

# Effects of Neglecting Buffers in Feed-Forward Design for Web Servers

Martin Ansbjerg Kjær and Anders Robertsson

**Abstract**— This paper addresses a problem related to control of web servers. Due to implementation issues, relevant measurements are only taken over a limit part of the server system, which leaves the controller unaware of what happens in buffers proceeding the actual web-server (for instance, in the TCP/IP layers). We model this and reveal that the unmeasured buffering introduces an extra zero in the relationship between the process control-signal and output. Furthermore, it is shown that a measurement of a disturbance-input actually is state dependent. We also investigate how these new properties affect a traditional feed-forward design. The stability analysis suggests that under certain circumstances, the system becomes unstable due to the feedback mechanism introduced by the designed feed-forward mechanism. We verify the analysis by both simulations and experiments.

## I. INTRODUCTION

Resource management has a long history in computer science, since poorly managed resources can degrade the performance of a computer system severely. In larger computer systems, work balancing is done in order to distribute the need for resources uniformly over a number of resource units (servers, CPUs, memory units, I/O etc.), thus avoiding that some units are overloaded while others are idle [1], [2]. When more resources are requested than available, e.g., a higher request rate than service rate at a web-server systems, the risk of overload occurs and to maintain some (reduced) service, admission control is needed. This has unwanted side-effects, since the price of not satisfying certain request for resources is, in many cases, loss of profit. In order to keep the side-effects as small as possible, a lot of focus has been put into developing methods to maintain a well behaving computer-system (not overloaded) while keeping the side-effects as small as possible.

Server systems have traditionally been designed from a queuing-theory perspective, but in recent years there has been an increasing interest in both academia and industry to apply control-theoretic design-methods for improved performance in e.g., admission control, where new requests are blocked in order to maintain an acceptable queue length (and thereby an acceptable queuing time for the requests) [3], [4], [5].

Server applications are seldom run directly on the hardware of the computer, but several layers of software are implemented below the web-server, often collected in the operating system. The philosophy behind the layering is that as long as the defined interfaces between the layers are respected, the implementation of one layer can be exchanged

without affecting the functionality of the system. Examples of layers in a modern computer is the *IP*-layer and the *TCP*-layer. A whole framework of layering is defined in the *OSI*-model and the *TCP/IP* stack, which both define a number of layers, where each layer only has to know the interface to the layer above and the layer below [6]. One of the methods to keep the layers independent of the behavior of the other layers is to insert buffers. When a layer has something to sent to the layer above, it might put the job in its own out-going buffer, and is then ready to proceed with other tasks. When the layer above receives a job, it might put it in its own in-going buffer until it is ready to handle it. If this is done throughout the whole stack of layers, a request sent to a web server might pass through several buffers of the system before reaching the actual web-server. All this buffering optimizes throughput, since the individual layers do not have to wait for neither the upper nor the lower layer to respond. The drawback is that it is not possible to determine how long it will take for a request to propagate through the whole stack, which may cause unexpected problems for control design.

When trying to control, for instance, the average response time in a server, the buffering will influence the dynamic properties of the whole queuing system. This paper models and analyzes the effect of these underlying buffers and their influence on the controlled server dynamics.

This papers investigates a web-server system with response-time control, as investigated in e.g., [4], [7], [8], [9]. The objective of the control is to maintain a pre-defined average response-time, while minimizing the *CPU*-consumption. The focus of the paper is not the control-design itself, but rather the stability properties of the system when including or excluding the dynamical effects of the buffers described above. For control of e.g., the response time different types of actuation have been considered in the literature; changing the admission probability or the dedicated *CPU*-resources, using *CPU* frequency scaling/dynamic-voltage scaling etc. However, the effect of buffering in the external queues has until now not been considered and analyzed, to the authors knowledge, in the server control setting.

## II. MODELING

Assume that the target problem is to control a web server running on top of a software stack as described above. The number of buffers is unknown, and naturally, it is not possible to measure any quantities (such as buffer length or buffering times) in these buffers.

M. A. Kjær and A. Robertsson are with the Department of Automatic Control, LTH, Lund University, Box 118, SE-221 00 Lund, Sweden {Martin.A.Kjaer, Anders.Robertsson}@control.lth.se

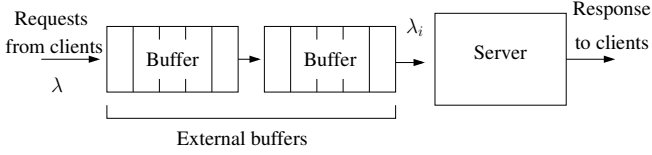


Fig. 1. Requests might pass through several buffers before reaching the server.

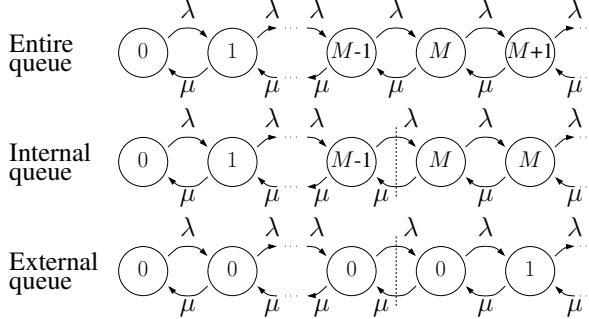


Fig. 2. *Markov* chain representing from top and downward: The entire queuing system and the internal and the external queue, respectively.

