USER-ORIENTED QUALITY OF SERVICE MAPPING MECHANISM FOR MPEG-4 IN DIFFERENTIATED SERVICE NETWORKS

By

KARYANTA PURNE

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1998

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August, 2002

USER-ORIENTED QUALITY OF SERVICE

MAPPING MECHANISM FOR MPEG-4

IN DIFFERENTIATED SERVICE

NETWORKS

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

Marcin Paprzycki

Timothy J. Pettibone

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my thesis adviser, Dr. Johnson P. Thomas for his intelligent supervision, constructive guidance, enthusiastic suggestions and friendship throughout this study. My appreciation extends to the other committee members, Dr. Debao Chen and Dr. Marcin Paprzycki, whose assistance, encouragement, and friendship are also invaluable.

In addition, I wish to express my appreciation to Dr. John P. Chandler, Dr. Rathindra Sarathy and Dr. Dursun Delen, who provided assistance and suggestions for this study.

Finally, I would like to give my special appreciation to my future wife, Imelda Oktavia, S.E., for her precious love, encouragement and understanding throughout the whole process of study. Special thanks also go to my parents, Mr. I Wajan Purne and Mrs. Jap Pauw Joeng, for their everlasting love, wisdom and supports.

PUBLICATION

Part of this thesis has been accepted for presentation at the IEEE International Conference on Networking (Networks 2002) and will be published in the proceedings. This is a refereed conference and will be held at the Georgia Centers for Advanced Telecommunications Technology (Atlanta, Georgia, United States of America) on August 26th – 29th, 2002.

Refereed Conference

K. Purne, and J. P. Thomas, "User-oriented Object-based Multimedia Communications over DiffServ Networks," *Proceedings of the IEEE International Conference on Networking '2002* (Networks 2002), August 2002 (to appear).

TABLE OF CONTENTS

APPENDIXES

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

The current internet service model is based on best-effort delivery model. This means the internet does not provide guarantees of any form, but will do its best to deliver data. In practical networking terms, IP is a best effort system because it makes a sincere attempt to forward all data grams. However, if the network becomes congested or routes change, data grams can be lost, delayed, or delivered out of order.

Todays, internet multimedia applications have very diverse requirements on the network service. In transferring data, loss is more critical than delay. For example, when we send or receive text email, we cannot afford to lose bits as this will result in loss of characters or numbers. However, delay experienced in receiving the text email is tolerable. On the other hand, in transferring an audio-visual scene, the overall delay of the scene is more critical than losing some packets. The human brain has an ability to filter noise and reconstruct missing information, that is, even if we lose some packets, perception and understanding will not suffer proportionally. If audio packets are lost during a newscast streaming, no sound comes out from the newscaster or the sound may be garbled. In an audio-visual transmission, synchronization between audio and video is also critically important. For example, lip movements must be synchronized with the sound caused by the lip movements. In the best-effort delivery model, all packets receive the same quality of service, and there is therefore no guarantee that the audio or video packets will be delivered at the time these packets are needed.

Thus the current best-effort model is inadequate. It is economically unrealistic to overcome the inadequacies of the internet by increasing the current overall bandwidth and replacing the entire existing internet infrastructure with the higher bandwidth fiber-optic cable. Therefore, given the current internet model, the demand of network services to support Quality of Service[1] (QoS) is a key solution to overcome the best-effort infrastructure limited capabilities. The idea is to let the network provide a different level of assurance in terms of network QoS parameters within its resource capacity [15].

The Internet Engineering Task Force (IETF) has proposed two models, which are Integrated Services (IntServ) [17] with the Resource Reservation Protocol (RSVP) [4] and the Differentiated Services (DiffServ) [2].

IntServ has some problems in term of scalability and manageability [16]. The DiffServ model provides a simpler approach, which handles well the scalability and manageability problems for the core routers [13]. We therefore use the DiffServ model in this research. The basic foundations of IntServ and DiffServ are briefly explained in Chapter II.

This DiffServ network model can be used to optimize the users' experiences of multimedia scenes. The concept of 'media objects' as specified in MPEG-4 [8] or 'regions of interests' (ROI) as specified in JPEG-2000 [9] is used to optimize the users' experiences of multimedia transmitted over DiffServ networks. In this research, we will focus on MPEG-4, which is a multimedia compression algorithm developed by MPEG (Motion Picture Expert Group).

[1] Quality of Service (QoS) is a measure of transmission rates, error rates, delay, loss, and other characteristics that can be delivered by the network. QoS generally involves prioritization of network traffic. The goal of QoS is to provide guarantees on the ability of a network to deliver predictable results.

