An Approach for Quality of Service Management

Chen Lee and Dan Siewiorek
October 1998
CMU-CS-98-165

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a translucent QoS management optimization framework for systems that must satisfy application needs along multiple dimensions such as timeliness, reliability, cryptographic security and other application-specific quality requirements. The architecture of the system consists of a semantically rich (in terms of customizable and expressiveness) QoS specification interface for multi-dimensional QoS provisioning, a quality-of-service index model to help the user make the quality trade-off decision, and a unified QoS-based admission control and resource planning system. The

semantically rich QoS specification interface allows the user or system administrator to define fine-grained service requests in terms of quality or rate of service. The QoS index model is designed to be flexible and policy driven. The unified QoS-based admission and resource control facilitates the deployment of various QoS policies to meet performance objectives for specific service optimizations. Finally, the overall architecture enables us to quantitatively measure QoS, and to analytically plan and allocate resource.

This work was supported in part by the Defense Advanced Research Projects Agency under agreements N66001-97-C-8527 and E30602-97-2-0287.

Keywords: QoS management and optimization, QoS index and utility model, QoS tradeoff, resource tradeoff, multi-media

1 Introduction

Quality of Service (QoS) control is considered an important user demand and therefore receives wide attention, especially in the areas of computer network and real-time multimedia system research, and in commercial markets as well.

Typically, service characteristics in existing systems are fixed when systems are built, therefore they often do not give users any real influence over the QoS they can obtain. On the other hand, multimedia applications and their users can differ enormously in their requirements for the rate of services and resources available to them at the time of use of the application systems. Therefore there is an increasing need for customizable services that can be tailored for the end users' specific requirements.

In the meantime, new and improved systems [Pro96] are placing more and more complex demands on the quality of service that are reflected in multiple criteria over multiple quality dimensions. These QoS requirements can be objective in some aspects and subjective in others. Moreover, because of the manifold and subjective nature of user quality demands, it is very hard to measure whether the provided quality fulfills the stated demands without guidance and input from end clients.

One issue is the *QoS Tradeoff* where a user of an application might want to emphasize certain aspects of quality, but not necessarily others. Users might tolerate different levels of service, or could be satisfied with different quality combination choices, but the available system resource might only be able to accommodate some choices but not others. In situations where a user is able to identify a number of desirable qualities and rate them, the system should be able to reconcile these different demands to maximize the user's preference and to make the most effective use of the system. So it is important for a system to provide a large variety of service qualities and to accommodate specific user quality requirements and delivery as good service as it can from the users' perspective.

A similar issue to QoS Tradeoff is *Resource Tradeoff*. In this case the tradeoff refers to the reconciling or balancing of competing resource demands. Resource Tradeoff is often transparent to the user but can be of great help in accommodating user requirements including QoS Tradeoff, especially when the availability of several different resources is not balanced. It arises when an application is able to use an excess of one resource, say CPU power, to lower its demands on another, say network bandwidth, while maintaining the same level of QoS. For example, video conferencing systems often use compression schemes that are effective, but computationally intensive, to trade CPU time for network bandwidth. If the bandwidth is congested on some intermediate links (which is often the case), this benefits the system as a whole. In the case of a mobile client with limited CPU and memory capacity but sufficient link speed with a nearby intermediate powerful server, the computational expensive speech recognition, silence detection and cancelation, and video compression could be carried out on the nearby server. For proxy servers which act as transcoders/transceivers besides caching data, the proxy servers can distill data for low bandwidth clients (when both server and client have fast CPU, memory and disk bandwidth, but the network link speed in between is limited).

The rest of the paper is organized as follows: Section 2 gives an overview of the QoS management optimization system. Section 3 classifies and formulates our QoS optimization problems. Section 4 discusses the user specification interface and mechanisms for QoS specification acquisition. Section 5 discusses the design principles for the QoS optimization. Section 6 and 7 presents various optimization algorithms for solving our QoS management optimization problems.

2 System Overview

We have proposed a translucent QoS framework[Lee97][RLLS97] for QoS management in systems that must satisfy application needs along multiple dimensions such as timeliness, reliability, cryptographic security and other application-specific quality requirements. The architecture consists of a semantically rich (in terms of customizability and expressiveness) QoS specification interface for multi-dimensional QoS provision, a quality-of-service index model to help the user make the quality trade-off decision, and a unified QoS-based admission control and resource planning system.

The semantically rich QoS specification interface allows the user or system administrator to define fine-grained service requests in terms of quality or rate of service. The QoS index model is designed to be flexible and policy driven. The unified QoS-based admission and resource control facilitates the deployment of various QoS policies to meet performance objectives for specific service optimizations. Finally, the overall architecture enables us to quantitatively measure QoS, and to analytically plan and allocate resources.

The system can be used to continuously monitor and adjust clients' level of service in light of the dynamically changing operational environment (of clients and resources). We also develop quality measures and indicators that will enable the system to emphasize quality, effectiveness and efficiency in the delivery of the services.

Our QoS specification allows applications and users to put values on the different levels of service that the system can provide. When "value" is taken literally, this means that our model is able to facilitate market-efficient resource distribution. Such a system has considerable potential, especially in solving bandwidth problems of the increasingly crowded Internet.

Figure 1 gives a pictorial view of our QoS management optimization system.

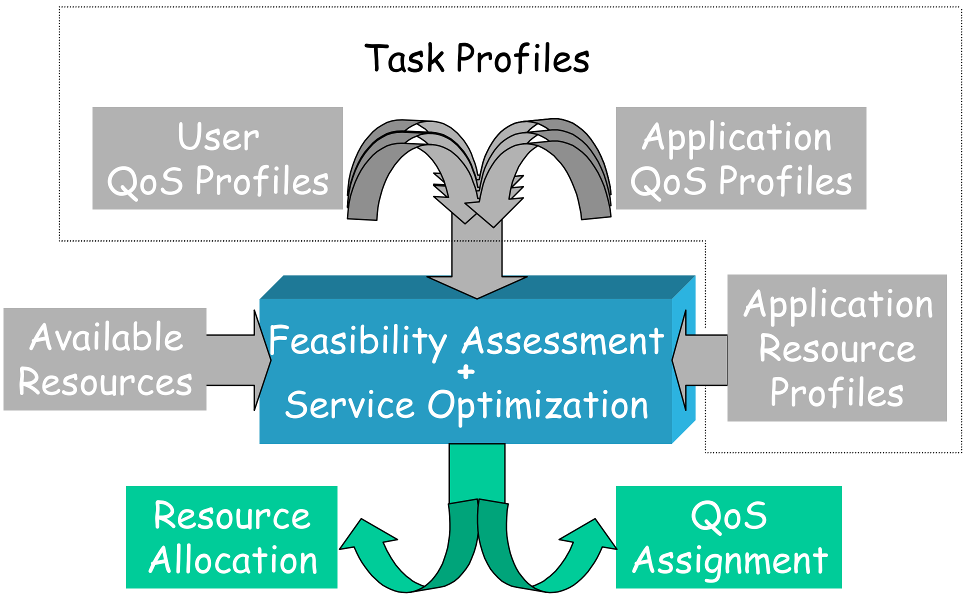

Figure 1: In and Out of the QoS Management Optimization Module

3 Problem Taxonomy, Modelling and Complexity

3.1 Problem Taxonomy

We classify the problem based on resources and QoS dimensions as follows:

- Single Resource Single QoS Dimension: SRSD
- Single Resource Multiple QoS Dimension: SRMD
- Multiple Resource Single QoS Dimension: MRSD
- Multiple Resource Multiple QoS Dimension: MRMD

Since SRMD is a superset of SRSD, and MRMD a superset of MRSD, SRMD and MRMD will be treated directly.

The reason for addressing SRMD is that we could develop efficient schemes that might not be easily achievable for MRMD. The schemes we have for SRMD readily lead us to a QoS-driven single resource allocation when only a single resource is of concern (either it is the only resource under consideration, or it is relatively more scarce and other resources are abundant). For instance, these schemes can be used for QoS-driven disk, memory, network bandwidth or other shared resource management and allocation, as well as for processor scheduling.

Since resource tradeoff is one of the main interests of our research, we will consider resource tradeoffs whenever applicable. That is, whenever multiple resources are involved, tradeoffs among them will be handled by default.

3.2 Problem Formulation

Consider a system with multiple independent applications and multiple resources. Each application, with its own quality-of-service requirements, contends with other applications for finite system resources. Let the following be given

$T_1, T_2, \ldots, T_n$	— tasks (or applications)
$R_1, R_2, \ldots, R_m$	— shared system resources
$Q_{i1}, Q_{i2}, \ldots, Q_{id_i}$	— quality-of-service dimensions for task T_i

Each R_i is a set of non-negative values representing the possible allocation choices of the ith shared resource. The set of possible resource vectors, denoted as R, is given by $R = R_1 \times \cdots \times R_m$. Each shared resource is finite, so we also have $r^{\max} = (r_1^{\max}, \ldots, r_m^{\max})$.

Similarly, each Q_{ij} is a finite set of quality choices for the ith task's jth quality-of-service dimension and we define the set of possible quality vectors by $Q_i = Q_{i1} \times \cdots \times Q_{id_i}$.

Associated with each T_i is an *Application Profile* and *User Profile.* An Application Profile comes from an application designer, while a User Profile provides user-specific quality requirement associated with each session. A user can either instantiate the attributes of the default application profile, by selecting one of many templates supplied with the application, or the user can supply their own merit and reward functions with respect to different levels of qualities.

