

Using Control Theory to Achieve Service Level Objectives In Performance Management

Sujay Parekh
IBM
sujay@us.ibm.com

Neha Gandhi
University of Michigan
gandhin@engin.umich.edu

Joe Hellerstein
IBM
hellers@us.ibm.com

Dawn Tilbury
University of Michigan
tilbury@umich.edu

T. S. Jayram
IBM
jayram@us.ibm.com

Joe Bigus
IBM
bigus@us.ibm.com

October 23, 2000

Abstract

A widely used approach to achieving service level objectives for a target system (e.g., an email server) is to add a controller that manipulates the target system's tuning parameters. We describe a methodology for designing such controllers for software systems that builds on classical control theory. The classical approach proceeds in two steps: system identification and controller design. In system identification, we construct mathematical models of the target system. Traditionally, this has been based on a first-principles approach, using detailed knowledge of the target system. Such models can be difficult to build, and too complex to validate, use, and maintain. In our methodology, a statistical (ARMA) model is fit to historical measurements of the target being controlled. These models are easier to obtain and use and allow us to apply control-theoretic design techniques to a larger class of systems. When applied to a Lotus Notes groupware server, we obtain model fits with R^2 no lower than 75% and as high as 98%.

In controller design, an analysis of the models leads to a controller that will achieve the service level objectives. We report on an analysis of a closed-loop system using an integral control law with Lotus Notes as the target. The objective is to maintain a reference queue length. Using root-locus analysis from control theory, we are able to predict the occurrence (or absence) of controller-induced oscillations in the system's response. Such oscillations are undesirable since they increase variability, thereby resulting in a failure to meet the service level objective. We implement this controller for a real Lotus Notes system, and observe a remarkable

correspondence between the behavior of the real system and the predictions of the analysis. This allows us to select the proper parameter for the controller from the analysis alone.

1 Introduction

Wide-spread reliance on IT systems has focused increasing attention on service level management, especially achieving response time and throughput objectives. A commonly used approach is to take an existing **target system** and add a controller that has access to the metrics and tuning parameters of the system. Based on the feedback information present in the metrics, the controller manipulates the tuning parameters to achieve the desired service level objectives. Examples of such **closed-loop** software systems abound: the network dispatcher [9, 8], which adjusts load balancing parameters in clusters of web servers; the Multiple Virtual Storage (MVS) workload manager [1], which adjusts memory allocations and other operating system tuning parameters to achieve response time and throughput objectives; and fair share schedulers [6], which adjust Unix *nice* tuning parameter to achieve fractional allocations of CPU.

While considerable attention has been focused on the software mechanisms needed to enable closed-loop systems (e.g., instrumentation and tuning control access), much less attention has been paid to a rigorous evaluation of the behavior of the controller. Computer scientists most frequently use simulations to understand and evaluate controller behavior. However, simulation studies can be time-consuming, expensive, and error prone.

The design of closed-loop or feedback control systems is also studied extensively in other engineering disciplines, such as mechanical and aeronautical engineering. In these engineering disciplines, designers most often employ *linear control theory* [15], which uses input-output relationships of linear systems to study controller properties such as: *stability* (finite inputs produces finite outputs), *bias* (how well objectives are achieved), *rise time* (how quickly the system responds to a change in objective), and *settling time* (how long until steady state is reached). This theory provides sound and rigorous mathematical principles for the design and analysis of closed-loop systems.

The goal of our work is to develop a methodology for, and assess the value of, applying control theory to the evaluation of controllers used for service level management of software systems. While this is not a novel idea in itself, we believe that our approach enables the application of these techniques to a wider variety of systems than the traditional approach. In this paper, we also demonstrate the appeal and power of a control theoretic analysis on a controller for doing admission control of a Lotus Notes workgroup server.

A primary concern with applying linear control theory to computer systems is that the assumption of a linear system is a poor fit to the realities of queueing in computer systems, which are highly non-linear. While there is a well-developed theory of non-linear control, it is much more difficult to apply,

does not generalize across systems and provides much less insight. Our perspective is more pragmatic. We pose the question “Can we construct and analyze the properties of real-life closed-loop software systems using the linear system assumption?” Even in mechanical engineering, a discipline where control theory is well-established, linearity often does not hold (e.g., turbulent fluid flows). Rather, the success of linear control theory has resulted from creativity in its application.

The classical controller design methodology consists of two steps:

System identification: Construct a **transfer function** which relates past and present input values to past and present output values. These transfer functions constitute a model of the system.

Controller design: Based on properties of the transfer function and the desired objectives, a particular **control law** is chosen. Techniques from control theory are used to predict how the system will behave once the chosen controller is added to it.

Previous work on the application of control theoretic techniques to computer systems ([5, 10, 14, 11, 17] to name a few) has generally used first principles to perform system identification. For example, the congestion control work typically constructs state transition equations based on detailed knowledge of (or assumptions about) the protocol, workload, losses, etc. Unfortunately, there are several short-comings with a first principles approach. First, for complex systems, it is difficult to construct a model from first principles, so often some unrealistic assumptions are made. This difficulty has been a major barrier to applying control theory to computer systems. Second, the first-principles models often employ detailed information about the target system. Since these details may change frequently (e.g., with each software release), a first-principles approach may require expert involvement on an on-going basis. This is expensive and often impractical. Third, the first-principles approach often does not address model validation. Without model validation, it is unclear how the insights obtained using control theory relate to the system being studied.

Rather than proceeding from first principles, we advocate an *empirical* approach to system identification. Here, the input and output parameters need to be identified, just as before. But rather than deriving the transfer functions based on first-principles knowledge, an autoregressive, moving average (ARMA) model is constructed, and standard statistical techniques are employed to estimate the ARMA parameters. This approach treats the system as a black box, and thus is not affected by system complexity or lack of expert knowledge. Moreover, changes in the target system can easily be accommodated by re-estimating model parameters. In this paper, we show that the approach works well for the Notes server: R^2 is no lower than 75%, and is as high as 98%.

For the controller law, we use a saturated integral controller. The behavior of such a controller is determined by one parameter, called the *gain*. Control theory tells us that the gain should be large to obtain a fast response to changing inputs, but if it is too large, it can lead to undesirable behaviors in the system,

such as controller induced oscillations. The goal of the analysis, then, is to identify how large gain can be without causing these undesirable behaviors. The particular form of the models allows us to use standard techniques from control theory to perform the analysis. Our results demonstrate that there is a remarkable correspondence between the predictions made by control theory and the observed behavior of an actual Notes server. In particular, we are able to identify the feasible gain values that satisfy the control objectives.