MPEG-4 audio-visual scene are composed of several media objects, organized in a hierarchical fashion. At the leaves of the hierarchy, we find primitive objects, such as still images, video objects, audio objects, etc. MPEG-4 also allows both natural and synthetic video to be coded and provides content-based access to individual objects in a scene [8].

However, although the MPEG-4 standards specify interaction with the objects, the MPEG-4 products currently available do not permit user's prioritization of objects. This is because objects of different priorities have to be segmented and video object segmentation is a very difficult problem. We therefore envisage the prioritization of MPEG-4 objects by user as applying to the future MPEG-4 player.

As mentioned before, DiffServ is currently in the proposal stage before the IETF. DiffServ is therefore not implemented on the existing internet yet. So, this research is directed at communication and video systems for which standards have been proposed but not yet implemented. This research is a first step in defining the transmission of MPEG-4 video over DiffServ for the ultimate benefit of the users.

Objectives of the Study

The prioritization of the MPEG-4 objects is theoretically possible. We can categorize the importance of the object based on the end-user's experience and differentiate the service for each object in the DiffServ network.

However, DiffServ is not envisioned as a free service. The user has to provide a budget for the entire video and the budget is per unit period of time (e.g. per minute) or size (e.g. per kilobytes). The budget per unit period of time or size may be a varying function of time.

Each DiffServ level will carry a different cost and different QoS parameters, such as average packet delays and losses. We assume that the network will provide the average packet delays and losses for each DiffServ level. Ideally, all the objects will be transmitted at the highest QoS, i.e., DiffServ level 1. However, this is not feasible at all times unless the user has a large budget.

Hence, there are constrains to be considered when assigning packets to the appropriate level of DiffServ network. Factors to be considered include:

- the total budget, which is the money the user wants to spend to use the DiffServ in a certain period of time or for the duration of a video transmission,
- the price of each level of DiffServ (e.g. level 1 DiffServ will cost more than level 3 DiffServ),
- the average packet delays and losses of each level of DiffServ . Each level has a certain range of packet delays and losses (e.g. since level 1 DiffServ costs more than level 3 DiffServ, level 1 DiffServ will have smaller packet delay and less loss than level 3 DiffServ).

The average user is not concerned with technical parameters such as packet delays or losses. It is unrealistic to ask the average non-technical user to specify the importance of packets and expect him/her to supply the optimum number of packets to be transmitted in each DiffServ level within the available budget. However, the user can be expected to provide an indication of the relative importance of the different objects presented in a multimedia scene. For example, in a news scene, the audio object and the newscaster object will have high priority whereas the background object will have a lower priority from a user's perspective.

Therefore, given the average packet delays and losses for each level of DiffServ and the price of each level of DiffServ, a mechanism is needed to optimally and efficiently map and distribute all the incoming packets to the DiffServ network with respect of total budget to spend and a tolerable range of packet delays and losses per kilobyte for a particular MPEG-4 scene.

In figure 1, the user specifies his/ her objects preferences (or relative importance of objects) at the client machine. These preferences are mapped to weighted QoS parameters (delay and loss). At the server, labeled packets are assigned based on their QoS weightings to the appropriate DiffServ level given budgetary constraints. The MPEG-4 scene is transmitted to the client machine using the differentiated service of the DiffServ network.