Application Profile: consists of a *QoS Profile* and a *Resource Profile.*

- A *QoS Profile* consists of
  - Quality Space — Q_i
  - Quality Index — a bijective function

    $$f_{ij} : Q_{ij} \rightarrow \{1, 2, \ldots, |Q_{ij}|\}$$

    that preserves the ordering, i.e., if q_1 is "better than" q_2, then $f_ij(q_1) > f_ij(q_2)$. Since f_{ij} is bijective, it has an inverse $f_{ij}^{-1} : \{(y, x) \mid (x, y) \in f\}$, i.e. $f_{ij}^{-1} : \{1, 2, \ldots, |Q_{ij}|\} \rightarrow Q_{ij}$. For simplicity, in the rest of this paper we will also use Q_{ij} and q_{ij} to represent their corresponding f_{ij}–indexed quality sets and quality points. This should not cause any confusion as the context clearly determines whether the original quality specification or index value is under consideration.
  - Dimension-wise Quality Utility — $u_{ij} : Q_{ij} \rightarrow \mathbb{R}$
  - *Application Utilities* — a QoS or rate of service measure

    $$u_i : Q_i \rightarrow \mathbb{R}$$

    It could be defined as a weighted sum of u_{ij}

    $$u_i(q_i) = \sum_{j=1}^{d_i} w_{ij} u_{ij}(q_{ij})$$

    We require that u_i is non-decreasing in all d_i arguments and that it is non-negative.
- A *Resource Profile* for T_i defines a relation between R and Q_i:

  $$r \models_i q$$

  which describes a list of potential resource allocation schemes to achieve each quality point q.

Note that both R and Q_i have partial orderings which $\models_i$ must respect. That is, if $r_1 \models_i q_1$, $r_2 \models_i q_2$, and $r_1 > r_2$, then we will have $q_1 \not< q_2$. This partial ordering is required to ensure that utility is non-decreasing with respect to resources. In other words, more resources should not lead to reduced quality (and thus utility), which is reasonable and natural.

It is important to note that we can only define a relation but not a function between Q_i and R. For a given value of q, multiple resource allocation schemes could be used to achieve the same level of quality; likewise, for a given resource allocation, one could use the resource(s) to improve different QoS dimensions, which could yield different quality results. The scatter plot in Figure 2, which depicts a possible relation between resource and quality, might help us visualize this.

User Profile: mostly an instantiation of the Application Profile

- An Application QoS Profile provides a template, which a user could instantiate to create a User Profile. A user could also supply his/her own QoS Profile which supersedes those provided by the application. Furthermore, a User Profile could specify its QoS constraints.

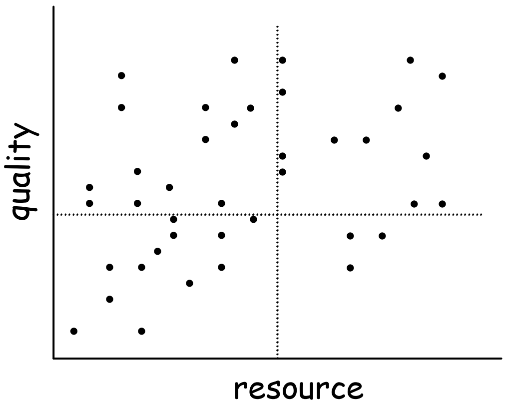

Figure 2: Scatter of Resource and Quality

- **QoS Constraint**: is the minimum QoS requirement specification

$$q_i^{\min} = (q_{i1}^{\min}, q_{i2}^{\min}, \ldots, q_{id_i}^{\min}).$$

  When the minimum requirements cannot be satisfied, the user of task T_i might choose not to run T_i at all.

  Alternatively we could let the user implicitly specify the $q_i^{\min}$ through utility functions by setting $u_i(q) = 0$ for all $q < q_i^{\min}$. We have yet to complete a user-interface study to decide whether this approach will compromise the simplicity of the user-interface. For now, we will use this QoS Constraint approach.

- A user might explicitly specify a cap, or *Saturation Point,* $q_i^{\max}$, on its quality requirement to indicate that further improvements beyond it are not likely to be perceived or appreciated. Similar to the discussion of $q^{\min}$ above, the maximum quality constraint could be handled by setting $u_i(q) = u_i(q_i^{\max})$ for all $q > q_i^{\max}$.

  To simplify our presentation, we did not explicitly use this aspect in the algorithms presented in this paper.

In the rest of the paper, the **Task Profile** will be used to represent the effect of using an application profile that has been instantiated by the user profile. The terms Application Profile, User Profile, and Task Profile might be used interchangeably which denotes an instantiated Application Profile with possibly added QoS Constraint and QoS Saturation Point.

For the overall system, with multiple applications possibly requiring multiple resources, we have the

System Utility $u : Q_1 \times \cdots \times Q_n \rightarrow \mathbb{R}$, which could be defined as:

- A (weighted) sum of *Application Utilities*

$$u(q_1, \ldots, q_n) = \sum_{i=1}^{n} w_i u_i(q_i)$$

  for *differential services,* where u_i is non-decreasing, and $0 \leq w_i \leq 1$ could be the priority[1] of T_i, or

[1] Note that the algorithms or schemes presented in this paper are for the weighted sum where the weights are set to 1 for simplification to present the algorithms.

- $u = u^*$, where

$$u^*(q_1, \ldots, q_n) = \min_{i=1\ldots n} u_i(q_i)$$

for *"fair" sharing.*

The goal is to assign qualities (q_i) and allocate resources (r_i) to tasks or applications, such that the system utility u is maximized. Therefore we have the following *Problem Function* formulation

$$\begin{array}{lllr}
\text{maximize} & u(q_1, \ldots, q_n) & & \\
\text{subject to} & q_i \geq q_i^{\min} \text{ or } q_i = 0 \ , & i = 1, \ldots, n, & \textbf{(QoS Constraints)} \\
& \sum_{i=1}^{n} r_{ij} \leq r_j^{\max} \ , & j = 1, \ldots, m, & \textbf{(Resource Constraints)} \\
& r_i \models_i q_i \ , & i = 1, \ldots, n. &
\end{array} \tag{1}$$

3.3 Problem Complexity

This combinatorial problem could also be formulated as follows. Let $\kappa_{i1}, \ldots, \kappa_{i|Q_i|}$ be an enumeration of the quality space, Q_i, for task T_i. Let $\rho_{ij1}, \ldots, \rho_{ijN_{ij}}$ be an enumeration of the resource usage choices (tradeoffs among different resources) associated with κ_{ij} for T_i, where N_{ij} is the number of such resource usage choices. (In particular we should always have $\rho_{ijk} \models_i \kappa_{ij}$.)

Let $x_{ijk} = 1$ if task T_i has been given quality point κ_{ij} and resource consumption ρ_{ijk}, and $x_{ijk} = 0$ otherwise.

$$\begin{array}{lll}
\text{maximize} & \sum_{i=1}^{n} \sum_{j=1}^{|Q_i|} \sum_{k=1}^{N_{ij}} x_{ijk} u_i(\kappa_{ij}) & \\
\text{subject to} & \sum_{i=1}^{n} \sum_{j=1}^{|Q_i|} \sum_{k=1}^{N_{ij}} x_{ijk} \rho_{ijk\ell} \leq r_\ell^{\max}, & \ell = 1, \ldots, m, \\
& \sum_{j=1}^{|Q_i|} \sum_{k=1}^{N_{ij}} x_{ijk} \leq 1 \ , & i = 1, \ldots, n, \\
& x_{ijk} \in \{0, 1\}, & i = 1, \ldots, n, \ j = 1, \ldots, |Q_i| \ , \ k = 1, \ldots, N_{ij}.
\end{array} \tag{2}$$

Note, that $\rho_{ijk\ell}$ is just the ℓth coordinate of the vector ρ_{ijk}.

Therefore all the instances of our problem can be viewed as special cases of the general (mix)Integer or Nonlinear Programming problems.

Proposition 1 *SRSD, SRMD, MRSD, and MRMD are all NP-Hard problems.*

Proof Since SRSD is a special case of the other three, we only have to show that SRSD is NP-Hard.

For SRSD, we have $m = N_{ij} = 1$ and thus $k = \ell = 1$. System (2) becomes

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{|Q_i|} x_{ij1} u_i(\kappa_{ij}) \\
\text{subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{|Q_i|} x_{ij1} \rho_{ij11} \leq r_1^{\max}, \\
& \sum_{j=1}^{|Q_i|} x_{ij1} \leq 1, \qquad i = 1, \ldots, n, \\
& x_{ij1} \in \{0, 1\}, \qquad i = 1, \ldots, n, \; j = 1, \ldots, |Q_i| \, .
\end{aligned}
\tag{3}
$$

The 0–1 Knapsack Problem is known to be NP-Hard [MT90]. It can be described as follows. Given a set of n items and a knapsack of capacity c, with p_i and w_i the profit and weight of item i respectively, select a subset of the items so as to

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\
\text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c \\
& x_i \in \{0, 1\}, \quad i = 1, \ldots, n,
\end{aligned}
\tag{4}
$$

We can therefore reduce the 0–1 Knapsack Problem to SRSD by setting

$$
\begin{aligned}
Q_i &= \{(1)\} \\
u_i(q_i) &= p_i \\
r_1^{\max} &= c \\
\rho_{i111} &= w_i
\end{aligned}
$$

and have the 0–1 Knapsack Problem's x_i represented by x_{i11} in the SRSD case. □

4 User Specification Interface for QoS Provision

At the crux of our translucent QoS management optimization system lies the QoS specification. First, it is important that we provide powerful and semantically rich QoS specifications that they system and the user can use for service optimization. Equally important we need to provide a *user friendly interface* that facilitates specification acquisition.