It is assumed that all the buffers implemented before the web server are ideal buffers – that is, they do not impose any delay if the proceeding element in the chain is willing to accept the job. The effect of other jobs passing through the same buffers is neglected, assuming that the requests to the web server are not delayed by jobs of other purpose, such as *ftp* and *ssh*. The buffers proceeding the web server (the buffers which are passed when the answer is sent to the client) are also neglected. The simplified view of a web-server system is illustrated Fig. 1.

The web-server is modeled as a single processor-sharing server with a limited number of  $M$  jobs allowed simultaneously. The individual external queues do not hold any independent processors and the external buffers should therefore not be modeled as individual queuing systems, but rather as one long queue. The only processor in the system is that of the web-server, and the dynamics of the entire queuing system is therefore modeled as one infinite queue and a single processor.

#### A. Static Modeling

First, the system is considered to be in steady state, and all dynamics are neglected. For the queuing system to be in a stable mode, we have that the service rate  $\mu$  is larger than the arrival rate  $\lambda$ . The queue is modeled with a *Markov*-chain as illustrated in Fig. 2, and the probability distribution for the number of jobs in the system is given by

$$p_k = \left(1 - \frac{\lambda}{\mu}\right) \left(\frac{\lambda}{\mu}\right)^k \quad (1)$$

assuming an *M/M/1*-queue [10]. The average length of the entire queue  $N$  is then

$$N = \sum_{k=0}^{\infty} p_k k = \left(1 - \frac{\lambda}{\mu}\right) \sum_{k=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^k k = \frac{\frac{\lambda}{\mu}}{1 - \frac{\lambda}{\mu}} \quad (2)$$

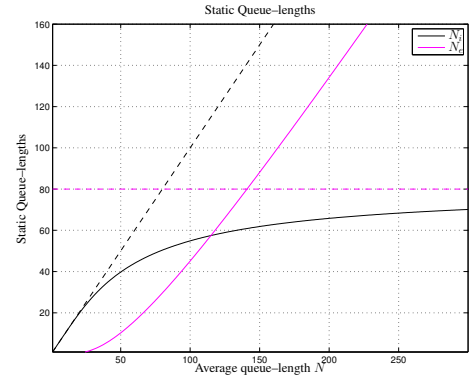


Fig. 3. Static average queue-lengths for  $M=80$ . The horizontal dashed-line represent the upper limit for how many jobs the web server can hold.

which is a well-known result from queuing theory. In order to evaluate the average length of the internal queue  $N_i$  and the external queue  $N_e$ , the *Markov* chain is divided into two parts as illustrated in Fig. 2. The left hand-side of the chain represents the situations where the number of jobs in the entire system does not fill up the web server. Here, the internal queue is treated the same way as when evaluating the entire queue. The external queue does not hold any jobs in this case, and the contribution to the average value is therefore zero. The right hand-side of the chain represent the situation where the web server is full, and queuing in the external queue might occur. Here, the internal queue observes a number of  $M$  jobs, and the external queue observes an increasing number of jobs, starting from zero. The expected values of the internal and external queues therefore become

$$N_i = \sum_{k=0}^{M-1} p_k k + \sum_{k=M}^{\infty} p_k M \quad (3)$$

$$N_e = \sum_{k=0}^{M-1} p_k 0 + \sum_{j=M}^{\infty} p_j (j - M) \quad (4)$$

$$N_i + N_e = \sum_{k=0}^{\infty} p_k k = N \quad (5)$$

by comparison to (2). Elaborating further reveals

$$N_e = N \left(\frac{\lambda}{\mu}\right)^M = N \left(\frac{N}{1+N}\right)^M \quad (6)$$

$$N_i = N - N_e = N \left(1 - \left(\frac{N}{1+N}\right)^M\right) \quad (7)$$

Figure 3 illustrates the internal and external queue-length as functions of the total queue-length, which follow the intuition: The internal queue grows with  $N$ , but does never exceed  $M$ , which is the upper limit for the number of jobs that the web server can hold (the horizontal dashed-line). For low values of  $N$ , the external queue is empty because the web server can hold all the jobs, but as  $N$  becomes higher, the external queue grows. At very high values of  $N$  the external queue holds all the jobs except for the  $M$  jobs which are hold in the web server.

## B. Dynamic Modeling

The queue-length model for the entire queuing system follows *Tipper's model*

$$\dot{x} = \lambda(t) - \mu \frac{x(t)}{1+x(t)} \quad (8)$$

assuming exponentially distributed inter-arrival times and service times [11], [12], [13], [14]. Here, the arriving traffic is that entering the first external queue, which might not be measurable in a real system. Assuming that the service rate  $\mu$  can be changed online, by altering the reserved *CPU*-bandwidth  $p$ , the service rate is described by  $\mu(t) = p(t)/\bar{w}$  where  $\bar{w}$  is the average time that the jobs would require to be served, if only one job were processed at a time.