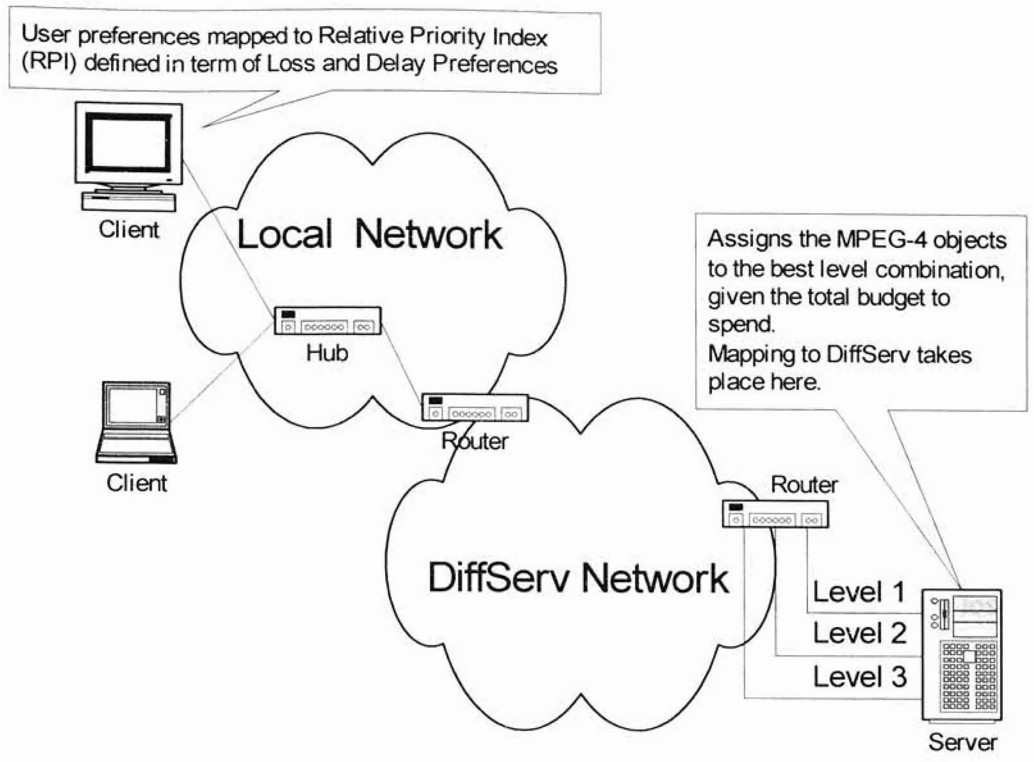

User preferences mapped to Relative Priority Index (RPI) defined in term of Loss and Delay Preferences
Client
Local Network
Hub
Client
Router
Assigns the MPEG-4 objects to the best level combination, given the total budget to spend.
Mapping to DiffServ takes place here.
Router
DiffServ Network
Level 1
Level 2
Level 3
Server

Figure 1. Overall QoS Mapping General Framework in DiffServ Network

Before this mechanism can be built, the first thing to do is to know the priority of each primitive object defined in MPEG-4. Assigning priorities of audio-visual scenes is described in detail in Chapter III.

After successfully assigning priority to each object, we come up with the efficient and adaptive QoS mapping mechanism from the given information, i.e., the priority of the objects, average loss and delay of each DiffServ level, price of each DiffServ level, and the total budget.

Assumptions

In this study, we assume that the total incoming MPEG-4 packets that want to be mapped to DiffServ levels are given and have already decoded. Hence, we do not need to decode each streaming incoming MPEG-4 packets. These incoming packets will be measured in kilobytes.

DiffServ is assumed to have only three levels, which are level 1, 2 and 3. Level 1 is always more expensive than level 2; level 2 is always more expensive than level 3, but level 1 always provides a better QoS than level 2, in terms of packet delays and losses; level 2 is always better than level 3, in terms of QoS parameters.

It is important to note that there is no direct relationship between the price of each DiffServ level with the packet delays and losses of each DiffServ level.

DiffServ is also assumed running in the router. So, DiffServ is only an algorithm and/ or mechanism to differentiate the given services by level, inside the router. Each level is given the average packet delays and losses.

Scope and Limitation

Rather than employing real MPEG-4 video over DiffServ, this study develops a simulation of the mapping mechanism where total incoming packets are mapped to the DiffServ levels. This is because currently MPEG-4 is still in the conceptual phase, and real MPEG-4 video data is very rare. Moreover, currently existing MPEG-4 implementation do not permit object specification and prioritization. Similarly, DiffServ network standards are still being developed. Hence, for this study purpose, we will use artificial data for the incoming MPEG-4 packets.

Scenes in a video change rapidly. This requires continuous input from the user on user's preferences. Similarly, if there are multiple JPEG-2000 images to be transmitted, multiple user specification of ROI is required. This problem is not discussed in this thesis.

Also, in this research we are not concerned with the mechanisms of assigning user-defined priorities to individual MPEG-4 objects.

CHAPTER II

REVIEW OF THE LITERATURE

Integrated Service

The Integrated Services (IntServ) focuses on individual packets [17]. Each packet can request a specific level of service from the router; the router will grant or reject the requests based on availability and capacity of resources.

There are three major components of the IntServ architecture [3]. They are packet scheduler, which manages the forwarding of different packet streams using a set of queues and perhaps other mechanisms like timers; packet classifier, which map the incoming packets to some class, the choice of the class may based upon the content of the existing packet header; and admission control, which implements the decision algorithm that a router or host uses to determine whether a new packet can be granted the requested QoS without impacting earlier guarantees.

The IntServ approach seems to have a solid foundation to support the QoS. However, as mentioned before, there are some problems in the scalability and manageability. IntServ requires router to maintain state information of each packet. From an implementation point of view, maintaining and processing large number of large-sized packets with millions of simultaneous incoming packets, are very difficult.

