APPLYING FORMAL CONTROL THEORETIC TECHNIQUES TO
COMPUTER SYSTEM PERFORMANCE MANAGEMENT

BY
MEHMET HADI SUZER
BS, Inonu University, 2003
MS, Binghamton University, 2007

DISSERTATION
Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2011
Accepted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Binghamton University
State University of New York
2011

August 5, 2011

Dr. Kyoung-Don Kang, Chair and Faculty Advisor
Department of Computer Science, Binghamton University

Dr. Leslie Lander, Member
Department of Computer Science, Binghamton University

Dr. Kartik Gopalan, Member
Department of Computer Science, Binghamton University

Dr. Edward Li, Outside Examiner
Department of Electrical and Computer Engineering, Binghamton University
Abstract

Computer systems have limited amounts of resources to serve applications’ growing demands. Most systems tend to allocate resources to applications by offline analysis of application requirements, which often results in inefficient resource usage due to the dynamic, time-varying nature of workloads. Formal control theory is known to effectively support the desired performance of the controlled system. However, it is challenging to support the performance of computational systems, since there is no definitive methodology to model computer system dynamics unlike physics laws applied to model physical systems such as a cruise control system. Modeling computer systems, selecting proper control theoretic tools and tuning them according to the needs of specific applications - the research problems to be investigated in this proposed work- are the key ingredients for successful application of control theory to computer system performance management.

To support the desired system performance even in the presence of dynamic workloads, we have applied advanced control theoretic approaches, namely fuzzy control theory, model predictive control theory and event-driven control theoretic techniques. Specifically, we apply these techniques to manage the CPU utilization in a real-time operating system, network bandwidth consumption for video streaming, and link congestion in network gateways.

I aim to demonstrate the applicability of formal control theoretic techniques to support desired system behaviors even in the presence of dynamic workloads and uncertain environments. In this way, I intend to improve the predictability and reliability of computer systems that need to process highly dynamic workloads in uncertain environments.
# Table of Contents

Chapter 1  Introduction ............................................................... 1

Chapter 2  Robust Fuzzy CPU Utilization Control for Dynamic Workloads 8

2.1  Problem Formulation .............................................................. 10

2.1.1  Objective and Real-Time Task Model ................................... 11

2.1.2  High-level System Architecture .......................................... 12

2.1.3  Overview of QoS Adaptation in the Fuzzy Closed-Loop System . 13

2.2  Fuzzy Logic Control .............................................................. 16

2.2.1  Fuzzy Logic Control Components for Control Signal Derivation . 16

2.3  Fuzzy Rule-Base Design .......................................................... 19

2.4  Stability Analysis and Tuning ................................................... 21

2.4.1  Stability Analysis .............................................................. 21

2.4.2  Fuzzy Controller Tuning ...................................................... 23

2.5  Performance Evaluation ........................................................... 24

2.5.1  Experimental Settings ......................................................... 24

2.5.2  Experiment Results ............................................................ 26

2.5.3  Ramp Workload ................................................................. 27

2.5.4  Step Workloads ................................................................. 28

2.5.5  Sawtooth Workload ............................................................ 30

2.5.6  TASKx6 Workload ............................................................. 32

2.5.7  Step5-Random Workload .................................................... 33

2.6  Summary ................................................................................. 35
### Chapter 3  Bandwidth Consumption Control and Service Differentiation for Video Streaming

- **3.1 System Architecture and Service Differentiation** ........................................ 40
- **3.2 Fuzzy Control of the Transmission Rate** ...................................................... 45
- **3.3 Fuzzy Utilization Control** ........................................................................... 46
- **3.4 Fuzzy Rule Base** ...................................................................................... 48
- **3.5 Performance Evaluation** ............................................................................ 50
- **3.6 Summary** ................................................................................................. 53

### Chapter 4  Active Queue Management via Event-Driven Feedback Control

- **4.1 Problem Formulation** ................................................................................ 58
- **4.2 Event-Driven Control for AQM** ................................................................. 61
- **4.3 System Architecture** ................................................................................ 61
- **4.4 TCP and Queue Mathematical Models** ..................................................... 62
- **4.5 Transformation from Time Domain to Spatial Domain** ......................... 63
- **4.6 Linearizing the TCP and Queue Models** .................................................. 64
- **4.7 Event-Driven Congestion Controller Design** .......................................... 66
- **4.8 Performance Evaluation** .......................................................................... 67
- **4.9 Summary** ................................................................................................. 72

### Chapter 5  Related Work

- **5.1 Robust fuzzy CPU utilization control for dynamic workloads** ............... 74
- **5.2 Bandwidth Consumption Control and Service Differentiation for Video Streaming** ........................................................... 75
- **5.3 Active Queue Management via Event-Driven Feedback Control** ........ 76

### Chapter 6  Conclusions and Future Work

- **References** ..................................................................................................... 82
List of Tables

2.1 Fuzzy Rule-Base .......................................................... 17
2.2 System parameters ...................................................... 24
2.3 Performance Summary .................................................. 36
2.4 Control Overhead Comparisons ....................................... 37
3.1 Utilization Control Rules ................................................. 50
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>System Architecture</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Fuzzy Logic Control System</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Input/Output Membership Functions</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>Utilization Control Characteristics</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Tested Workloads</td>
<td>25</td>
</tr>
<tr>
<td>2.6</td>
<td>Transient Utilization for the Ramp Load</td>
<td>28</td>
</tr>
<tr>
<td>2.7</td>
<td>Transient Utilization for the Step-5 Load</td>
<td>29</td>
</tr>
<tr>
<td>2.8</td>
<td>Transient Deadline Miss Ratio for the Step-5 Load</td>
<td>30</td>
</tr>
<tr>
<td>2.9</td>
<td>Transient Utilization for the Sawtooth Load</td>
<td>31</td>
</tr>
<tr>
<td>2.10</td>
<td>Transient Deadline Miss Ratio for the Sawtooth Load</td>
<td>31</td>
</tr>
<tr>
<td>2.11</td>
<td>Transient Utilization for the TASKx6 Load</td>
<td>32</td>
</tr>
<tr>
<td>2.12</td>
<td>Transient Deadline Miss Ratio for the TASKx6 Load</td>
<td>33</td>
</tr>
<tr>
<td>2.13</td>
<td>Transient Utilization for the Step5-Random Load</td>
<td>34</td>
</tr>
<tr>
<td>2.14</td>
<td>Transient Deadline Miss Ratio for the Step5-Random Load</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>Size of the base-layer over time</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>QoS-Aware Streaming Structure</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Computer systems have limited amounts of resources to serve applications’ growing demands. Most systems tend to allocate resources to applications by offline analysis of application requirements, which often results in inefficient resource usage due to the dynamic, time-varying nature of workloads. Formal control theory is known to effectively support the desired performance of the controlled system. However, it is challenging to support the performance of computational systems, since there is no definitive methodology to model computer system dynamics unlike physics laws applied to model physical systems such as a cruise control system.

Modeling computer systems, selecting proper control theoretic tools and tuning them according to the needs of specific applications - the research problems to be investigated in this proposed work - are the key ingredients for successful application of control theory to computer system performance management. To support desired system performance even in the presence of dynamic workloads, we have applied advanced control theoretic approaches, namely fuzzy control theory, model predictive control theory and event-driven control theoretic techniques. Specifically, we apply these techniques to manage the CPU utilization in a real-time operating system, network bandwidth consumption for video streaming, and link congestion in network gateways.

First, to reduce the difficulty of modeling real-time systems with stringent timing constraints, we apply formal fuzzy control theory that is very effective to support the desired performance in a nonlinear dynamic system without requiring a system model. Traditional real-time scheduling techniques [29] requiring precise a priori knowledge of the workload
are not directly applicable to support timing constraints. Thus, it is critical to continuously measure and control the utilization in a feedback loop to avoid severe underutilization or overload in real-time systems operating in dynamic environments. Linear PID (proportional, integral, and differential) control techniques [41] have been applied to manage real-time performance in dynamic environments [3, 31]. However, PID controllers and their variants, e.g., P, PI, or PD controllers, usually approximate the system dynamics in a piecewise linear fashion [16, 31]. PID controllers are guaranteed to support the set-point only if system dynamics do not deviate from a specific operating range derived offline. If the workload varies dynamically exceeding the operating range, PID controllers and their variants, may largely fail to support the set-point [16]. Model predictive control theory [34] is applied to manage the utilization in dynamic environments by continuously modeling the system behavior online [32, 50]. However, approximate models are often used to reduce the complexity of online predictive modeling of the controlled real-time system. For example, the authors of [32, 50] assume that the actual execution times of real-time tasks are equal to their estimated execution times to decrease the complexity of system modeling. Also, the predictive system model derived online may have non-trivial errors when workloads change fast [4].

In this chapter, we apply formal fuzzy logic control theory [39] to adapt workloads, if necessary, to make the utilization converge to the specified set-point even given dynamic workloads. Unlike PID and model predictive control techniques, fuzzy control is not tied to a mathematical model of the controlled system or an operating range. We support direct nonlinear mappings between the utilization error (target utilization − current utilization) and the workload adjustment required to achieve the target utilization via IF-THEN rules. Rather than relying on an approximate system model, we develop a novel fuzzy closed-loop system to control the utilization based on the logical understanding of the relation between the workload and utilization changes.

Via the Lyapunov direct method [4, 39], we mathematically prove the stability of the fuzzy closed-loop system. Further, we extend the Real-Time Application Interface (RTAI) for Linux kernel [43]. We implement and evaluate our fuzzy logic utilization controller as
well as two existing utilization controllers based on the linear and model predictive control theory for an extensive set of workloads in RTAI. Among the tested approaches, our approach shows the smallest deviation from and the fastest convergence to the specified utilization set-point when the system is in a transient status.

Second, we applied fuzzy control theory to network bandwidth consumed by multimedia streaming. In this way, we aim to avoid undesirable situations in which multimedia streaming starves other applications such as file transfer sharing the network, for example, in a smart home. To bound the bandwidth usage of streaming, we leverage the layered encoding technique, in which a video frame consists of a base layer and multiple enhancement layers. We always transmit the base layer, because it is required to display a scene. However, under overload, we degrade the video quality by dropping high enhancement layers without affecting the underlying layers, if necessary, to support the specified bit rate bound. A key challenge is how to determine how many enhancement layers to transmit for concurrent video streams to support the specified bit rate bound. This is not a trivial problem, since the size of a frame consisting of a base layer and enhancement layers may significantly vary in time depending on the complexity of the scenes and their inter-relations in a video. Additionally, new multimedia streaming sessions may start, while others may terminate, further increasing the complexity of bit rate control. We also differentiate the video quality for streams with different levels of importance. We have implemented our transmission rate control and service differentiation schemes and evaluated them in our department network where a number of different applications may coexist at the same time.

In this chapter, we present our QoS-aware video streaming framework (qVSF) implemented on top of an open source video streaming server, QStream [42]. We apply fuzzy logic control to support the specified end-to-end bit rate bound at the node with the bottleneck link that suffers the most significant network congestion, since it may have the lowest physical bandwidth and/or the largest amount of streaming data to deliver. The controller at the bottleneck node informs streaming servers, i.e., stream data sources, how many enhancement layers they can transmit during the next sampling period. If streams
going through the bottleneck link have different levels of importance, qVSF differentiates the service by allowing more important stream sources to transmit more enhancement layers. To this end, we apply fuzzy control theory, since it is known to be very effective to support the desired performance when the system model is complex and nonlinear [39].

For performance evaluation, we have run experiments across the shared department network in the Department of Computer Science at SUNY Binghamton where a considerable number of different applications usually coexist at the same time. Performance evaluation results show that our video streaming system can support the specified bit rate bound and differentiate the service to efficiently utilize the limited bandwidth without severely degrading the visual quality of low priority video streams.

Third, we applied formal event-driven control theory to address the congestion control problem in gateways. Packets injected into the network by a source can be dropped before reaching their destinations due to congestion, wasting all the resources consumed by them. It is known that, in an extreme case, congestion collapse may happen causing users suffer severe network performance degradation [20]. Active Queue Management (AQM) is investigated to avoid incipient congestion in gateways to complement congestion control run by the transport layer protocol such as the TCP. Usually, AQM is implemented in gateways that can distinguish between the propagation delay and persistent queuing delay for effective congestion detection. As a gateway is shared by many active connections with a wide range of round trip times, delay tolerances, and throughput requirements, decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself. Most existing work on AQM is either ad-hoc event-driven feedback approaches or time-driven formal control theoretic approaches. Ad-hoc event-driven approaches for congestion control, such as RED (Random Early Detection), lack a mathematical model therefore it is difficult to analyze its dynamics and tune its numerous parameters, while time-driven control theoretic approaches sample the queue length and run the AQM algorithm at every fixed time interval. A time-driven approach for feedback-based congestion control may not be adaptive enough to an abrupt load surge, if a low
sampling rate for feedback control is used or a large number of packets arrive in a short
time period that is shorter than the sampling period. To avoid this problem, a short sam-
pling period should be selected based on pessimistic assumptions about the network load.
As a result, the controller is executed unnecessarily often when the load is not high, wasting
precious resources at the gateway.

To seamlessly integrate the advantages of both event-driven and time-driven control-
theoretic approaches, we present an event-driven feedback control approach based on for-
mal control theory [15, 40]. The key idea of our approach is to design a feedback-based
congestion controller that is invoked upon the arrivals of a specified number of packets
rather than being invoked at every fixed sampling period. As our approach is based on a
mathematical model, its performance is easier to analyze and more predictable than ad-hoc
event-driven approaches are. If a large number of packets arrive in a short time interval,
event-driven controller autonomously executes more often. In this way, the latency for con-
gestion control reduces, enhancing the reactivity to bursty network loads. In contrast,
a time-driven controller has to wait until the next sampling period even in the presence
of a dramatic increase in packet arrivals during the current sampling period. If the packet
arrival rate decreases, the event-driven controller automatically adapts itself to execute less
frequently. As a result, under light loads, it consumes less computational resources than a
time-driven controller does.

We thoroughly evaluate the performance of our approach via an extensive simulation
study in OMNeT++ [37]. We compare it to five advanced approaches for AQM: (1) RED
[13] with the ’gentle’ parameter turned on, (2) the time-driven feedback-based Proportional
Integral (PI) congestion controller developed by Hollot et al.[19], (3) Proportional Integral
based series compensation, and Position feedback compensation (PIP) Controller [17], (4)
Adaptive Optimized Proportional Controller (AOPC) [49] and (5) Fuzzy Logic Controller
(FLC) [10]. The simulation results show that our event-driven controller effectively main-
tains the queue length around the specified reference, while reducing queue length fluctu-
ations compared to the tested baseline approaches. At the same time, it achieves shorter
E2E (end-to-end) delays and noticeably smaller E2E delay fluctuations than RED, PI and
PIP controllers, while achieving almost the same E2E delays and E2E delay fluctuations with AOPC and FLC controllers. Further, it is invoked only 8 times per second on average. In contrast, RED is activated 30 times/s in average while PI [19], PIP [17], AOPC [49] and FLC [10] congestion controllers are activated 160 times/s.

In summary, I aim to demonstrate the applicability of formal control theoretic techniques to support desired system behaviors even in the presence of dynamic workloads and uncertain environments. In this way, I intend to improve the predictability and reliability of computer systems that need to process highly dynamic workloads in uncertain environments.

As of August 2011, we published the following papers. In these papers, we applied formal control theoretic techniques to computer system performance management:


Also, the following paper is under review:

Chapter 2

Robust Fuzzy CPU Utilization Control for Dynamic Workloads

Real-time systems are deployed in mission critical applications such as target tracking, traffic control, and electric grid management where the workload may dynamically vary [6, 30]. For example, the execution times of real-time tasks for target tracking or traffic control may vary significantly when the number of targets or traffic density dynamically changes. In these systems, traditional real-time scheduling techniques [29] requiring precise a priori knowledge of the workload are not directly applicable to support timing constraints. Thus, it is critical to continuously measure and control the utilization in a feedback loop to avoid severe underutilization or overload in real-time systems operating in dynamic environments.