The response time, from the requests arrive at the first queue to the requests finish at the processor, is modeled according to *Little's law* [10],

$$d(t) = \frac{x(t)}{\lambda(t)} \quad (9)$$

It is assumed that only quantities known inside the web server are measurable.

The internal average number of jobs is denoted  $x_i$ . It is assumed that (7) also holds dynamically, and thus

$$x_i(t) = x(t) \left( 1 - \left( \frac{x(t)}{1+x(t)} \right)^M \right) =: f_i(x(t)) \quad (10)$$

The internal arrival rate  $\lambda_i$  is measurable, simply by counting the number of arriving request over a certain time interval. Consider the external queue, and denote the average number in the external queue by  $x_e$ . Assuming that this queue acts as a simple integrator, where the queue length is given by the integral of the inflow and outflow of requests. This is similar to a conservation law from physics, as *e.g.*, flow of liquid in and out of a container. The model becomes

$$\dot{x}_e = \int_0^t (\lambda(\tau) - \lambda_i(\tau)) d\tau \quad (11)$$

The static relation described in (5) is assumed to hold also in the dynamic case, and by differentiation, the following model is obtained

$$\lambda_i(t) = \lambda(t) - \frac{d}{dt} \{x(t) - x_i(t)\} \quad (12)$$

The internal average response time  $d_i$  is regarded as the time from the arrival to the web server of the requests until the requests have been fully processed. Using *Little's law*, the following model is derived

$$d_i(t) = \frac{x_i(t)}{\lambda_i(t)} \quad (13)$$

## C. Linearization

The result of the linearization of the model is illustrated as a block diagram in Fig. 4 with the parameters

$$\alpha = \frac{p^0}{\bar{w}} \frac{1}{(x^0+1)^2}, \quad \beta = \frac{x^0}{\bar{w}(x^0+1)} \quad (14)$$

$$\gamma_1 = \gamma_{i1} = \frac{1}{\lambda^0}, \quad \gamma_2 = \frac{-x^0}{(\lambda^0)^2}, \quad \gamma_{i2} = \frac{-x_i^0}{(\lambda^0)^2} \quad (15)$$

$$\gamma_f = \frac{x^0+1-x^0 \left( \frac{x^0}{x^0+1} \right)^M - \left( \frac{x_i^0}{x^0+1} \right)^M - M \left( \frac{x^0}{x^0+1} \right)^M}{x^0+1} \quad (16)$$

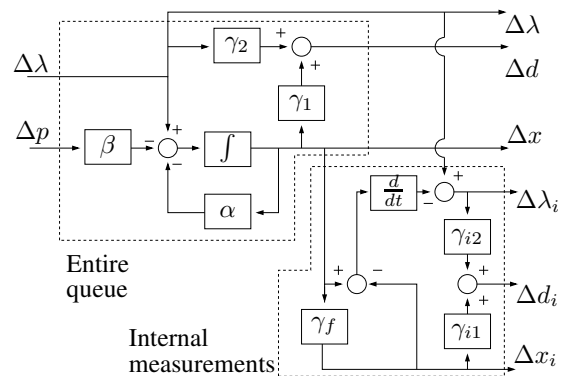


Fig. 4. Block diagram of the server with outputs both from the full queue, and from the web server.

The figure shows that the full queuing system can be represented by a simple first-order system from the control input  $p$  to the output  $d$ . Also, the disturbance  $\lambda$  affects the system in a fairly simple manner. It is observed that if the external queue really can be neglected ( $\gamma_f=1$ ), the measured quantities matches those of the whole queue ( $\lambda_i(t) = \lambda(t)$  and  $d_i(t) = d(t)$ ). The model also reveals that the external queue imposes a zero into the transfer function from the control signal  $p$  to the output  $d_i$ , and that the measurement of the disturbance  $\lambda_i$  now becomes state dependent.

## III. TESTBED AND SYSTEM IDENTIFICATION

### A. Testbed

In order to evaluate the model towards a real system, a testbed was developed. The server computer was a Pentium 4, 1 GB memory, 3 GHz PC, with a Linux Fedora 8 operating system and modified kernel 2.6.25.4. Virtualization was achieved with Control Group functionality [15]. Also, an Apache server, version 2.2.8, configured by using the `prefork` module, was installed on the server computer, [16]. 10 client computers, Athlon, 1.5 GHz PC with 2 GB memory, Linux Fedora 9 and kernel 2.6.26.3-29, generated traffic with the CRIS tool [17], which is a java-program developed on data obtained from real-life applications (from log-files of a news-site). The client computers were connected to the server by an 100 Mb Ethernet switch. A master computer was connected through a local Ethernet network for administration tasks. The Apache web-server was chosen mainly because it is one of the most used web servers on the Internet. A detailed description of the Apache architecture and model structure is found in *e.g.*, [18], [19].

The Linux 2.6 kernel provides functionality to group different processes and perform scheduler-specific operations on group-basis. The project is called Control Group, and is accessed through a virtual file-system [15], [20]. According to measurements, not presented here, the access is done on the scale of 0.2 ms, and the scheduler can be considered as true processor-sharing and without dynamics down to a time resolution of around 100 ms.