The remainder of this paper is organized as follows. Section 2 describes the Notes server and how this target system is embedded into a closed-loop to achieve service level objectives. Section 3 details our approach to system identification. Section 4 discusses controller design and uses empirical studies to access the accuracy of insights obtained from control theory. We provide a summary and future work discussion in Section 5. Finally, Section 6 discusses related work.

2 Lotus Notes and Its Closed-Loop Control

This section describes relevant features of the Lotus Notes server and provides more details on how closed-loop control is obtained for this target system.

Architecturally, Lotus Notes is a client-server system. Client software converts high-level user activity (mouse clicks, etc) into remote procedure calls (RPCs) that are sent to the server. The server maintains a queue of these in-progress RPCs. Once an RPC is serviced, an entry is made in the server log, and the appropriate response is sent to the client. Clients operate in a synchronous manner – waiting for the previous request to complete before sending a new request. The client/server protocol is session-oriented. A new session is begun after a session-initiating RPC is accepted by the server. We use the term **offered load** to refer to the load imposed on the server by client requests. In the case of homogeneous clients, offered load is expressed in terms of the number of clients. Our service level metric is the length of the queue of in-progress RPC requests, hereafter just referred to as **queue length**.

The tuning parameter `SERVER_MAXUSERS` regulates the number of users allowed to access the server at any time. This is a session-level control (as opposed to packet-level RPC controls). It operates by rejecting session-initiating RPCs once the number of connected users exceeds `SERVER_MAXUSERS`. As such, this parameter has a somewhat complex effect on queue length. In particular, changing `SERVER_MAXUSERS` has no effect until a session-initiating RPC arrives, so existing sessions are not affected.

Unfortunately, we do not have direct measurements of RPC rates and queue length. Rather, we obtain these values from a measurement sensor that samples the server log at a rate of once a minute. The queue length computation is performed by counting RPCs that were active in the previous time quantum. However, since RPCs currently waiting in the queue are not present in the log, this approach underestimates the true queue length and true RPC rates. That is, measurement is **lossy**. We can improve the approximation by delaying one or

more time units before reporting the measurements since doing so allows more RPCs to complete and hence gives us a more accurate count of the RPCs that were executing.

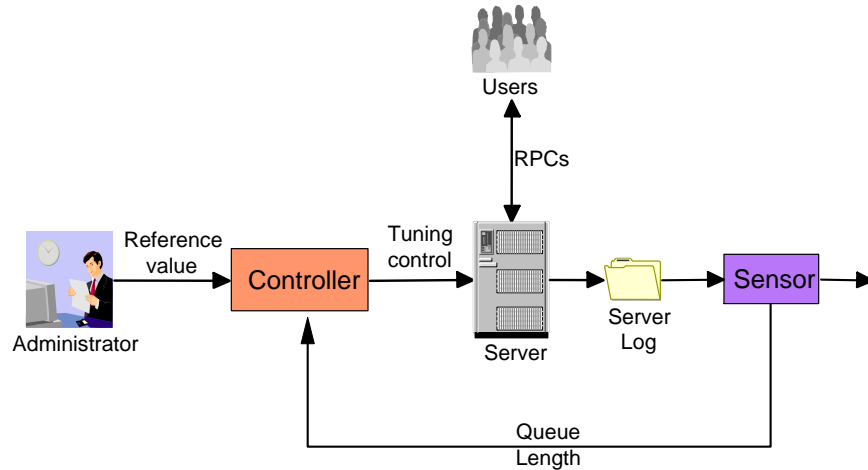


Figure 1: Closed-Loop Control of Lotus Notes

Notes administrators are keenly interested in controlling queue length since this provides a way to manage trade-offs between response times and throughputs. Figure 1 shows how we construct a closed-loop system to control Notes queue length. Notes provides an interface for manipulating tuning parameters such as `SERVER_MAXUSERS`. We use the measurement sensor described above to obtain values of queue length. The administrator specifies a desired value, or **reference value**, for queue length. The reference value specifies a management policy that the closed loop system tries to achieve. The controller takes as input the **control error**, which is the difference between the reference value and measured value of queue length. Depending on the current (and past) values of the error, `SERVER_MAXUSERS` is adjusted. The algorithm that determines the value of `SERVER_MAXUSERS` is called the **control law**.

3 System Identification and Validation

This section describes our approach to system identification and its application to Lotus Notes. System identification has three parts. The first, block diagram construction, identifies the significant functional components and their input-output relationships. The second, transfer function formulation, models the input-output relationships of each element in the block diagram. The particular form of the models we construct (linear transfer functions) is important because it enables us to leverage a large set of analysis techniques that are available in

control theory. The third, parameter estimation and evaluation, assesses the quality of the model developed.

3.1 Block Diagram Construction

A block diagram depicts the components of a system and the flow of information between them. Figure 1 provides a convenient starting point for modeling the Notes server. Here, RPC rates and `SERVER.MAXUSERS` are inputs, and queue length is the output. This is depicted in Figure 2(a).

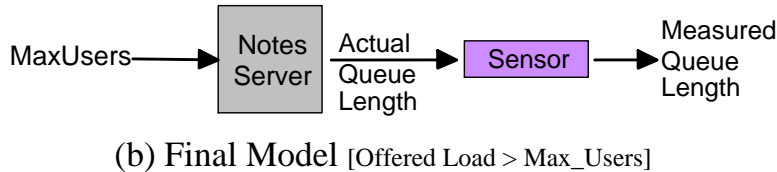
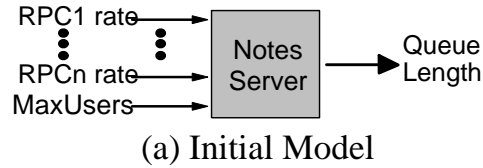


Figure 2: Models of Notes server (open-loop)

There are two problems with the foregoing. First, it is incomplete in that lossy measurements are not considered. Thus, a separate sensor component should be included.

The second problem is more involved. Since `SERVER.MAXUSERS` has an indirect effect on RPC rates, the two inputs are not independent. Hence this is not a linear system. To address this, we divide the operating region of the Notes server into two regions. In the first, `SERVER.MAXUSERS` exceeds offered load and so the tuning parameter has no effect. In the second, `SERVER.MAXUSERS` is lower than offered load and so exactly `SERVER.MAXUSERS` users are allowed onto the system. We focus on the second operating region. Here, the offered load value is not relevant (as long as we stay in this region) and hence, it can be ignored. In other words, there is no need to consider RPC rates as an input.