Differentiated Service

In the DiffServ model, resources are allocated differently for various aggregated traffic flows[2] based on a set of bits. The basic idea is to support a set of traffic class, e.g.: a premium service (PS), which expects the virtual leased line service to support low loss and delay/ jitter and an assured service (AS), which provides better than best-effort but without guarantee [13].

Differentiated services are intended to provide a framework and building blocks to enable deployment of scalable service discrimination in the internet. This architecture contains two main components [2]. One is the forwarding path; the other is background policy and allocation component that configures parameters used in the forwarding path.

The forwarding path behavior includes the differential treatment an individual packet receives, as implemented by queue service and/ or queue management disciplines. Background policy is performed by traffic conditioners at network boundaries, including the edges of the network and administrative boundaries. These traffic conditioners may include the primitives of marking, metering, policing and shaping.

Then, services are realized by the use of particular packet classification and traffic conditioning mechanisms at boundaries coupled with the concatenation of per-hop behaviors along the transit path of the traffic [13].

The DiffServ architecture is based on a simple model where traffic entering a network is classified and conditioned at the boundaries of the network, and assigned to

[2] A flow is stream of packets with common source address, destination address and port number. Based on this definition, the flow will be assumed as a group of packets that form a single MPEG-4 object. Therefore, we will use the term of flow (instead of packet) throughout the paper.

different behavior aggregates [2]. Hence, it is possible to separate the behavior of flows by their priorities to obtain different network services.

MPEG-4

MPEG-4 was originally intended for very high compression coding of audio-visual information at very low bit-rates of 64 kbit/s or under. However in July 1994, its scope was expanded to include coding of scenes as a collection of individual audio-visual objects and enabling a range of advanced functionalities not supported by other standards. On of the key functionalities supported by MPEG-4 is robustness in error prone environment [14].

So, the mission, focus and scope of MPEG-4 were redefined. According to ISO/IEC standard developed by Moving Picture Expert Group [8], MPEG-4 now provides standardized ways to:

- represent units of aural, visual or audio-visual content, called "media objects". These media objects can be of natural or synthetic origin; this means they could be recorded with a camera or microphone, or generated with a computer;
- describe the composition of these objects to create compound media objects that form audio-visual scenes;
- multiplex and synchronize the data associated with media objects, so that they can be transported over network channel providing a QoS appropriate for the nature of the specific media objects; and
- interact with the audio-visual scene generated at the receiver's end.

Hence, the nature of each MPEG-4 audio-visual scene is the composition of individual media object. Therefore the manipulation of these objects, such as giving a priority to particular object based on end-user's perception, is theoretically possible.

JPEG-2000

JPEG-2000 is an image compression algorithm developed by JPEG (Joint Photographic Experts Group). In this research, we are interested in the concept of ROI as specified in JPEG-2000, which is used for image optimization.

ROI (Regions Of Interest) coding [9] is one of the innovative functionalities supported by JPEG 2000, the ISO/ITU-T still image coding standard. It enables a non-uniform distribution of the image quality between a selected region (the ROI) and the rest of the image (background). An ROI is a region of the image that is expected to have a better quality than the rest at any decoding bit-rate. In other words, this implies a non-uniform distribution of the quality inside the image.

The similarity definitions of 'regions of interest' and 'media objects' make the prioritization of the ROI in JPEG-2000 using the mapping mechanism is hypothetically possible.

Mapping Mechanism

Shin, et. al. [16] have proposed the QoS mapping framework for video flows in DiffServ network, where service differentiation is expressed in terms of loss-rate and delay associated with forwarding queues. Each video flow of a user application has to be classified in the loss-rate and delay preference and each packet is associated with RPI (relative priority index) composed of two normalized indexes, RLI and RDI.

These RPI is associated packets are categorized into immediate DiffServ categories in fine-grained manner, albeit independent of underlying network if required. Then, pre-marked RPI categorized packets are conveyed into the DiffServ-aware node for QoS mapping, which can be located at the end-system itself. Thus, given a video application and the responding DiffServ network, their QoS mapping is accomplished by mapping the relative prioritized packets to maximize end-to-end video quality under a given cost contraint. Then, at the DiffServ boundary node, the packets are classified, conditioned and re-marked to certain network DiffServ levels by considering the traffic profile based on the current network status [16].

Although the result clearly gives the advantage of the proposed QoS mapping mechanism, we are aware that there are a couple of issues should be elaborated, i.e.: RDI association should be extended to include more characteristics within a video stream and in order to get a persistent QoS mapping, better end-system adjustment with feedback is needed.