The reason for the emphasis on QoS specification and interface design might not be obvious, but the reader should see the point shortly as the quality dimensions of typical multimedia systems, QoS tradeoff and resource tradeoff issues are presented.

4.1 Quality Dimensions

We consider the following example quality dimensions, by no means exclusive, with their corresponding dimensional space ordered from worst to best:

- Cryptographic Security (encryption key-length) : 0(off), 56, 64, 128
- Data Delivery Reliability, which could be
  - maximum packet loss : in percentage
  - expected packet loss : in percentage
  - packet loss occurrence : in probability
- Video Related Quality
  - picture format[2]: SQCIF, QCIF, CIF, 4CIF, 16CIF
  - color depth(bits): 1, 3, 8, 16, 24, ...
    black/white, grey scale to high color
  - video timeliness — frame rate(fps): 1, 2, ..., 30
    low-frame-rate cartoon or animation to high motion picture video
- Audio Related Quality
  - sampling rate(kHz): 8, 16, 24, 44, ...
    AM, FM, CD quality to higher fidelity audio
  - sample bit(bits): 8, 16, ...
  - audio timeliness — end-to-end delay(ms): ..., 100, 75, 50, 25, ...
    (Note that we list these in worst-to-best order, not numerically increasing.)

The specification above contains ellipses ("...") to indicate that more choices could have been listed. Ignoring extra choices for a moment, the total number of different choices ("quality points") in the example can be calculated:

$$|Q_i| = \prod_{j=1}^{d_i} |Q_{ij}| = 4 \times 1 \times 5 \times 5 \times 30 \times 4 \times 2 \times 4 = 96{,}000$$

(A single option in data delivery reliability and 30 different frame rates were chosen for this example.) With these many quality points it would be completely out of the question to have the user specify the quality on a point-by-point basis. Therefore a pragmatic method is needed to address the issue.

4.2 Application Utility and QoS Tradeoff

Because QoS is often multi-dimensional, and because its measure could be objective or subjective (user or session dependent), a user might want to make some quality tradeoff, especially when resources (processing power or the link speed) on or between the end and intermediate nodes might dynamically change. For example, a user (or task T_i) might have a desired quality level, but be able to tolerate certain lower quality if there are insufficient resources to obtain the desired quality. It is therefore to the user's advantage for a system to provide an interface that allows the user to make implicit or explicit quality tradeoffs.

[2] The choices listed here come from [ITU95] [Say96]. Other standards, such as MPEG [MGF96] [LG91] [Say96] could have been used instead.

In our previous work [LRM96], specifically the RT-Phone (a video conferencing system) , we used a leveled QoS specification scheme of a simple one to ten scale. These scales are statically mapped to certain quality choice combinations, where many of the quality choices therefore left out (as we can think about it as a "QoS digitization"). At another level (for more involved users), individual knobs (sliding bars) were provided that a user could tune on each quality dimension. This second level had many fewer choices than listed above in Section 4.1. We therefore have extended our previous work, to allow each task T_i to specify its minimum acceptable quality ($q_i^{\min}$), saturation level ($q_i^{\max}$) and application utility function as part of the task profile (application profile instantiated with user profile). Thus our QoS management optimization engine will work most effectively to help each task achieve as high level of quality as possible, subject to resource constraints and the management policy deployed in the system.

Application utility functions are conceptually easy to imagine but difficult to construct. As pointed out in Section 4.1, it is clearly infeasible to make the user specify the utility of every quality choice on a point-by-point basis. There are simply too many choices. Instead, one could make the user specify the utility of selected points and then interpolate in order to get the utility of the rest. This might work well in the one-quality dimension case, but in the multi-dimensional case one would need a dense net of selected points and therefore again need too many points.

While we would like a user to provide the service optimization system with the translucent and semantically rich service requirement specification so that the optimization module can best accommodate the user's request, we also want to ensure that methods and mechanisms are in place in the system that will facilitate the delivery of these specifications from the user. In other words, we want to develop a measure and merit scheme as well as a reasonably user-friendly interface that will pose less of a burden on the user without sacrificing the semantically rich capability of the specification interface. Therefore a QoS index model is proposed from which dimension-wise quality utility functions are defined.

4.3 Quality Index

Certain quality dimensions, such as frame rate, have easily defined utility functions while others, such as picture format and end-to-end delay, are in non-numeric, non-uniform, or non-increasing order which require a quality to numeric mapping. Therefore the Quality Index is introduced, which maps qualities to indices.

The concept and the use of Quality Index is illustrated in the context of an example application.

Consider task T_i, which could be a video conferencing system. T_i's quality dimensions, quality space and Quality Index are as follows:

Picture format: Assume it uses the H263 [ITU95] standard format

Format:	SQCIF	QCIF	CIF	4CIF	16CIF
Quality Index:	1	2	3	4	5

The corresponding Quality Index is therefore $Q_{i1} = \{1, 2, 3, 4, 5\}$.

Color depth: Assume that T_i has 1, 3, 8, 16, and 24 bit color depths available for the user to choose.

Depth:	1	3	8	16	24
Quality Index:	1	2	3	4	5

Therefore $Q_{i2} = \{1, 2, 3, 4, 5\}$.

Frame rate: T_i allows frame rates ranging from 1 fps to 30 fps in steps of 1 fps. These will map directly onto $Q_{i3} = \{1, 2, \ldots, 30\}$.

Rate (fps):	1	2	...	30
Quality Index:	1	2	...	30

Encryption key length: For T_i, encryption will be either on with 56-bit encryption or off. Therefore we have $Q_{i4} = \{1, 2\}$.

Key length:	(none)	56-bit
Quality Index:	1	2

Audio sampling rate: Assume T_i provides audio sampling rates from AM-quality (8 kHz) to CD-quality (44 kHz).

Sampling rate (kHz):	8	16	24	44
Quality Index:	1	2	3	4

Thus we have $Q_{i5} = \{1, 2, 3, 4\}$.

Audio bit count: Assume that T_i provides only two sampling sizes, 8 bits and 16 bits.

Bit count:	8	16
Quality Index:	1	2

Therefore $Q_{i6} = \{1, 2\}$.

End-to-end delay: Assume that end-to-end delays ranging from 125 ms to 25 ms in steps of 25 ms. Since high numbers for end-to-end delay are worse than low numbers, $Q_{i7} = \{1, 2, \ldots, 5\}$ maps high number to low indices.

Delay (ms):	125	100	...	25
Quality Index:	1	2	...	5

4.4 Dimension-wise Utilities

Quality points in the multi-dimensional case do not have a complete ordering. The individual dimensions, however, do. Moreover, some common properties associated with dimensional quality utility are observed including: non-decreasing, often quasi-continuous and piecewise concave. Figure 3 depicts some typical utility function shapes.

Recall that the application utility u_i for T_i is defined in terms of the value accrued when T_i achieves a certain quality, i.e. $u_i : Q_i \rightarrow \mathbb{R}$. As discussed above, when many quality dimensions are involved, it is often very difficult for a user to express his/her quality preferences. We therefore provide the user with the capability to specify dimension-wise quality utilities. As a result, the application utility can then be defined as a weighted sum of dimension-wise quality. This creates an interesting issue regarding how weights should be assigned. Currently the Analytic Hierarchy Process (AHP) [Saa92a][Saa92b] model and the Simple Multi-Attribute Rating Technique (SMART) [Edw97] are used to cope with the problem.

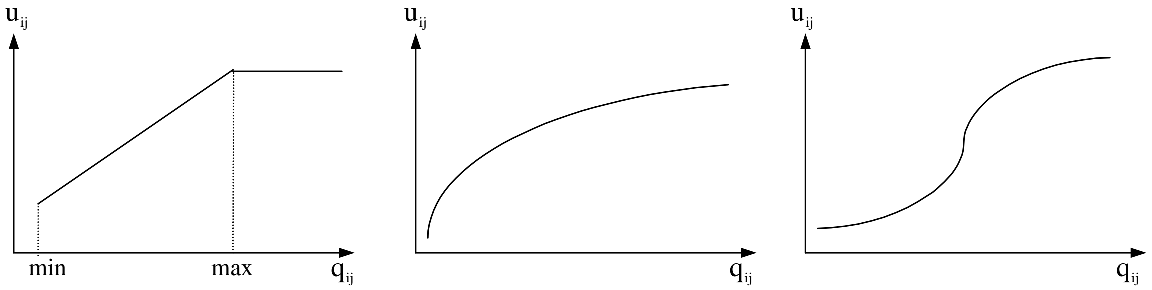

Figure 3: Typical Dimension-wise Utility Functions

Given the Quality Index, a dimension-wise utility could be defined and hence the application utility. Again, an example task profile is presented in the next subsection to illustrate the possible structure of dimension-wise utility functions and application utility functions.