Can we adequately model the Notes server if `SERVER.MAXUSERS` is the only input to the transfer function? To answer this question, Figure 3 plots queue length where the offered load is 300 users and `SERVER.MAXUSERS` increased by 20 every 20 minutes. (These data are obtained using the experimental set up described in Section 4.4.) The impact of `SERVER.MAXUSERS` is clear, suggest-

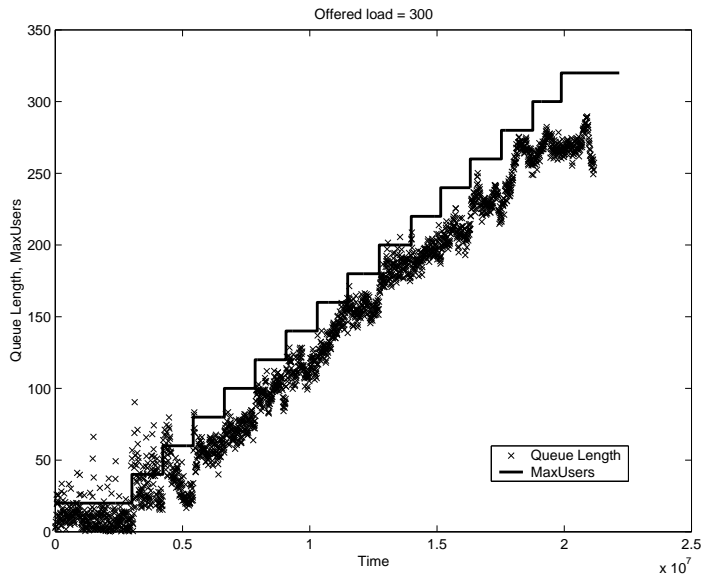


Figure 3: Effect of SERVER_MAXUSERS on Queue Length

ing that it would be sufficient. A more quantitative assessment is provided in Section 3.3.

This results in the block diagram in Figure 2(b). Note that this is a single-input, single-output system.

3.2 Transfer Function Formulation

We must now be more precise about quantifying input-output relationships. Throughout, we assume that time is discrete with uniform interval sizes. Consider a linear system with input $x(t)$ and output $y(t)$. By input-output relationships, we mean an autoregressive, moving average (ARMA) model of the form

$$y(t) = \sum_{i=1}^n a_i y(t-i) + \sum_{j=0}^m b_j x(t-j) \quad (1)$$

where (n, m) is the order of the model, and the a_i, b_j are constants that are estimated from data. When values for n, m, a_i, b_i are specified, this is the transfer function of the linear system. For analysis purposes, it is much more convenient to convert the transfer functions from the time domain into the z (frequency) domain, where z is a complex number. That is, we want to use $Y(z) = \sum_{t=0}^{\infty} y(t)z^{-t}$, which is known as the z -transform.

z -transforms have several nice properties. For example, consider two linear systems with transforms $A(z)$ and $B(z)$. Then the transform of the system

formed by connecting these two in series is $A(z)B(z)$. If outputs of the two systems are summed, then the combined system has the transform $A(z) + B(z)$. Also, if the input to $A(z)$ is multiplied by k , then the associated transform is $kA(z)$.

Applying these principles to Eq. (1) and assuming that $n \geq m$ (which is a typical constraint), we obtain the z -transform:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^m b_j z^{n-j}}{z^n - (\sum_{i=1}^n a_i z^{n-i})} \quad (2)$$

We use the general form of the transfer function in Eq. (2) to determine $N(z)$, the transfer function for the Notes-Server, and $S(z)$, the transfer function of the Sensor component. Let $q(t)$ be the value of queue length at time t and $u(t)$ be the value of `SERVER.MAXUSERS`. Note that this $q(t)$ is the actual queue length, not the one produced by the Sensor. For $N(z)$, it turns out that a good fit is obtained if $n = 1$ and $m = 0$ (see Table 1). That is,

$$N(z) = \frac{Q(z)}{U(z)} = \frac{zb_0}{z - a_1}$$

This is a first order model since $\max(n, m) = 1$.

Modeling $S(z)$ is a bit more involved. Let $m(t)$ be the measured queue length as output by the Sensor. As discussed earlier, the Sensor output is lossy, so in general $m(t)$ underestimates the real queue length $q(t)$. Once again we use a first order model:

$$m_0(t) = a_1 m_0(t-1) + b_0 q(t) + b_1 q(t-1)$$

whose transform is:

$$\frac{M_0(z)}{Q(z)} = \frac{zb_0 + b_1}{z - a_1}$$

However, this is not enough. Recall that to get an accurate estimate of $q(t)$, we could delay d time units so that long-running RPCs present during time t complete their execution. To model this effect, define $m(t)$ such that: $m(t) = m_0(t-d)$. In the z -domain, this is simply z^{-d} . Folding this into the previous equation, we obtain:

$$S(z) = \frac{M(z)}{Q(z)} = \frac{zb_0 + b_1}{(z - a_1)z^d}$$

Note that in modeling $N(z)$ and $S(z)$, we have treated the components as a black boxes, and have not used any details about their internal operation.

3.3 Parameter Estimation and Model Evaluation

Given the functional forms of $N(z)$ and $S(z)$, we must estimate their parameters. Our approach is statistical. First, measurements of the target system are

obtained while varying the input parameters in a controlled way, such as the data in Figure 3. Then, we use least-squares regression to estimate the a_i, b_j for different values of (n, m) . In general the fit of the model improves as (n, m) are increased. We seek a model that has adequate fit and a low order.

	Delay	R^2	\mathbf{a}_1	\mathbf{b}_0	\mathbf{b}_1
Notes Server	All	97.6	0.4261	0.4709	0.0
Sensor	0	75.5	0.6371	0.1692	-0.1057
	1	83.7	0.7991	0.7182	-0.6564
	2	91.2	0.9237	0.9388	-0.9092

Table 1: Model R^2 Values and Coefficients

How well do these transfer functions characterize the input-output relationships in the real system? One way to answer this question is to use the metric R^2 , the fraction of the variability of the output variable that is explained by the transfer function. It turns out that a first order model provides a good fit for both $N(z)$ and $S(z)$. Table 1 reports values of the a_i, b_j and R^2 for these transfer functions. For the Notes-Server, R^2 is quite large, almost 98%! This is an excellent fit. The quality of this model can be further assessed by plotting observed values of queue length versus those predicted by the model, shown in Figure 4. Note that almost all observations lie close to the line of unit slope where the predicted value equals the actual value.