Linear PID (proportional, integral, and differential) control techniques [41] have been applied to manage real-time performance in dynamic environments [3, 31]. However, PID controllers and their variants, e.g., P, PI, or PD controllers, usually approximate the system dynamics in a piecewise linear fashion [16, 31]. PID controllers are guaranteed to support the set-point only if system dynamics do not deviate from a specific operating range derived offline. If the workload varies dynamically exceeding the operating range, PID controllers and their variants, may largely fail to support the set-point [16].

Model predictive control theory [34] is applied to manage the utilization in dynamic environments by continuously modeling the system behavior online [32, 50]. However, approximate models are often used to reduce the complexity of online predictive modeling of the controlled real-time system. For example, the authors of [32, 50] assume that the actual execution times of real-time tasks are equal to their estimated execution times to
decrease the complexity of system modeling. Also, the predictive system model derived online may have non-trivial errors when workloads change fast [4].

In this chapter, we apply formal fuzzy logic control theory [39] to adapt workloads, if necessary, to make the utilization converge to the specified set-point even given dynamic workloads. Unlike PID and model predictive control techniques, fuzzy control is not tied to a mathematical model of the controlled system or an operating range. Because of the model-free nature of a fuzzy logic controller, there is less risk of introducing design errors due to, for example, statistical inaccuracies existing in a black-box plant model [34, 41].

Rather than relying on an approximate system model, we develop novel fuzzy closed-loop system to control the utilization based on the logical understanding of the relation between the workload and utilization changes. Intuitively, it is clear that the utilization increases as the load increases before it saturates at 1 and vice versa. After the utilization saturates at 1, any further load increase does not affect the utilization. In this chapter, we develop a fuzzy logic utilization controller based on the logical understanding of the nonlinear relation between utilization and load changes. We prove the stability of our fuzzy logic controller via the Lyapunov direct method [4, 39]. By leveraging the stability analysis result, we also tune the fuzzy logic controller to avoid repetitive tuning and testing.

For fair and realistic performance evaluation, we extend the Real-Time Application Interface (RTAI) for Linux kernel [43] to implement our fuzzy logic utilization controller (FLC), the PI utilization controller (PIC) designed via an offline piecewise linear approximation of system dynamics [31], and the advanced model predictive utilization controller (MPC) [32]. By performing extensive experiments, we thoroughly compare their performance with each other. Among the tested approaches, the FLC shows the smallest deviation from and the fastest convergence to the specified utilization set-point when the system is in a transient status. Further, it only consumes 0.53% CPU utilization and a small amount of memory to store fuzzy rules and a few control variables.

Despite the effectiveness of fuzzy logic control theory, little prior work has been done to apply it to manage the performance of real-time systems [26, 45]. A summary of the key contributions of this study follows:
• This study presents a new closed-loop approach to supporting the specified set-point utilization even in the presence of dynamic workloads. Especially, we directly manage the nonlinear relation between the load and utilization via formal fuzzy logic control theory that is very effective to support the desired performance in nonlinear, dynamic systems [39].

• Unlike the most existing work on fuzzy control of real-time performance [26, 45], we do formal stability analysis to prove that the utilization converges to the specified set-point in our fuzzy closed-loop system.

• Different from [31, 32, 45, 50] based on simulations, we compare the performance of our approach to PIC and MPC in a real-time kernel. Although Wang et al [52, 53] have implemented and evaluated their approaches based on model predictive control theory [34] for utilization control in a real-time middleware, we are not aware of any prior work that thoroughly compares the performance of fuzzy logic, model predictive, and PI control approaches for performance management in a real-time kernel.

The remainder of this chapter is organized as follows. The problem formulation of fuzzy logic control is given in section 2.1. The design of our fuzzy closed-loop system is described in section 2.2. In section 2.4, the stability of our fuzzy logic controller is proved. Performance evaluation results are discussed in section 2.5. Our work is compared to the current state of art in section 5.1. Finally, we conclude the chapter and discuss future work in section 2.6.

2.1 Problem Formulation

In this section, the key objective of our fuzzy closed-loop approach, real-time task model, and QoS adaptation approach taken in this chapter are described.
2.1.1 Objective and Real-Time Task Model

- **Goal:** In this chapter, we aim to ensure that the utilization converges to the specified set-point even in the presence of dynamic workloads. In this way, the real-time system controlled by our fuzzy closed-loop scheme is desired to avoid overload or underutilization as much as possible.

- **Average and Transient Performance:** A real-time system operating in a dynamic environment may suffer transient overload or underutilization. Therefore, it is necessary to monitor and control not only the long-term average utilization but also the transient utilization in a closed-loop.

- **Task Model:** In this chapter, we assume that there are $N$ periodic real-time tasks in the system. Task $\tau_i$ ($1 \leq i \leq N$) is described by $(C_i, T_i, U_i, D_i)$ where $C_i$ is the estimated execution time, $T_i$ is the period, $U_i (= C_i/T_i)$ is the estimated utilization and $D_i$ is the relative deadline of $\tau_i$.

  A job $\tau_{ij}$ is the $j^{th}$ instance of the periodic task $\tau_i$. We assume that every task starts at time 0. Also, we assume that an arbitrary job’s deadline is equal to the period; therefore, $\tau_{ij}$’s absolute deadline $D_{ij} = jT_i$ where $j \geq 1$.

  In our approach, $\tau_i$’s period $T_i$ can be adapted at run-time, if necessary, to support the utilization set-point within the specified lower and upper bounds, similar to [5]. Hence, $T_i$ always meets the following condition:

\[
T_{i,\text{min}} \leq T_i \leq T_{i,\text{max}} \tag{2.1}
\]

where the minimum period, $T_{i,\text{min}}$, and maximum period, $T_{i,\text{max}}$, are determined by the application of interest. For example, $T_{i,\text{min}}$ and $T_{i,\text{max}}$ may determine the highest and lowest QoS provided by $\tau_i$ for target tracking or traffic monitoring, respectively. Also, based on the relative importance of tasks, different tasks can be assigned different minimum and maximum periods, similar to [5].
• **Workload Estimation:** In this chapter, we assume that only the estimated task execution times are known but the accurate execution times are unknown, because it is very difficult, if at all possible, to know precise workloads a priori in real-time systems operating in dynamic environments as discussed before. Therefore, we can only compute the estimated load \( L = \sum_{i=1}^{N} U_i \).

• **Scheduling:** In this chapter, real-time tasks are scheduled in an earliest deadline first (EDF) manner [29]. Thus, a set of real-time tasks are admitted to the system, if \( L \leq 1 \). As tasks are admitted and scheduled based on the estimated load, the system can be overloaded (or underutilized), if the execution times are underestimated (or overestimated). Note that our approach is not tied to any specific real-time scheduling algorithm. Thus, another real-time scheduling algorithm such as rate monotonic [29] can be used instead.

### 2.1.2 High-level System Architecture

![System Architecture](image)

**FIG. 2.1.** System Architecture

The high level structure of our closed-loop real-time system is shown in Figure 2.1. The admission controller (AC) admits tasks based on the estimated utilization. An incoming task \( \tau_c \) is admitted to the system only if total utilization \( U_t = \sum_{i=1}^{N} U_i \) (for each task \( \tau_i \)) after admission does not exceed utilization set-point \( U_s \). Once admitted, periodic instances of task \( \tau_c \) are continuously executed by the scheduler. At the \( k^{th} \) sampling point, current system utilization \( u(k) \) is provided to the controller which then computes necessary workload adjustment \( \Delta w(k) \) to support \( U_s \). QoS manager enforces the workload adjustment by
modifying periods of all active real-time tasks. A task’s period is always kept within the bounds specified for that specific task. After the periods are adjusted, scheduler schedules tasks using the adjusted periods. The period adjustment for a specific task is proportional to $\Delta w(k)$ and its current period so that tasks with small periods receive small adjustments and vice versa (Eq 2.5). In our system, tasks are notified via a signal after their periods are updated.

### 2.1.3 Overview of QoS Adaptation in the Fuzzy Closed-Loop System

![Fuzzy Logic Control System](image)

**FIG. 2.2. Fuzzy Logic Control System**

Figure 2.2 shows the structure of the fuzzy closed-loop system designed in this chapter. Let SP stand for the sampling period for control. We use the same SP for the PIC, MPC and FLC for fair performance comparisons in section 2.5. The utilization $u(k)$ is measured at the $k^{th}$ sampling point, i.e., time $k \times SP$, for the jobs executed in the $k^{th}$ sampling period, i.e., the time interval $[(k-1)SP, kSP)$.

Given the current utilization $u(k)$, the fuzzy logic controller computes the required workload adjustment to support the utilization set-point $U_s$ such as 0.7. The system is considered to be overloaded, if $u(k)$ exceeds the set-point and vice versa. The fuzzy control signal becomes negative (positive) when the system is overloaded (underutilized). Accordingly, the QoS manager in the real-time system determines how much to increase the task periods under overload and vice versa within certain bounds. A more detailed description of the procedure follows.

In the fuzzy logic closed loop system, the error, $e(k)$ in Figure 2.2, is defined as follows:

\[
e(k) = U_s - u(k) \tag{2.2}
\]
where $U_s$ is the utilization set-point. Also, we monitor the change in error:

$$\Delta e(k) = e(k) - e(k-1)$$  \hspace{1cm} (2.3) $$

Based on the measured error and change in error, we directly manage the utilization rather than relying on a black-box model that may involve non-trivial statistical errors, if the load changes fast [34, 41]. Based on $e(k)$ and $\Delta e(k)$, the FLC in Figure 2.2 computes the required workload adjustment $\Delta W(k)$ for the next sampling period. The fuzzification interface converts $e(k)$ and $\Delta e(k)$ to linguistic values such as negative small (NS) and positive small (PS). The inference mechanism looks up the knowledge base that has IF-THEN rules to find the corresponding control signal. For example, an IF-THEN rule for utilization control may state that if error is NS and change in error is PS, then the control signal is NS. This rule dictates the QoS manager to reduce the load by a small amount. The defuzzification\(^1\) interface converts the linguistic control signal to a crisp control signal $\Delta W(k)$ expressed as a real number such as -0.25. A detailed discussion of fuzzy control is given in section 2.2.

Given the control signal $\Delta W(k)$, the QoS manager computes the period adaptation factor $F_e(k+1)$ for the next sampling period:

$$F_e(k+1) = F_e(k) \cdot (1 - K_{\Delta W} \Delta W(k))$$  \hspace{1cm} (2.4) $$

Note that the control signal in Eq 2.4, i.e. $\Delta W(k)$, is derived based on the potentially nonlinear relationship between the load and utilization as described before. $K_{\Delta W}$ in Eq 2.4 is the control gain that needs to be tuned to support the stability of the closed-loop system. (The stability of our closed-loop system is analyzed in section 2.4.)

As the control signal $\Delta W(k)$ is inverted in Eq 2.4, $F_e(k+1) > F_e(k)$ and the periods of real-time tasks will be increased to reduce the utilization if $\Delta W(k) < 0$ due to overload conditions and vice versa. If the system is overloaded at the $k^{th}$ sampling point, the period

\(^1\)Fuzzification and defuzzification are standard terms in fuzzy logic control theory [39].
of $\tau_i$ ($1 \leq i \leq N - 1$) is increased for the next sampling period; that is, $T_i(k + 1) > T_i(k)$. Thus, the estimated load $L$ is decreased by $C_i / (T_i(k+1) - T_i(k))$. Assuming the tasks are sorted in descending order of the importance, QoS adaptation is applied to $\tau_{i+1}$ and the next task(s) until the sum of the estimated load adaptation becomes equal to $K \Delta w \Delta w(k)$ or no task period can be increased any further. Similarly, task periods are decreased according to the control signal, if the system is underutilized.

Using the adaptation factor, the QoS manager in the real-time system computes:

$$\hat{T}_i(k + 1) = T_i(k) \cdot F_e(k+1)$$

for an arbitrary task $\tau_i$ in the real-time system and determines $\tau_i$’s period for the $(k+1)^{th}$ sampling period, $T_i(k+1)$, as follows:

$$T_i(k+1) = \begin{cases} \hat{T}_i(k + 1) & \text{if } T_i,\text{min} \leq \hat{T}_i(k + 1) \leq T_i,\text{max} \\ T_i,\text{min} & \text{if } \hat{T}_i(k + 1) < T_i,\text{min} \\ T_i,\text{max} & \text{if } \hat{T}_i(k + 1) > T_i,\text{max} \end{cases}$$

Given $\Delta w(k)$, to support the utilization set-point, the period of every task in the system is increased or decreased by the QoS manager and scheduler according to Eq 2.5 and Eq 2.6. In this way, we avoid an unfair case in which one task’s period is increased (or decreased) substantially within its minimum and maximum bounds, while others are not. Hence, the required workload to support utilization set-point $U_s$ for the next period is calculated as:

$$w(k+1) = w(k) + K \Delta w \Delta w(k)$$

From Eq 2.1 and Eq 2.6, we observe that it may not always be possible to adapt the task period as much as indicated by $K \Delta w \Delta w(k)$. This is especially a problem when the system is currently overloaded and no task period can be extended anymore. In this case, newly incoming tasks, if any, are rejected. Also, the least important tasks in the system are temporarily suspended to fully enforce the control signal.

In this chapter, a certain load that lets the system to converge to the set-point is called
of the convergent load $W$. The difference between $W$ and the current workload is formulated as:

$$\tilde{w}(k) = W - w(k)$$  \hspace{1cm} (2.8)

In reality, $W$ is unknown and it may vary in time depending on execution time estimation errors. Thus, the purpose of fuzzy control is to adapt the workload based on $e(k)$ and $\Delta e(k)$ to support $U_S$ by minimizing $|\tilde{w}(k)|$, i.e., the absolute value of $\tilde{w}(k)$.

### 2.2 Fuzzy Logic Control

In this section, the key components of the FLC and control signal computation process are described. Further, a detailed discussion of our rule-base design is given.

#### 2.2.1 Fuzzy Logic Control Components for Control Signal Derivation

In this section, we describe standard fuzzy control terminologies [39] and describe how to derive the control signal.

![Input/Output Membership Functions](image.png)

**FIG. 2.3. Input/Output Membership Functions**

The universe of discourse is the domain of an input (output) to (from) the FLC [39]. Figure 2.3 shows the universe of discourse for the utilization error, change in error, and control output. In this chapter, the universe of discourse for $e(k)$ and $\Delta e(k)$ is $[-1, 1]$, while the universe of discourse for the control output is set to $[-0.75, 0.75]$ to bound the range of the control signal.
Table 2.1. Fuzzy Rule-Base

Linguistic variables describe the input/output variables in fuzzy control. For instance, two inputs to the fuzzy controller at time $k_{SP}$ are error, i.e., fuzzified $e(k)$, and change in error, i.e., fuzzified $\Delta e(k)$. Also, the output from the FLC is called control signal—the required workload adjustment expressed linguistically.

Linguistic variables are associated with linguistic values to describe characteristics of the variables. A linguistic variable error, for example, could be associated with linguistic values Large, Small, or Zero at a sampling point. Figure 2.3 shows linguistic values for the linguistic variables error, change in error, and workload control signal used in this chapter.

A set of IF premise THEN consequent linguistic rules are used to map the inputs to output(s) of a FLC. For example, if error $= \text{NL}$ and change in error $= \text{NM}$ at the $k$th sampling point, i.e., time $k_{SP}$, then the system is overloaded and the degree of overload is increasing considerably according to Eq 2.2 and Eq 2.3. Thus, the corresponding rule in Table 2.1 generates a NL signal that dictates the real-time system to significantly reduce the load to achieve $U_S$. The rule-base in Table 2.1 has a set of IF-THEN rules stating how to achieve the utilization set-point according to the current error and change in error. (The design of the rule-base in Table 2.1 is discussed in section 2.3).