The controller must have access to the measured variables, and is implemented in the *Apache* server as a special request,

which enters an infinite loop under the logging phase. For each loop a control action is performed, and the loop sleeps for a specified amount of time, before waking up for a new sample. More details on the testbed are presented in [21].

### B. Parameter Estimation and Model Validation

The nonlinear model was implemented in Simulink<sup>®</sup> for Matlab<sup>®</sup>. The model consists of the full queuing-system ((8) and (9)), the internal measurements ((10), (12), and (13)), along with a first-order filter to filter the arrival rate:

$$\dot{\lambda}_f(t) = -\frac{1}{T_\lambda}\lambda_f(t) + \frac{1}{T_\lambda}\lambda_i(t) \quad (17)$$

where  $T_\lambda$  is the filter constant, chosen to  $T_\lambda = 10$  seconds.

The model has only two parameters; the average time for service of a single job,  $\bar{w}$ , and the limit of the internal queue,  $M$ . The parameter  $\bar{w}$  is usually not set explicitly, so it has to be estimated. The parameter  $M$  is often set explicitly by the system administrator. For example, it is defined by the parameter `MaxClients` in *Apache*. Therefore, only one parameter must be estimated.

The model specifically describes the influence of the external queue, so for the parameter estimation and for the validation it is important that the experiments conducted excite the external queue. This is ensured by limiting the internal queue ( $M=80$ ), forcing the external queue to hold more jobs. In practice, this short internal queue is not realistic, but it is chosen to demonstrate the affect of the external queue, which could become significant, even under normal configuration, but under high load.

The estimation of  $\bar{w}$  is done by steady-state measurements. It is not possible to measure the response time from the entrance of the server, but it is possible to measure the response time at the clients in steady-state. This measurement includes the network-delay, which is not part of the model. 970 packages were sent by the `ping`-program under Linux, which recorded an average network round-trip-time of 0.266 seconds. It is assumed that the response time experienced at the clients is the average round-trip-time added to the response time of the web-server. The average round-trip-time is used as offset to the response-times of the full queue, predicted by the model.

In order to utilize the external queue, the web-server is stressed by operating close to the stability limit (in the queuing-theoretical sense). Several steady-state experiments with arrival rate of 51.3 requests/s were conducted with different CPU bandwidth. Transients were removed, leaving approximately 2000 measurements for averaging for each experiment. The sampling interval was 1 second.

Figure 5 shows the results of the experiments along with the theoretical values obtained with the value  $\hat{w}=0.00504$ , which was considered as the best choice. The steady-state model is sensitive to the choice of  $\hat{w}$  at loads close to the stability line, but this is a problem inherited from queuing theory. The parameter also enters the dynamic properties of the system, but here the sensitivity is not as profound. The priority of the parameter estimate was to match the model to

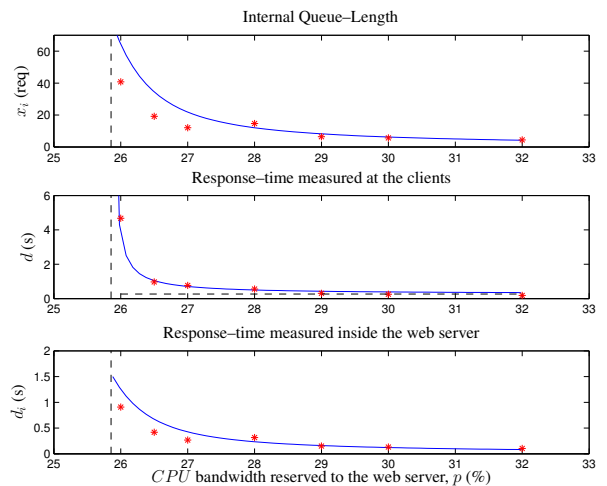


Fig. 5. Validation of the model versus experimental data. The full line represent the model, and the stars represent steady-state experiment data of approximately 2000 seconds (2000 measurements). The vertical dashed lines represent the theoretical limit for queuing stability (where  $\lambda=\mu$ ). The horizontally dashed line in the middle-figure represent the estimated network round-trip-time.

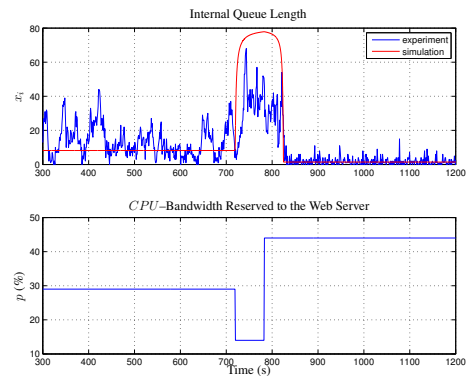


Fig. 6. Validation of the model versus experimental data. The lower part of the figure shows the control signal  $p$ , and the top of the figure shows the behavior of the internal queue-length.

the experiment data for the entire queue, since this model is well established within the queuing-theory community (see the middle of Fig. 5). The lower part of the same figure shows the response times measured inside the web-server. The match between the model and the measurements are not as accurate as in the middle of the figure, but still acceptable. The queue-length of the entire queue is not measurable, so comparison is only conducted for the internal queue, illustrated in the top of Fig. 5. Some deviation is observed, but the model follows the experimental results quite well.