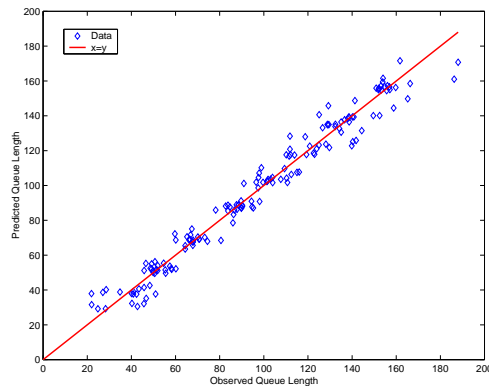


Figure 4: Comparing $N(z)$ model predictions with observed values

For the Sensor transfer function, R^2 is smaller, although still acceptable. Note that as d increases so does R^2 , and a_1 approaches 1. Both effects are expected since with longer delays, measured values approach the actual value.

4 Controller Design and Assessment

Having completed system identification, the next step is to design and assess one or more controllers. We begin by describing how to construct the controller from a control law. This is done under the assumption that the system is linear. Unfortunately, linearity does not always hold. Hence, a preliminary analysis is required to determine the conditions under which linearity is reasonable. We then use control theory to gain insights into controller behavior, especially the presence of controller induced oscillations. These predictions are assessed using measurements of a real Notes server.

4.1 Control Law and Closed-Loop Analysis

Control theory provides a systematic way to study feedback systems. Here, we show how to construct a transfer function of a closed loop system based on the transfer function of the target system in Figure 2(b). By constructing a closed loop system, we mean that the output of Figure 2(b) is fed back to the controller, which in turns compares this to the reference value $r(t)$. Based on the difference between these two values, the controller computes a new setting $u(t)$ for the control, which in our case is the value of `SERVER_MAXUSERS`. This is shown in Figure 5.

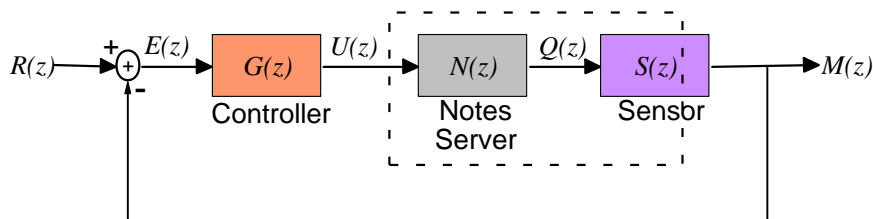


Figure 5: System with controller

The starting point for controller design is a control law that describes how the controller operates. We focus on **integral control** [15], a widely used technique that is a reasonable approach for the Notes server. Only one control law is considered since our objective is to demonstrate the value of our methodology.

A time-domain expression of the integral control law is

$$u(t) = u(t-1) + K_i e(t) \quad (3)$$

where $u(t)$ is the new control value at time t , and $e(t) = r(t) - m(t)$ is the control error. The parameter $K_i > 0$ is called the *gain*. Intuitively, this control law dictates that `SERVER_MAXUSERS` be adjusted incrementally based on its previous value and the gain-weighted control error. From the definition of a transfer function, we have

$$G(z) = \frac{U(z)}{E(z)} = K_i \frac{z}{z-1} = K_i D(z) \quad (4)$$

For an integral controller, the intuition is that higher K_i values lead to a faster response. However, care is required since larger values of K_i can cause oscillations or even instabilities.

There is a problem with directly translating this control law into software that is used for controlling Lotus Notes. Specifically, if $|K_i e(t)|$ is too large, `SERVER.MAXUSERS` is set to a value that can cause a software error. To avoid such situations, we limit the range of `SERVER.MAXUSERS` by extending the control law: $\forall t : Min \leq u(t) \leq Max$. Such **saturated controllers** are not linear. Thus, our modeling is restricted to regions in which these bounds are not reached.

Using the principles of z -transforms discussed earlier, we have

$$\begin{aligned} M(z) &= E(z) * K_i D(z) N(z) S(z) \\ E(z) &= R(z) - M(z) \end{aligned}$$

Solving these equations, we get the following transfer function for the system in Figure 5:

$$\frac{M(z)}{R(z)} = \frac{K_i D(z) N(z) S(z)}{1 + K_i D(z) N(z) S(z)} \quad (5)$$

4.2 Preliminary Analysis

Since our model has been developed under the assumption that $Min \leq u(t) \leq Max$, any analysis based on the model is necessarily restricted to this region as well. We perform a preliminary analysis to determine the values of K_i for which this holds.

Our approach is as follows. We divide the control region into three parts: $u(t) = Max$, $u(t) = Min$ and $Max < u(t) < Min$. We designate these states as *Max*, *Min*, and *Intermediate*, respectively. We seek to understand the conditions under which control values will be in states *Min* and *Max*. If we stay away from these regions, then the assumptions of our analysis should hold.

Figure 6 shows the state transitions obtained from the control law. We see that as $K_i \rightarrow \infty$, all transitions are between states 1 and *Max*. Clearly, we want to avoid large K_i .

How big can K_i be without encountering states *Min* or *Max*? Let ϵ be the largest error that occurs once the closed-loop system is in operation. Then, if

$$K_i < \frac{Max - Min}{\epsilon}$$

we never transition into the extreme states. In our empirical studies of an uncontrolled Notes system, queue lengths range from approximately 20 to 100 if $d = 0$ and 60 to 140 if $d = 2$. So, if there is no bias, then in either case, $\epsilon = 40$. We set $Max = 200$ so that it equals offered load, and $Min = 1$. That gives us: $K_i < 5$.