A membership function (MF) in Figure 2.3 quantifies the certainty an $e(k)$, $\Delta e(k)$, or $\Delta w(k)$ value to be associated with a specific linguistic value. Specifically, the horizontal axis of Figure 2.3 represents $e(k)$, $\Delta e(k)$, or $\Delta w(k)$, while the vertical axis indicates the membership value. For MFs (except for the leftmost or rightmost ones), we use symmetric
triangles of an equal base and 50% overlap with adjacent MFs, similar to [36, 39].

Unlike traditional set theory, in fuzzy set theory underlying fuzzy control theory, set membership is not binary but continuous to deal with uncertainties [23, 39, 55]. Thus, a fuzzy input or output may belong to more than one sets—maximum two adjacent sets in Figure 2.3—with different certainty values. For example, if $e(k) = -0.25$, then $e(k)$ belongs to the fuzzy set NS in Figure 2.3 with certainty 1, which is expressed as: $\mu_{NS}(-0.25) = 1$. If $\Delta e(k) = 0.0625$, $\mu_{ZE}(0.0625) = 0.75$ and $\mu_{PS}(0.0625) = 0.25$.

Based on the fuzzified $e(k)$ and $\Delta e(k)$, the inference mechanism in Figure 2.2 determines which rules to apply at the $k^{th}$ sampling point. Thus, in the previous example, the IF-THEN rules, rule(NS,ZE) = NS and rule(NS, PS) = ZE, in Table 2.1 apply. To compute the certainty value(s) of the corresponding IF premise THEN consequent rule(s), we take the minimum between the certainty values of the premise, i.e., $e(k)$ and $\Delta e(k)$, because the consequent cannot be more certain than the premise [36, 39, 54]. Thus, $\mu(NS,ZE) = \min\{1, 0.75\} = 0.75$ and $\mu(NS,PS) = \min\{1, 0.25\} = 0.25$ in the previous example.

Note that maximum four rules apply at a sampling point, since the error or change in error can belong to up to two MFs in Figure 2.3. Thus, the worst case time complexity of our fuzzy logic control is $O(1)$. Also, storing the rule-base (Table 2.1) consumes little memory.

Finally, the control signal is computed via defuzzification. Let $i$ and $j$ ($1 \leq i, j \leq 7$) represent the row and column indexes in Table 2.1. Further, let $\mu(i, j)$ denote the certainty of the corresponding $rule(e(i, j))$ in the table derived as described before and let $c(i, j)$ denote the center of the MF of the $rule(e(i, j))$’s consequent. For triangle MFs, the center is the middle of the triangle’s base and the fuzzy utilization control output is [39]:

$$\Delta w(k) = \frac{\sum_{i,j} c(i, j) \cdot \mu(i, j)}{\sum_{i,j} \mu(i, j)} \quad (2.9)$$

In Figure 2.3, the center of NS and ZE is $-0.25$ and $0.0$, respectively. Thus, in the previous example, $\Delta w(k) = (((-0.25) \cdot 0.75 + (0.0) \cdot 0.25)/(0.75 + 0.25) = -0.1875$. 

18
2.3 Fuzzy Rule-Base Design

As shown in Figure 2.4, there are five zones that characterize dynamic real-time system’s behaviors from which we derive the rule-base for utilization control in Table 2.1.

**Zone 1.** $e(k) \geq 0$ and $\Delta e(k) \leq 0$: In this zone, the actual utilization is smaller than the set point, but it comes closer to the set point. The control signal to be applied is carefully determined by comparing the magnitude of $"e(k)"$ and $"\Delta e(k)"$ where $"e(k)"$ and $"\Delta e(k)"$ represent the fuzzified $e(k)$ and $\Delta e(k)$, since the current workload may be lower than, equal to, or higher than the convergent load $W$.

- If $|"e(k)"| > |"\Delta e(k)"|$, then $\bar{w}(k) \geq 0$ in Eq. 2.8; that is, the current load is lower than $W$. For example, if $"e(k)" \in PM, PL$ and $"\Delta e(k)" \in NS$, then the current load is lower than $W$. In this case, the utilization is increasing too slow. Thus, the controller should apply a positive signal to further increase the load. As a result, $\Delta w(k) > 0$.

- If $|"e(k)"| = |"\Delta e(k)"|$, then $\bar{w}(k) = 0$ in Eq. 2.8. For example, if $"e(k)" \in PS$ and $"\Delta e(k)" \in NS$, then the current load is equal to $W (\bar{w}(k) = 0)$. Thus, $\Delta w(k) = 0$.

- If $|"e(k)"| < |"\Delta e(k)"|$, then $\bar{w}(k) < 0$. For example, if $"e(k)" \in PS$ and $"\Delta e(k)" \in NM, NL$, then the current load is higher than $W$. In this case, the utilization increases too fast. Thus, the controller applies a negative signal, $\Delta w(k) < 0$, to avoid an overshoot.

\footnote{Note that, in Zones 1–4, $e(k)$ and $\Delta e(k)$ are not both zero at the same time. In only Zone 5, $e(k)$ and $\Delta e(k)$ can be zero at the same time.}
Zone 2. \( e(k) < 0 \) and \( \Delta e(k) \leq 0 \): In this zone, the utilization is higher than the set-point and it is further increasing. It indicates that the current load is higher than \( W \); that is, \( \tilde{w}(k) < 0 \). Hence, the controller applies \( \Delta w(k) < 0 \) to reverse the current trend.

Zone 3. \( e(k) \leq 0 \) and \( \Delta e(k) \geq 0 \): In this zone, the utilization is higher than the set point, but it comes closer to the set point. The control signal should be carefully determined by comparing the magnitude of \( "e(k)" \) and \( "\Delta e(k)" \) as the current workload value may be lower than, equal to, or higher than \( W \) value.

- If \( |"e(k)"| > |"\Delta e(k)"| \), then \( \tilde{w}(k) < 0 \). For example, if \( "e(k)" \in NM , NL \) and \( "\Delta e(k)" \in PS \) then the current load is higher than \( W \); that is, \( \tilde{w}(k) < 0 \). As the utilization is decreasing too slow, the controller should apply a negative signal to further reduce the load.

- If \( |"e(k)"| = |"\Delta e(k)"| \), then \( \tilde{w}(k) = 0 \). For example, if \( "e(k)" \in NS \) and \( "\Delta e(k)" \in PS \), then the current load is equal to \( W \). Thus, \( \Delta w(k) = 0 \).

- If \( |"e(k)"| < |"\Delta e(k)"| \), then \( \tilde{w}(k) > 0 \). For example, if \( "e(k)" \in NS \) and \( "\Delta e(k)" \in PM , PL \), then the current load is lower than \( W \). The utilization is decreasing too fast in this case. Thus, the controller should apply a positive signal to increase the load to support \( U_\delta \), i.e., \( \Delta w(k) > 0 \).

Zone 4. \( e(k) > 0 \) and \( \Delta e(k) \geq 0 \): In this zone, the actual utilization is lower than the set-point and it is further decreasing. It indicates that the current workload is lower than \( W \), i.e., \( \tilde{w}(k) > 0 \). Thus, \( \Delta w(k) > 0 \).

Zone 5. \(|e(k)| \leq \varepsilon \) and \(|\Delta e(k)| \leq \varepsilon \) where \( \varepsilon \) is a small predefined real number: In this case, the real-time system is in the steady state. \( \Delta w(k) = 0 \), as the current workload is equal to \( W \), i.e., \( \tilde{w}(k) = 0 \). In section 2.4, we prove that the fuzzy closed-loop system asymptotically convergences to the \( \varepsilon \) neighborhood of the set-point.

To summarize, the relationship between the control output and inputs in Table 2.1 can be formulated in linguistic terms:

\[ "\Delta w(k)" = "e(k)" + "\Delta e(k)" \]
The linguistic value of ”\( \bar{w}(k) \)” can be determined from these five zones. Our fuzzy logic rule-base containing the five zones implies the following linguistic equation:

\[
"\bar{w}(k)" = "\Delta w(k)"
\]  

(2.10)

which can be validated by inspecting the rule base and explanation of the fuzzy control actions in the five zones. In our rule-base, the sign of \( \Delta w(k) \) is equal to the sign of \( \bar{w}(k) \). This is because, in each zone, the sign of \( \Delta w(k) \) is determined based on the sign of \( \bar{w}(k) \) as discussed earlier in this section. Also, the control signal’s magnitude is proportional to the difference between \( W \) and current load.

2.4 Stability Analysis and Tuning

In this section, the stability of the closed-loop system is analyzed and the control gains, i.e., \( K_e, K_{\Delta e} \) and \( K_{\Delta w} \) in Figure 2.2, are tuned.

2.4.1 Stability Analysis

In this chapter, we prove the stability of our fuzzy closed-loop system via the Lyapunov Direct Method [4, 39].

**Theorem 2.4.1** Lyapunov Direct Method [4, 39]. If the following conditions are true for an arbitrary function \( V(x(k)) : \mathbb{R}^n \to \mathbb{R} \) where \( n \geq 1 \),

\[
\begin{align*}
V(x(k)) &= 0, \text{ if } x(k) = 0 \\
V(x(k)) &> 0, \text{ if } x(k) \in \mathbb{R}^n - \{0\} \\
V(x(k+1)) - V(x(k)) &< 0
\end{align*}
\]

then \( V(x(k)) \) is a Lyapunov candidate function (LCF) in some region \( D \in \mathbb{R}^n \) which contains the origin. \( V(x(k)) \) guarantees the asymptotic stability around zero. (Any nonzero equilibrium point can be transformed to the origin via change of variables.)
We apply Theorem 2.4.1 to prove the stability of our closed loop fuzzy control system. Specifically, we choose the LCF function as:

\[ V(\tilde{w}(k+1)) = \tilde{w}^2(k+1). \]  

(2.11)

**Theorem 2.4.2** If the \( V(\tilde{w}(k+1)) \) has the LCF function properties, then the closed loop fuzzy control system is asymptotically stable around the set point.

**Proof** The LCF function has the following properties:

\[
V(\tilde{w}(k+1)) = 0, \text{ if } \tilde{w}(k+1) = 0 \\
V(\tilde{w}(k+1)) > 0, \text{ if } \tilde{w}(k+1) \in \mathbb{R} - \{0\}
\]

To meet all the requirements to be a LCF, this function should also have the following property:

\[
V(\tilde{w}(k+1)) - V(\tilde{w}(k)) = \tilde{w}^2(k+1) - \tilde{w}^2(k) < 0
\]  

(2.12)

Using Eq. 2.7 and Eq. 2.8, we get:

\[
\tilde{w}(k+1) = W - w(k+1) \\
= W - [w(k) + K_{\Delta w}\Delta w(k)] \\
= W - w(k) - K_{\Delta w}\Delta w(k) \\
= w(k) - K_{\Delta w}\Delta w(k)
\]  

(2.13)

From Eq. 2.12 and Eq. 2.13, we derive that:

\[
V(\tilde{w}(k+1)) - V(\tilde{w}(k)) = [w(k) - K_{\Delta w}\Delta w(k)]^2 - \tilde{w}^2(k) \\
= K_{\Delta w}\Delta w(k) [K_{\Delta w}\Delta w(k) - 2\tilde{w}(k)] < 0
\]
To ensure this inequality, the following constraints should be met:

\[
\text{sign}(\bar{w}(k)) = \text{sign}(\Delta w(k)) \tag{2.14}
\]

\[
|\Delta w(k)| < \frac{2}{K_{\Delta w}}|\bar{w}(k)| \tag{2.15}
\]

The first constraint (Eq. 2.14) is met, since “\(\bar{w}(k)\)” = “\(\Delta w(k)\)” (Eq. 2.10). As \(W\) and thus \(\bar{w}(k)\) are not measurable directly, we can change the second constraint (Eq. 2.15) by replacing \(\bar{w}(k)\) with a small positive real number \(\varepsilon\):

\[
|\Delta w(k)| < \frac{2}{K_{\Delta w}}\varepsilon, \ \varepsilon \in \mathbb{R}^+ \tag{2.16}
\]

If this inequality holds for \(\bar{w}(k) \geq \varepsilon\), then \(w(k)\) will asymptotically converge to an \(\varepsilon\) neighborhood of the convergent load. Specifically, \(0 < K_{\Delta w} < 2/0.75\) since \(\Delta w(k) = [-0.75, 0.75]\) as discussed in section 2.2.1. This concludes the proof of the stability of our fuzzy closed-loop system.

### 2.4.2 Fuzzy Controller Tuning

We need to tune \(K_e, K_{\Delta e}\) and \(K_{\Delta w}\) in Figure 2.2 for good performance. To support the stability of the fuzzy closed-loop system, we must meet the condition that \(0 < K_{\Delta w} < 2.6\) as derived in Theorem 2.4.2. \(K_{\Delta w}\) of a larger value reduces the settling time, but it may cause a higher overshoot. In this chapter, we set \(K_{\Delta w} = 1\) to balance the settling time and overshoot, while focusing slightly more on reducing potential overshoots. Generally, \(K_{\Delta w}\) has the largest effect on the system performance, because it directly affects the stability in addition to the settling time and overshoot. On the other hand, \(K_e\) and \(K_{\Delta e}\) do not directly affect the stability according to Theorem 2.4.2. In this chapter, \(K_e\) is set to 1 so that the controller can utilize the whole rule base for the error input. On the other hand, we set \(K_{\Delta e} = 0.1\) to damp potentially jittery change-in-error values. Generally, a large \(K_{\Delta e}\) reduces the settling time, but increases the overshoot.
2.5 Performance Evaluation

In this section, a description of the experimental set-up for evaluating the FLC, MPC, and PIC is given. Also, the performance evaluation results are discussed.

2.5.1 Experimental Settings

We have implemented the FLC, MPC, and PIC in the RTAI 3.6 [43]. The Linux kernel version 2.6.22 is installed on a 2.3GHz Pentium 4 machine with 1 GB RAM. We have modified the real-time scheduler provided by RTAI to collect performance statistics and implement the controllers. We have implemented and tuned the PIC as described in [31]. Also, we have implemented the MPC described in [32] with the prediction horizon $P = 2$ and control horizon $M = 1$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-point ($U_s$)</td>
<td>0.7</td>
</tr>
<tr>
<td>Sampling period (SP)</td>
<td>1 second</td>
</tr>
<tr>
<td>Algorithm</td>
<td>EDF</td>
</tr>
<tr>
<td>Deadline semantics</td>
<td>Firm</td>
</tr>
<tr>
<td>Estimated execution time</td>
<td>$[50\mu s, 100\mu s]$</td>
</tr>
<tr>
<td>Task period</td>
<td>$[300\mu s, 4\text{ms}]$</td>
</tr>
<tr>
<td>Run length</td>
<td>300 seconds</td>
</tr>
<tr>
<td>Runs per load profile</td>
<td>10</td>
</tr>
<tr>
<td>Load profiles</td>
<td>Ramp, Step, Sawtooth, TASKx6 &amp; Step5-Random</td>
</tr>
</tbody>
</table>

Table 2.2. System parameters

As described in section 2.1, all the controllers output the period adaptation factor $F_e$ in Eq. 2.4 used to adapt the periods of real-time tasks in the system. Each controller is invoked at every sampling point to compute the required workload adjustment to support the utilization set-point $U_s$. In this chapter, SP is set to 1s and $U_s$ is set to 0.7 as shown in Table 2.2. Tasks are scheduled according to the EDF (Earliest Deadline First) algorithm. The deadlines are firm; that is, a task instance is canceled as soon as it misses its deadline.

For performance evaluation, we generate periodic real-time tasks. As shown in Table 2.2, the estimated execution time of a task to be generated is uniformly selected in the
range of [50±μs, 100±μs]. Further, the period of the task is uniformly selected in the range of [300±μs, 4ms]. Each job is associated with an actual execution time: \( \text{AET}_{ij} = \alpha \cdot \text{EET}_{ij} \) where \( \text{EET}_{ij} \) is the estimated execution time of job \( \tau_{ij} \) in the system and \( \alpha \) is the execution time factor, similar to [31, 32]. In this way, fair performance comparisons are possible among the PIC, MPC, and FLC.

