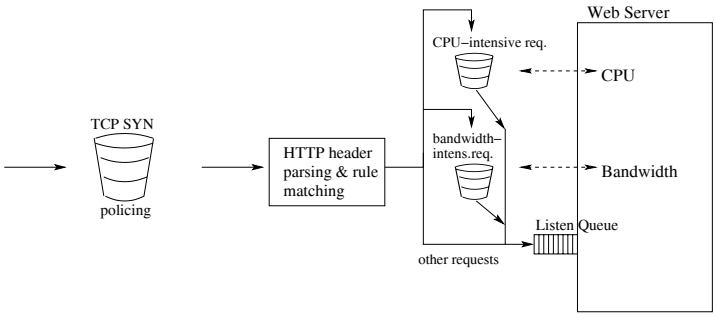


Figure 7: Admission Control Architecture

Our idea is to collect all web objects that, when requested, are the main consumers of the same server resource into one directory. Thus, we have one directory for each supervised resource. We associate a filter rule with each directory in that maps resource-intensive requests to the corresponding token bucket policer. Hence, we can use HTTP header-based connection control to avoid overutilization of specific resources. For example, CPU-intensive scripts can reside in the web server's `/cgi-bin` directory and a filter rule specifying the URL prefix `/cgi-bin` can be associated with it. For each of the critical resources, we use a feedback control loop that adapts the token rate at which we accept requests in order to avoid overutilization of the resource. We call our approach, illustrated in Figure 7, resource-based admission control.

We do not perform resource-based admission control on all requests. Requests such as those for small static files do not put significant load on one resource. However, if requested at a sufficiently high rate, these requests can still cause server overload. When such situations arise we use *TCP SYN policing* applied to all requests, independent of resource requirements. Using SYN policing the admission of connection requests is based on network-level attributes, i.e. on IP addresses and port numbers, and not on fine-grained HTTP header attributes.

We have implemented our admission control architecture in the Linux operating system. Our experiments show that the combination of resource-based admission control and TCP SYN policing works and adapts the rates as expected for our load scenarios. When resource-based admission control alone cannot prevent server overload, TCP SYN policing becomes active and high throughput and low response times can be sustained even when the request demand is high. We achieve more than 40% higher throughput and several magnitudes lower

response times during overload compared to a standard Apache on Linux configuration. We also show that the adaptation mechanisms can cope with bursty request arrival distributions.

The architecture is targeted towards single node servers or to back-end servers in a web server cluster. We believe that the architecture can easily be extended to web server clusters and enhance sophisticated request distribution schemes such as the Harvard Array of Clustered Computers (HACC) [55] and Locality-aware Request Distribution (LARD) [46]. In an extended architecture the front-end performs resource-based admission control. The back-end servers monitor the utilization of each critical resource and propagate the values to the front-end. Based on these values, the front-end updates the rates for the different token bucket policers. After the original distribution scheme has selected the node that is to handle the request, compliance with the corresponding token bucket ensures that critical resources on the back-ends are not overutilized.

The main contribution of this paper is the adaptive admission control architecture that handles multiple resource bottlenecks in server systems.

#### Comments

This paper has been accepted for the seventh International Workshop on Protocols for High-Speed Networks, to be held in Berlin, Germany, in April 2002, where I will present the paper.

The work described in the paper was done by me with discussions with Per Gunningberg.

## 2.5 Paper E

## SILK: Scout Paths in the Linux Kernel

A lot of research effort has been invested into operating system architectures for providing QoS to applications. Some efforts have focused on new QoS features and abstractions to existing operating systems, while others have built new operating systems from scratch. However, the results of these efforts have hardly been put to general use, so far. New QoS mechanisms for mainstream operating systems are often only available for specific versions of the operating system. Due to feature interaction problems between different kernel patches, it is often impossible to combine several of these mechanisms into one system.