4.5 Example Dimension-wise Utilities and Application Utilities

Recall that application utility u_i for T_i is defined as a weighted sum of the dimension-wise quality utilities.

$$u_i(q_i) = \sum_{j=1}^{7} w_{ij} u_{ij}(q_{ij})$$

where u_{ij} are the dimensional utility functions. Example definitions could be:

Function	**Comments**
$u_{i1}(q_{i1}) = 20q_{i1}$	Picture format: linear.
$u_{i2}(q_{i2}) = 100q_{i2}/3$	Color depth: linear.
$u_{i3}(q_{i3}) = 100(1 - e^{aq_{i3}+b})$	Frame rate: exponential decay, assume T_i achieves 50% at $q_{i3} = 5$ and 95% at $q_{i3} = 20$. Therefore $a = -0.1535, b = 0.0744$.
$u_{i4}(q_{i4}) = 20(q_{i4} - 1)$	Encryption: linear.
$u_{i5}(q_{i5}) = 100(1 - e^{-1.5q_{i5}})$	Audio sampling rate: exponential decay, T_i achieves 95% at $q_{i5} = 2$ or 16 kHz.
$u_{i6}(q_{i6}) = 50q_{i6}$	Audio sampling bits: linear.
$u_{i7}(q_{i7}) = 20q_{i7}$	End-to-end delay: linear, achieves 100% at the best quality point, $q_{i7} = 5$ or 25 ms delay.

Figure 4 depicts the utility curve described above for frame rate.

Suppose T_i is a remote surveillance system, where video is much more important to the user than audio. Assume that SQCIF, gray-scale, low frame rate is fine for video, and there is no need for encryption. Therefore, in the example system of section 4.3, we could have the following minimum quality specification

$$q_i^{\min} = (1, 1, 2, 0, 1, 1, 2)$$

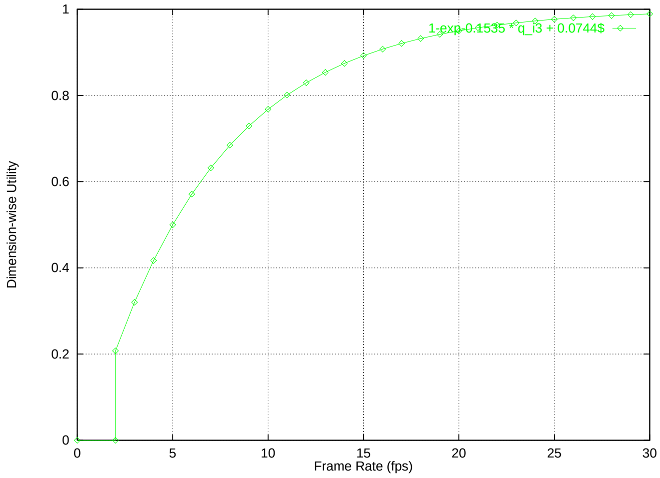

Figure 4: Dimension-wise Utility Function for Frame Rate

which corresponds to the following minimum quality

$$(\text{SQCIF}, 1\,\text{bpp}, 2\,\text{fps}, \text{no encryption}, 8\,\text{kHz}, 8\,\text{bps}, 100\,\text{ms}).$$

Since video is more important to the user than audio, an example application utility function for T_i could be:

$$u_i(q_1, \ldots, q_7) = 5\Big(\underbrace{u_{i1}(q_{i1}) + \cdots + u_{i4}(q_{i4})}_{\text{video}}\Big) + 1\Big(\underbrace{u_{i5}(q_{i5}) + \cdots + u_{i7}(q_{i7})}_{\text{audio}}\Big)$$

where video quality is weighted five times more than that of audio.

4.6 User Interface Consideration

If a user were to choose quality on a scale of 1 to 10 with some pre-determined quality choices preset by the system, the interface would be very simple.

A more flexible, but also more complicated, scheme would be to have a set of parameterized utility curves available for each quality dimension, and to have the user pick the curves and instantiate appropriate parameters/coefficients. In our system, the instantiation is carried by letting the user graphically specify *Satisfaction Knee Point parameters*. For the exponential-decay used in the previous example ($u_{i3}(q_{i3}) = 1 - e^{aq_{i3}+b}$), the user could specify the 50% and 95% levels. This is enough to uniquely determine a and b. For example, a user could specify (5fps, 0.50) and (20fps, 0.95), and the corresponding utility curve would then be the one shown in Figure 4, with $a = -0.1535$ and $b = 0.0744$.

It would be ideal to have an interface that could help the user digitize the quality to a certain range of scale, and acquire the corresponding utility accordingly. One way could be to move the dimension-wise utility function method to the user interface part to synthesize or digitize quality-utility data, as it could significantly reduce the quality space searched by the QoS management optimizer.

5 Issues on Algorithm Choice and Methodology

5.1 Algorithm Design Issues — Solution Quality vs. Computational Complexity

As is shown in [LS98] the QoS management optimization problems are NP-Hard. As a consequence, there are no optimal solution techniques other than an (possibly complete) enumeration of the solution space. On the other hand, QoS management calls for on-line solutions as the optimization module will ideally be in the heart of an admission control and adaptive QoS management system. Therefore the goal is to strike the right balance between solution quality and computational complexity.

For more than two decades, many researchers from the fields of mathematics, computer science and operations research have been working on the combinatorial optimization and solving NP-Hard problems. There are three algorithmic approaches [AL97] [MT90] that have been well studied and widely used :

- **enumerative methods** that are guaranteed to produce an optimal solution [Iba87a][Iba87b],
- **approximation algorithms** that run in polynomial time [Sah75][IK75], and
- **heuristic techniques** (under the general heading of local search) that do not have *a priori* guarantee in terms of solution quality or running time, but provide a robust approach to obtaining a high-quality solution to problems of a realistic size in reasonable time [AL97].

An important attribute is the incremental and state-reuse property of a scheme, so as to avoid having to completely redo expensive computations to accommodate the dynamic arrival and departure of tasks. Also, we ensure that all algorithms should be formulated so that the the search for an optimal solution can be terminate at any time while still reaching a *feasible,* but sub-optimal and hopefully good, solution. These two properties are essential for an algorithm to be used in an online (or near-online) environment.

Therefore a series of schemes have been developed that give approximation, approximation with bound, and exact solutions, with increased asymptotic computational complexity. These algorithms use various optimization techniques including linear and nonlinear programming, constraint relaxation, basic dynamic programming, branch-bound, advanced dynamic programming with addition of dominance rules, direct and local search schemes. In addition parallel algorithms are being developed to speed up the computation process.

It will be necessary to conduct extensive empirical studies to evaluate the practical performance of these algorithms when deployed under different system setups and task profiles. For instance, the systems to which QoS management optimization engine could be deployed could range from an end-node multi-media workstation, small or medium scale proxy/transceiver[3] servers, medium or large (with firewall and routing capability) gates [Lab97], and on-demand media (news, video, stock quote, game) servers. These studies allow comparison of the relative performance of the the algorithms and answer questions such as whether algorithms are robust [Pis95] enough to cover multiple cases, or whether combination algorithms might prove useful. In the latter case, the QoS management

[3] Data distillation for low-link-speed mobile or other clients for instance.

optimization engine could fire an algorithm based on the particular data instance exhibited by the profiles of application/user sessions in the system.

Another important issue, which is policy-dependent but would affect the actual algorithm design, is the stability of the task quality assigned to existing tasks in the system. In the case where policy requires that quality not degrade for certain tasks, some algorithms might not be suitable , while others might be more appropriate.

5.2 Resource-Utility Scatter to Graph Structure Composition

Due to the multi-dimensional and potentially subjective nature of quality of services, there is often no complete ordering among quality-of-service points, even for individual tasks. Only a scatter for R and U can be drawn shown in Figure 5. So some structural composition is required for those algorithms that call for mapping from resource to utility. Specifically, an *R-U* (Resource to Utility) function/graph is constructed for each task through *QoS Profile* and *Resource Profile.* An *R-U* graph can be constructed by listing each valid quality point's resource usage and its corresponding utility.

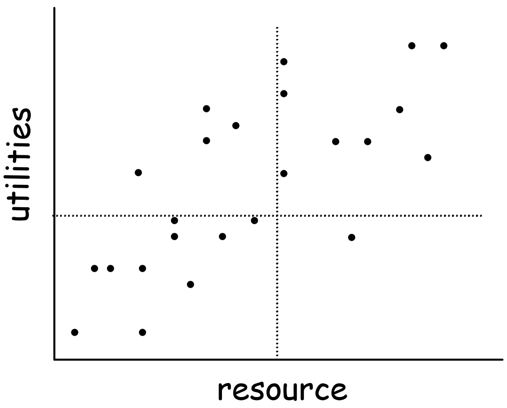

Figure 5: Scatter of Resource and Utilities

Recall that given a resource allocation to a task, one could use the resource to improve different QoS dimensions, which could therefore lead to different utility values. But the most valued QoS point for each resource value can be picked, as intuitively, we certainly want to assign resources to those quality points with the highest utility value.

We therefore define a function $g_i : R \rightarrow \mathbb{R}$, such that

$$g_i(r) = \max\{\, u_i(q) \mid r \models_i q \,\} \tag{5}$$

and define $h_i : R \rightarrow \mathcal{P}(Q_i)$ to retain the quality points associated with the utility value $g_i(r)$:

$$h_i(r) = \{\, q \in Q_i \mid u_i(q) = g_i(r) \wedge r \models_i q \,\} \tag{6}$$

Then an *R-U* graph can be generated for each task, each of which would be a step function (perhaps with multiple level of steps).

6 SRMD Algorithms

6.1 An Approximation Scheme for SRMD

By constructing the convex hull for each of g_i (see Definition 5) functions we get piece-wise linear relaxation functions g_i°, $i = 1 \dots n$. The gradients of of g_i° can be used as a heuristic to allocate resources among these tasks[4].