Alternatively, the mapping of QoS is mapped onto the qualities of resources depends on the nature of the service itself. For example, assuming that a service consists of a linear configuration of resources, then the total delay of the composed service is the sum of the delays of its resources [12].

QoS can be modeled using sets of parametric functions, i.e.: linear, linear-exponential, exponential, or reciprocal linear exponential. It is reported in [12]. Very little research has been reported on the modeling of QoS parameters from a user's perspective. In other words, modeling of QoS parameters in term of user's perception is still not well understood.

CHAPTER III

DESIGN OF MAPPING ARCHITECTURE

As mentioned before, the user can be expected to provide some form of an indicator of the relative importance of the different objects presented in a multimedia scene. The user-prioritized objects are assigned to appropriate DiffServ levels in two mapping phases:

- the first mapping phase assigns weights to QoS parameters based on user's preferences. In this phase, flows associated with an object are labeled with a relative weighting for each QoS parameter. The weighting is an indication of the importance of the indicated QoS parameter. For example, delay is more critical for flows with a relative delay weighting of 10 than for the ones with a relative delay weighting of 5.
- the second mapping phase assigns labeled flows based on their QoS weightings to the appropriate DiffServ level given the budgetary constraints.

Therefore, we envisage the architecture of future MPEG-4 player for DiffServ network as shown below:

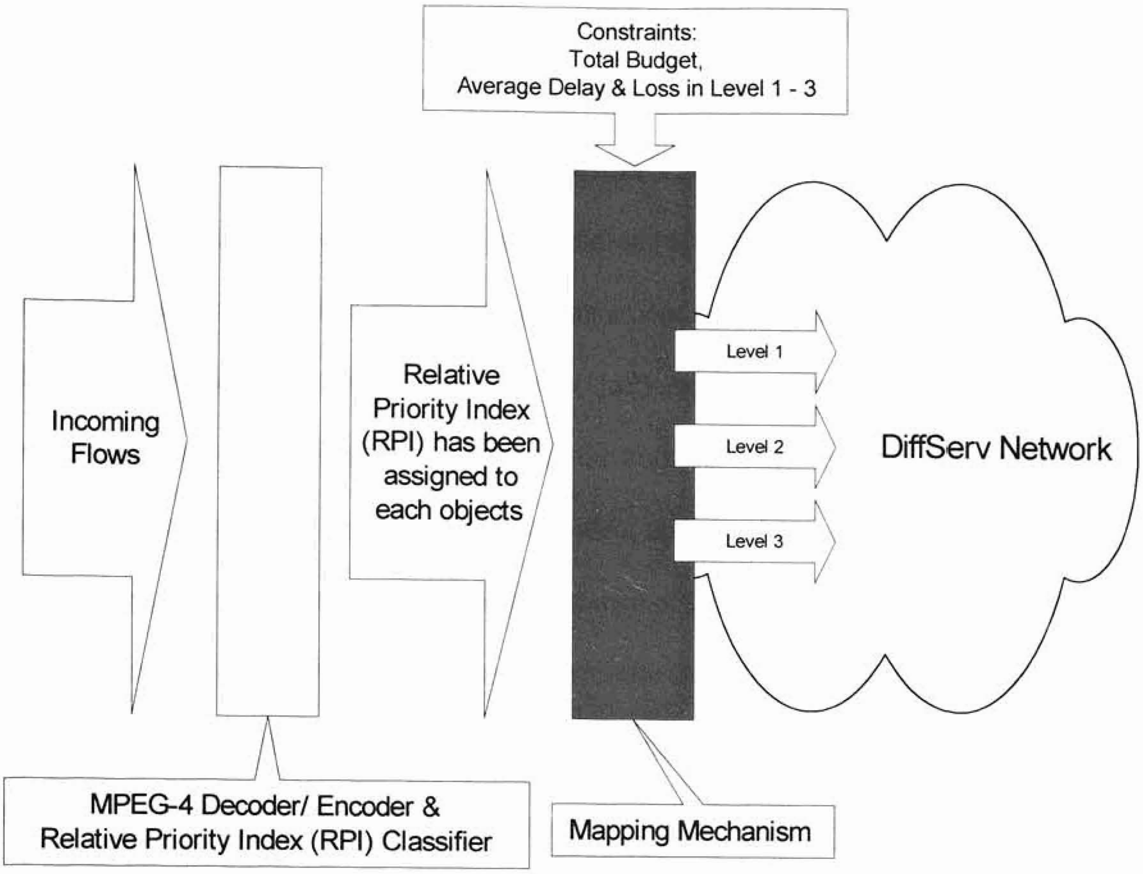

Constraints:
Total Budget,
Average Delay & Loss in Level 1 - 3
Incoming Flows
Relative Priority Index (RPI) has been assigned to each objects
Level 1
Level 2
Level 3
DiffServ Network
MPEG-4 Decoder/ Encoder & Relative Priority Index (RPI) Classifier
Mapping Mechanism

Figure 2. General Mapping Mechanism Framework

Prioritization of Audio-visual Objects

The proposed method of the first phase of developing QoS mapping mechanism is to assign relative priorities to a service in the DiffServ architecture such that the end-to-end video quality is maximized under the available bandwidth. The maximization is mainly measured by the user's experience of the multimedia.