4.3 Analytical Studies

This subsection uses classical control theory to evaluate the closed-loop system described by Eq. (5). We know from control theory that K_i should be as large

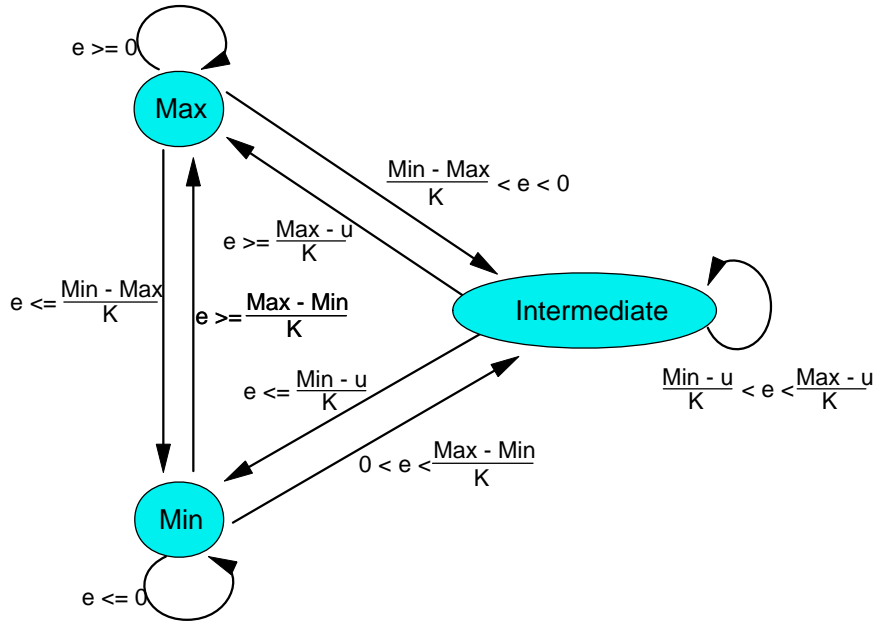


Figure 6: Preliminary Analysis: control state transitions

as possible to provide a fast response. The issue addressed here is to predict when K_i will be so large that there are controller induced oscillations. Such a prediction is made by studying properties of the transfer function for the closed-loop system, that is Eq. (5).

First, some background is required to understand the logic of this analysis. Note that the transfer functions we consider can be expressed as a ratio of two polynomials in z . The roots of the numerator are called its *zeros* and the roots of the denominator are its *poles*. Specifically, if $A(z)$ and $B(z)$ are two polynomials of z and $H(z) = \frac{A(z)}{B(z)}$ is a transfer function, then the zeros are the values of z where $A(z) = 0$ and the poles are where $B(z) = 0$.

The poles and zeros of a transfer function provide insight into stability and controlled-induced oscillations. Recall that z is an imaginary number. If any of the poles of $H(z)$ lie outside the unit circle, then $H(z)$ is unstable. That is, a bounded input produces an unbounded output. This is a result of the mapping from the z domain to the time domain: the time domain equation is expressed in terms of a geometric series in which the poles are raised to the t power. Thus, if a pole has magnitude that exceeds one, it becomes unbounded as t becomes large.

Another fact of interest relates to poles for which $\text{Im}(z) \neq 0$. Such poles contain time domain terms of the form $e^{j\omega t}$, where $j = \sqrt{-1}$. This is a sinusoid and so oscillations are present that increase the variability of $q(t)$.

Root-locus plots provide a systematic way to study the location of poles in the Complex plane. Figure 7 shows root-locus plots for a unit step response (unit change in the tuning parameter) of the system described by Eq. (5). Consider the left most plot, which addresses $d = 0$. The horizontal axis of the plot corresponds to $\text{Re}(z)$ and the vertical axis is $\text{Im}(z)$. To provide a frame of reference, there is a unit circle centered at 0. The x's indicate poles in $G(z)N(z)S(z)$, and the o's indicate its zeros. The root-locus is the curve inside the unit circle that traces the poles as K_i increases from 0 to ∞ . Since all poles lie within the unit circle, there is no problem with stability. Further, observe that for large K_i (e.g. 0.1 and 1), the poles lie on the real axis. Thus, there is no sinusoidal component associated with the step response for these gains. However, for larger K_i (e.g. 5 and 50), there is a non-zero imaginary component to the poles. This suggests the presence of controller induced oscillations that increase the variance of queue length.

Now consider the the root locus plot for $d = 2$, which is the right plot in Figure 7. While $K_i = 0.1$ lies on the real axis, poles for the other gains have nonzero imaginary components. Hence, we expect controller induced oscillations that result in higher variability for queue length.

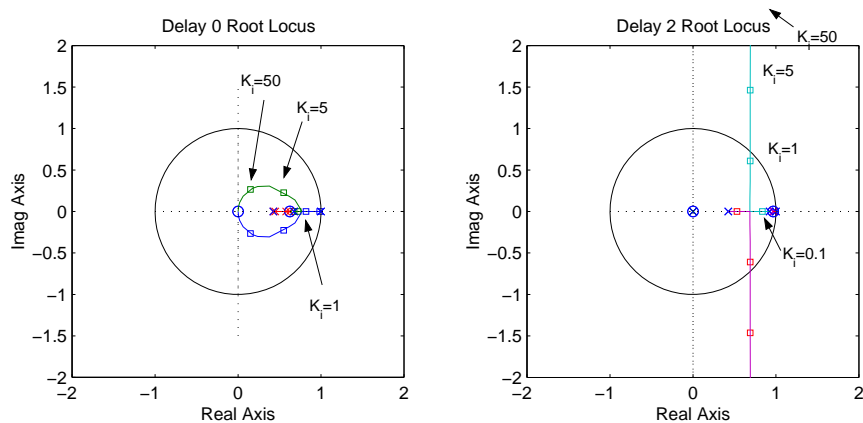


Figure 7: Root-locus plots for delay=0 and delay=2

We further observe that for $d = 2$, the pole at $K_i = 5$ lies outside the unit circle. This suggests a stability problem. Of course, the system cannot become unstable since we have bounded the range of values that `SERVER_MAXUSERS` is assigned. However, large gains can cause another problem—a **limit cycle** in which the tuning parameter only takes on values in $\{u_{\min}, u_{\max}\}$. We discuss this further in Section 4.4. The analysis thus reveals that if we wish to introduce a delay in order to obtain more accurate queue length information, it severely limits the range of gain values that we may use, and thus limits the responsiveness of the control system. Thus, we have a rigorous way of trading off accuracy for recency in the sensor data.

4.4 Empirical Assessments

Here, we present empirical results for various values of K_i used in an integral controller in a real system with a synthetic workload. We study how the predictions made by control theory compare with the behavior of the real system.