Scout [43] is a modular, configurable, communication-oriented operating system tailored for small network appliances. Scout combines features such as early demultiplexing, early dropping, resource accounting, explicit scheduling and extensibility into a single abstraction called a *path*.

SILK is a port of the Scout operating system to run as a downloadable kernel module in a standard Linux 2.4 kernel. SILK can replace the Linux networking subsystem. Regular Linux applications can use SILK which is demonstrated using the popular Apache web server. Introducing the path concept into the Linux

networking subsystem enables prioritizing and other forms of service differentiation between different network connections. Additionally, SILK coordinates the scheduling of applications and paths in the networking stack by *extending paths* into the application.

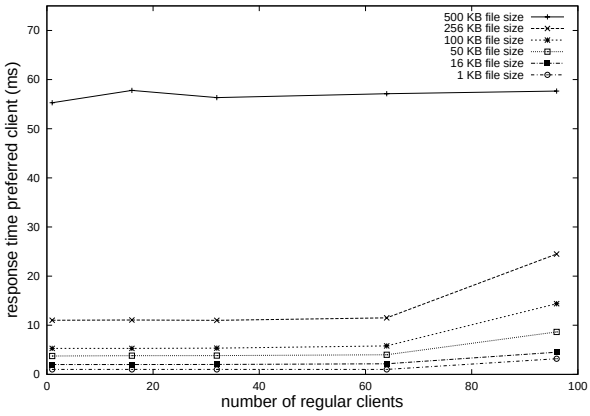


Figure 8: Response time preferred client

Our results show that SILK's performance is still comparable to the native Linux networking stack, i.e., using paths does not lead to performance loss. We also compare latency and throughput for preferred and regular clients using a fixed priority scheduler in SILK. Figure 8 illustrates that SILK provides almost constant response time for preferred clients independent of the number of regular clients accessing the server simultaneously. Without priorities, the response time would increase linearly with the number of clients, since each client receives about $1/n$ of the resources when $n$ clients are active simultaneously.

The contributions of this paper are a networking subsystem for Linux based on the Scout path architecture and the concept of extending paths into the application.

### Comments

This paper has been published as Technical Report 2002-009, Department of Information Technology, Uppsala University, Uppsala, Sweden, February 2002.

Andy Bavier implemented most of SILK. I assisted in debugging the system (in particular its TCP implementation) and conducted most of the evaluation experiments. Andy and I wrote the paper together, my focus was on the experimental section.

# 3 Related Work

This section presents related work. The discussion is divided into the two problem areas presented in Section 1.2.

## 3.1 Overload Protection and Service Differentiation for Web Servers

One of the main design objectives addressed by our admission control architectures is to employ efficient early connection discard mechanisms that provide overload protection and service differentiation for web servers. Many architectures ignore the importance of efficient admission control and presumably reject requests after passing them to user space [2, 19, 20, 33, 38]. Other admission control and service differentiation architectures such as WebQoS [13] and Web2k [14] are deployed in user space. In these architectures admission control is less efficient than in our kernel-based architecture.

A few web server admission control architectures adhere to the principle of early discard. Kant *et al.* moved overload protection into intelligent network cards [30]. While more efficient by off-loading the host, their approach is less flexible since it relies on special hardware. The performance gains are unknown. Jamjoom *et al.* use a mechanism similar to TCP SYN policing to avoid server overload [31]. Their mechanism bases the admission decision on network-level information such as IP addresses and port numbers. Hence, they cannot discriminate between different resource bottlenecks, but have to reduce the acceptance rate for all requests when only one resource is overutilized.

Service differentiation can also be realized by scheduling of server processes and by dynamically partitioning server nodes in a web server cluster. Almeida *et al.* assign different priorities to the processes handling requests [3]. In their approach the application, i.e., the web server, classifies the requests and assigns scheduling priorities. In a similar approach, Eggert and Heidemann propose to lower the priority of processes serving less important requests [24]. They also propose limiting the available bandwidth and the number of server processes for less important requests. While we perform service differentiation before a request is accepted by the web server, the approaches above perform service differentiation when scheduling or processing requests. Combining these approaches would further decrease the impact of low priority requests on the service high priority requests receive.