Very little work has been reported on the impact of varying QoS on user perception of multimedia. Apteker, et. al. [1] showed that the relationship between frame loss and user satisfaction with multimedia is not a linear relationship. Ghinea and Thomas extended this work by examining the effect of varying QoS on user's perception of multimedia [6]. User's perception is not only a measure of user's satisfaction with the multimedia presentation, but also includes the user's absorption and assimilation of the information present in the multimedia presentation [6]. Moreover, Ghinea and Thomas have proposed a mechanism for approximating a user's perception of multimedia [7]. They called the term 'Quality of Perception' (QoP) and have proposed a mapping from user QoP to network QoS. In their approach, the authors define the user's experience of a multimedia as dependent on the primacy of video, audio and text as the carriers of information. Table I shows the relative importance of QoS parameters for each media [18]. For example, the table shows that bit error rate is of low importance to video whereas delay is of medium importance, indicating that bit error rate are more tolerable than delay in video transmission.

TABLE I

CONVERSION MATRIX LINKING QoP to QoS

QoP To QoS MAPPING		QoP		
		VIDEO	AUDIO	TEXT
QoS	Bit error rate	Low	Low	Low
	Delay	Medium	Medium	Low
	Jitter	Medium	Low	Medium
	Loss	Low	High	High
	Order	High	Medium	Medium

In this thesis project, we consider only delay (DEL) and loss (LOSS) priority index of the QoS.

The mathematical representation of QoP to QoS mapping is defined as follows:

$$QoP_{IT} \cong \frac{1}{V+A+T}[(V\alpha_{V,DEL} + A\alpha_{A,DEL} + T\alpha_{T,DEL}) * DEL + (V\alpha_{V,LOSS} + A\alpha_{A,LOSS} + T\alpha_{T,LOSS}) * LOSS]$$

where:

- QoP_{IT} is Quality of Perception of Information Transfer
- V, A and T are relative importance of the video, audio and textual components as conveyors of information
- DEL and LOSS are actual run-time values of the considered network parameters.
- $\alpha_{V,DEL}$ ... $\alpha_{T,LOSS}$ are relative priority values of QoP to QoS mapping. They correspond to the elements of Table I. For instance, $\alpha_{V,DEL}$ = medium while $\alpha_{T,LOSS}$ = high

The formula above was originally taken from Ghinea and Thomas [7]. However, since we deal with multiple objects in a video scene and we are only concern about delay and loss priority index, a slight modification of the original formula will take place.

Each video frame will have a number of video objects. So, the QoP of video objects can be captured with this representation:

$$QoP_{V,IT} \cong \frac{1}{V_1 + V_2 + \cdots + V_N}[(V_1\alpha_{V,DEL} + V_2\alpha_{V,DEL} + \cdots + V_N\alpha_{V,DEL}) * DEL$$

$$+ (V_1\alpha_{V,LOSS} + V_2\alpha_{V,LOSS} + \cdots + V_N\alpha_{V,LOSS}) * LOSS]$$

Therefore, by merging two formulas above, we obtain the relative priority of the different video objects, audio and text in term of delay and loss:

$$QoP_{IT} \cong \frac{1}{V_1+V_2+\cdots+V_N+A+T}[(V_1\alpha_{V,DEL}+V_2\alpha_{V,DEL}+\cdots+V_N\alpha_{V,DEL}+A\alpha_{A,DEL}+T\alpha_{T,DEL})*DEL$$

$$+(V_1\alpha_{V,LOSS}+V_2\alpha_{V,LOSS}+\cdots+V_N\alpha_{V,LOSS}+A\alpha_{A,LOSS}+T\alpha_{T,LOSS})*LOSS] \tag{3.1}$$

where:

- QoP_{IT} is Quality of Perception of Information Transfer
- $V_1 \ldots V_N$ are relative importance of video object 1 to video object N
- A and T are relative importance of the audio and textual components as conveyors of information
- DEL and LOSS are actual run-time values of the considered network parameters.
- $\alpha_{V,DEL}$ … $\alpha_{T,LOSS}$ are relative priority values of QoP to QoS mapping. They correspond to the elements of table I. For instance, $\alpha_{V,DEL}$ = medium while $\alpha_{T,LOSS}$ = high

We obtain the relative loss priority and delay priority values from the formula above. The second phase maps flows to one or more different DiffServ levels, while satisfying constraints, such as total budget.