The dynamical model is validated against experiment data. In order to show the dynamic influence of the external queue, the control signal illustrated in the bottom of Fig. 6 was applied in an experiment. The arrival rate was kept constant at 51.3 requests/second throughout the entire experiment. The results of the experiment are shown in Figs. 6 and 7 along with the corresponding simulation-results. The top of Fig. 6 shows the behavior of the internal queue. When the

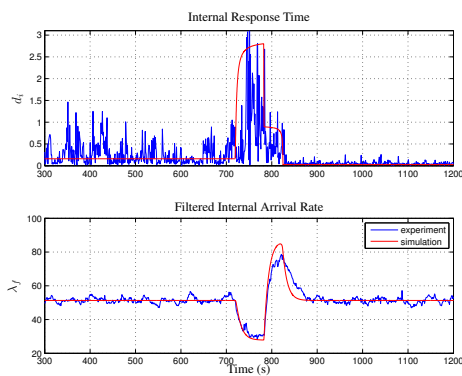


Fig. 7. Validation of the model versus experimental data. The upper part of the figure shows the internal response–time, and the lower part of the figure shows the filtered internal arrival–rate.

control signal is decreased and the server is overloaded, the queue builds up, but because it is limited to 80, it does not grow that much. The top of Fig. 7 shows the effect of the queue building up; the response times become larger, but still, the full effect is not seen, since the response times over the full queue cannot be measured. The bottom of the figure shows some interesting affects of the external queue. When the control signal is decreased, the service rate becomes smaller than the arrival rate. Since the internal buffer is often full, the rate of the jobs entering the web–server is limited by the service rate, and therefore, the measured internal arrival–rate decreases. When the control signal is increased, the flow into the web server will still be limited by the service rate, until the external buffer has emptied. At this time, the service rate is around 87 req/s, which is clearly observed as the top peak in the bottom of Fig. 7. When the queue has settled at the new operating point, the arrival rate measured inside the web–server matches that of the offered. It is worth to remember that arrival rate entering the full queuing system remains constant under the whole experiment.

The match between the model and the measurements is acceptable. The problems of matching the right levels of response times and internal queue–length observed in the steady–state experiments remains, but the dynamics of the system is matched quite well by the model. Especially the filtered internal arrival–rate is matched well.

#### IV. CONTROL DESIGN NEGLECTING THE EXTERNAL QUEUE

If the system is operated in medium or light load, the external queue can be neglected, simplifying the control design significantly. The question is then how the control system behaves when operating under higher loads, where the external queue becomes significant, but where the control system does not encounter the effect of the extra queuing. That is the scope of the following.

A control–strategy based on feed–forward from the disturbance  $\lambda$  is considered, where it is assumed that the external queue can be neglected. The response time is chosen as main metric, since it is important from a client–perspective.

An analysis then follows, where the control structure and parameters remains the same, but the system now includes the external queue. The system is modeled by a first–order system with two inputs; one input for the control action  $p$ , and one input where the ”disturbance”  $\lambda$  enters the system.

Several recent publications propose to measure the disturbance  $\lambda$ , and to utilize it for feed–forward compensation/control, and thereby reduce the undesired effects of changes in the work load. The term *feed–forward* is used to stress that the only measurement used for this controller is an input—no outputs are used. Later, the measurement used for the feed–forward controller will be exchanged with an internal measurement of the arrival rate which is an output of the process. The feed–forward mechanism will then actually become a feedback mechanism, but it will still be denoted as *feed–forward* to state that it was originally intended as a feed–forward mechanism.

Because the measurement of the disturbance can be quite noisy, it is filtered with the first–order low–pass filter from (17). The feed–forward controller is based on the queuing–theoretical relationships from (2) and (9) which are rewritten to a function of the desired response time  $d_r$  and the filtered arrival rate  $\lambda_f$ .

$$p(t) = \frac{\hat{w}}{d_r} + \hat{w} \lambda_f(t) \quad (18)$$

This feed–forward signal is a proportional controller with proportional gain  $\hat{w}$  and bias a term  $\hat{w}/d_r$ .

It is noted that the bias–term is proportional to  $\hat{w}$ , which makes it difficult to evaluate the performance with different values of  $\hat{w}$ , since the operation point changes with  $\hat{w}$ . Therefore the feed–forward controller is rewritten into

$$p(t) = p^0 + \hat{w} (\lambda_f(t) - \lambda^0) \quad (19)$$

Linearization and *Laplace*–transformation of the feed–forward controller gives

$$P(s) = \frac{\hat{w}}{T_\lambda s + 1} \Lambda(s) \quad (20)$$

#### V. STABILITY ANALYSIS INCLUDING THE EXTERNAL QUEUE

Consider the case where the control design is conducted as described above (assuming that the external queue can be neglected), but the external queue is now significant. The analysis then follows that of the above, except that  $\lambda$  and  $d$  are exchanged with  $\lambda_i$  and  $d_i$ . This has the consequence that an extra zero is included in the server model, and the arrival rate  $\lambda$ , which before was treated as a disturbance, now becomes a dynamical state ( $\lambda_i$ ), which introduces an extra feedback loop, as illustrated in Fig. 8.