![Graphs of ramp, step, and sawtooth workloads](image)

** FIG. 2.5. Tested Workloads**

The worst case execution time of \( \tau_{ij} \) is equal to the maximum of the possible \( \text{AET}_{ij} \) values that are varied by \( \alpha \). Note that the EDF scheduler and controllers (i.e., PIC, MPC, and FLC) are aware of neither the actual nor the worst case execution times, because \( \alpha \) is unknown to them. When \( \alpha > 1 \), they may underestimate execution times. As a result, they may overload the system, missing deadlines. On the other hand, when \( \alpha < 1 \), they may underutilize the system. Thus, we evaluate how closely the FLC, MPC, and PIC can support \( U_S \) when \( \alpha \) varies. To this end, we have created several different experimental load profiles summarized in Table 2.2. For each profile, 10 runs are executed and the average of the 10 runs is reported. Each run executes a random task set for 300s.

In Figure 2.5, the ramp load continuously increases as \( \alpha \) increases from 0.3 to 5 over
300s. The step load tests the robustness of the controller given a sudden load increase and decrease in a step manner. There are five variations of the step load. Each of them starts with \( \alpha = 1 \) and an initial load of 60\%. At 100s, \( \alpha \) is increased to 2, 3, 4, and 5 for Step-2, Step-3, Step-4, and Step-5, respectively. Further, \( \alpha \) is decreased to 0.3 at 200s.

The ramp and step workloads are widely used to evaluate control performance [16, 31, 32, 41]. We use them for fair performance comparisons between the fuzzy controller and the PIC [31] and MPC [32]. In addition, we consider the sawtooth load that concatenates multiple ramp loads to stress the real-time system by increasing or decreasing \( \alpha \) at a constant rate.

At the beginning of an experimental run, each task \( \tau_i \) runs at its minimum period \( T_{i,\text{min}} \). To satisfy Eq 2.1, the maximum period of a task \( \tau_i \) is:

\[
T_{i,\text{max}} = xT_{i,\text{min}}
\]  

For the set of experiments presented in sections 2.5.3 - 2.5.5, \( x = 4 \). For the experiments presented in section 2.5.6, we fix \( \alpha \) to 2 and randomly choose \( x \) in Eq 2.17 in the range \([2, 6]\) for each task, but we increase the number of tasks by six times. This workload is called TASKx6 workload in this chapter. In section 2.5.7, we use a different workload called Step5-Random where \( \alpha \) is increased to 5 at 100s and \( x \) is randomly selected in the range \([2, 6]\) to further evaluate FLC, MPC, and PIC in dynamic environments. In our experiments reported in this chapter, approximately 340,000 – 1,500,000 jobs, i.e., periodic task instances, are generated for a 300s run depending on the specific workload.

### 2.5.2 Experiment Results

In this section, the performance evaluation results of the FLC, MPC, and PIC for the ramp, step, and sawtooth workloads are discussed. In our experiments, all the tested approaches admitted all real-time tasks, because the estimated total utilization computed based on the estimated execution times is smaller than 1—the schedulable utilization bound of the EDF scheduling algorithm [29]. Also, no task was suspended in this chapter.
In our experiments, all the tested closed-loop approaches successfully supported the average utilization set-point for most of the experiments by adapting task periods according to the feedback control signal as directed in Eq 2.6. Therefore, we focus on the transient performance results in the following. Note that it is critical to manage not only the long-term average but also transient performance in a mission-critical real-time system.

### 2.5.3 Ramp Workload

The results for the ramp load are given in Figure 2.6. The PIC has non-zero steady state errors that do not decay until the end of the experiment at 300s as shown in Figure 2.6. Thus, we observe that the PIC clearly fails to support the set-point, which is important as the set-point support not only provides a safety-margin for bursts but it also allows for resource sharing possibly between non-real-time tasks and real-time tasks. In contrast, the MPC cancels an initial utilization overshoot. MPC’s settling time is approximately 40s. As shown in Figure 2.6, the FLC’s settling time is only about 10s. Also, it shows the smallest overshoot. From these results, we observe that the FLC achieves the best performance among the tested approaches for the ramp load.

For the ramp workload, all the tested approaches meet all the task deadlines as the $\alpha$ value increases only gradually. Although the PIC meets all the deadlines, its utilization does not converge to the set-point and it is higher than the set-point as discussed before. Thus, the PIC leaves less CPU cycles available to non-real-time tasks (if any). Also, it is possible for the PIC to miss deadlines, if a higher utilization set-point is used.

To further analyze the set-point tracking performance, we define the aggregated error $E_{agg}$:

$$
E_{agg} = \sqrt{\frac{1}{n} \sum_{k=1}^{n} (U_s - u(k))^2}
$$

(2.18)

where $n$ is the number of the sampling points in one experimental run. For the ramp workloads, the FLC reduces $E_{agg}$ by 56% and 74% compared to the MPC and PIC. Specifically, $E_{agg} = 0.0056$ for the FLC, while $E_{agg} = 0.0127$ and $E_{agg} = 0.0215$ for MPC and PIC, respectively. Overall, the FLC supports the smallest deviations from the set-point and shortest
settling times for the ramp load.

For all the tested workloads, the FLC, MPC, and PIC adjust the task periods in a similar fashion. The average period adaptations achieved by them are almost equal. However, the transient period adaptation of the FLC is faster than the others. This result shows the higher adaptivity of the FLC to dynamic workloads.

2.5.4 Step Workloads

For the clarity of presentation without repetitive discussions, we only show the results for the Step-5 load under which the real-time system is most stressed. Figure 2.7 shows the results for the Step-5 load that tests the robustness of the controllers against abrupt changes in task execution times. Since the $\alpha$ value suddenly jumps from 1 to 5 at 100s, all the tested approaches show utilization overshoots. As a result, the utilization saturates at 1 at 100s in Figure 2.7. However, the FLC’s settling time is only about 7s as shown in Figure 2.7. The
settling time of the MPC, 55s, is approximately eight times longer than the FLC’s.

Further, the FLC achieves the smallest $E_{agg}$. Specifically, $E_{agg} = 0.0611, 0.0714,$ and $0.1227$ for the FLC, MPC, and PIC, respectively. Thus, the FLC reduces $E_{agg}$ by approximately 50% compared to the PIC. Furthermore, it reduces $E_{agg}$ by more than 14% compared to the MPC with the less complex controller design than the MPC.

Figure 2.8 shows the transient deadline miss ratios (DMRs) of all the tested approaches for the critical time interval that includes the highest non-zero DMR. Due to the step increase of $\alpha$ to 5 and potential workload accumulation after that, the tested approaches generally show transient DMR overshoots shortly after 100s.

As shown in Figure 2.8, the FLC responds to the step workload increase faster than the PIC and MPC do. As a result, it resumes to meet all deadlines faster than the PIC and MPC do.
2.5.5 Sawtooth Workload

The performance results for the sawtooth load are shown in Figure 2.9. Similar to the ramp and step load results, the FLC shows the most reliable performance.

In Figure 2.9, the FLC shows the substantially smaller utilization overshoots and undershoots than the MPC and PIC. The FLC achieves the fastest convergence to the set-point, even though it is difficult to numerically compare the settling time of the tested approaches due to the highly dynamic system behavior as shown in Figure 2.9.

Figure 2.10 shows the DMRs for the sawtooth workload. As the $\alpha$ value in Figure 2.5(c) is sharply increased shortly after being decreased to 0.2 at 220s, all the tested approaches show the highest transient DMRs in the time interval plotted in Figure 2.10. As shown in Figure 2.10, the observed DMR overshoot of the FLC is considerably smaller than that of the PIC and MPC. Also, it converges back to the zero DMR faster than the PIC and MPC do.
FIG. 2.9. Transient Utilization for the Sawtooth Load

FIG. 2.10. Transient Deadline Miss Ratio for the Sawtooth Load
Moreover, $E_{agg} = 0.0568, 0.073, \text{ and } 0.0994$ for the FLC, MPC, and PIC. Thus, the FLC reduces $E_{agg}$ by more than 22% and 42% compared to the MPC and PIC, respectively. These results demonstrate the effectiveness and robustness of fuzzy control.

2.5.6 TASKx6 Workload

![Figure 2.11. Transient Utilization for the TASKx6 Load](image)

In the TASKx6 workload, we abruptly increase the number of real-time tasks in the system rather than increasing the $\alpha$ value, which is kept fixed at $\alpha = 2$. The number of tasks in the system is increased by 6 times at time 100s. As a result, the load increases from 70% to 420%. Also, the $x$ value in Eq. 2.17 is randomly selected within the range $[2, 6]$ for each task. As shown in Figure 2.11, FLC improves the settling time by 29% and 72% compared to MPC and PIC, respectively. Also, it decreases $E_{agg}$ by 17% and 32%, while reducing the total number of deadline misses by 50% and 46% over MPC and PIC, respectively. For these reasons, in Figure 2.12, the FLC achieves the smallest DMR overshoot and fastest...
2.5.7 Step5-Random Workload

In this experiment, \(x\) value is randomly selected in the range \([2, 6]\). In addition \(\alpha\) is increased to 5 at 100s. The results are presented in Figure 2.13. FLC decreases the settling time by 63\% and 84\%, while reducing \(E_{agg}\) by 29\% and 35\% over MPC and PIC, respectively. Further, FLC reduces the number of total deadline misses by 59\% and 52\% compared to MPC and PIC. Also, from Figure 2.14, we observe that the FLC decreases the transient DMR back to zero faster than the PIC and MPC do. Although the PIC cancels transient deadline misses faster than the MPC does in Figure 2.14, it shows generally higher DMR overshoots.

Overall, the FLC achieves the most robust control performance based on the logical understanding of the system behavior requiring no mathematical modeling of the underly-
ing controlled system, which is tied to an operating range or subject to modeling errors due to simplified approximations or online/offline statistical modeling errors. Especially, the FLC is more robust than the PIC and MPC when the load changes fast.

Table 2.3 summarizes the settling time (T₅), Eₐgg, and the total number of deadline misses normalized to the FLC. Further, it shows the transient deadline miss ratio overshoot (DMRₘₐₓ), i.e., the highest transient DMR, observed for each approach. In this way, we measure the performance of the tested approaches in terms of the accuracy of set-point tracking, the timeliness of system adaptation under overload or underutilization conditions, and the transient DMRs. We consider these features, because they are very important to avoid and recover quickly from overload or underutilization conditions, if any, in a real-time system. From Table 2.3, we observe that the FLC improves the settling time by up to 75% and 96%, Eₐgg by up to 56% and 74%, and number of total deadline misses by up to 62% and 71% compared to the MPC and PIC. Also, the DMRₘₐₓ of the FLC is smaller that...
Finally, Table 2.4 shows overhead of the tested controllers. All the controllers are lightweight and consume less than 1% CPU utilization for the sampling period of 1s. The PIC has the lowest overhead while the MPC has the highest overhead due to the complexity. The FLC consumes approximately 0.5% CPU utilization and a small amount of memory.

2.6 Summary

In a number of real-time applications such as target tracking and traffic control, it is challenging to support the desired real-time performance. To closely support the specified utilization set-point in the presence of dynamic workloads and system behaviors, we design a fuzzy closed-loop system, while mathematically proving the stability of the fuzzy closed-loop system. Also, extensive experiments are performed to thoroughly evaluate the fuzzy, PI [31], and model predictive [32] controllers in a real-time kernel. Among the
<table>
<thead>
<tr>
<th>Load</th>
<th>Approach</th>
<th>Norm. T₅</th>
<th>Norm. Error</th>
<th>Total Misses</th>
<th>DMR&lt;sub&gt;max&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramp</td>
<td>FLC</td>
<td>1 (10s)</td>
<td>1 (0.0056)</td>
<td>1 (0)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>4</td>
<td>2.27</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>1.1 (eₜ &lt; 0)</td>
<td>3.84</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Step 2</td>
<td>FLC</td>
<td>1 (4s)</td>
<td>1 (0.0415)</td>
<td>1 (2716)</td>
<td>0.5486</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>1.25</td>
<td>1.12</td>
<td>2.60</td>
<td>0.5464</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>3.25</td>
<td>1.4</td>
<td>1.31</td>
<td>0.5557</td>
</tr>
<tr>
<td>Step 3</td>
<td>FLC</td>
<td>1 (6s)</td>
<td>1 (0.0490)</td>
<td>1 (6994)</td>
<td>0.8473</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>1.33</td>
<td>1.21</td>
<td>2.65</td>
<td>0.8491</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>3.66</td>
<td>1.72</td>
<td>1.87</td>
<td>0.8500</td>
</tr>
<tr>
<td>Step 4</td>
<td>FLC</td>
<td>1 (7s)</td>
<td>1 (0.0559)</td>
<td>1 (9477)</td>
<td>0.9469</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>1.29</td>
<td>1.19</td>
<td>2.54</td>
<td>0.9496</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>4.28</td>
<td>1.88</td>
<td>1.96</td>
<td>0.9492</td>
</tr>
<tr>
<td>Step 5</td>
<td>FLC</td>
<td>1 (7s)</td>
<td>1 (0.0611)</td>
<td>1 (11198)</td>
<td>0.9797</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>7.86</td>
<td>1.17</td>
<td>2.32</td>
<td>0.9809</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>10</td>
<td>2</td>
<td>1.96</td>
<td>0.9816</td>
</tr>
<tr>
<td>Sawtooth</td>
<td>FLC</td>
<td>N/A</td>
<td>1 (0.0568)</td>
<td>1 (7430)</td>
<td>0.1772</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>N/A</td>
<td>1.29</td>
<td>1.40</td>
<td>0.2626</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>N/A</td>
<td>1.75</td>
<td>3.46</td>
<td>0.6665</td>
</tr>
<tr>
<td>TASKx6</td>
<td>FLC</td>
<td>1 (7s)</td>
<td>1 (0.0286)</td>
<td>1 (4058)</td>
<td>0.0763</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>1.40</td>
<td>1.21</td>
<td>2</td>
<td>0.1020</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>3.57</td>
<td>1.47</td>
<td>1.84</td>
<td>0.1072</td>
</tr>
<tr>
<td>Step5-Random</td>
<td>FLC</td>
<td>1 (7s)</td>
<td>1 (0.0733)</td>
<td>1 (23531)</td>
<td>0.9979</td>
</tr>
<tr>
<td></td>
<td>MPC</td>
<td>2.71</td>
<td>1.41</td>
<td>2.45</td>
<td>0.9996</td>
</tr>
<tr>
<td></td>
<td>PIC</td>
<td>6.14</td>
<td>1.54</td>
<td>2.08</td>
<td>0.9990</td>
</tr>
</tbody>
</table>

Table 2.3. Performance Summary

We tested approaches, our fuzzy logic controller shows the smallest overshoots, undershoots, and reference tracking error as well as the shortest settling time to the set-point across all the tested workloads. To the best of our knowledge, no prior work has designed a fuzzy control system for real-time performance management with formal stability analysis, while comparing it to the PI and model predictive controllers. In the future, we will develop more advanced fuzzy control techniques for real-time performance management.
<table>
<thead>
<tr>
<th>Controller</th>
<th>CPU Utilization</th>
<th>Code Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC</td>
<td>0.25%</td>
<td>3 lines</td>
</tr>
<tr>
<td>FLC</td>
<td>0.53%</td>
<td>100 lines</td>
</tr>
<tr>
<td>MPC</td>
<td>0.95%</td>
<td>600 lines</td>
</tr>
</tbody>
</table>

Table 2.4. Control Overhead Comparisons
Chapter 3

Bandwidth Consumption Control and Service Differentiation for Video Streaming

For real-time streaming, a streaming server may take a greedy approach, in which it simply transmits every frame with the highest possible quality. As a result, however, multimedia streams may starve other applications, e.g., file transfer, web surfing, email, and instant messaging in smart homes and buildings. Thus, a system administrator may specify the bit rate bound for streaming. Also, different videos may have different levels of importance. For instance, a live stream of the cradle in the baby’s room may have a higher degree of importance than a movie. Hence, the quality of videos need to be differentiated according to their importance to efficiently utilize limited resources.