The testbed for our experiments consists of a workload generator, product level Notes server, a sensor running on the Notes server, and a controller running on a third machine (so as not to perturb the Notes server). The workload generator simulates the activity of multiple clients by running n copies of an identical script that sends RPCs to the server. These scripts are executed repeatedly with a one minute delay between executions. The script was selected from the NotesBench suite, a standard for such workload generation. During the experiment, the offered load to the server (i.e, the number of users trying to issue requests) is kept constant at 200 users. The reference queue length is initially set to 10, and after 60 minutes is changed to 25.

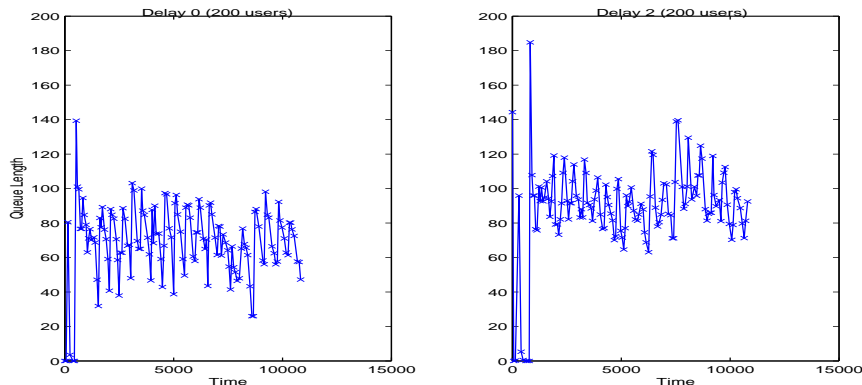


Figure 8: Uncontrolled system run ($\text{SERVER_MAXUSERS} > \text{offered load}$)

Now consider the behavior of the closed loop system if the controller is disabled. This is effected by modifying the control law so that SERVER_MAXUSERS is set to offered load. Figure 8 displays the result for both $d = 0$ and $d = 2$. In the former, queue length hovers around 80. In the latter, it's around 100. These results are consistent with the fact that $d = 0$ is more lossy than $d = 2$. We also see substantial variability in both cases, with changes in queue length of 40 being common. This variability comes, in large part, from having random think times between script executions.

Note further that there appears to be some oscillatory pattern in that queue length alternates between small and large values. To understand why, note that the duration of a script is also one minute. Further, recall that clients operate synchronously in that only one request can be outstanding. Thus, we tend to alternate between sets of clients that are executing and those that are waiting to execute. Thus, even without a controller modifying SERVER_MAXUSERS ,

significant variability is present.

Figure 9 shows the effect of the controller for $K_i \in \{0.1, 1, 5\}$ and delays of 0 and 2. The figure consists of 12 plots presented in two columns. The left column is $d = 0$ and the right column is $d = 2$. There are three parts to the figure. Each part considers a single value of K_i . For example, part (a) consists of the first two rows of plots. The first row plots queue length (and the reference value) versus time. The the second row shows the value of the control at the same time as the queue length plot. Part (b) does the same for $K_i = 1$, and Part (c) displays $K_i = 5$.

Consider Part (a). While there is an initial transient, the queue length converges to the reference value. There is some variability, but variance is considerably smaller than in Figure 8. This observation holds for both $d = 0$ and $d = 2$. These results are consistent with the root-locus analysis that found $\text{Im}(z) = 0$ for $K_i = .1$, where z is a pole of the transfer function of the closed loop system.

Now consider Part (b) in which gain has increased by a factor of ten. We begin with $d = 0$. Here, variability is substantially larger than for $K_i = 0.1$. However, there does not appear to be a controller induced oscillation. Also, queue length values remain centered on the reference value suggesting that bias is small. The situation for $d = 2$ is not so good. There is a pronounced cycle in the queue length values, a cycle that corresponds to a cycle in the values of the control. This suggests a controller induced oscillation. We note that root-locus analysis predicted both of these results in that: (a) there is no pole with a non-zero imaginary component for $K_i = 1, d = 0$ and (b) there is such a pole for $K_i = 1, d = 2$.

What is the reason for the oscillations in queue length? Primarily, this is the result of overcompensation. That is, $K_i = 1$ is so large for the $e(t)$ in $d = 2$ that large positive errors cause the controller to increase `SERVER_MAXUSERS` and this in turn causes the next $e(t)$ to be so negative that `SERVER_MAXUSERS` greatly reduced, and so on. This is evident from the plot of $u(t)$ in Figure 9 Part (b).

In Part (c), gain is fifty times larger than in Part (a). Variability is quite large, even larger than in the uncontrolled system. Indeed, there are oscillations that are clearly related to changes in $u(t)$. Recall that root-locus analysis predicts the presence of controller induced oscillations for this case.

It is instructive to consider an extreme example, one that clearly violates the constraint of $K_i < 5$ that was established in our preliminary analysis. Figure 10 displays the results of studies done for $K_i = 50$. Here, we see a strong limit cycle for the control value, and the resultant queue length plot shows large oscillations. Indeed, changing the reference value has no apparent impact on the system's behavior. Instead, the control oscillates between extreme values in a dysfunctional way.

Table 2 quantifies these results for the region where the reference value $r(t) = 25$. Several metrics are reported: mean queue length, standard deviation of queue length, and RMS (root-mean-square) error. Mean queue length relates to bias in that we are interested in the difference between this number and 25, the reference value. Standard deviation of queue length reflects variability in the