The expressions for stability analysis becomes too complicated to draw any general conclusions if no parameters are fixed. Therefore, it is chosen to evaluate the stability for testbed–system. The parameters and operating point is chosen according to the following arguments.

**The model parameters** are chosen according to the system identified in Section III–B:  $M=80$  and  $\hat{w}=0.00504$  sec.

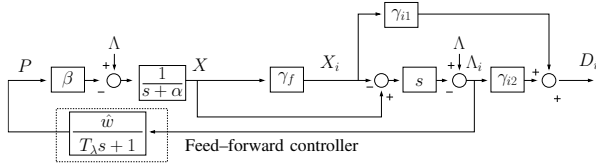


Fig. 8. Block diagram of the server with feed-forward control.

TABLE I  
OPERATING POINTS

| Variable | $p^0$ | $\lambda^0$ | $\mu^0$  | $x^0$ | $x_i^0$ | $d^0$ | $d_i^0$ |
|----------|-------|-------------|----------|-------|---------|-------|---------|
| Unit     | %     | req./sec    | req./sec | req.  | req.    | sec.  | sec.    |
| Value    | 26.16 | 51.30       | 51.91    | 83.56 | 51.3    | 1.63  | 1.0     |

**The operation point** is chosen as acceptable response time for the end user. Since the full response time is not measured, it is chosen as reference for the internal response-time. That is,  $d_i^0=1.0$  sec. Also, a constant arrival rate of  $\lambda^0=51.3$  req/s is chosen, since this was the arrival rate used for the parameter estimation for of the model. The remaining operating-point parameters are listed in Table I. The chosen operating point represents a highly loaded situation.

The feed-forward controller introduces a feedback from the number of jobs in the system, which alters the stability properties of the system. The denominator of the system is then

$$\frac{D_i(s)}{\Lambda(s)} = \frac{\dots}{T_\lambda s^2 + (1 - \beta \hat{w}(1 - \gamma_f) + \alpha T_\lambda) s + \alpha} \quad (21)$$

which indicates that the system becomes unstable for

$$\hat{w} > \frac{\alpha T_\lambda + 1}{\beta(1 - \gamma_f)}. \quad (22)$$

The test-bed system is, according to the parameters chosen in the beginning of this chapter, unstable for  $\hat{w} > 7.281$  ms.

Several nonlinearities are present in the server system. The nonlinear model prevents the queue from holding negative jobs ( $x < 0$ ) but this is not captured by the linear model. The second nonlinearity is an actuator saturation, since CPU bandwidth is limited from both below and above. These nonlinearities are not relevant at the investigated operating-points, and therefore are not treated any further. The third nonlinearity is the static function of (10), relating the internal queue-length to the full queue-length, which holds for  $x \geq 0$ . In the case where  $x < 0$ , the function must be limited from below, such that  $x_i \geq 0$ . Furthermore, the function is altered to operate on the linearized variables:

$$\tilde{f}_i(\Delta x) := \begin{cases} f_i(\Delta x + x^0) - x_i^0 & \text{for } \Delta x \geq -x^0 \\ -x_i^0 & \text{for } \Delta x < -x^0 \end{cases} \quad (23)$$

Instability can expose itself in two different ways; either the system goes to an extreme configuration, determined by the the limitations of the system (such as saturated actuator, saturated states, etc.), and remains there until external events (such as user interaction) occur, or the system starts to oscillate. Methods to predict the type of instability exist,

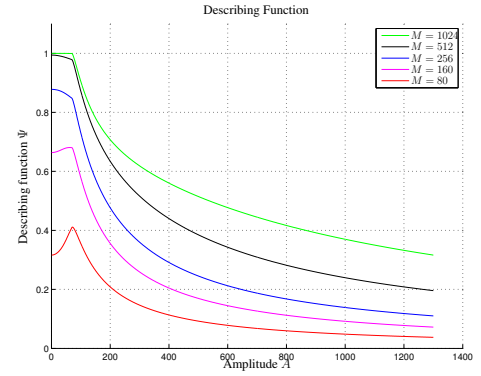


Fig. 9. Describing function for  $x^0 = 70$ , for different values of  $M$ .

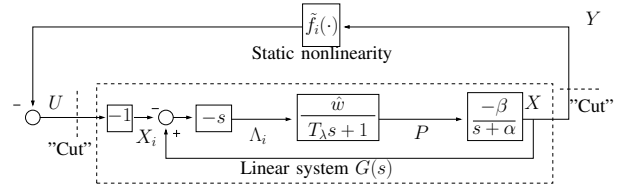


Fig. 10. Block diagram of the server with feed-forward control. The system is divided into a linear part and a nonlinear part, interacting through the signals  $U(s)$  and  $Y(s)$ .

and one such method is the describing-function method. The notation presented here follows that of [22], but also [23] gives a comprehensive treatment.

The describing function  $\Psi$  is not evaluated analytically due to the complicated integrals, but is computed numerically and presented in Fig. 9 for different values of  $M$ .