One approach to provide service differentiation in server clusters is by dynamically partitioning server nodes and forwarding different classes of requests to different partitions [56, 16]. The aim is to dynamically adjust the server partitions, not to perform efficient admission control. A similar approach designed for single server nodes is to reallocate the number of server processes for each service class. Abdelzaher *et al.* implement such an approach [39]. They enforce relative delays among service classes using a feedback control loop to reallocate the number of server processes for each service class. In their work, none of the

critical resources is overutilized. Instead, a peculiar bottleneck introduced by persistent connections causes large delays.

There are also other approaches to deal with server overload such as adapting the delivered content [1].

## 3.2 Operating System Support for Predictable Service

Both Scout and Nemesis are operating systems built from scratch to provide QoS to applications. Designing and implementing operating systems from scratch is a major effort, which is one reason why a lot of novel mechanisms and abstractions for QoS have been implemented in mainstream operating systems instead.

Among these new abstractions are resource containers, virtual services and processor reserves. Resource containers [9] present an abstraction that encompasses all system resources that a server uses to perform an independent activity, such as serving one client connection. All user and kernel processing time and other resource consumption such as memory is accounted to a resource container. Resource containers are used in conjunction with Lazy Receiver Processing (LRP), a network subsystem architecture that includes early demultiplexing and protocol processing at the priority of the receiving application [23]. Cluster reserves extend resource containers to server clusters [6] to provide differentiated and predictable quality of service in clustered web server systems. On one hand, these abstractions require more changes to the operating system than, for example, the admission control architecture presented in Paper D. On the other hand, the abstractions are implemented in an existing operating system and not in an operating system designed from scratch such as Nemesis or Scout.

Reumann *et al.* have presented *virtual services*, an abstraction that provides resource partitioning and management [47]. Virtual services can enhance web server overload protection architectures by dynamically controlling the number of processes a web server is allowed to fork.

Processor reserves [40] are used to provide QoS for multimedia applications in microkernel environments such as the Mach microkernel. Applications can make CPU reservations, which are guaranteed by the system, even in the presence of shared servers. Nemesis avoids this potential problem by using shared libraries instead of shared servers.

Several other researchers have explored ways to provide QoS guarantees to networked applications by controlling bandwidth and CPU. Examples include Yau and Lam's migrating sockets [54], Lakshman *et al.*'s *Adaptive Quality of service Architecture* [35] and Gopalakrishnan and Parulkar's real-time upcalls [28]. QLinux provides fair queuing mechanisms for CPU and network packets as well as an LRP networking subsystem and in addition an advanced disk scheduling algorithm [51]. None of these QoS provisioning methods is able to completely solve the problem of "QoS crosstalk", or in other words, to provide perfect performance isolation.

Rialto [32] is designed and built from scratch to support coexisting independent real-time and non-real-time programs. An abstraction called an *activity* is

the entity to which resources are allocated and charged. Another QoS operating system of this kind is Eclipse. Eclipse's proportional share schedulers and the corresponding API have also been ported to FreeBSD [15]. *Resource kernels* provide applications with explicit guarantees to system resources through abstractions such as CPU Reserves [45]. The portable implementation of a resource kernel implemented in Linux shares some goals with SILK, for example modularity and minimal changes to Linux. In order to increase performance, flexibility and functionality of applications, Exokernels [25] provide applications with a large degree of control over the physical resources, similar to the Nemesis operating system.

# 4 Conclusions and Future Work

In this thesis I present mechanisms and architectures for service differentiation and overload protection in web servers. In order to reduce the resources spent on requests that are eventually discarded, admission control should be performed as early as possible in the lifetime of web transactions. However, the earlier admission control is performed, the less information about the request, its resource requirements and its originator, is available. Deferred admission control enables a more informed control decision. Therefore, many web server admission control schemes are implemented in the web server application. Our research demonstrates that it is both desirable and possible to perform early but yet informed web server admission control.