To bound the bandwidth usage of streaming, in this chapter, we leverage the layered encoding technique, in which a video frame consists of a base layer and multiple enhancement layers. We always transmit the base layer, because it is required to display a scene. However, under overload, we degrade the video quality by dropping high enhancement layers without affecting the underlying layers, if necessary, to support the specified bit rate bound. A key challenge is how to determine how many enhancement layers to transmit for concurrent video streams to support the specified bit rate bound. This is not a trivial problem, since the size of a frame consisting of a base layer and enhancement layers may significantly vary in time depending on the complexity of the scenes and their interrelations in a video. Figure 3.1 is a plot of base-layer size for a single stream and shows the varying nature of a frame size clearly. Also, new multimedia streaming sessions may start,
while others may terminate, further increasing the complexity of bit rate control.

![Graph](image)

**Fig. 3.1.** Size of the base-layer over time

In this chapter, we present our QoS-aware video streaming framework (qVSF) implemented on top of an open source video streaming server, QStream [42]. We apply fuzzy logic control to support the specified end-to-end bit rate bound at the node with the bottleneck link that suffers the most significant network congestion, since it may have the lowest physical bandwidth and/or the largest amount of streaming data to deliver. The controller at the bottleneck node informs streaming servers, i.e., stream data sources, how many enhancement layers they can transmit during the next sampling period. If streams going through the bottleneck link have different levels of importance, qVSF differentiates the service by allowing more important stream sources to transmit more enhancement layers. To this end, we apply fuzzy control theory, since it is known to be very effective to support the desired performance when the system model is complex and nonlinear [39]. Fuzzy control theory provides formal techniques to represent, manipulate, and implement human experts’ heuristic knowledge for controlling a plant, e.g., a video streaming application, via if-then rules rather than relying on a mathematical system model developed for specific workloads,
e.g., specific video streams.

qVSF is implemented as a middleware without requiring a change of routers or switches. For performance evaluation, we have run experiments across the shared department network in the Department of Computer Science at SUNY Binghamton where a considerable number of different applications usually coexist at the same time. The performance evaluation results show that qVSF can support the specified end-to-end bit rate bound. For efficient utilization of the bounded bandwidth, it also differentiates the service by transmitting more enhancement layers for more important streams without severely degrading the visual quality of low priority video streams.

The rest of the chapter is organized as follows. Section 3.1 describes the architecture of qVSF as well as the rate control and service differentiation schemes. Section 3.2 describes the design of our fuzzy control scheme. Section 3.5 presents the performance evaluation results. Key related work is reviewed in Section 5.2. Finally, Section 3.6 concludes the chapter and discusses future work.

3.1 System Architecture and Service Differentiation

In this chapter, we adopt the UDP protocol for video streaming. We do not use the TCP protocol as the AIMD (additive increase and multiplicative decrease) nature of the TCP congestion control mechanism may adversely affect the timeliness of video streams. Our approach is reasonable, since a few packet losses could be tolerated for media streaming, but large delays due to retransmissions and transmission rate reductions caused by the TCP congestion control protocol may considerably impair the user perceived QoS. We control the transmission rate of UDP flows for streaming, while differentiating the video quality between flows.

In our approach, frames are transmitted in an EDF (earliest deadline first) manner. Thus, the frame with the shortest playtime deadline is transmitted first. Figure 3.2 shows the structure of qVSF that consists of the admission controller (AC), streaming engine, fuzzy controller, service differentiator (Diff), and chopper. A description of these components
and their interactions follows.

**Admission Control.** A straightforward approach to admission control may rely on the peak bit rate requirement of a video stream. However, this simplistic approach is too pessimistic. As the frame size may significantly vary over time, a reservation approach based on the peak bit rate can waste a large fraction of the reserved bandwidth for most of the time. Also, carefully dropping a subset of enhancement layers under overload might not severely degrade the user perceived QoS, while considerably reducing the bandwidth consumptions. For these reasons, we take an optimistic approach to admission control that accepts a streaming request as long as its average bit rate computed at the video encoding time can be met without affecting the existing streams. Suppose there are \( m \) multimedia streams already in the system and stream \( j (1 \leq j \leq m) \) requires \( r_j \) average bps (bits per second). When a new streaming request arrives, the AC in Figure 3.2 admits it, if \( r_{\text{new}} + \sum_{j=1}^{m} r_j \leq R_s \) where \( r_{\text{new}} \) is the required average bit rate of the newly incoming request and \( R_s \) is the bit rate bound specified by a system administrator.

**Fuzzy Control.** The stream engine periodically transmits frames. Even if the sum of the
average bit rates does not exceed the specified bit rate bound, several streams may need to transmit large frames at the same time. As a result, the rate bound could be violated. In Figure 3.2, the fuzzy controller computes the bit rate control error:

\[ e(k) = R_s - r(k) \]  

(3.1)

where \( r(k) \) is the bandwidth used during the \( k^{th} \) \( (k \geq 1) \) sampling period, i.e., \([(k-1)P, kP), \] where \( P \) is the sampling interval (=1s in this chapter). Our fuzzy controller also computes the change of the error:

\[ \delta e(k) = e(k) - e(k - 1). \]  

(3.2)

Based on \( e(k) \) and \( \delta e(k) \), the controller computes the required bandwidth usage adaptation \( \delta b(k) \) at the \( k^{th} \) sampling instant, i.e., \( kP \). When \( r(k) > R_s \), the rate control signal \( \delta b(k) \) is negative to reduce the bandwidth usage in the \( (k+1)^{th} \) sampling period and vice versa. A detailed discussion of the fuzzy control signal computation is given in Section 3.2.

**Chopping.** Popular video encoding techniques such as MPEG or SPEG result in variable sizes of frames in a video. Thus, there could be times at which large bursts of multiple streams overlap. As a result, a feedback controller may not be able to completely avoid overshoots—bandwidth consumptions exceeding the specified bound. One can fine-tune the controller for specific workloads to minimize potential overshoots. However, this approach is against the qVSF design concept, since fine-tuned controllers may perform poorly for different workloads. To address this problem, qVSF relies on fuzzy control theory, which does not require a precise mathematical model of the controlled system or fine-tuning. Also, qVSF has a chopper to constantly monitor the transmission rate and drop enhancement layers when there is an overshoot due to potentially insufficient QoS adaptation directed by the fuzzy controller. For these reasons, in Figure 3.2, we intentionally place the chopper outside the fuzzy control loop to have no overshoot in qVSF. Also, to enhance the perceived video quality via effective QoS adaptation, the fuzzy controller tries to reduce the error measured before potential chopping.

**Service Differentiation.** The importance of a stream can be determined based on many
factors such as the number of individuals interested in the stream or the importance of the content itself. In this chapter, we assume that weights, i.e., importance levels, of streams are given by users based on their subjective notion of relative importance among video streams. Given the bandwidth usage control signal $\delta b(k)$, we adapt QoS based on the relative weights at times of stream bursts to support the bit rate bound, while differentiating services for efficient bandwidth utilization.

In the current version of qVSF, each client specifies an initial weight when it requests a multimedia stream. For stream $j$, let $w^I_j$ and $w^e_j$ represent the initial weight and effective weight, respectively. The effective weight is computed: $w^e_j = w^I_j / \sum_{j=1}^{n} w^I_j$ where $n$ is the total number of streams. When a new streaming request is accepted, the effective weights are recomputed. Thus, it always holds that $\sum_{j=1}^{n} w^e_j = 1$.

Effective weights are used to distribute the control signal $\delta b(k)$ to streams. When the network is congested, low weight streams are forced to drop more enhancement layers. When the network is underutilized, higher importance streams can transmit more enhancement layers. Separately, the priority mapper of QStream [42] assigns a priority to each layer ranging from 0 to 15 where 15 is the highest priority. These priorities are assigned according to the utility of a layer estimated in terms of the user perceived QoS considering the type of the frame (i.e., I, B, or P) and its rank in the layer hierarchy. To support bandwidth control and service differentiation at the same time, qVSF performs the following procedure at the $k^{th}$ ($\geq 1$) sampling instant.

1. Compute the bandwidth control signal $\delta b(k)$.

2. Let $N_p$ represent the number of priority levels. In QStream, $N_p = 16$. Compute the transmission threshold for $s_j$, i.e., stream $j$: $\theta_j(k) = [\theta_j(k-1) - \delta b(k) \cdot f(\delta b(k), w^e_j) \cdot (N_p - 1)]$ where

   - $f(\delta b(k), w^e_j) = w^e_j$ if $\delta b(k) \geq 0$.
   - $f(\delta b(k), w^e_j) = 1 - w^e_j$ otherwise.

   If $\theta_j > N_p - 1$, set $\theta_j = N_p - 1$. If $\theta_j < 0$, set $\theta_j = 0$. Note that $\forall j \theta_j(0) = N_p - 1$. 

43
3. During the \((k+1)\text{th}\) sampling period, transmit an enhancement layer of \(s_j\), if its QStream priority is higher than or equal to \(\theta_j(k)\). Otherwise, drop it. For example, if \(\theta_j(k) = 14\), qVSF transmits only the base layer, which always has the highest priority \(N_p - 1\) and the enhancement layer with priority \(N_p - 2\) (=14) for \(s_j\)'s frame(s) during the \((k+1)\text{th}\) sampling period. Since \(\forall j, k \theta_j(k) \leq N_p - 1\) in step 2, the base layer is always transmitted. Also, the higher \(w_j\), the lower \(\theta_j(k)\) in step 2. Thus, qVSF transmits more (or drops fewer) enhancement layers for a higher weight stream when the network is underutilized (or overloaded). Further, \(\theta_j(k) \geq \theta_j(k-1)\) (or \(\theta_j(k) \leq \theta_j(k-1)\)) when \(\delta b(k-1)\) and \(\delta b(k)\) are both negative (or positive) due to continuous overload (or underutilization) conditions. Hence, fewer enhancement layers are transmitted when the network is continuously overloaded and vice versa.

**Multi-hop Settings.** Figure 3.3 shows a possible multi-hop layout of qVSF. In this figure, the intermediate relay node 2 has the most demanding resource requirements to forward real-time streams originating at the servers 1, 2, ..., N+1. As a result, the network link between the relay node 2 and its downstream client becomes the bottleneck. In this case, the controller at the relay node 2 takes control of the bandwidth usage control, while the other nodes turn off their controllers and follow the control decisions made by the bottleneck node. On the other hand, each node always runs its own chopper to ensure that the bit rate
on its outgoing link does not exceed \( R_s \). Thus, the end-to-end bandwidth usage does not exceed \( R_s \). In summary, via fuzzy control and chopping, we ensure that the UDP traffic rate for streaming conforms to the bit rate bound specified by the system administrator. By supporting rate control and service differentiation, our work could provide a basis for important real-time and non-real-time applications such as network services in smart homes and buildings. A system administrator can allocate the bandwidth with the confidence that the allocated bandwidth limits and service differentiation policy will be enforced by our approach. A thorough investigation of bandwidth allocation between different types of applications, e.g., video streaming and file transfer, in smart environments is reserved for future work.

### 3.2 Fuzzy Control of the Transmission Rate

In this section, the design of our fuzzy logic controller is discussed. For the clarify of presentation, we convert the rate control problem to the problem of controlling the utilization of the bottleneck link. To achieve the target utilization, our fuzzy controller computes the workload control signal based on the error \( e(k) \) and change of the error \( \delta e(k) \) redefined as follows:

\[
e(k) = U_s - u(k) = U_s - r(k)/R_s
\]

\[
\delta e(k) = e(k) - e(k-1) \tag{3.4}
\]

where the utilization \( u(k) \) is the total bandwidth utilization in the \( k^{th} \) sampling period. In this chapter, we set the target utilization \( U_s = 1 \) to fully utilize \( R_s \). As discussed before, there is no overshoot due to possible chopping.

Based on \( e(k) \) and \( \delta e(k) \) computed in Eq. 3.3 and Eq. 3.4, the fuzzy rate controller computes the required utilization adjustment \( \delta u(k) \) for the next sampling period. \( K_e, K_{\delta e}, \) and \( K_{\Delta r} \) in Figure 3.4 are control gains tuned by trial and error. Specifically, we choose \( K_e = 1.0, K_{\delta e} = 0.5, \) and \( K_{\Delta r} = 0.1 \). Intuitively, \( e(k) \) at the \( k^{th} \) sampling instant needs to be fully considered, while \( \delta e(k) \) and (especially) \( \delta u(k) \) need to be damped to avoid
oscillations. Note that key to effective fuzzy control is good understandings and insights of the controlled system behaviors and corresponding rule-base design [36, 39]. Flexible tuning is a virtue of fuzzy control.

3.3 Fuzzy Utilization Control

![Membership Functions](image)

The universe of discourse (UD) is the domain of the inputs to a fuzzy controller. In Figure 3.5, the UD is \([a, b]\). Note that \(a \rightarrow -\infty\) for both \(e(k)\) and \(\delta e(k)\). If there is no bandwidth usage control, a streaming engine may consume the bandwidth with no bound. As a result, the utilization error (Eq 3.3) could theoretically go to the negative infinity. In practice, it will consume the maximum bandwidth physically available in the worst case and, therefore, \(a\) will be bounded. By having \(a \rightarrow -\infty\), however, our UD becomes...
independent of the physical bandwidth limit, which may change from network to network.

Note that $b = 1$ for $e(k)$, since the maximum utilization set-point $U_s = 1$ and $U_s \geq u(k)$ when $e(k) \geq 0$ in Eq 3.3. In contrast, $b \to \infty$ for $\delta e(k)$, because, for example, $e(k) = 0$ and $e(k-1) \to -\infty$ due to theoretically unbounded bandwidth consumption. Finally, the UD = $[-1, 1]$ for the utilization control signal after defuzzification in Figure 3.4 due to Eq 3.5 to be discussed shortly. Thus, the UD of $\delta u(k)$ is $[-K_{\Delta r}, K_{\Delta r}]$, i.e., $[-0.1, 0.1]$ in this chapter.

By limiting the amplitude of the control signal to a small range, we can reduce potential performance oscillations due to too large (positive or negative) control signals. In summary, the UDs for $e(k)$, $\delta e(k)$, and $\delta u(k)$ are $(-\infty, 1]$, $(\infty, \infty)$, and $[-0.1, 0.1]$ in this chapter.

Linguistic variables describe the inputs and output(s) of a fuzzy controller. The fuzzification interface in Figure 3.4 converts $e(k)$ and $\delta e(k)$ to the corresponding linguistic values defined next.

A linguistic variable such as the error (fuzzified $e(k)$) or change in error (fuzzified $\delta e(k)$) is associated with linguistic values to describe characteristics of the variables. Figure 3.5 shows linguistic values for the linguistic variables error, change in error, and workload control signal used in this chapter.

A linguistic rule is a “IF premise THEN consequent” rule used to map the inputs to output(s) of a fuzzy controller. For example, if $error = NL$ (negative large) and change in error $= NL$ at the $k^{th}$ sampling instance, then the link is overloaded, i.e., $U_s < u(k)$, and the degree of overload is increasing significantly. Thus, the corresponding rule generates a $NL$ signal to reduce the workload by a large amount.

The rule-base in Figure 3.4 has a set of IF-THEN rules dictating how to achieve $U_s$ according to the error and change in error. The inference mechanism in Figure 3.4 evaluates which control rules are relevant at the current time to compute the fuzzy control signal by looking up the rule-base table based on the fuzzified $e(k)$ and $\delta e(k)$ values.