Consider the following example: there are 5 objects V_1, V_2, V_3, A, and T in a MPEG-4 scene (3 video objects, 1 audio object and 1 text object). Users prioritize the objects of this particular MPEG-4 scene as follows:

TABLE II

MPEG-4 OBJECTS AND THEIR PRIORITIES EXAMPLE

Objects	**Priority**
V_1	High
V_2	Medium
V_3	Low
A	High
T	Low

From (3.1):

$$QoP_{IT} \cong (\frac{6}{10}V_1 + \frac{4}{10}V_2 + \frac{2}{10}V_3 + \frac{6}{10}A + \frac{1}{10}T) * DEL + (\frac{3}{10}V_1 + \frac{2}{10}V_2 + \frac{1}{10}V_3 + \frac{9}{10}A + \frac{3}{10}T) * LOSS$$

The weightings associated with object indicate the relative delay priority index and the relative delay index of each object from a user perspective. For example, from a user's point of view, for the particular video scene, the delay weighting of object V_1 (6/10) is greater than for object V_2 (4/10). In other words, user's perception benefits by attaching more importance to the delay of object V_1 rather than the delay associated with object V_2.

Given user's preference for individual objects in a scene, the relative loss priority and delay priority values can thus be derived from the formula above. We now describe an efficient way to map each flow to different DiffServ levels.

Mapping Priorities to DiffServ Levels

The objective of mapping each flow to a DiffServ level while satisfying the budgetary constraints is to maximize QoS. Maximum QoS is obtained when loss and delay are minimized. This can be represented as:

$$\max QoS = \min(f(\sum_{i=1}^{N} RLI_q \bullet l_{i(q)}, \sum_{i=1}^{N} RDI_q \bullet d_{i(q)}))$$

where:

- RLI_q and RDI_q are relative loss priority and relative delay priority for packet q
- $l_{i(q)}$ and $d_{i(q)}$ are loss and delay at the DiffServ level i for packet q

Practically, the value of $l_{i(q)}$ and $d_{i(q)}$ can be determined by calculating the average of the loss and delay at each level of DiffServ after several runs or simulations. And also, the price or the cost of each different level of DiffServ involved when computing the average loss and delay at all level of DiffServ. The price constraint can be demonstrated as:

$$\sum_{i=1}^{N} P_{i(q)} \leq P$$

where:

- $P_{i(q)}$ is the price of DiffServ level i for packet q
- P is the total price or budget of the user

Since the function f has not been defined, it is part of this thesis project to find the optimum function and demonstrate it using the computer language (such as C).

Adaptable Mapping Architecture

As we know the essence of the modern network service is adaptation capability. Hence, it is reasonable to expect that the prices of the DiffServ level will not remain constant for the duration of the video transmission. For instance, the prices of all level of DiffServ will be higher at peak times than off-peak times.

The audio-visual scene over DiffServ environment is therefore highly dynamic, due to the huge number of flows being transmitted and the varying price over time. Moreover, as video scenes change rapidly, the mapping cannot be considered as a one-off computation; instead the mapping is continuous process for the duration of the video transmission. Therefore rather than finding an optimum, but computationally expensive solution to this non-linear optimization problem, this work focuses on finding a fast heuristic approach that provides a solution that is close to the optimum.

We are aware that there are many approaches to solve this particular optimization problem. However our aim is to find a solution using the heuristic approach rather than mathematical or other approaches, because the heuristic approach in most cases has better asymptotic complexity in solving the problem.

Finally, we will develop a working simulation program that will demonstrate and validate the idea of the user-oriented mapping mechanism in the DiffServ networks.

CHAPTER IV

QoS MAPPING FRAMEWORK IN DIFFSERV NETWORK

Loss and Delay Representations

QoS parameters should be modeled from a user's perspective. The representation of quality of service parameters as linear, linear-exponential, and other functions is reported in [12].