Let

$$C_i = \left\langle \begin{pmatrix} u_{i1} \\ r_{i1} \end{pmatrix}, \dots, \begin{pmatrix} u_{ik_i} \\ r_{ik_i} \end{pmatrix} \right\rangle$$

be the utility function g_i's discontinuity points in increasing r-order (therefore increasing u-order as well), and we will refer it as r-u-pair list. Denote by r^c the current remaining resource capacity after certain resource has been allocated; s_list[i].t, s_list[i].r, s_list[i].u the task id, the associated r-value and u-value of the corresponding r-u-pair list; and r[i] the resource allocated for T_i.

```
approx_srmd1(n, C_1, ..., C_n)
 1.  for i = 1 to n do
 2.     C'_i := convex_hull_frontier(C_i)
 3.     u[i] := 0
 4.     r[i] := 0
 5.  s_list= merge(C'_1, ..., C'_n)
 6.  r^c := r^max
 7.  u := 0
 8.  for j = 1 to |s_list| do
 9.     i := s_list[j].t
10.     β = s_list[j].r − r[i]
11.     if (β ≤ r^c) then
12.        r^c := r^c − β
13.        r[i] := s_list[j].r
14.        u[i] := s_list[j].u          /* Update allocation info for T_i. */
15.     else
16.        break
17.  for i = 1 to n do
18.     q[i] := h_i(r[i])               /* See Definition 6. */
19.     u := u + u[i]
20.  return (q[1], ..., q[n], u)
```

Note that each $q[i]$ provides a set of quality choices from which T_i (its user, or session manager) could choose to make further QoS tradeoffs.

Notice that in implementation, we actually replace "break" in line 16 with continue (i.e., let the loop continue when condition at step 11 does not hold). This means that after the optimal condition is violated, the residual capacity (r^c) will be greedily filled. The continuation can be thought as a post-optimization process. The error bound property to be proved below holds for either cases.

[4] The algorithm and analysis in this section is a clarification and a slight improvement over a similar algorithm described in [RLLS98].

Let $L = \max_{i=1}^{n} |C_i|$. After the procedure *convex_hull_frontier*[5] (which takes time $O(nL)$) a convex hull frontier with non-increasing slope segments (piece-wise concave) is obtained for each task. The segments are merged at step 5 using a divide-and-conquer approach with $\log_2 n$ levels, each level has nL comparisons. Merging thus takes time $O(nL \log n)$. Steps 8 through 16 require $O(|s\_list|) = O(nL)$. Steps 17 through 19 take $O(n)$. The total running time of the algorithm is thus $O(nL \log n) + O(nL) = O(nL \log n)$.

Denote by δ_i the maximum utility difference between adjacent discontinuity points of C_i', i.e., the largest increase in utility for task T_i on the convex hull frontier. Let $\chi = \max_{i=1}^{n} \delta_i$. Denote by U_{opt} the optimal utility result and U_{srmd1} the approximation result obtained by algorithm *approx_srmd1*.

Proposition 2 *U_{srmd1} is within χ of U_{opt}, i.e. $U_{opt} - \chi < U_{srmd1} \leq U_{opt}$.*

Proof Note first, that if the residual resource, r^c, ends up being zero before executing "**break**" at step 15 (or if j reaches the end of $|s\_list|$), then the solution found is in fact optimal based on Kuhn-Tucker condition[PSJJU80], as each g_i° (represented by C_i' in *approx_srmd1*) is essentially a piece-wise concave function.

Approx_srmd1 produces a utility value, U_{srmd1}, which is feasible. Therefore we have $U_{\text{srmd1}} \leq U_{\text{opt}}$.

Suppose that convex hull frontier segments (ordered and stored in *s_list*) are consecutively used (with corresponding quality upgrade and added utility) until the first segment, s, is found that requires more resource than residual resource capacity r^c to realize the extra utility at the end of the segment s (remember that the convex hull segments are imaginary linear relaxation of the real utility functions).

Let the two end points of the critical segment s be (r_{si}, u_{si}) and (r_{si+1}, u_{si+1}) in C_i'.

Based on Kuhn-Tucker condition and Dantzig[Dan57] upbound, we have

$$\begin{aligned} U_{\text{opt}} &\leq U_{\text{srmd1}} + (r^c - r_{si}) \frac{u_{si+1} - u_{si}}{r_{si+1} - r_{si}} \\ &< U_{\text{srmd1}} + (r_{si+1} - r_{si}) \frac{u_{si+1} - u_{si}}{r_{si+1} - r_{si}} \\ &= U_{\text{srmd1}} + u_{si+1} - u_{si} \end{aligned}$$

and we know that $u_{si+1} - u_{si} \leq \chi$, therefore $U_{\text{opt}} - U_{\text{srmd1}} <^{6} \chi$. □

Remark: To give a feel for how tight the bound is from below, examine two cases (see Figure 6) when the results are suboptimal. The reason for the first case is sub-optimality is due to the convex hull approximation error (where one or more intermediate utility points are bridged and removed when we construct g_i° (or C_i'), from g_i (or C_i); the reason for the second case is the consequence of the greedy heuristic (no costly backtracking after optimal condition is violated) near the end of the *approx_srmd1* optimization process.

Case 1. When interior (intermediate) points are bridged over and dominated by the critical convex hull segment s.

[5] Overmars & Leeuwen's [OL81] algorithm, or simply the quickhull [PS85] or Graham-Scan [CLR90] when C_i are not pre-sorted.

[6] Except in the degenerate case where $\chi = 0$, and $U_{\text{opt}} - U_{\text{srmd1}} = \chi = U_{\text{opt}} = U_{\text{srmd1}} = 0$.

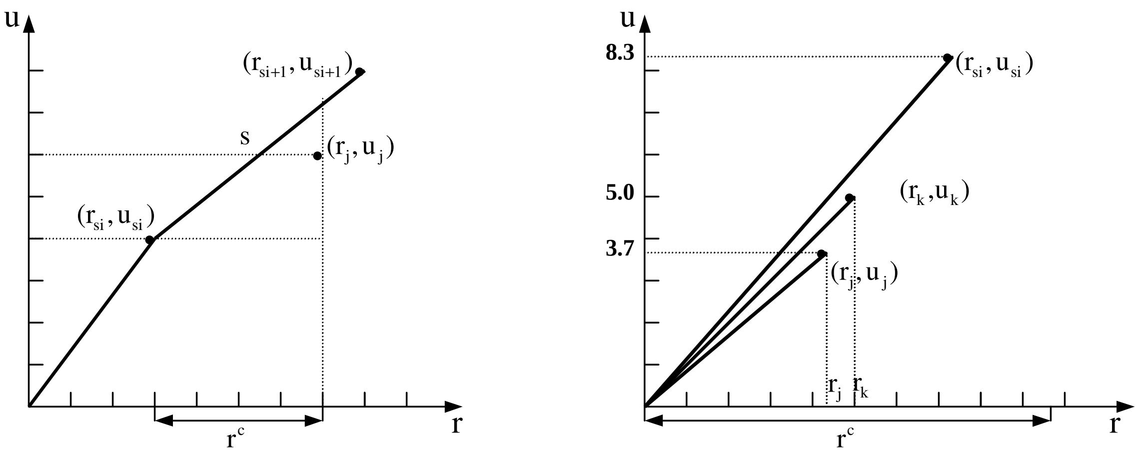

Figure 6: Suboptimal Cases

Let the inferior point bridged over by s with the largest utility be (r_j, u_j), where $j > si$ in the original C_i list of T_i. Further assume that $r_j - r_{si} \leq r^c$, and there is no more elements left in *s_list*. Then when *approx_srmd1* stops and reports the achieved utility of U_{srmd1}, which excludes (r_{si+1}, u_{si+1}), the optimal is in fact $U_{\text{opt}} = U_{\text{srmd1}} + (u_j - u_{si})$. Since $u_j - u_{si} < \chi$, $U_{\text{srmd1}} < U_{\text{opt}} - \chi$.

Case 2. When $r^c > r_{si}$, and there are (r_{sj}, u_{sj}) and (r_{sk}, u_{sk}) [7] in s_list, where $sj, sk > si$, and their slopes are lower than that of $\binom{u_{si}}{r_{si}}$, but $(r^c - r_{si}) < r_{sj}$, $(r^c - r_{si}) < r_{sk}$, $r_{sj} + r_{sk} \leq r^c$, and $(u_{sj} + u_{sk}) > u_{si}$. By the Dantzig bound, the U_{srmd1} would be well within χ as well.

Although *approx_srmd1* is a polynomial approximation algorithm with a describable and potentially small error bound from the optimal result, the bound is not controllable. Section 6.2 presents another polynomial scheme with a controllable error bound.

6.2 An Optimal Solution Scheme for SRMD

Assume that the resources are allocated in units of $r^{\max}/P$ for some integer P. If, for example, $P = 100$ this would mean that allocation is in integer percentage. Under this assumption, we can characterize the structure of the optimal solution and recursively define its value as follows:

Denote by $v(i, p)$ the maximum utility achievable when the first i tasks are considered with resource $r^{\max}p/P$ available for allocation, and define

$$v(i, p) = \max_{p' \in \{0, \ldots, p\}} \{g_i(p') + v(i-1, p-p')\} \tag{7}$$

The set of interesting p' values is in fact just all the (starting) discontinuity points of g_i (see Definition 5).

Therefore $v(n, P)$ will be the maximum utility achievable by allocating up to $r^{\max}$ to the n tasks, i.e., the best allocation overall.