The defuzzification interface in Figure 3.4 converts the fuzzy control signal reached by the inference mechanism to the control signal $\delta u(k)$ expressed as a real number. $\delta u(k)$ is the input to the plant, i.e., the controlled real-time streaming system, which is required to adjust the bit rate consumption according to $\delta u(k)$ by QoS adaptation.
The horizontal axis of Figure 3.5 represents $\varepsilon(k)$ or $\delta\varepsilon(k)$ and the vertical axis indicates the membership value. A membership function (MF) quantifies the certainty an $\varepsilon(k)$ (or $\delta\varepsilon(k)$) value to be associated with a certain linguistic value. For MFs, we use symmetric triangles of an equal base and 50% overlap with adjacent MFs. A triangular MF is one of the most popular MFs in fuzzy control [39]. For example, suppose $\varepsilon(k) = 0.25$. In Figure 3.5, $\mu_{PS}(0.25) = 1$; that is, 0.25 belongs to the fuzzy set PS with certainty 1.

Based on the fuzzified $\varepsilon(k)$ and $\delta\varepsilon(k)$, the inference mechanism in Figure 3.4 determines which rules to apply at the $k$th sampling instance. For example, suppose $\varepsilon(k) = 0.25$ and $\delta\varepsilon(k) = 0.0625$. According to Figure 3.5, the certainty $\mu_{PS}(0.25) = 1$ for $\varepsilon(k)$ and $\mu_{ZE}(0.0625) = 0.75$ and $\mu_{PS}(0.0625) = 0.25$ for $\delta\varepsilon(k)$. To compute the certainty value of the premise in the corresponding IF premise THEN consequent rule(s), we take the minimum between the certainty values of $\varepsilon(k)$ and $\delta\varepsilon(k)$, following the one of the most common approaches [36, 39, 54]. Thus, $\mu(\text{error, change in error}) = \mu(PS, ZE) = \min\{1, 0.75\} = 0.75$ and $\mu(PS, PS) = \min\{1, 0.25\} = 0.25$.

By referring to our rule base in Table 3.1, the inference engine finds that $\text{rule}(PS, ZE) = PS$ and $\text{rule}(PS, PS) = PM$. Let $\mu(i, j)$ denote the membership function and $c(i, j)$ denote the center of the MF of the consequent of the rule $i$, $j$). For triangle MFs, the center is the value on the x axis at the middle of the triangle [39]. For example, in Figure 3.5, the center of $PS$ and $PM$ is 0.25 and 0.5, respectively. Then the fuzzy utilization control output is:

$$\delta u(k) = \frac{\sum_{i,j} c(i, j) \cdot \mu(i, j)}{\sum_{i,j} \mu(i, j)}$$  \hspace{1cm} (3.5)

In the previous example, the defuzzified control signal $\delta u(k) = \frac{0.25 \cdot 0.75 + 0.5 \cdot 0.25}{0.75 + 0.25} = 0.3125$.

For more information about fuzzy control, readers are referred to [39]. A discussion of key ideas behind our rule base design follows.

### 3.4 Fuzzy Rule Base

As shown in Figure 3.6, there are five zones that characterize the utilization controller’s action, from which we derive the rule-base for fuzzy utilization control described...
in Table 3.1:

1. If $e(k)$ is positive and $\delta e(k)$ is negative, the actual utilization is smaller than the target utilization, but it comes closer to the set-point. The controller applies a small signal to avoid a potential overshoot while increasing the utilization.

2. If $e(k)$ is negative and $\delta e(k)$ is negative, the actual utilization is larger than the set-point and it is further increasing. The controller must reduce the load to reverse the current trend.

3. $e(k)$ is negative and $\delta e(k)$ is positive, the actual utilization is higher than the target utilization, but it comes closer to the set-point. Since the actual utilization is converging to the target, the controller applies a small signal to avoid a potential undershoot, i.e., underutilization, while reducing the utilization.

4. $e(k)$ is positive and $\delta e(k)$ is positive, the actual utilization is lower than the set-point and it is further decreasing. Thus, the controller increases the utilization to reverse the current trend.

5. If $|e(k)| < \epsilon$ and $|\delta e(k)| < \epsilon$ where $\epsilon$ is a predefined small positive real number, the utilization converges to the set-point and $|e(k)|$ and $|\delta e(k)|$ are small. Hence, the utilization adjustment $\delta u(k) = 0$.
Table 3.1. Utilization Control Rules

<table>
<thead>
<tr>
<th>e/δe</th>
<th>NL</th>
<th>NM</th>
<th>NS</th>
<th>ZE</th>
<th>PS</th>
<th>PM</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
<td>NM</td>
<td>NS</td>
<td>ZE</td>
</tr>
<tr>
<td>NM</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
<td>NM</td>
<td>NS</td>
<td>ZE</td>
<td>PS</td>
</tr>
<tr>
<td>NS</td>
<td>NL</td>
<td>NL</td>
<td>NM</td>
<td>NS</td>
<td>ZE</td>
<td>PS</td>
<td>PM</td>
</tr>
<tr>
<td>ZE</td>
<td>NL</td>
<td>NM</td>
<td>NS</td>
<td>ZE</td>
<td>PS</td>
<td>PM</td>
<td>PL</td>
</tr>
<tr>
<td>PS</td>
<td>NM</td>
<td>NS</td>
<td>ZE</td>
<td>PS</td>
<td>PM</td>
<td>PL</td>
<td>PL</td>
</tr>
<tr>
<td>PM</td>
<td>NS</td>
<td>ZE</td>
<td>PS</td>
<td>PM</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
</tr>
<tr>
<td>PL</td>
<td>ZE</td>
<td>PS</td>
<td>PM</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
</tr>
</tbody>
</table>

Overall, our fuzzy logic control is lightweight, only requiring a small rule-base, efficient table look-ups, and control signal computation (Eq 3.3 − Eq 3.5).

3.5 Performance Evaluation

The results given in this section are taken from a streaming session during which four clients are served. The duration of the session was 13 minutes. Client requests were sent at arbitrary times within the first minute of the session. Each client requested a different video from the server over our department network. The set-point for transmission rate is set as 2 Mbps, while the four streamed videos require approximately 1.8 Mbps in average. The choice of the set-point was an arbitrary one in our case, driven by our desire to create a harsh environment. For the same settings, QStream [42] without our QoS management schemes often consumes more than 4Mbps. We have considered different settings too. However, due to space limitations, we only present key performance evaluation results.

Figure 3.7 shows the bandwidth usage of the streaming server for the entire session. Transient bandwidth is the transmission rate of the server measured every second. The average bandwidth usage at time \( t \) is the average bandwidth consumption over the time period \([0, t]\). As shown in the figure, cooperation between the controller and chopper eliminates overshoots for the entire streaming session period. Without chopping, our fuzzy controller experiences approximately 10 overshoots in the 13 minute experiment and the largest transient bandwidth usage was approximately 2.5Mbps. Moreover, due to the nature of video
streaming, undershoots may happen when the frames of multiple streams to be transmitted are small. For example, the current frames could be relatively small when they are similar to their previous frames. Since the peak bursts are kept at the set point and the bandwidth consumption is smaller than the set-point except the peaks, the average transmission rate is lower than the specified bound in qVSF. In the future, we will investigate an approach to further increase the efficiency of bandwidth utilization without violating the specified bit rate bound.

Figure 3.8 shows the average number of enhancement layers transmitted for four individual streams. According to this figure, our QoS management scheme effectively differentiates services. For the stream with smallest effective weight of 0.1, 6 enhancement layers were sent in average. For the stream with the highest effective weight of 0.4, an average of 14 enhancement layers out of 15 enhancement layers were sent. Note that QoS for each stream is not perfectly proportional to its effective weight. There are two reasons for this. First, a base-layer is larger than its enhancement layers and layer sizes also vary.
from frame to frame. Since our service differentiation policy is based on bandwidth consumptions, we expect total bytes sent per stream to be proportional to its effective weight rather than the number of transmitted layers. Second, a control signal cannot be applied to saturated streams; that is, when a stream is already at the highest/lowest acceptable QoS level, a positive/negative control signal will not apply to that particular stream. A thorough investigation of a more sophisticated service differentiation scheme is reserved for future work.

Figure 3.9 shows the number of enhancement layers sent per stream. As shown in Figure 3.9(a), the stream with the lowest priority receives the smallest number of enhancement layers throughout the session. In contrast, the highest priority stream is served with the largest number of enhancement layers as shown in Figure 3.9(d).

Finally, we stream a single video with four different priorities to compare their visual quality. Figure 3.10 shows the screen captures of a video simultaneously streamed with different effective weights. Observe that our QoS management scheme supports visually
acceptable QoS despite the rate control and service differentiation. The lowest priority video with effective priority 0.1 is not severely worse than the highest priority video with effective priority 0.4. This is because we always transmit the base layer, while transmitting enhancement layers according to prioritized fuzzy control logic.

3.6 Summary

If greedy QoS management techniques are applied, real-time video streams may starve other applications such as file transfer. Also, different video streams may have different importance levels to users. To address these research issues, we present a real-time video streaming system called qVSF in this chapter. A summary of the key contributions follows.

- In this chapter, we present optimistic admission control and fuzzy logic control
schemes developed to support the specified bandwidth usage bound.

- We present a service differentiation scheme to efficiently utilize the limited bandwidth for streamed videos with different priorities.

- We have implemented qVSF and evaluated it in a local area network shared with different users and applications. qVSF supports the specified transmission rate bound and effectively differentiate the service without unduly degrading the quality of a low priority video.

In the future, we will investigate more advanced approaches for real-time streaming such as dynamic bandwidth allocation among different applications and packet scheduling geared towards real-time streaming.
Chapter 4

Active Queue Management via Event-Driven Feedback Control

Congestion control and avoidance is critical. Packets can be dropped before reaching their destinations due to congestion, wasting all the resources consumed by them. It is known that, in an extreme case, congestion collapse may happen causing users suffer severe network performance degradation [20]. For congestion control and avoidance, AQM (Active Queue Management) has been investigated. Usually, AQM is implemented in gateways that can distinguish between the propagation delay and persistent queuing delay for effective congestion detection. As a gateway is shared by many active connections with a wide range of round trip times, delay tolerances, and throughput requirements, decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself.

The notion of feedback control has been applied to manage the queue length. RED (Random Early Detection) [13], which is one of the earliest approaches for AQM, controls the queue length based on ad hoc feedback control. The objective of RED is to detect any incipient congestion early and provide congestion notification to the sources to let them reduce their transmission rates before packets are dropped due to overflow in network queues. To detect congestion, RED keeps track of an exponentially weighted moving average of the queue length. If the average queue length exceeds a minimum threshold, RED randomly drops or marks packets with an explicit congestion notification bit. Additionally, all packets are marked or dropped if the average queue length exceeds the specified maximum threshold. RED is an ad hoc feedback-based approach, as it neither mathematically models the TCP and queue length dynamics nor develops a feedback controller for congestion.
control by applying formal control theoretic techniques [40]. Instead, RED tries to avoid congestion by monitoring the average queue length and manipulating the packet drop rate if necessary. Due to its superior performance to the previous approaches such as the Drop Tail mechanism, it is recommended by the Internet Engineering Task Force and adopted by many commercial routers. However, it is difficult to analyze the dynamics of RED, because the RED mechanism [13] lacks a mathematical model. Another drawback of RED is the difficulty of tuning its parameters. RED received remarkable attention in the research community. It is followed by a number of projects including [8, 9, 11, 12, 27, 38, 44] just to name a few.

Although the time-driven feedback controller [19] is shown to achieve better performance than RED, it has shortcomings too. A time-driven controller uses equidistant sampling of the controlled system behavior, e.g., the queue length in a router, in time and compares the measured value to the specified reference to compute the error, e.g., the difference between the current queue length and the specified reference value. Based on the error, the control signal is computed to achieve the desired performance such as the reference queue length. However, a time-driven approach for feedback-based congestion control may not be able to support good performance, if a low sampling rate for feedback control is used or a large number of packets arrive in a short time period that is shorter than the sampling period. To avoid this problem, a short sampling period should be selected based on pessimistic assumptions about the network load. As a result, the controller is executed unnecessarily often when the load is not high, wasting precious resources at the gateway.

In this chapter, to seamlessly integrate the event-driven nature of RED and control theoretic approaches for congestion control (and avoidance), we develop an event-driven feedback controller based on formal control theory [15, 40]. The key idea of our approach is to design a feedback-based congestion controller that is invoked upon the arrivals of a specified number of packets rather than being invoked at every fixed sampling period. The advantages of our event-driven approach for congestion control are as follows:

- The nature of congestion control is event-driven and initiated by packet arrivals. For
this reason, RED is designed to be event-driven. Our approach is also event-driven. Further, it is based on a rigorous mathematical model and formal control theory [15, 40] unlike RED. Thus, we can apply well established control theory [15, 40] to tune and mathematically analyze and support the stability of our feedback-based congestion control scheme.

- If a large number of packets arrive in a short time interval, event-driven controller autonomously executes more often. As a result, the latency for congestion control reduces, enhancing the reactiveness to bursty network loads. In contrast, a time-driven controller has to wait until the next sampling period even in the presence of a dramatic increase in packet arrivals during the current sampling period.

- If the packet arrival rate decreases, an event-driven controller automatically adapts itself to execute less frequently. As a result, under light loads, it consumes less computational resources than a time-driven controller does.

To support event-driven congestion control, we convert the time domain TCP and queue model [18, 35] to the corresponding spatial domain—event domain—model. A summary of our key contributions follows:

- We transform the time-domain nonlinear TCP and queue model [18] to the corresponding spatial-domain model. For this transformation and event-driven controller design, we adapt the event-driven control theoretic techniques [15] developed for motor synchronization. The key idea behind [15] is to measure the time between angular movements around a motor’s axis to compute the speed and acceleration in an event-driven fashion rather than calculating the speed and acceleration using a fixed (periodic) sampling rate. In this chapter, we adapt this approach to measure the time taken for a specified number of packets to arrive at the queue. Thus, our approach is not tied to a fixed sampling rate but purely driven by events, i.e., packet arrivals.

- We linearize the spatial-domain nonlinear model and design an event-driven controller based on the linearized model. The basic approach is similar to [19] that
linearize the time-driven model; however, we linearize the event-driven model in the spatial domain unlike [19].

- Also, we thoroughly evaluate the performance of our approach via an extensive simulation study in OMNeT++ [37]. We compare it to five advanced approaches for AQM: (1) RED [13] with the 'gentle' parameter turned on, (2) the time-driven feedback-based Proportional Integral (PI) congestion controller developed by Hollot et al.[19], (3) Proportional Integral based series compensation, and Position feedback compensation (PIP) Controller [17], (4) Adaptive Optimized Proportional Controller (AOPC) [49] and (5) Fuzzy Logic Controller (FLC) [10]. The simulation results show that our event-driven controller effectively maintains the queue length around the specified reference, while reducing queue length fluctuations compared to the tested baseline approaches. At the same time, it achieves shorter E2E (end-to-end) delays and noticeably smaller E2E delay fluctuations than RED, PI and PIP controllers, while achieving almost the same E2E delays and E2E delay fluctuations with AOPC and FLC controllers. Further, it is invoked only 8 times per second in average. In contrast, RED is activated 30 times/s in average while PI [19], PIP [17], AOPC [49] and FLC [10] congestion controllers are activated 160 times/s.

The remainder of this chapter is organized as follows. A problem formulation is given in Section 4.1. Our event-driven approach for active queue management is described in Section 4.2. The performance of our approach and five baselines is compared in Section 4.8. Related work is discussed in Section 5.3. Finally, Section 4.9 concludes the chapter and discusses future work.

4.1 Problem Formulation

This section describes the scope of this chapter, while reviewing the main objectives of AQM.