This research has been conducted under the assumption of a single node web server. Modern web servers are often built as clusters with one or several front-ends. In Paper D we discuss how to extend the presented architecture towards web clusters. The implementation and evaluation of the extended architecture would highlight both the performance gains and potential problems, such as the scalability of the cluster. This is left for future work.

The admission control architectures in this thesis have been implemented as prototypes. They include most, but not all, details required for a production system. It would also be interesting to study the impact of the proposed mechanisms on user perception during overload.

My other contribution to providing service differentiation is the work on SILK and Nemesis. These novel operating system architectures can provide applications with fine-grained QoS guarantees. SILK suggests that it is feasible to adopt main abstractions from an operating system developed from scratch to provide fine-grained QoS guarantees in mainstream operating systems.

The work on SILK is still ongoing and will include work with more advanced schedulers such as Weighted Fair Queueing to provide web server QoS. We also plan to port our adaptive admission control architecture to SILK. Then we could study how to regulate both access to the web server itself and access to the critical server resources.

# References


[1] T. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. In *8th International World Wide Web Conference*, Toronto, Canada, May 1999.

[2] T. Abdelzaher and C. Lu. Modeling and performance control of Internet servers. In *IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[3] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of service in web content hosting. In *Internet Server Performance Workshop*, Madison, WI, USA, March 1999.

[4] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In *Proc. of ACM SIGMETRICS*, pages 126–137, Philadelphia, PA, USA, April 1996.

[5] M. Aron. *Differentiated and Predictable Quality of Service in Web Server Systems*. PhD thesis, Rice University, Houston, TX, USA, October 2000.

[6] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Proc. of ACM SIGMETRICS*, pages 90–101, Santa Clara, CA, USA, June 2000.

[7] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *USENIX Annual Technical Conference*, June 2000.

[8] G. Banga and P. Druschel. Measuring the capacity of a web server. In *USENIX Symposium on Internet Technologies and Systems*, pages 61–71, Monterey, CA, USA, December 1997.

[9] G. Banga, P. Druschel, and J. Mogul. Resource containers: a new facility for resource management in server systems. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 45–58, New Orleans, LA, USA, February 1999.

[10] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. of ACM SIGMETRICS*, pages 151–160, Madison, WI, USA, June 1998.

[11] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - http/1.0. Internet RFC 1945, May 1996.

[12] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into web server design. In *9th International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000.

[13] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 7(4):36–43, September 1999.

[14] P. Bhoj, S. Ramanathan, and S. Singhal. Web2k: Bringing QoS to web servers. Technical Report HPL-2000-61, HP, May 2000.

[15] J. Bruno, J. Brustoloni, E. Gabber, B. Özden, and A. Silberschatz. Retrofitting quality of service into a time-sharing operating system. In *Proc. of Usenix Annual Technical Conference*, pages 15–26, Monterey, CA, USA, June 1999.

[16] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM Performance Evaluation Review*, 29(2):20–25, September 2001.

[17] J. Carlström and R. Rom. Application-aware admission control and scheduling in web servers. In *IEEE Infocom 2002*, New York, NY, USA, June 2002.

[18] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for web clusters providing multiple services. In *10th International World Wide Web Conference*, Hong Kong, China, May 2001.

[19] X. Chen, H. Chen, and P. Mohapatra. An admission control scheme for predictable server response time for web accesses. In *10th International World Wide Web Conference*, Hong Kong, China, May 2001.

[20] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving the performance of an overloaded web server. Technical Report HPL-98-119, HP, 1999.

[21] I. Cooper, I. Melve, and G. Tomlinson. Internet web replication and caching taxonomy. Internet RFC 3040, January 2001.