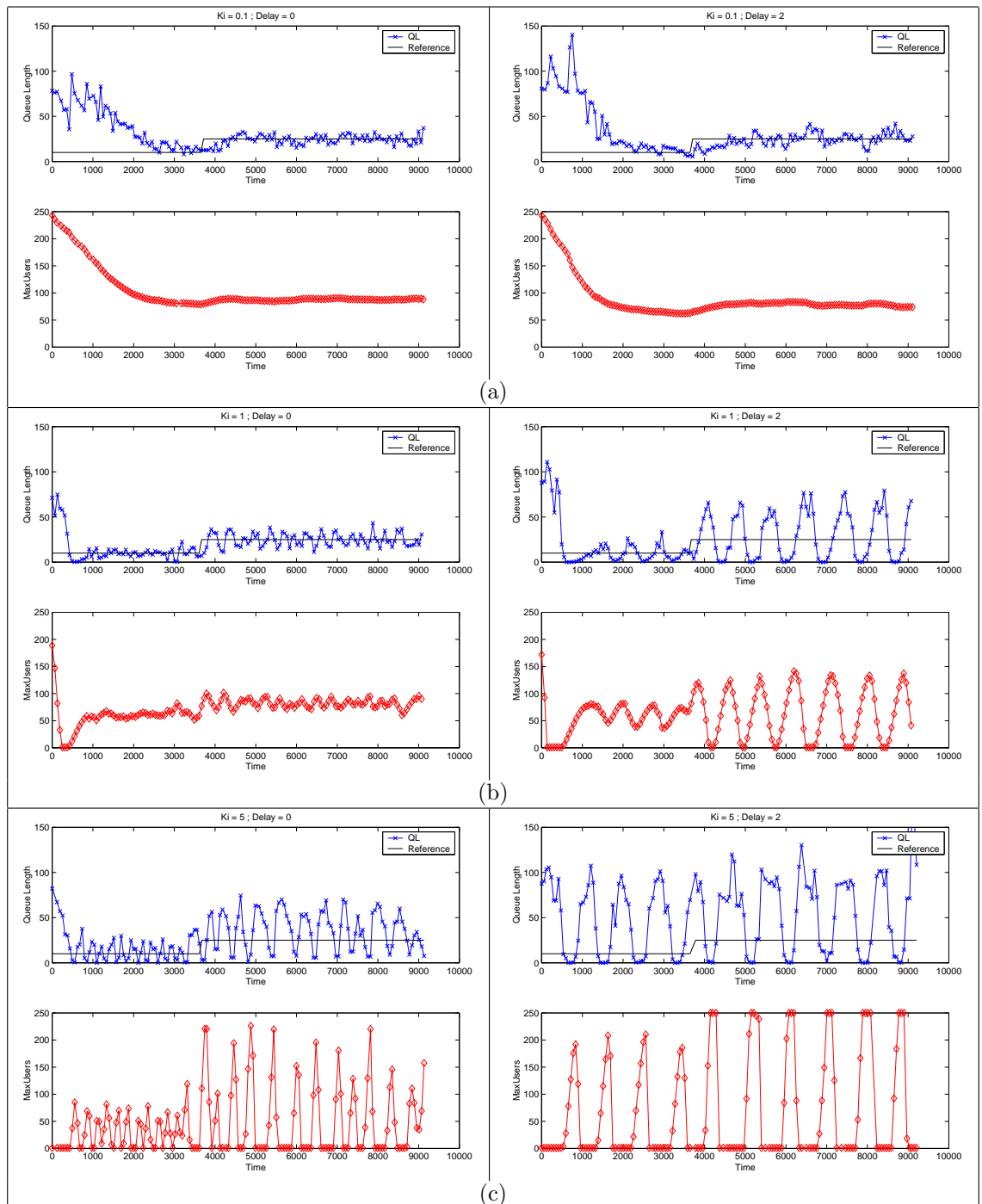


Figure 9: Effect of controller on a real system

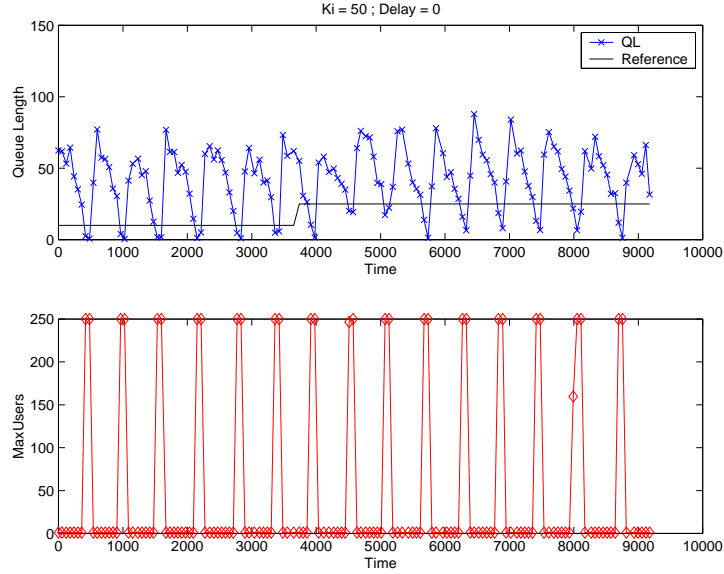


Figure 10: (Unstable) controller with high K_i

system. Root-mean-squared (RMS) error with respect to the reference value quantifies how extreme the bias becomes.

K_i	Delay	Queue Length		RMS Error
		Mean	St. Dev.	
0.1	0	23.95	5.58	5.64
0.1	2	23.80	7.37	7.43
1	0	24.66	7.57	7.54
1	2	29.70	25.67	25.96
5	0	35.39	20.94	23.27
5	2	59.99	41.63	54.17
50	0	42.73	21.41	27.70

Table 2: Controller performance statistics

Note that standard deviations are small for those values of (K_i, d) that root-locus identified as only having real-valued poles. On the other hand, standard deviations are large for those values of (K_i, d) that do have complex poles. We also observe that for larger K_i, d , there is a problem with bias. This is indicated by large values of RMS and the difference between average queue length and the reference value of 25. This validates our analysis that for larger delays, we cannot get a fast-responding system (large K_i), otherwise the system quickly

becomes unstable.

5 Summary and Future Work

Wide-spread reliance on IT services has created intense interest in automated techniques for achieving service level objectives for target systems (e.g., web servers, email servers). In order to design systems that respond dynamically based on feedback about their current state, a commonly-used approach is to create a controller that manipulates tuning parameters of the target system to achieve these objectives. However, a rigorous analysis of the behavior of such a closed-loop system is often lacking. If care is not taken in the design of the controller, then controller induced oscillations can arise that can degrade the quality of service delivered.

In this paper, we have demonstrated a methodology for constructing and analyzing closed-loop systems. Our starting point is classical control theory, a widely used approach in other engineering disciplines. While others have used control theory to analyze computer and communications systems, their work has not provided a general approach to modeling control systems nor has it included empirical validation. We suggest a statistical approach to system identification which is more generally applicable than the conventional first-principles approach, and also has the potential to adapt to changes in the underlying system (such as new software releases).

We have applied this methodology to the closed-loop control of a Lotus Notes server, where a controller manipulates the Notes `SERVER_MAXUSERS` tuning parameter. For system identification, we use least-squares regression to estimate the parameters of the ARMA models for the components of the system. The fit for these models is quite good: R^2 is no lower than 75%, and is as high as 98%. Interestingly, we find that it is important to model the fact that queue lengths are sampled on the server since this can introduce delays that affect controller performance.