[7] Or a single element with higher utility value than u_{si} given $r_{si} + r^c$. This case is not shown in Figure 6

Based on Equation 7, the following algorithm *srmd* can be constructed through dynamic programming. Let

$$C_i = \left\langle \begin{pmatrix} u_{i1} \\ r_{i1} \end{pmatrix}, \ldots, \begin{pmatrix} u_{ik_i} \\ r_{ik_i} \end{pmatrix} \right\rangle$$

denote the utility function g_i's discontinuity points in increasing u-order, and $qos(i,p)$ the list of QoS allocation choices for T_1 through T_i towards $v(i,p)$.

srmd$(n, P, C_1, \ldots, C_n)$

  for $p = 0$ **to** P **do**
    $qos(0,p) := nil, v(0,p) := 0$
  for $i = 1$ **to** n **do**
    $qos(i,0) := nil, v(i,0) := 0$
    for $p = 1$ **to** P **do**
      $u^* := 0, j^* := 0$
      for $j = 1$ **to** $|C_i|$ **do**
        if $(r_{ij} > p$ **or** $h_i(r_{ij}) < q_i^{\min})$ **break**
        $u := u_{ij} + v(i-1, p-r_{ij})$
        if $u > u^*$ **then**
          $u^* := u$
          $j^* := j$
      $qos(i,p) := qos(i-1, p-r_{ij^*})$ **concat** $[h_i(r_{ij^*})]$
      $v(i,p) := u^*$
  return $v(n,P)$ and $qos(n,P)$

The result $v(n,P)$, the utility accrued when 100% of the resource is available, is optimal. Let $L = \max_{i=1}^{n} |C_i|$. The time complexity of the algorithm is $O(nLP)$ or $O(nP^2)$, which is pseudo-polynomial.

One of the plus sides of this scheme (also true for the MRMD scheme described in Section 7.1) is its incremental and state-reuse property in which when a new task arrives, previous results can be directly reused to avoid the expensive recomputation of the complete new task set.

When the session length information of tasks are available, the task lists are generally ordered in decreasing session length order, so when a task T_n finishes and departs the system (and therefore releases some resources), the result for T_i, $i = 1, \ldots, n-1$ is already computed and kept in the system, that could be reused to make a quick decision (not necessarily to be optimal especially when stability policy is in use) on which tasks' qualities could be improved.

When a priority-based policy alone is emphasized, the task list to be fed into the algorithm will be in non-increasing order of task priorities.

Srmd could to be a practical method for QoS-driven single resource allocation, such as processor scheduling in operating systems which support QoS. The algorithm, with minor change, would be suitable to deal with the *stability* problem when a user prefers (or a policy requires) a relative consistent quality.

6.3 A Polynomial Scheme with Controllable Bound for SRMD

The algorithm *approx_srmd2* to be described will give an approximate quality and resource allocation which is guaranteed to have a maximum relative error, ε, where $0 < \varepsilon < 1$ is a user-specified value. A relative error of ε means that the utility U_{srmd2} found by the algorithm satisfies

$$(1-\varepsilon)U_{\mathrm{opt}} \leq U_{\mathrm{srmd2}} \leq U_{\mathrm{opt}}$$

where U_{opt} is the optimal utility.

Before presenting *approx_srmd2,* let us define some data structures and operations to be used in the algorithm. All utility function g_i's discontinuity points are listed in increasing u-order as

$$C_i = \left\langle \binom{u_{i1}}{r_{i1}}, \ldots, \binom{u_{ik_i}}{r_{ik_i}} \right\rangle$$

where $\binom{0}{0}$ is the first element, and referred to as r-u-pair lists. We also define the following operation for r-u-pair lists and r-u-pair elements.

$$\left\langle \binom{u_1}{r_1}, \ldots, \binom{u_k}{r_k} \right\rangle + \binom{u}{r} = \left\langle \binom{u_1+u}{r_1+r}, \ldots, \binom{u_k+u}{r_k+r} \right\rangle$$

Note, that this operation produces a new r-u-list that is sorted non-decreasingly in u-value. From now on such sorting will be assumed.

Let A and B be r-u-pair lists. The procedure *combine_and_merge* will combine A and B into a single r-u-pair list.

combine_and_merge(A, B)
1. **foreach** $b_i \in B$
2. $\quad A_i := A + b_i$ $\quad$ /* A_i *is now increasing in u-value.* */
3. $C :=$ **merge**$(A_1, \ldots, A_k)$
4. **return** C.

where $k = |B|$, and A_i, $1 \leq i \leq k$, are intermediate r-u-pair lists.

Steps 1 and 2 takes $O(|A|\,|B|)$, step 3 takes $O(|A|\,|B| \log |B|)$ if we do it through divide-and-conquer and merge lists in pairs recursively. So *combine_and_merge* is $O(|A|\,|B| \log |B|)$.

The procedure *resource_sieve* trims those r-u-pair elements of list $L = \left\langle \binom{u_{i1}}{r_{i1}}, \ldots, \binom{u_{in}}{r_{in}} \right\rangle$ which do not satisfy $r < r^{\max}$; and those inefficient elements. By inefficient we mean: for each element $\binom{u_i}{r_i}$ and element $\binom{u_{i+1}}{r_{i+1}}$ from L, if $r_{i+1} \leq r_i$ (and $u_i \leq u_{i+1}$ since elements are sorted) then $\binom{u_i}{r_i}$ is inefficient and should be removed from L. Intuitively, we only want to keep those choices that use less resource while achieving the same or higher utility. The procedure takes $O(|L|)$.

resource_sieve$(L, r^{\max})$

$i := 1$
while $i < |L|$ **do**
$\quad$ **if** $r_{i+1} > r^{\max}$ **then**

4.   Remove $\binom{u_{i+1}}{r_{i+1}}$ from L
5.   **else**
6.   **while** $i \geq 1$ **and** $r_{i+1} \leq r_i$ **do**
7.   Remove $\binom{u_i}{r_i}$ from L
8. $i := i - 1$
9. $i := i + 1$
10. **if** $r_i > r^{\max}$ **then**
11.   Remove $\binom{u_i}{r_i}$ from L
11. **return** L.

Procedure *representative_list* trims the r-u-pair list further in $O(|L|)$ by removing elements that are too close to other element in terms of u-value. That is, for each adjacent $\binom{u_i}{r_i}$ and $\binom{u_{i+1}}{r_{i+1}}$ from L, if $(u_{i+1} - u_i)/u_{i+1} \leq \delta$, then $\binom{u_{i+1}}{r_{i+1}}$ can be presented by $\binom{u_i}{r_i}$ with a discrepancy of at most δ w.r.t. the u-value of $\binom{u_{i+1}}{r_{i+1}}$, and therefore $\binom{u_{i+1}}{r_{i+1}}$ can be removed from L.

representative_list(L, δ)
1. $L' := \left\langle \binom{u_1}{r_1} \right\rangle$
2. $u^* := u_1$
3. **for** i $= 2$ **to** $|L| - 1$ **do**
4.   **if** $(u^* < u_i(1 - \delta))$ **then**
5.   append $\binom{u_i}{r_i}$ to L'
6. $u^* := u_i$
7. **return** L'

Given the above procedures, the bounded approximation scheme can be constructed as follows. For the sake of simplicity of the complexity analysis to be followed, we introduce some intermediate lists L_{ia}, L_{ib} and L_i.

approx_srmd2$(C_1, ..., C_n, \varepsilon)$
1. $L_0 := \left\langle \binom{0}{0} \right\rangle$
2. $\delta := \varepsilon / n$
3. **for** $i = 1$ **to** n **do**
4. $L_{ia} :=$ **combine_and_merge**(L_{i-1}, C_i)
5. $L_{ib} :=$ **resource_sieve**$(L_{ia}, r^{\max})$
6. $L_i :=$ **representative_list**(L_{ib}, δ)
7. let $\binom{u}{r}$ be the element with the largest utility value in L_n
8. **return** $\binom{u}{r}$

Without *resource_sieve* and *representative_list* the length of the list obtained at step 4 in *approx_srmd2* could increase exponentially. We will show that with those steps, the length of of L_i will be bounded by $\left\lfloor \frac{n \ln(u_{\text{up}}/u_{\text{low}})}{\varepsilon} + 2 \right\rfloor$, where u_{up} and u_{low} are easily determined from C_i and f is a suitable constant.

Lemma 1 *Given two sorted r-u-pair lists A and B,* combine_and_merge *generates a sorted r-u-pair list which contains all the possible combinations of a choice element from A and a choice element from B.*

Proof Since A is sorted, each A_i in step 2 of *combine_and_merge* maintains its order. Moreover A_i contains all new combinations of choices that can be generated by selecting one choice from A and the other as b_i from B. Therefore after the loop at step 1 of *combine_and_merge* finishes, all possible combinations of one element chosen from A and one element chosen from B are stored in A_i, where $1 \leq i \leq |B|$. The merge at step 3 will therefore generate a single combined sorted list. □

Theorem 1 *The approximation of* approx_srmd2 *is within a bound of* ε *w.r.t. the optimal.*

Proof If we were not to have the trimming operations *resource_sieve* and *representative_list* in steps 5 and 6 (denote such lists generated without trimming by L_i°), we could prove, based on Lemma 1, by induction on i that *combine_and_merge* at step 4 would list all the possible r-u-pair combinations for i tasks. It would then lead us to an optimal solution at the expense of exponential time complexity in general, since the length of L_i° would grow exponentially.

With trimming that removes from L_i every element that is greater (in terms of r-value) than $r^{\max}$ in step 5, and the trimming in step 6, the property that every remain element in L_i is a member of the complete solution space is maintained. Therefore, the r-u-pair returned in step 7 is indeed one valid allocation scheme. It remains to show that the u-value of the returned pair is not smaller than $1 - \varepsilon$ times an optimal solution.