[22] M. Crovella, R. Frangioso, and M. Harchol-Balter. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems*, pages 243–245, Boulder, CO, USA, October 1999.

[23] P. Druschel and G. Banga. Lazy receiver processing (LRP): a network subsystem architecture for server systems. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 91–105, Seattle, WA, USA, October 1996.

[24] L. Eggert and J. Heidemann. Application-level differentiated services for web servers. *World Wide Web Journal*, 3(2):133–142, September 1999.

[25] D. Engler, F. Kaashoek, and J. O'Toole Jr. Exokernel: an operating system architecture for application-level resource management. In *15th ACM SIGOPS Symposium on Operating Systems Principles, Operating Systems Review*, pages 251–266, Copper Mountain Resort, CO, USA, 1995.

[26] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol - http/1.1. Internet RFC 2616, June 1999.

[27] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-based scalable network services. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 78–91, Saint-Malo, France, October 1997.

[28] R. Gopalakrishnan and G. M. Parulkar. Efficient user space protocol implementations with QoS guarantees using real-time upcalls. *IEEE/ACM Transactions on Networking*, 6(4):374–388, August 1998.

[29] S. Hand. Self-paging in the Nemesis operating system. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 73–86, New Orleans, LA, USA, February 1999.

[30] R. Iyer, V. Tewari, and K. Kant. Overload control mechansims for web servers. In *Performance and QoS of Next Generation Networks*, Nagoya, Japan, November 2000.

[31] H. Jamjoom and J. Reumann. Qguard: Protecting internet servers from overload. Technical Report CSE-TR-427-00, University of Michigan, 2000.

[32] M. B. Jones, J. S. Barrera III, A. Forin, P. J. Leach, D. Rosu, and M. Rosu. An overview of the Rialto real-time architecture. In *ACM SIGOPS European Workshop*, pages 249–256, Connemara, Ireland, September 1996.

[33] V. Kanodia and E. Knightly. Multi-class latency-bounded web servers. In *International Workshop on Quality of Service*, pages 231–239, Pittsburgh, USA, June 2000.

[34] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement.* Addison-Wesley, 2001.

[35] K. Lakshman, R. Yavatkar, and R. Finkel. Integrated CPU and network-I/O QoS management in an endsystem. In *7th International Workshop on Quality of Service*, pages 167–178, New York, NY, USA, April 1997.

[36] C. Lee, K. Yoshida, C. Mercer, and R. Rajkumar. Predictable communication protocol processing in real-time Mach. In *Real-Time Technology and Application Symposium*, pages 115–123, Boston, MA, USA, June 1996.

[37] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297, Sep. 1996.

[38] K. Li and S. Jamin. A measurement-based admission controlled web server. In *IEEE Infocom 2000*, Tel Aviv, Isreal, March 2000.

[39] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Real-Time Technology and Application Symposium*, TaiPei, Taiwan, June 2001.

[40] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Boston, MA, USA, May 1994.

[41] J. C. Mogul. The case for persistent-connection HTTP. In *SIGCOMM '95 Conference Proceedings*, pages 299–313, Cambridge, MA, USA, August 1995. ACM SIGCOMM Computer Communication Review, 25(4).

[42] J. C. Mogul and K. K. Ramakrishan. Eliminating receive livelock in an interrupt-driven kernel. In *Proc. of USENIX Annual Technical Conference*, San Diego, CA, USA, January 1996.

[43] D. Mosberger and L. Peterson. Making paths explicit in the Scout operating system. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 153–168, Seattle, WA, USA, October 1996.

[44] Nua. Nua Internet Surveys. http://www.nua.ie/surveys/analysis/weekly_editorial/archives/issue1no197.html, October 2001.

[45] S. Oikawa and R. Rajkumar. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In *Real-Time Technology and Application Symposium*, Vancouver, Canada, June 1999.

[46] V. Pai, M. Aron, G. Banga, M. Svendsen, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, USA, October 1998.