“Cuts” are made in the block diagram on each side of the nonlinearity, and the two variables  $U(s)$  and  $Y(s)$  are inserted at the places of “cutting”. The whole block diagram is rearranged so these variables becomes input and output of a linear system  $G(s)$ , as shown in Fig. 10.

The transfer function from  $U(s)$  to  $Y(s)$  is now

$$G(s) = \frac{\beta \hat{w} s}{T_\lambda s^2 + (1 - \beta \hat{w} + T_\lambda \alpha) s + \alpha} \quad (24)$$

which has a zero in the origin and two poles, which mean that the phase of  $G(\omega i)$  will be in the interval  $] -90^\circ \ 90^\circ [$ . Since the negative inverse describing-function is real and strictly negative, it will never intersect with the the *Nyquist*-plot, and thus, the describing-function method cannot predict any limit-cycle. The conclusion is therefore that because the system is unstable (for  $\hat{w} > 7.281$ ) and without limit cycles, the system will go towards some extreme.

## VI. VALIDATION

### A. Validation by Simulations

The simulation-model developed for validation was expanded to include the feed-forward controller. A saturation was imposed to limit the control signal  $p$  between one and zero. The focus here is the stability, so all simulations were initiated at the operation point, but a small deviation in the

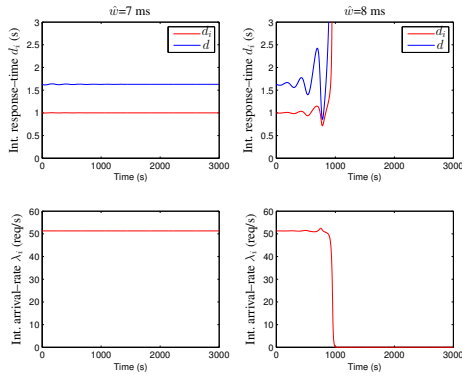


Fig. 11. Simulation results showing the internal measured response-time  $d_i$  and the internally measured arrival rate when utilizing feed-forward control. The estimate  $\hat{w}$  used in the feed-forward controller was 7 ms in the simulation results presented in the left hand-side, and 8 ms in the simulation results presented in the right hand-side.

control signal in the beginning of the simulation ensured that instability revealed itself. The operating point was as described in in Table I.

Fig. 11 illustrates the results of two simulations. The figures indicate that the system remained stable and behaved well for low values of  $\hat{w}$ , but became unstable for higher values. The stability analysis suggested that the limit for stability is 7.3 ms, which corresponds well with the presented simulation-results. Also, the simulation results indicate that the system does not enter stable limit cycles in the unstable operating point, but goes to an extreme saturation, where the control signal goes to the lower limitation, and the response times goes to infinite (since the queue grows to infinite). This matches the conclusion from the describing-function analysis.

### B. Experimental Validation

The controllers were implemented on the testbed described in Section III-A. The condition of all the experiments were as described in Table I. A saturation was imposed to limit the control signal  $p$  between 0.01 and 0.89. (10% of the CPU bandwidth was reserved to the operating system and other tasks, and the web server was guaranteed at least 1% of the CPU bandwidth). Figure 12 illustrates the results of three experiments. Note that the feed-forward controller was first included after 300 s in the experiment with  $\hat{w}=8$  ms (the rightmost part of the figure). The initial conditions for the experiments were not the same, so only the steady-state conditions can be compared. For the case of  $\hat{w}=7$  ms, the system was stable for a while, but after around 2500 seconds, the system suddenly became unstable. This suggests that the stability limit for the actual system is around 7 ms, which corresponds well with the theoretical analysis, which predicted the stability limit to 7.3 ms.

## VII. CONCLUSIONS

This paper has considered the problem where several buffers, denoted the external queue, was inserted between

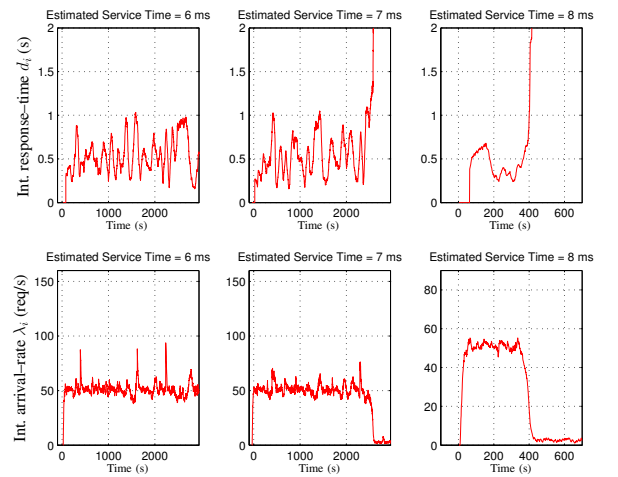


Fig. 12. Experimentally results showing the internal measured response-time  $d_i$  and arrival-rate  $\lambda_i$  when utilizing feed-forward control. The estimate  $\hat{w}$  used in the feed-forward controller is varied between 6 ms and 8 ms.

the client and the web server, and the parameters of the external queue were unmeasurable. The paper investigated the dynamic effects of these extra buffers, when measurements were taken inside the web server, and the investigations reveled several interesting properties:

- The (internal) measurement of the arrival rate becomes state dependent.
- The (internal) measurement of the response times now incorporates a zero, which was not present in the model of the response time for the entire queue.