Since *resource_sieve* at step 5 only throws away invalid elements that violate the *resource constraint,* or those that for sure cannot contribute toward the optimal solution, any error will only be caused by *representative_list.* So it remains to be shown that the relative error caused by *representative_list* is bounded.

When L_i is trimmed by *representative_list,* a relative error of at most δ (or ε/n) is introduced between the representative values remaining in the list and the values before the trimming. By induction on i, it can be shown that for every element $\binom{u^\circ}{r^\circ}$ in L_i° with $r^\circ \leq r^{\max}$, there is an $\binom{u}{r}$ in L_i such that

$$(1 - \varepsilon/n)^i u^\circ \leq u \leq u^\circ.$$

If $\binom{U_{\text{opt}}}{r^*} \in L_i^\circ$ denotes an optimal solution to the SRMD problem, then there is an $\binom{u}{r} \in L_i$ such that

$$(1 - \varepsilon/n)^n U_{\text{opt}} \leq u \leq U_{\text{opt}}$$

The $\binom{u}{r}$ with the largest u is the value returned by *representative_list* and $u = U_{\text{srmd2}}$. The value of $(1 - \varepsilon/n)^n$ increases with n, as it can be shown that

$$\frac{d}{dx}\left(1 - \frac{\varepsilon}{x}\right)^x > 0 \quad \text{for } x \geq 1,$$

so that $n > 1$ implies $1 - \varepsilon < (1 - \varepsilon/n)^n$, and therefore

$$(1 - \varepsilon)U_{\text{opt}} \leq U_{\text{srmd2}} \leq U_{\text{opt}}$$

That is, the result returned by *representative_list* has a maximum relative error of less than ε. □

We will show that the algorithm is of polynomial time complexity. Begin by investigating L_i in *representative_list.* After trimming, successive elements $\binom{u_i}{r_i}$ and $\binom{u_{i+1}}{r_{i+1}}$ of L_i must satisfy $u_i < u_{i+1}(1 - \delta)$, that is

$$\frac{u_{i+1}}{u_i} > \frac{1}{1 - \delta}\,.$$

Let $f = 1/(1-\delta)$ and $K = \lfloor \log_f(u_{\text{up}}/u_{\text{low}}) + 2 \rfloor$, where $u_{\text{up}} > 0$ is the u in step 7 of *approx_srmd2* and $u_{\text{low}} > 0$ is the smallest utility value, among all tasks, other than 0.

Lemma 2 *There are at most K elements in each L_i of step 6 of* approx_srmd2*.*

Proof Not counting the first element (whose u-value is zero), *representative_list* at step 6 removes elements that differ in u-value from each other by a factor of less than f. Therefore, the number of elements in L_i will be at most

$$\begin{aligned} 1 + \max\{\, k \geq 0 \mid f^k u_{\text{low}} \leq u_{\text{up}} \,\} &= 1 + \lfloor \log_f(u_{\text{up}}/u_{\text{low}}) + 1 \rfloor \\ &= \lfloor \log_f(u_{\text{up}}/u_{\text{low}}) + 2 \rfloor \\ &= K. \end{aligned}$$

□

Theorem 2 approx_srmd2 *is a polynomial approximation for SRMD.*

Proof Since steps 4 through 6 in *approx_srmd2* are all polynomial in the lengths of the lists they handle, and since step 6 by Lemma 2 reduces the number of elements to less than K, it remains to be shown that the number of elements after steps 4 and 5 are bounded.

For step 5 this is trivial since it reduces the number of elements. For step 4, the number of elements grows by a factor of $|C_i|$, so the number of elements after step 4 is bounded by $KC^{\max}$ where

$$C^{\max} = \max_{i=1,\ldots,n} |C_i|$$

The total number of steps in *approx_srmd2* therefore is bounded by

$$\begin{aligned} cnKC^{\max} &= cnC^{\max} \left\lfloor \log_f(u_{\text{up}}/u_{\text{low}}) + 2 \right\rfloor \\ &= cnC^{\max} \left\lfloor \log_{1/(1-\varepsilon/n)}(u_{\text{up}}/u_{\text{low}}) + 2 \right\rfloor \\ &\leq cnC^{\max} \left\lfloor \frac{n \ln(u_{\text{up}}/u_{\text{low}})}{\varepsilon} + 2 \right\rfloor \end{aligned}$$

for some constant $c > 0$.

Therefore, the algorithm is polynomial in time in terms of the input and $1/\varepsilon$. And it is clear that the algorithm is polynomial in space as well. □

The analysis of *approx_srmd2* is, in part, modelled after [IK75].

7 MRMD Algorithms

7.1 Optimal Solution Schemes for MRMD

7.1.1 Dynamic Programming

The scheme and algorithm described in this section is an extension of the algorithm described in Section 6.2. We will concentrate on the two resources (i.e., $m = 2$) case, but the scheme and results described below extends easily to higher dimensions.

The challenge here is to extend the tabular or dynamic programming scheme described under Section 6.2 in the presence of multiple resources. As in the single resource case, allocation is in units of $r_1^{\max}/P_1$ and $r_2^{\max}/P_2$.

For the two-resource case, the structure of an optimal solution of the problem can be characterized as follows:

Denote by $v(i, p_1, p_2)$ the maximum utility achievable when only the first i tasks are considered with $r_1^{\max}p_1/P_1$ of resource R_1 and $r_2^{\max}p_2/P_2$ of resource R_2 for allocation. Define the value of an optimal solution recursively in terms of the optimal solutions to subproblems as

$$v(i, p_1, p_2) = \max_{\substack{p_1' \in \{0,\ldots,p_1\} \\ p_2' \in \{0,\ldots,p_2\}}} \{g_i(p_1', p_2') + v(i-1, p_1 - p_1', p_2 - p_2')\}$$

As for the single source case, $v(n, P_1, P_2)$ will be the maximum utility achievable. The set of interesting p_1' and p_2' values are just all the (starting) discontinuity points of g_i. The time complexity of the algorithm is $O(nP_1^2P_2^2)$, which is pseudo-polynomial as well. To save space, the complete algorithm which is very similar to *approx_srmd2*, is omitted.

The above algorithm extends to multiple resources with time complexity $O(nP_1^2 \cdots P_m^2)$, where m is the number of resources involved for consumption and tradeoff. Due to its pseudo-polynomial complexity, we expect that its use will be limited for on-line systems, instead it will mainly be used for off-line and solution quality measurement of other heuristic and approximation schemes.

7.1.2 Integer Programming

From problem formulation 2 (Section 3.3), Integer Programming can be applied. For efficiency, we use CPLEX [Div97] MIP callable library which employs a branch-and-bound algorithm. In the branch-and-bound method, a series of LP subproblems is solved. A tree of subproblems is built, where each subproblems is a node of the tree. The root node is the LP relaxation of the original IP problem.

One negative aspect of the branch-and-bound techniques for solving integer programming problem is that the solution process can continue long after the optimal solution has been found, as the tree is being exhaustively searched in an effort to guarantee that the current feasible integer solution is indeed optimal. As we know, the branch-and-bound tree may be as large as 2^n nodes, where n equals the number of binary variables. A problem containing only 30 variables could produce a tree having over one billion nodes. Its applicability for practical large MRMD problem is yet to be determined.

7.2 Approximation and Heuristic Schemes

To improve the performance of our integer programming with branch-and-bound, described in the last section, we use task priority and gradient of dimension-wise quality utility function as heuristics for developing integer solution at root node and selecting branching node, variable, and direction. Optimality tolerance (such as the gap between the best result and utility of the best node remaining) or limits on time, nodes, memory etc. can be set for fast approximation results.

Local and direct search algorithms are also under development. Searching heuristic will be drawn from task profiles including the (approx)gradient of the dimension-wise utility functions, the resource-quality relations and other informations provided in the profiles.

Unlike the algorithms presented in the previous sections, currently we do not have *theoretical* measures of performance for the heuristic approximation schemes. Their experimental performance evaluation will be presented in a separate report.

8 Related Work

Research on Quality of Service for multimedia applications has gained significant momentum over the last few years. Much research has been being conducted on the end-system or end-to-end architectures for QoS support [HLP91, KJS91, CSZ92, Nah93, NT94, CCH94, MST94, LNS95, CCR95, VWHW95, RP95, LKRM96, KT97], and much more is on link, network and transport layer ([ZDE+93, ZF94, FJ95, SCFJ96, SZN97, MS97, SFZ97, CFKS98] to name a few). Most of this research has been focused on low-level system mechanisms. The authors consider and work on such parameters as period, buffer size, jitter, bandwidth and so on. No doubt these are important issues and factors for QoS control, but they are hardly tangible for the ultimate end-users for whom QoS is meant to.

Research on adaptive QoS control [TTCM92, TK93, MJ95, LRM96, NSN+97] brings us a step closer to the QoS support from a user's perspective by providing a mechanism in an application to accommodate potential dynamic changes in the operating environment. But these mechanisms are still mainly system-oriented in that a user has limited influence over the quality of the service to be delivered or adapted.

In coping with the shortage of QoS support from an end-user point of view, we proposed a basic framework [Lee97, RLLS97, LS98] that enables the end users to give guidance on the qualities they care about and the tradeoffs they are willing to make under potential resource constraints. Working from the user's perspective and maximizing the user perceived quality or utility has also been addressed in [JLDB95, AAS97, BG98]. In [JLDB95], a user-centric approach is taken, where a user's preferences are considered for application runtime behavior control and resource allocation planning. Example preferences include statements that a video-phone call should pause a movie unless it's being recorded and that video should be degraded before audio when all desired resources are not available. These are useful hints for high-level QoS control and resource planning, but are inadequate for quantitatively measuring QoS, or analytically planning and allocating resources.