[47] J. Reumann, A. Mehra, K. Shin, and D. Kandlur. Virtual services: A new abstraction for server consolidation. In *Proc. of USENIX Annual Technical Conference*, San Diego, CA, USA, June 2000.

[48] P. Rodriguez, C. Spanner, and E. Biersack. Web caching architectures: hierarchical and distributed caching. In *4th International Caching Workshop*, San Diego, CA, USA, April 1999.

[49] J. Schiller and P. Gunningberg. Feasibility of a software-based ATM cell-level scheduler with advanced shaping. In *Broadband Communications'98*, Stuttgart, Germany, April 1998.

[50] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. RFC 2212, September 1997.

[51] V. Sundaram, A. Chandra, P. Goyal, P. Shenoy, J. Sahni, and H. Vin. Application performance in the QLinux multimedia operating system. In *Eighth ACM Conference on Multimedia*, pages 127–136, Los Angeles, CA, USA, November 2000.

[52] M. Welsh, D. Culler, and E. Brewer. SEDA: an architecture for well-conditioned, scalable internet services. In *ACM Symposium on Operating Systems Principles*, pages 230–243, Chateau Lake Louise, Banff, Canada, October 2001.

[53] J. Wroclawski. Specification of the controlled load quality of service. RFC 2211, September 1997.

[54] D. K.Y. Yau and S. S. Lam. Migrating sockets - end system support for networking with Quality of Service guarantees. *IEEE/ACM Transactions on Networking*, 6(6):700–716, Dec. 1998.

[55] X. Zhang, M. Barrientos, J. Chen, and M. Seltzer. HACC: An architecture for cluster-based web servers. In *Third Usenix Windows NT Symposium*, pages 155–164, Seattle, WA, July 1999.

[56] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *IEEE Infocom 2001*, Anchorage, AK, USA, April 2001.

# Paper A

Thiemo Voigt and Bengt Ahlgren. Scheduling TCP in the Nemesis Operating System. *IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks*, Salem, MA, USA, August, 1999.

# Paper B

Thiemo Voigt, Renu Tewari, Douglas Freimuth and Ashish Mehra. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. *Proceedings of Usenix Annual Technical Conference*, pages 189 – 202, Boston, MA, USA, June 2001.

# Paper C

Thiemo Voigt and Per Gunningberg. Kernel-based Control of Persistent Web Server Connections. *ACM Performance Evaluation Review*, 29(2):20–25, September 2001.

# Paper D

Thiemo Voigt and Per Gunningberg. Handling Multiple Bottlenecks in Web Servers Using Adaptive Inbound Controls. *Seventh International Workshop on Protocols for High-Speed Networks (PfHSN 2002)*, Berlin, Germany, April 2002.[5]



[5]The version in this thesis has been extended by adding the sections on service differentiation.

# Paper E

Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson, Per Gunningberg. SILK: Scout Paths in the Linux Kernel. Technical Report 2002-009, Department of Information Technology, Uppsala University, Uppsala, Sweden, February 2002.