AQM aims to support congestion avoidance by controlling the queue length to be shorter than the specified upper bound, even if the transport layer protocols in the traffic
sources do not support any congestion control. In this way, AQM aims to reduce the number of packet drops when a network traffic burst arrives. Although there is an upper bound on the queue length, AQM is desired to allow queue length fluctuations to let the queue absorb bursty traffic spikes and accommodate transient congestion. This can be accomplished by either controlling the average queue length instead of the transient queue length or by employing a low pass filter, such as the integrator in a PID (proportional, integral, and differential) or PI controller [40], on the control path. Thus, the queue length limit reflects the size of bursts needed to be absorbed rather than the steady state queue length desired to be maintained. Queue length control affects the overall throughput and E2E delay. Especially, there is a trade-off between throughput and E2E delay. Enforcing a smaller upper bound on the queue length translates into lower E2E delay, but this comes at the expense of lower throughput and vice versa.

In AQM, the gateway needs to randomly select a victim connection to drop packets from. If the gateway drops packets from all connections at the same time, this will result in global synchronization where all the flows throttle back their transmission rates under congestion and potentially increase the rates later in a synchronized manner. As a result, the network performance may oscillate widely. Via random selection, AQM also aims to avoid biases against bursty traffic that is observed, for example, in a Drop Tail queue [13].

Moreover, AQM should be compatible with TCP. Generally, AQM notifies the traffic source of a congestion condition implicitly (by dropping packets) or explicitly (by forwarding an explicit control notification to sources). Therefore, AQM is appropriate to be used with TCP flows that can react to congestion detection.

In addition, there are several features desired to be supported by AQM:

- An AQM algorithm is desired to react quickly to workload changes. Ideally, an overshoot, i.e., the queue length longer than the desired reference value, and settling time, i.e., the time taken to cancel an overshoot (if any), should be minimal in the presence of dynamic load changes.

- Even if the load changes abruptly, the algorithm is desired to control the queue length
to be stable and avoid large queue length oscillations.

- The implementation and tuning of the algorithm is desired to be easy.
- It is desirable for the algorithm to not need any further tuning upon load changes.
- The algorithm itself should not consume excessive amounts of system resources.

The bursty nature of network traffic and dynamic behavior of network components make these performance criteria difficult to meet. RED controls the packet drop rate by monitoring the average queue length. However, RED lacks a mathematical or control theoretic model. Thus, it shows weak performance in the presence of highly dynamic network traffic [18]. Also, RED has many parameters to tune, which makes it difficult to obtain best performance. The time driven controllers such as ([19], [17], [49], [10]) showed superior performance to RED; however, they cannot adapt their sampling rate according to the network traffic. Thus, they must be designed in a pessimistic way to support acceptable performance under heavy loads. As a result, they have to sample the queue length and run the AQM algorithm unnecessarily often under light loads as discussed before.

To address these problems, we develop an EDC (Event Driven Controller). Different from most of existing approaches for AQM that is based on either ad hoc event-driven feedback control [11–13, 38, 44] or time-driven control theoretic techniques [10, 17, 19, 49, 56], our approach for congestion control leverages both the event-driven nature of the RED scheme and control theoretic aspect of the feedback PI controller scheme. EDC has only two parameters to tune for application dependent performance requirements: (1) the desired queue length set-point and (2) event threshold that specifies the number of packet arrivals that triggers an event. By applying control theoretic techniques [15, 40], we aim to tune EDC to support its stability. Also, EDC aims to reduce the consumption of system resources for AQM via systematic event-driven control of the queue length.
4.2 Event-Driven Control for AQM

In this section, the architecture of our event-driven AQM scheme is described. For event-driven AQM, we first transform the time-domain nonlinear TCP and queue model [18] that shows the relation between packet arrivals and queue length variations [18] to the spatial-domain correspondent. Also, we linearize the model around the operating point, at which the spatial-domain derivatives of the queue length and TCP window size are zero, in order to support the stability of our EDC. Using the linearized model, an event-driven feedback controller for congestion control is designed as discussed in the following sections.

4.3 System Architecture

![System Architecture Diagram]

**FIG. 4.1. System Architecture**

Figure 4.1 shows the system architecture of our closed loop control system. Arriving packets are added to the queue or randomly dropped by the Admission Controller (AC) according to the drop probability calculated by our EDC. EDC is invoked if the number of arriving packets becomes equal to (or exceeds) the specified event threshold. When invoked, EDC computes the difference between the current queue length and specified reference queue length (i.e., set-point) to compute the control signal, i.e., the drop rate adjustment, needed to avoid congestion. The selection of the event threshold is based on the trade-off between control performance and resource usage: A smaller event threshold value generally makes our EDC more reactive to dynamic loads and vice versa. However,
simply picking the smallest possible event threshold may not be a solution, because too frequent EDC invocations may consume excessive computational resources at the router. In our approach, the event threshold is computed based on the network parameters usually available, such as the link capacity and typical packet size. For example, let us assume that the link capacity is 10Mbps and the measured average packet size is 1,000 bits. In this case, to invoke EDC for a number of new packet arrivals estimated to consume 1% link utilization, the event threshold is set to 100 packets. If many packets arrive in a short time interval, EDC is invoked frequently and vice versa. From this example, we observe that, to select the event threshold, we do not need to assume heavy network loads that may happen only occasionally. Hence, selecting the event threshold is much less complex and less pessimistic than choosing the sampling period for a time driven controller.

4.4 TCP and Queue Mathematical Models

Let \( \dot{x} \) denote the time derivative of \( x \). At time \( t \), the following nonlinear differential equations model the TCP and queue dynamics [18] in the time domain:

\[
\dot{W}(t) = \frac{1}{R(t)} - \frac{W(t) \cdot W(t - R(t))}{2R(t - R(t))} \cdot p(t - R(t)) \\
q(t) = \frac{W(t)}{R(t)} \cdot N(t) - C \tag{4.1}
\]

where \( W \) is the expected TCP window size, \( q \) is the expected queue length, \( C \) is the link capacity (packets/s), \( N \) is the load factor expressed in terms of the number of TCP sessions, \( T_p \) is the propagation delay, \( R \) is the average round trip time = \( \frac{q}{C} + T_p \) (seconds) and \( p(\in [0,1]) \) is the packet mark/drop probability.

In Eq. 4.1, the queue length \( q \) and the window size \( W \) are bounded positive quantities; therefore, \( q \in [0, \bar{q}] \) and \( W \in [0, \bar{W}] \) where \( \bar{q} \) and \( \bar{W} \) denote the maximum queue size and maximum window size, respectively.
4.5 Transformation from Time Domain to Spatial Domain

For event-driven AQM, we transform the window size and queue length models in Eq. 4.1 to the spatial domain correspondents by applying the techniques presented in [15].

If $\theta(t)$ denotes the number of packets arrived at the queue at time $t$, the packet arrival rate at time $t$ is:

$$\frac{d\theta}{dt} = \frac{N(t)}{R(t)} \cdot W(t) \quad (4.2)$$

The transformation from the time domain ($t$) to the spatial domain ($\theta$) can be performed based on this relationship. The key idea is to consider $\theta$ no longer as a function of time $t$, but to let time $t$ be a function of packet arrivals $\theta$. The notation $t(\theta)$ then denotes the time at which $\theta$ packets arrived at the queue. With this interpretation, $W(\theta)$, $R(\theta)$ and $N(\theta)$ denote the window size, round trip time and number of TCP sessions at the time when $\theta$ packets arrive at the queue. Based on this observation, we transform Eq. 4.2 to the event-driven, spatial-domain as follows:

$$dt = \frac{R(\theta)}{N(\theta) \cdot W(\theta)} \cdot d\theta \quad (4.3)$$

Let $\tilde{W}(\theta) = \frac{dW}{d\theta}$ and $q(\theta) = \frac{dq}{d\theta}$. Since $\tilde{W}(t) = \frac{dW(t)}{dt}$ in the time-driven model (Eq. 4.1), we can transform this model by substituting $dt$ with Eq. 4.3 and setting the independent variable to $\theta$:

$$f(\theta) = \tilde{W}(\theta) = \frac{R(\theta)}{N(\theta) \cdot W(\theta)} \cdot \left[ \frac{1}{R(\theta)} - \frac{W^2(\theta)}{2R(\theta)} \cdot p(\theta) \right]$$

$$= \frac{1}{N(\theta) \cdot W(\theta)} \cdot \frac{W(\theta)}{2N(\theta)} \cdot p(\theta) \quad (4.4)$$
After the transformation, we have obtained another set of nonlinear differential equations. In order to apply linear control theory [40] to these models, we linearize them around an operating point next.

### 4.6 Linearizing the TCP and Queue Models

In this chapter, \((W(\theta), q(\theta))\) in Eq. 4.4 and Eq. 4.5 is defined as the state of event-driven AQM. Also, the spatial domain expression of the drop probability \(p(\theta)\) is the control input to the AC in Figure 4.1. Therefore, the operating point \((W_0(\theta), q_0(\theta), p_0(\theta))\) is defined by \(\dot{W}(\theta) = 0\) and \(\dot{q}(\theta) = 0\). From this and Eq. 4.4 and Eq. 4.5, the following is derived:

\[
W_0^2 p_0 = 2 \quad \text{and} \quad W_0 = \frac{R_0 C}{N} \Rightarrow p_0 = \frac{2N^2}{(R_0 C)^2} \quad (4.6)
\]

where \(R_0 = \frac{q_0}{C} + T_p\). Assuming that \(N(t) \equiv N\) and \(R(t) \equiv R_0\), we linearize the model around the operating point [48] as follows:

\[
\delta W(\theta) = j_{11} \cdot \delta W(\theta) + j_{12} \cdot \delta q(\theta) + j_{13} \cdot \delta p(\theta)
\]

\[
\delta q(\theta) = j_{21} \cdot \delta W(\theta) + j_{22} \cdot \delta q(\theta) + j_{23} \cdot \delta p(\theta) \quad (4.7)
\]

where \(\delta X = X - X_0\) and \(jj_j's\) are elements of Jacobian Matrix \(^1\) of the system:

\(^1\)The third row of Jacobian Matrix, which is the feedback controller model, is omitted since it is not needed to linearize the nonlinear open loop model.
\[
\begin{bmatrix}
  j_{11} & j_{12} & j_{13} \\
  j_{21} & j_{22} & j_{23}
\end{bmatrix}
= \begin{bmatrix}
  \frac{\partial f}{\partial W} & \frac{\partial f}{\partial q} & \frac{\partial f}{\partial p} \\
  \frac{\partial g}{\partial W} & \frac{\partial g}{\partial q} & \frac{\partial g}{\partial p}
\end{bmatrix}
\]

which can be calculated by taking partial derivatives of the nonlinear model around the operating point \((f_0, g_0)\) and using relationships given in Eq. 4.6:

\[
\begin{align*}
j_{11} &= \frac{\partial f}{\partial W} = -\frac{1}{NW_0^2} - \frac{p_0}{2N} = -\frac{p_0}{N} = -\frac{2N}{(R_0C)^2} \\
j_{13} &= \frac{\partial f}{\partial p} = -\frac{W_0}{2N} = -\frac{R_0C}{2N^2} \\
j_{21} &= \frac{\partial g}{\partial W} = -\frac{C R_0}{N} - \frac{1}{W_0^2} = \frac{N}{R_0C} \\
j_{22} &= \frac{\partial g}{\partial q} = -\frac{1}{NW_0} = -\frac{1}{R_0C} \\
j_{12} &= j_{23} = 0
\end{align*}
\] (4.8)

Finally, we derive our linearized spatial domain model as follows:

\[
\begin{align*}
\delta W(\theta) &= -\frac{2N}{(R_0C)^2} \cdot \delta W(\theta) - \frac{R_0C}{2N^2} \cdot \delta p(\theta) \\
\delta q(\theta) &= \frac{N}{R_0C} \cdot \delta W(\theta) - \frac{1}{R_0C} \cdot \delta q(\theta)
\end{align*}
\] (4.9)

The eigenvalues of the linearized TCP and queue dynamics (4.9) are \(-\frac{2N}{(R_0C)^2}\) and \(-\frac{1}{R_0C}\), respectively. Since all the network parameters are positive quantities, these negative eigenvalues indicate that the equilibrium state of the nonlinear dynamics is locally asymptotically stable. A steady state equilibrium is locally asymptotically stable if there exists an \(\varepsilon\) neighborhood of the steady state equilibrium such that from an arbitrary initial condition within this neighborhood, the system converges to this steady state equilibrium. Formally, a steady state equilibrium, \(\bar{y}\), of the difference equation \(y_{\theta+1} = ay_\theta + b\) is locally asymptotically stable, if \(\lim_{\theta \to \infty} y_\theta = \bar{y}\) \(\forall\) initial condition \(y_0\) such that \(|y_0 - \bar{y}| < \varepsilon\) for some small \(\varepsilon > 0\).
4.7 Event-Driven Congestion Controller Design

For event-driven congestion control, we implement a PI (Proportional and Integral) controller to manage the drop probability for congestion control. A PI controller is a variation of a popular PID (Proportional, Integral and Differential) controller. A proportional controller computes the control signal in proportion to the error, i.e., the difference between the measured performance of the controlled system and the specified reference performance. A proportional controller by itself cannot support the stability of the feedback control system [40]. An integrator is a low pass filter and it can support the stability of the closed-loop system. In this chapter, an integrator is employed to allow traffic bursts as discussed before, while supporting the stability of the closed-loop congestion control system. We do not use a differential controller, because it may show unreliable performance when workloads are highly dynamic. The time-driven congestion controller developed by Hollot et al. [19] is also implemented using a PI controller.

We formulate the PI Controller as a transfer function, which characterizes the behavior of a closed-loop system [40] in the discrete time $z$ domain rather than in the continuous time $s$ domain, because most of computational systems, such as routers, usually work in the discrete time domain. Further, the notation $\tilde{z}$ is used instead of $z$ in the following PI transfer function to emphasize that the discretization has been made in the spatial domain, instead of the time domain:

$$ C(z) = \frac{\alpha(z - \beta)}{z - 1} \quad (4.10) $$

where $\alpha = K_p (K_i + 1)$, $\beta = \frac{1}{K_i + 1}$. $K_p$ and $K_i$ are the proportional and integral control gains. To tune the control gains, $K_p$ and $K_i$, to support the stability of the closed-loop system for AQM, we apply the Root Locus design method [40], in which one can graphically tune feedback control gains. For more details about the Root Locus method, readers are referred to [40].
4.8 Performance Evaluation

We compare the performance of EDC to five baseline approaches: (1) RED [13] with the ‘gentle’ parameter turned on, (2) PI [19], (3) PIP [17], (4) AOPC [49] and (5) FLC [10] congestion controllers. For performance evaluation, we simulate 10 clients and a server connected by a gateway in OMNeT++ [37]. As shown in Figure 4.2, each client is connected to the gateway with a 10 Mbps link. The 1.5Mbps link between the gateway and the server constitutes a bottleneck. Our simulation starts with 60 ftp and 180 http applications and continues until 100s. At 100s, 60 more ftp applications are activated in order to observe how the tested approaches for AQM react to the abrupt load increase. The simulation ends at 200th second.