An utility model for QoS control is also used in [AAS97]. In [AAS97] the authors propose a mechanism for QoS (re-)negotiation as a way to ensure graceful degradation. They suggest that a user should be able to express, in his/her service requests, the spectrum of QoS levels the user can accept from the provider, as well as the perceived utility of receiving service at each of these levels. But the authors did not address the resource tradeoff problem. Also, no specification method and mechanism is provided to facilitate utility data acquisition. Interesting research is being conducted in [BG98], where the author presents a framework for the construction of network-aware applications. The basic idea is to allow an application to adapt to its network environment, e.g. by trading off the volume (and with it the quality) of the data to be transfered and the time needed for the transfer. Given the framework, the application developer must specify functions to determine the relations between quality and size as well as to provide estimates about the effectiveness of various transformations to reduce size, and therefore to trade off the volume of the data to be transfered and the time need for the transfer. The above mechanism coincides with one of our schemes for implementing the resource tradeoff. The model defined in [LS98] can be considered a generalization of [BG98]. Attempts to optimize the system in terms of allocating CPU cycles for feedback control applications have been studied in [SLSS96].

9 Acknowledgement

The authors would like to thank John Hooker at the GSIA of CMU for many insightful and interesting discussions on our work, especially on various heuristic schemes for the MRMD problem. Thanks also goes to John Lehoczky, Raj Rajkumar and other members of the Amaranth project at Carnegie Mellon University.

References

[AAS97] E. Atkins, T. Abdelzaher, and K. Shin. QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, June 1997.

[AL97] E. Aarts and J. Lenstra, editors. *Local Search in Combinatorial Opitmization.* John Wiley & Sons, 1997.

[BG98] J. Bolliger and T. Gross. A Framework-Based Approach to the Development of Network-Aware Applications. In *IEEE Trans. Software Engineering (Special Issue on Mobility and Network-Aware Computing)*, volume 24, pages 376–390, May 1998.

[CCH94] A. Campbell, G. Coulson, and D. Hutchison. A Quality of Service Architecture. *Computer Communication Review*, 24(2):6–27, April 1994.

[CCR95] G. Coulson, A. Campbell, and P. Robin. Design of A QoS Controlled ATM Based Communication System in Chorus. In *IEEE Journal of Selected Areas in Communications (JSAC)*, Special Issues on ATM LANs: Implementation and Experiences with Emerging Technology, May 1995.

[CFKS98] P. Chandra, A. Fisher, C. Kosak, and P. Steenkiste. Network Support for Application-Oriented Quality of Service. In *Sixth IEEE/IFIP International Workshop on Quality of Service*, May 1998.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press / McGraw-Hill, 1990.

[CSZ92] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proceedings of the SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pages 14–26, October 1992.

[Dan57] B. Dantzig. Discrete Variable Extremum Problems. In *Operations Research*, volume 5, pages 266–277, 1957.

[Div97] CPLEX Division. *Using the CPLEX Callable Library.* ILOG Inc., 1997.

[Edw97] W. Edwards. How to Use Multiattribute Utility Theory for Social Decision Making. In *IEEE Trans. Systems Man, Cybern. 7*, pages 326–340, 1997.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. In *IEEE/ACM Transactions on Networking*, volume 3, pages 365–386, August 1995.

[HLP91] J. M. Hyman, A. A. Lazar, and G. Pacifici. Real-Time Scheduling with Quality of Service Constraints. *IEEE Journal on Selected Areas in Communications*, 9(7), September 1991.

[Iba87a] T. Ibaraki. Enumerative Approaches to Combinatorial Optimization — Part 1. *Annals of Operations Research*, 10, 1987.

[Iba87b] T. Ibaraki. Enumerative Approaches to Combinatorial Optimization — Part 2. *Annals of Operations Research*, 11, 1987.

[IK75] O. Ibarra and C. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of ACM*, 22:463–468, 1975.

[ITU95] ITU. ITU-T Recommendation H.263 - Video Coding for Low Bit Rate Communication, July 1995.

[JLDB95] M. Jones, P. Leach, R. Draves, and J. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, May 1995.

[KJS91] D. L. Stone K. Jeffay and F. D. Smith. Kernel support for live digital audio and video. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 10–21, November 1991.

[KT97] K. Kawachiya and H. Tokuda. A Negotiation-Based Resource Management Framework for Dynamic QoS Control. In *Proceedings Real-Time Mach Workshop '97*, August 1997.

[Lab97] AT&T Labs. GeoPlex — The Enhanced Network Infrastructure For 21st Century Services. AT&T White Paper, May 1997.

[Lee97] C. Lee. A Translucent QoS Architecture. In *Proceedings of the RT-Mach Workshop 9́7*, August 1997.

[LG91] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *CACM*, 34(4):46–58, April 1991.

[LKRM96] C. Lee, Y. Katsuhiko, R. Rajkumar, and C. Mercer. Predictable Communication Protocol Processing in Real-Time Mach. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 1996.

[LNS95] A. Lazar, L. Ngoh, and A. Sahai. Multimedia Networking Abstraction with Quality of Services Guarantees. In *Proc. SPIE Conference on Multimedia Computing and Networking*, February 1995.

[LRM96] C. Lee, R. Rajkumar, and C. Mercer. Experience with Processor Reservation and Dynamic QoS in Real-Time Mach. In *Proceedings of the Multimedia Japan ́96*, March 1996.

[LS98] C. Lee and D. Siewiorek. An Approach for Quality of Service Management. Technical Report CMU-CS-98-165, Carnegie Mellon University, October 1998.

[MGF96] J. Mitchell, D. Le Gall, and C. Fogg. *MPEG Video Compression Standard*. Chapman & Hall, 1996.

[MJ95] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *Proc. ACM Multimedia'95*, November 1995.

[MS97] Q. Ma and P. Steenkiste. Quality of Service Routing for Traffic with Performance Guarantees. In *IFIP International Workshop on Quality of Service*, May 1997.

[MST94] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 90–99, May 1994.

[MT90] S. Martello and P. Toth. *Knapsack Problems — Algorithms and Computer Implementations*. John Wiley & Sons Ltd., 1990.

[Nah93] K. Nahrstedt. Network Service Customization: End-point Perspective. Technical Report MS-CIS-93-100, University of Pennsylvania, December 1993.

[NSN+97] B. Noble, M. Satyanarayanana, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. *Proceedings of the 16th ACM Symposium on Operating System Principles*, oct 1997.

[NT94] T. Nakajima and H. Tezuka. A Continuous Media Application supporting Dynamic QOS Control on Real-Time Mach. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 289–297, October 1994.

[OL81] M. Overmars and J. Leeuwen. Maintenance of Configurations in the Plan. In *Journal of computer and System Sciences*, volume 23, pages 166–204, 1981.

[Pis95] D. Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, Dept. of Computer Science, February 1995.

[Pro96] Amaranth Project. Amaranth Project. Amaranth White Paper, December 1996.

[PS85] F. Preparata and M. Shamos. Computational Geometry : An Introduction. In *Texts and Monographs in Computer Science*. Springer-Verlag, 1985.

[PSJJU80] A. L. Peressini, R. E. Sullivan, and Jr. J. J. Uhl. *Convex Programming and the Karish-Kuhn-Tucker conditions*, chapter 5. Springer-Verlag, 1980.

[RLLS97] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A QoS-based Resource Allocation Model. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[RLLS98] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical Solutions for QoS-based Resource Allocation Problems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1998.

[RP95] R.Gopalakrishnan and G. Parulkar. A Framework for QoS Guarantees for Multimedia Applications within an Endsystem. In *Swiss German Computer Society Conference*, September 1995.

[Saa92a] T. Saaty. Decision Making for Leaders. *RWS Publications, Pittsburgh*, 1992.

[Saa92b] T. Saaty. Multicriteria Decision Making - The Analytic Hierarchy Process. Technical report, University of Pittsburgh, RWS Publications, 1992.

[Sah75] S. Sahni. Approximation algorithms for the 0-1 knapsack problem. In *Journal of ACM*, volume 23, pages 555–565, 1975.

[Say96] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc., 1996.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederic, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, 1996.

[SFZ97] P. Steenkiste, A. Fisher, and H. Zhang. Resource Management in Application-aware Networks. In *Workshop on the Integration of IP and ATM*, November 1997.

[SLSS96] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On Task Schedulability in Real-Time Control Systems. In *IEEE Real-Time System Symposium*, December 1996.

[SZN97] I. Stoica, H. Zhang, and E. Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings of SIGCOMM'97*, 1997.

[TK93] H. Tokuda and T. Kitayama. Dynamic QOS Control based on Real-Time Threads. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 113–122, November 1993.

[TTCM92] H. Tokuda, Y. Tobe, S. T.-C. Chou, and J. M. F. Moura. Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network. In *Proceedings of the SIGCOMM '92 Symposium on Communications Architectures and Protocols*, pages 88–98. ACM, October 1992.

[VWHW95] C. Volg, L. Wolf, R. Herrtwich, and H. Wittig. HeiRAT – Quality of Service Management for Distributed Multimedia Systems. In *Multimedia Systems Journal*, November 1995.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, pages 8–18, September 1993.

[ZF94] H. Zhang and D. Ferrari. Improving Utilization for Deterministic Service in Multimedia Communication. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 295–304, May 1994.