# Department of Computer Systems

**Dissertation Series**

85/03 Joachim Parrow, *Fairness Properties in Process Algebra*
87/09 Bengt Jonsson, *Compositional Verification of Distributed Systems*
90/21 Parosh A. Abdulla, *Decision Problems in Systolic Circuit Verification*
90/22 Ivan Christoff, *Testing Equivalences for Probabilistic Processes*
91/27 Hans A. Hansson, *Time and Probability in Formal Design of Distributed Systems*
91/31 Peter Sjödin, *From LOTOS Specifications to Distributed Implementations*
93/37 Linda Christoff, *Specification and Verification Methods for Probabilistic Processes*
93/40 Mats Björkman, *Architectures for High Performance Communication*
94/46 Fredrik Orava, *On the Formal Analysis of Telecommunication Protocols*
96/70 Lars Björnfot, *Specification and Implementation of Distributed Real-Time Systems for Embedded Applications*
97/80 Bengt Ahlgren, *Improving Computer Communication Performance by Reducing Memory Bandwidth Consumption*
98/98 Björn Victor, *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*
98/100 Ernst Nordström, *Markov Decision Problems in ATM Traffic Control*
99/101 Paul Pettersson, *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*
99/110 Mats Kindahl, *Verification of Infinite-State Systems: Decision Control and Efficient Algorithms*
00/114 Kristina Lundqvist, *Distributed Computing and Safety Critical Systems in Ada*
00/115 Jan Gustafsson, *Analyzing Execution-Time of Object-Oriented Programs Using Abstract Interpretation*
00/116 Jakob Carlström, *Reinforcement Learning for Admission Control and Routing*
00/117 Mikael Sjödin, *Predictable High-Speed Communications for Distributed Real-Time Systems*
01/118 Björn Knutsson, *Architectures for Application Transparent Proxies: A Study of Network Enhancing Software*
02/119 Thiemo Voigt, *Architectures for Service Differentiation in Overloaded Internet Servers*

# Swedish Institute of Computer Science

### SICS Dissertation Series

01. Bogumil Hausman, *Pruning and Speculative Work in OR-Parallel PROLOG*, 1990
02. Mats Carlsson, *Design and Implementation of an OR Parallel Prolog Engine*, 1990
03. Nabiel A. Elshiewy, *Robust Coordinated Reactive Computing in SANDRA*, 1990
04. Dan Sahlin, *An Automatic Partial Evaluator for Full Prolog*, 1991
05. Hans A. Hansson, *Time and Probability in Formal Design of Distributed Systems*, 1991
06. Peter Sjödin, *From LOTOS Specifications to Distributed Implementations*, 1991
07. Roland Karlsson, *A High Performance OR-parallel PROLOG System*, 1992
08. Erik Hagersten, *Towards Scalable Cache Only Memory Architectures*, 1992
09. Lars-Henrik Eriksson, *Finitary Partial Inductive Definitions and General Logic*, 1993
10. Mats Björkman, *Architectures for High Performance Communication*, 1993
11. Stephen Pink, *Measurement, Implementation and Optimization of Internet Protocols*, 1993
12. Martin Aronsson, *GCLA: The Design, Use, and Implementation of a Program Development System*, 1993
13. Christer Samuelsson, *Fast Natural-Language Parsing Using Explanation-Based Learning*, 1994
14. Sverker Jansson, *AKL—A Multiparadigm Programming Language*, 1994
15. Fredrik Orava, *On the Formal Analysis of Telecommunication Protocols*, 1994
16. Torbjörn Keisu, *Tree Constraints*, 1994
17. Olof Hagsand, *Computer and Communication Support for Interactive Distributed Applications*, 1995
18. Björn Carlsson, *Compiling and Executing Finite Domain Constraints*, 1995
19. Per Kreuger, *Computational Issues in Calculi of Partial Inductive Definitions*, 1995
20. Annika Wærn, *Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction*, 1996
22. Klas Orsvärn, *Knowledge Modelling with Libraries of Task Decomposition Methods*, 1996
23. Kristina Höök, *A Glass Box Approach to Adaptive Hypermedia*, 1996
24. Bengt Ahlgren, *Improving Computer Communication Performance by Reducing Memory Bandwidth Consumption*, 1997
25. Johan Montelius, *Exploiting Fine-grain Parallelism in Concurrent Constraint Languages*, 1997
26. Jussi Karlgren, *Stylistic Experiments in Information Retrieval*, 2000
27. Ashley Saulsbury, *Attacking Latency Bottlenecks in Distributed Shared Memory Systems*, 1999

28. Kristian Simsarian, *Toward Human-Robot Collaboration*, 2000
29. Lars-Åke Fredlund, *A Framework for Reasoning about Erlang Code*, 2001
30. Thiemo Voigt, *Architectures for Service Differentiation in Overloaded Internet Servers*, 2002