We illustrate the value of a control-theoretic analysis by applying it to an integral controller for the Notes server, a popular technique in control theory. To ensure software correctness, however, we are forced to limit the range of $u(t)$. This causes the controller to become non-linear.

With integral control, the design problem is to determine an appropriate value for the gain K_i . From control theory, we know that having a large gain makes the system more responsive. However, too large a gain can cause instabilities. Studying this using classical control theory requires that we restrict ourselves to linear regions of the controller's operation. We therefore use a simple state analysis to estimate values of the gain for which linearity should hold.

Our control-theory based analysis provides useful insights into this tradeoff about the gain value. Root-locus analysis, which is commonly used in classical control theory, allows us to predict which values of K_i cause controller induced oscillations. Our empirical studies using a real Notes system confirm these pre-

dictions. That is, in all cases where root-locus analysis predicts that a controller induced oscillation is present, this happens in our empirical studies. And in all cases where root-locus predicts that there should be no such oscillation, these oscillations are absent in our empirical results. Thus, we can choose a value for K_i that allows the system to be responsive and yet not be subject to controller-induced oscillations. Moreover, the analysis clarifies the effect of sensor-induced delays.

Note that both system identification and controller design are performed off-line, based on data that has been collected from either controlled or production runs of the target system. This allows us to use a large amount of sample data, perform more time-consuming analyses and even consult domain experts. We assume that the system evolves slowly, if at all, so the model does not need to be estimated often. An online changepoint detection [2] scheme can be employed to actively monitor the system and trigger the parameter re-estimation when required. Online adaptation of the target that is within the bounds of the estimated model is performed by the controller.

Much work remains. Our approach to identifying linear regions of operation is approximate at best. A better approach would employ describing functions, a technique used in non-linear control theory. In this paper we have restricted ourselves to a simple control law in order to demonstrate the value of this approach. We plan to study more complex controllers to assess if control theory provides useful insights as to their operation. More broadly, we are interested in applying our methodology to other service level management situations both to refine our methodology and to assess its value.

6 Related Work

The application of control theory to analyzing software systems has been prevalent mostly in the networking arena. The focus is usually on congestion and flow control algorithms for reservationless protocols. A high-level analysis is done by Chiu et al. [5] to derive optimal convergence and fairness policies for congestion avoidance. Keshav [10] provides a more detailed analysis for performing flow control in a network with a very specific set of assumptions about the networking infrastructure and protocols. They use the z -transform and root-locus analysis as well, and are also faced with a non-linearity in their system introduced by *Max* and *Min* terms. While the non-linear analysis is not carried out, they suggest using the second method of Liapunov [15]. Work by Benmohamed et al. [3] for packet-switched networks, and further extended by Mascolo et al. [13] for ATM also follow the first-principles approach. Both these papers have a very detailed system model and they do some sophisticated analyses. However, it is not clear how their ideas would generalize to other systems. More recent work includes Mascolo et al. [14]. and Shor et al. [16], both of which apply control theoretic ideas to analyzing the congestion control behavior in TCP.

Researchers have also applied control theory to QoS-oriented systems, such as OS schedulers, [17], distributed multimedia systems [11] and QoS-aware

caches [12]. In particular, Lu et al. [12] also investigate a controller whose behavior is controlled by one parameter, and they show similar instability effects when the controller gain is too high. Their controller design methodology works backwards from the control objective, but they do not present an analytical analysis of their controller's behavior. On an encouraging note, Goel et al. [7] have recognized the need to build general tools and algorithms for dealing with feedback-based systems. They provide a software toolkit called SWiFT for building and dynamically reconfiguring such systems.

In a somewhat different approach to system modeling, some researchers (for example, Bigus [4]) use neural networks for system modeling. While this approach also allows us to treat the target system as a black box, the resultant models are not as transparent as the ARMA models. While the NNs are non-linear and they can model more complex systems, we are not able to leverage any of the control theory analysis tools for designing or analyzing controllers.

References

- [1] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2), 1997.
- [2] M. Basseville and I. Nikiforov. *Detection of Abrupt Changes: Theory and Applications*. Prentice Hall, 1993.
- [3] Lotfi Benmohamed and Semyon M. Meerkov. Feedback control of congestion in packet switching networks: the case of a single congested node. *IEEE Transactions on Networking*, 1(6), December 1993.
- [4] Joseph P. Bigus. *Adaptive Operating System Control using Neural Networks*. PhD thesis, Lehigh University, 1993.
- [5] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1), June 1989.
- [6] Raymond B. Essick. An event-based fair share scheduler. In *Proceedings of ACM USENIX*, pages 147–161, Winter 1990.
- [7] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. SWiFT: A feedback control and dynamic reconfiguration toolkit. Technical Report 98-009, Oregon Graduate Institute CSE, 1998.
- [8] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee. Network dispatcher: A connection router for scalable internet services. In *Proceedings of the 7th International World Wide Web Conference*, April 1998.
- [9] Arun Iyengar, Jim Challenger, Daniel Dias, and Paul Dantzig. High-performance web site design techniques. *IEEE Internet Computing*, 4(2):17–26, February 2000.

- [10] Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM '91*, September 1991.
- [11] Baochun Li and Klara Nahrstedt. Control-based middleware framework for quality of service applications. *IEEE Journal on Selected Areas in Communication*, 1999.
- [12] Ying Lu, Avneesh Saxena, and Tarek F. Abdelzaher. Differentiated caching services: A control-theoretic approach. In *submitted to Int'l Conference on Decision and Control Systems 2001*, 2001.
- [13] S. Mascolo, D. Cavendish, and M. Gerla. ATM rate based congestion control using a smith predictor: an EPRCA implementation. In *Proceedings of IEEE INFOCOM '96*, 1996.
- [14] Saverio Mascolo. Classical control theory for congestion avoidance in high-speed internet. In *Proceedings of the 38th Conference on Decision & Control*, December 1999.
- [15] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 3rd edition, 1997.
- [16] Molly H. Shor, Kang Li, Jonathan Walpole, David C. Steere, and Calton Pu. Application of control theory to modeling and analysis of computer systems. In *Proceedings of the Japan-USA-Vietnam RESCCE Workshop*, June 2000.
- [17] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, February 1999.