This alter the properties of a controlled system significantly, so neglecting the external queue in the control design can become devastating if operating under high loads.

The model has only two parameters; one explicitly set by the system administrator ( $M$ ) and one has to be estimated ( $\bar{w}$ ). The model has been verified by comparing simulated behavior to experimental data, and showed sound correspondence. The levels of some of the variables were sensitive to the estimated parameter when the system was operated close to overload. This is a normal situation within queuing systems. The dynamic properties of the model was not as sensitive to the parameter variation.

The paper demonstrated the danger of neglecting the external queue during the control design. The investigations showed that using a feed-forward strategy can lead to instability if the gain of the feed-forward controller is chosen too high. The instability was clearly observed as growing response-times and the control signal going towards zero. The stability conclusions were supported by theoretically analysis, simulations, and experiments, which all showed correspondence in the qualitative conclusions.

## VIII. ACKNOWLEDGMENTS

This work has been funded by the Swedish Research Council, under project 621-2006-5522 and the Lund Center for Control of Complex Engineering Systems (LCCC).

## REFERENCES

- [1] Y. Diao, C. Wu, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco, "Comparative studies of load balancing with control and optimization techniques," in *American Control Conference, 2005. Proc. of the 2005*, Portland, Oregon, June 2005, pp. 1484–1490.
- [2] Y. Fu, H. Wang, C. Lu, and R. Chandra, "Distributed utilization control for real-time clusters with load balancing," in *IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006, pp. 137–146.
- [3] X. Chen, H. Chen, and P. Mohapatra, "ACES: An efficient admission control scheme for QoS-aware web servers," *Computer Communications*, vol. 26, no. 14, pp. 1581–1593, 2003.
- [4] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web applications using queuing predictor," in *Proc. 10<sup>th</sup> IEEE/IFIP Network Operation & Management Symp.*, Vancouver, Canada, April 2006.
- [5] M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark, "Control-theoretic analysis of admission control mechanisms for web server systems," *The World Wide Web Journal, Springer*, vol. 11, no. 1-2008, pp. 93–116, Aug. 2007, online Aug 2007, print March 2008 (DOI 10.1007/s11280-007-0030-0).
- [6] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1996.
- [7] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queuing-theoretic prediction for relative delay guarantees in web servers," in *Proc. 9th IEEE Real-Time and Embedded Technology and Application Symp. (RTAS'03)*, Toronto, Canada, May 2003.
- [8] D. Henriksson, Y. Lu, and T. Abdelzaher, "Improved prediction for web server delay control," in *Proc. 16th Euromicro Conf. on Real-Time Systems (ECRTS'04)*, Catania, Italy, June 2004.
- [9] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal, "AutoParam: Automated control of application-level performance in virtualized server environments," in *Proc. Second IEEE Int. Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'07)*, Munich, Germany, 2007, pp. 2–7.
- [10] L. Kleinrock, *Queueing Systems*. New York: John Wiley & Sons, Inc, 1975.
- [11] C. Agnew, "Dynamic modeling and control of congestion-prone systems," *Operations research*, vol. 24, no. 3, pp. 400–419, 1976.
- [12] D. Tipper and M. K. Sundareshan, "Numerical methods for modeling computer networks under nonstationary conditions," *Selected Areas in Communications, IEEE Journal on*, 1990.
- [13] S. Sharma and D. Tipper, "Approximate models for the study of non-stationary queues and their applications to communication networks," in *Proc. IEEE Int. conf. Communications*, vol. 1, Geneva, may 1993, pp. 352–358.
- [14] W. Wang, D. Tipper, and S. Banerjee, "A simple approximation for modeling nonstationary queues," in *proc. IEEE INFOCOM '96*, San Francisco, CA, March 1996, pp. 255–262.
- [15] Linux Headquarters, "What are CGroups ?" <http://www.linuxhq.com/kernel/v2.6/25/Documentation/cgroups.txt>, November 2008.
- [16] Apache Software Foundation, <http://www.apache.org>, November 2008.
- [17] A. Hagsten and F. Neis, "Crisis request generator for internet servers," Master's thesis, LTH, Lund University, 2006.
- [18] B. Laurie and P. Laurie, *Apache: The Definitive Guide*, 3rd ed. O'Reilly, December 2002.
- [19] Apache Software Foundation, "Developer documentation for Apache 2.0," <http://httpd.apache.org/docs/2.0/developer/>, November 2008.
- [20] Linux Headquarters, "This is the CFS scheduler," <http://www.linuxhq.com/kernel/v2.6/25/Documentation/sched-design-CFS.txt>, November 2008.
- [21] M. A. Kjær, M. Kihl, and A. Robertsson, "Resource allocation and disturbance rejection in web servers using slas and virtualized servers," *Network and Service Management, IEEE Trans. on*, 2009, submitted.
- [22] J.-J. Slotine and W. Li, *Applied Nonlinear Control*. Prentice-Hall, 1991.
- [23] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.