For the tested approaches, we set the congestion control parameters by following the recommendations provided in the original papers for each controller, while tuning their control gains to obtain best performance for each and support their stability. Parameter settings are as follows:

- **Gentle RED:**
  - Minimum queue length threshold: 150 packets
  - Maximum queue length threshold: 300 packets
  - Averaging weight \(wq\) in [13]): 0.00133

![Test Bed](image-url)
– Maximum drop probability: 0.1

• **PI Controller**
  – Reference (set-point) queue length: 200 packets
  – Sampling frequency: 160 Hz (as recommended in [19])
  – $K_p = 0.0015$ and $K_i = 0.0003$ (tuned via the Root Locus design method [40])

• **PIP**
  – Reference queue length: 200 packets
  – Sampling frequency: 160 Hz
  – $K_h = 0.0021$, $\tau = 0.47$ (tuned via the Root Locus design method [40])

• **AOPC**
  – Reference queue length: 200 packets
  – Sampling frequency: 160 Hz

• **FLC**
  – Reference queue length: 200 packets
  – Sampling frequency: 160 Hz
  – $K_e = 0.05$ and $K_{\delta e} = 0.01$ (tuned via trial and error)

• **EDC:**
  – Reference queue length: 200 packets
  – Event threshold: 30 packets
  – $K_p = 0.03$ and $K_i = 0.01$ (tuned via the Root Locus design method [40])

For performance analysis, we show (1) the average and transient queue length, (2) E2E delay, (3) number of packet drops and (4) controller activation frequencies. Since the measured link utilization is almost 100% for all the tested approaches, we do not plot it.
Figure 4.3 shows the transient and average queue lengths of all the baselines and our controller, EDC. From the figure, we observe that RED keeps the average queue length between the specified minimum and maximum thresholds. Also, all other time domain controllers and our EDC maintain the average queue length near the specified reference queue length as shown in Figure 4.3.

As shown in Figure 4.3, RED shows big transient overshoots (in terms of the queue length) at the beginning. It cancels the initial overshoots and then maintains the transient queue length between 150 and 300 packets. However, RED’s reaction to the abrupt load increase at 100s is much slower than all other controllers’ reactions as shown in Figure 4.3.
In Figure 4.3, PI and PIP Controllers show a transient overshoot of approximately 320 packets and 285 packets at the beginning, respectively. On the other hand, AOPC, FLC and EDC show a very small transient overshoot, which is almost negligible. All the approaches, except RED, maintain the queue length at desired set-point of 200. They also converge to the set-point substantially faster than RED does. However, PI and PIP controllers show considerably larger transient queue length fluctuations around the set-point than EDC does. When the load is increased suddenly at 100s, all baselines especially RED, PI and PIP controllers show overshoot while EDC maintain the desired queue length even in presence of this abrupt workload change since its activation rate is quickly adapted to the workload.

**Fig. 4.4. E2E Delays**
Figure 4.4 shows the average and transient E2E delays for all the baselines and EDC. In Figure 4.4, the average E2E delay of RED is approximately 0.75s, whereas all the other baselines and EDC maintain the average E2E delay around 0.5s. Further, from the transient E2E delay curves in Figure 4.4, we observe that EDC achieves smaller E2E delay fluctuations than RED, PI and PIP controllers, while achieving almost equal E2E delay fluctuations as AOPC and FLC.

**Figure 4.5. Packet Drop Rates**

Figure 4.5 shows the average and transient packet drop rates measured in terms of the number of packet drops/s for all the baselines and EDC. In average, RED drops approximately 30 packets/s, whereas all the other baselines and EDC drop approximately...
50 packets/s. RED drops a smaller number of packets, since its queue length is generally longer than the reference queue length of other baselines and EDC as shown in Figure 4.3.

![Graph of RED and EDC activation frequencies](image)

**FIG. 4.6. Activation Frequencies**

Finally, we compare the performance of the tested AQM schemes in terms of the activation frequency, i.e., the number of control algorithm executions per second. Figure 4.6 shows the activation frequencies of RED and EDC. PI, PIP, AOPC and FLC’s activation frequency is fixed at 160Hz following [19]. From the figure, we observe that EDC has the smallest transient activation frequencies among the tested approaches. In average, EDC is activated only approximately 8 times per second and RED is activated approximately 30 times/s. As shown in Figure 4.6, RED and EDC increase their activation frequencies upon the abrupt load increase at 100s and then decrease the frequencies as the load becomes stable after 100s.

### 4.9 Summary

Active Queue Management (AQM) is investigated to avoid incipient congestion in gateways to complement congestion control run by the transport layer protocol such as the TCP. To seamlessly integrate the advantages of both event-driven and control-theoretic time-driven approaches, we present an event-driven feedback control approach for AQM based on formal control theory. As our approach is based on a mathematical model, its performance is more predictable than ad hoc event-driven approaches are. Also, it is more
reactive to dynamic load changes than time-driven approaches due to its even-driven na-
ture. We analyze the stability of the open-loop model and tune the event-driven controller
to support the stability of the closed loop system by applying formal control theoretic tech-
niques. Our simulation results show that our event-driven controller effectively maintains
the queue length around the specified set-point. It generally achieves shorter E2E (end-
to-end) delays and smaller E2E delay fluctuations than the baselines. Further, our AQM
algorithm is invoked much less frequently than the tested baselines. In the future, we will
continue to investigate more cost-effective approaches for active queue management.
Chapter 5

Related Work

5.1 Robust fuzzy CPU utilization control for dynamic workloads

Feedback control has been applied to manage the real-time system performance in dynamic environments. A number of existing approaches for feedback control of real-time performance such as [3, 31] mathematically model real-time system behaviors via difference equations. To apply classical linear control theory, real-time system behaviors are approximated in a piecewise linear manner. As control gains are determined offline, however, these approaches may fail when workloads or system behavioral characteristics deviate from the ones used for offline modeling. Many existing linear control theoretic approaches share this problem [1, 16].

Our QoS adaptation scheme via task period adaptation is similar to [5]. Abeni et al [2] takes an alternative approach where the task budget rather than the task period is adapted under overload. Our approach can be integrated with the adaptive reservation scheme [2] too. This is reserved for future work.

Constrained predictive control techniques are applied to control the CPU utilization in a multiprocessor environment [32, 50]. Self-tuning regulators based on adaptive control theory [4] estimate the system model for automatic tuning of the controllers to manage the performance of e-commerce servers [21]. It is shown that a self-tuning regulator can converge to the target performance, if a set of conditions are met [21]. Adaptive control is also applied to differentiated web caching services [33]. However, model predictive control and adaptive control approaches are subject to online modeling errors. Therefore, they can
only handle moderate nonlinearity. In contrast, fuzzy logic control is very effective to manage the performance of nonlinear, complex systems due to the model-free nature [39].

Fuzzy control theory has been applied to maximize the profit in an e-mail server [7]. eQoS [54] applies fuzzy control theory to differentiate services in a web server. However, they do not consider real-time constraints.

Little prior work has been done to apply fuzzy control theory to real-time performance management. Li et al [26] apply fuzzy control to visual tracking; however, they do not consider the utilization control problem. Further, they do not analyze the stability of the fuzzy closed-loop system. Suzer et al [45] have developed a fuzzy utilization controller. However, this study presents a more advanced fuzzy rule-base to reduce potential overshoots and undershoots. Further, [45] does not provide stability analysis and evaluates performance via simulation.

5.2 Bandwidth Consumption Control and Service Differentiation for Video Streaming

Our bit rate management and service differentiation schemes are implemented atop QStream [25]. QStream supports real-time video streaming by EDF (earliest deadline first) scheduling of frames based on their playtime deadlines and synchronization between the client and server. qVSF significantly extends QStream to support the specified bandwidth limit and service differentiation.

A fuzzy controller [24] determines the emitted QoS level of video streams. The error value fed into the fuzzy controller is the difference between the stream quality emitted from the server and the stream quality perceived at the client. They aim to find the point where the difference between the emitted QoS level and the perceived one is zero. However, bit rate control and service differentiation are not considered in their work.

A fuzzy logic based cross-layer video streaming solution for wireless media is presented in [47]. The feedback is received from the underlying network stack to change the bit rate of a video according to link conditions. However, their work only considers a single
stream. Also, it only considers to perform fuzzy logic control and QoS adaptation at the source node that encodes the video.

Stok et al. [46] propose a system that can react to network condition variations in a wireless home network. They use TCP for streaming and apply priority based rules to delete lower priority packets from the buffer if the buffer overflows. Resource allocation is a complementary problem to our work presented in this chapter and it is reserved for our future work.

Layered encoding used for QoS adaptation in this chapter is similar to the milestone approach [28], in which a real-time task consists of a mandatory part and optional parts. The mandatory part, e.g., the base layer of a frame in our approach, should always be executed to support the minimum QoS. Optional parts, e.g., enhancement layers, can be omitted, if necessary, to meet deadlines under overload. The QoS monotonically increases as more optional parts are executed.

Classical linear control theory has recently been applied to performance management of software, especially a web server [1]. Model predictive control theory and adaptive control theory have been applied to manage the performance of a real-time middleware [51] and web server [22]. Fuzzy control theory is applied to maximize the profit in an email server [7] and to differentiate the service in a web server [54]. However, these approaches do not consider real-time streaming of videos with inherently nonlinear, unpredictable complexities.

5.3 Active Queue Management via Event-Driven Feedback Control

Most existing work on AQM can be categorized in two major classes: (1) ad hoc event-driven approaches and (2) time-driven feedback control approaches. Ad-hoc approaches are event-driven; however, they neither have a mathematical model nor apply formal feedback control techniques. Other approaches rely on a mathematical model of the TCP and queue dynamics to perform congestion control. They use linear (P, PI, or PID) and nonlinear (fuzzy logic) control techniques based on fixed-interval sampling and con-
trol algorithm execution for congestion control. Unlike these approaches, our work is not only event-driven but also based on formal control theoretic techniques. We discuss a few representative existing approaches from the two camps that are closely related to our work in the following.

**Ad Hoc Event-Driven Approaches.** RED [13] controls the queue length by monitoring the average queue length and manipulating the packet drop/mark probability, if necessary, to avoid congestion. Although RED is very effective, it has several parameters to tune. Further, tuning is not always straightforward. Lin et al. proposed Flow Random Early Drop (FRED) [27]. They evaluate the effectiveness of RED over traffic types categorized as nonadaptive, fragile and robust, according to their responses to congestion. They point out that RED allows unfair bandwidth sharing when a mixture of the three traffic types shares a link. FRED uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow’s buffer use. Ott et al. proposed Stabilized RED (SRED) [38] to statistically estimate the number of active flows at a link and also identify misbehaving flows. SRED uses the estimated number of active flows and the instantaneous queue size to calculate the packet-dropping probability. Feng et al. developed a self-configuring version of RED [9]. The authors suggest using an on-line algorithm for dynamically changing RED parameters according to the observed traffic. The authors show that this mechanism can reduce packet losses, while maintaining high link utilization. Feng et al. proposed BLUE [8] as a congestion control algorithm to be deployed in gateways. By considering packet loss and link idle events to manage congestion, BLUE significantly outperformed RED.

**Time-Driven Feedback Control Approaches.** Misra et al. modeled the interactions of TCP flows and AQM routers by using stochastic differential equations [35]. Using this model in their simulations, they analyzed the impact of RED parameters on the network performance. This mathematical model constitutes a basis for a number of projects regarding AQM. The authors of [19], [56], [17], [49] and [10] used this model to develop control-theoretic approaches for AQM. Hollot et al. analyzed RED in control theoretic aspects [18] using the mathematical model developed in [35]. Using this model, they also
developed a P and a PI controller in a companion paper [19]. Zhang et al. proposed an online self-tuning structure [56] based on the mathematical model [35] to estimate and correct network parameters online. Accompanied by a PI controller, this structure is used to manage the queue length. Heying et al. [17] developed a novel algorithm, called Proportional Integral based series compensation and Position feedback compensation (PIP), to manage the queue length. They used the model developed in [35] to implement their approach. Wang et al. [49] proposed an optimized version of the mathematical model in [35] and they built the Adaptive Optimized Proportional Controller (AOPC). AOPC measures the latest packet loss ratio and uses it as a complement to the queue length in order to dynamically adjust the packet drop probability. It measures an additional state information, i.e., latest packet loss ratio, to enhance the control performance. Fengyuan et al. [10] built a fuzzy logic controller based on the model developed by Misra et al. [35] to manage the queue length. Their approach is inspired by the fact that fuzzy control theory [39] is more versatile than the classical control theoretical solutions in the presence of nonlinear system behaviors. However, their approach is also time-driven.
Chapter 6

Conclusions and Future Work

Computer systems have limited amounts of resources to serve applications’ growing demands. Most systems tend to allocate resources to applications by offline analysis of application requirements, which often results in inefficient resource usage due to the dynamic, time-varying nature of workloads. Formal control theory is known to effectively support the desired performance of the controlled system. However, it is challenging to support the performance of computational systems, since there is no definitive methodology to model computer system dynamics unlike physics laws applied to model physical systems such as a cruise control system. Modeling computer systems, selecting proper control theoretic tools and tuning them according to the needs of specific applications—the research problems to be investigated in this proposed work—are the key ingredients for successful application of control theory to computer system performance management. To support desired system performance even in the presence of dynamic workloads, we have applied advanced control theoretic approaches, namely fuzzy control theory, model predictive control theory and event-driven control theoretic techniques. Specifically, we apply these techniques to manage the CPU utilization in a real-time operating system, network bandwidth consumption for video streaming, and link congestion in network gateways.

First, to reduce the difficulty of modeling real-time systems with stringent timing constraints, we apply formal fuzzy control theory that is very effective to support the desired performance in a nonlinear dynamic system without requiring a system model. We support direct nonlinear mappings between the utilization error (= target utilization – current utilization) and the workload adjustment required to achieve the target utilization via IF-
THEN rules. Rather than relying on an approximate system model, we develop a novel fuzzy closed-loop system to control the utilization based on the logical understanding of the relation between the workload and utilization changes. Via the Lyapunov direct method [4, 39], we mathematically prove the stability of the fuzzy closed-loop system. Further, we extend the Real-Time Application Interface (RTAI) for Linux kernel [43]. We implement and evaluate our fuzzy logic utilization controller as well as two existing utilization controllers based on the linear and model predictive control theory for an extensive set of workloads in RTAI. Among the tested approaches, our approach shows the smallest deviation from and the fastest convergence to the specified utilization set-point when the system is in a transient status.

Second, we applied fuzzy control theory to network bandwidth consumed by multimedia streaming. In this way, we aim to avoid undesirable situations in which multimedia streaming starves other applications such as file transfer sharing the network, for example, in a smart home. To bound the bandwidth usage of streaming, we leverage the layered encoding technique, in which a video frame consists of a base layer and multiple enhancement layers. We always transmit the base layer, because it is required to display a scene. However, under overload, we degrade the video quality by dropping high enhancement layers without affecting the underlying layers, if necessary, to support the specified bit rate bound. We have implemented our transmission rate control and service differentiation schemes on top of an open source video streaming server, QStream [42] and evaluated by running experiments across the shared department network in the Department of Computer Science at SUNY Binghamton where a considerable number of different applications usually coexist at the same time. Performance evaluation results show that our video streaming system can support the specified bit rate bound and differentiate the service to efficiently utilize the limited bandwidth without severely degrading the visual quality of low priority video streams.

Third, we applied formal event-driven control theory to address the congestion control problem in gateways. Most existing work on AQM is either ad-hoc event-driven feedback approaches or time-driven formal control theoretic approaches. To integrate the advantages
of both event-driven and time-driven control-theoretic approaches, we present an event-
driven feedback control approach based on formal control theory [15, 40]. The key idea of
our approach is to design a feedback-based congestion controller that is invoked upon the
arrivals of a specified number of packets rather than being invoked at every fixed sampling
period. As our approach is based on a mathematical model, its performance is easier to
analyze and more predictable than ad-hoc event-driven approaches are.

We thoroughly evaluate the performance of our approach via an extensive simulation
study in OMNeT++ [37]. We compare it to five advanced approaches for AQM. Our sim-
ulation results show that our event-driven controller effectively maintains the queue length
around the specified set-point. It achieves shorter E2E (end-to-end) delays and smaller E2E
delay fluctuations than several existing AQM approaches while achieving almost the same
E2E delays and E2E delay fluctuations as the two other advanced control theoretic AQM
approaches. Further, our AQM algorithm is invoked much less frequently than the tested
baselines. Therefore, it saves precious resources at routers.

In summary, I aim to demonstrate the applicability of formal control theoretic tech-
niques to support desired system behaviors even in the presence of dynamic workloads
and uncertain environments. In this way, I intend to improve the predictability and reli-
ability of computer systems that need to process highly dynamic workloads in uncertain
environments.
References