We model loss as a linear function. Loss is a summation function on the loss incurred at the different DiffServ levels (figure 3). The assumption here is that the impact of loss on the user is proportional. Therefore, the simplest formula would be:

$$L_i = F_i \cdot l_i \tag{4.1}$$

where:

- L_i is the score function for loss model level i
- i is DiffServ level i
- F_i are total flows sizes that will be transmitted through DiffServ level i in kilobytes
- l_i is the average loss in DiffServ level i

Since loss is additive and is based on the number of flows sizes transmitted through DiffServ, so we sum all the flow losses of each DiffServ level to get the score for loss. The score is a measure of the total loss. Therefore, from (4.1), given that *N* is the maximum level in DiffServ (including all sublevels) and *Ls* is the score function for overall loss model, we represent the loss model initially as follows:

$$Ls = \sum_{i=1}^{N} F_i \cdot l_i \tag{4.2}$$

The user's preferences for loss can be adjusted using the weighting value, the RLI (relative loss importance). The RLI value is calculated based on the table I and the relative importance of the objects given by the end-user. Hence, the final definition of loss is as follows:

$$Ls = \sum_{i=1}^{N} \frac{1}{RLI} F_i \cdot l_i$$

where:

- *Ls* is the score function for overall loss model
- *i* is DiffServ level *i*
- *N* is the maximum level in DiffServ, including all sublevels
- F_i are total flows sizes that will be transmitted through DiffServ level *i* in kilobytes
- l_i is the average loss in DiffServ level *i*
- *RLI* is the relative loss priority index

The empirical value of RLI is 1 ≥ RLI ≥ 25, where the smaller the RLI value, the less important the delay, vice versa.

A graph of loss model is shown below:

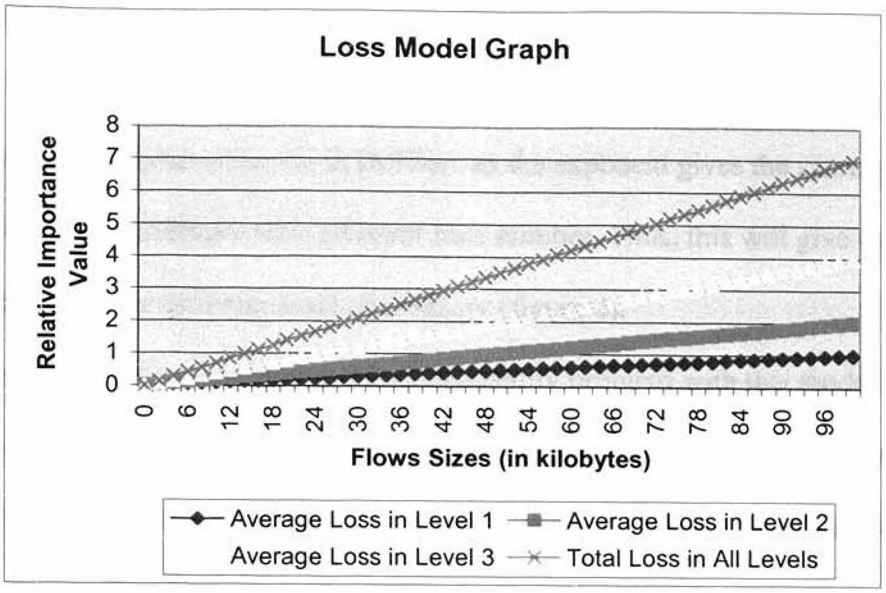


Figure 3. Loss Model Graph

Delay is modeled as a polynomial function. The impact of a small number of delayed flows on the user is negligible, as the human eye will not perceive the delay. However the effect of a large number of flow delays will be discernible to the user resulting in an unsatisfactory multimedia experience to the user. As flows sizes grow, the delay become more important. So the simplest function for the delay model would be:

$$D_i = F_i^M \tag{4.3}$$

where:

- D_i is the score function for the delay model level i
- i is DiffServ level i
- M is the number of main level of DiffServ

- *Fi* are total flows sizes that will be transmitted through DiffServ level *i* in kilobytes

Using the number of levels in DiffServ as the exponent gives the same growth rate at each level of DiffServ with different base number. Thus, this will give the "range" effect to the graph for different level of DiffServ (figure 4).

It's worth noting that there is minor scalability problem with this model. That is, the model cannot accommodate large number of DiffServ levels. One way to approach the scalability issue is to say that there are 3 main levels of DiffServ with 5 sublevels of each main level of DiffServ.

To avoid a combinatorial explosion, since we do not have the maximum number of flows sizes, we need to scale down the sizes of the flows proportionally. Practically, we can divide the flows sizes in the particular level with the maximum sizes that can be transmitted through that level. Hence, from (4.3) and given that *N* is maximum level of DiffServ (including all sublevels), the updated function would look like this:

$$D_i = \left(\frac{F_i}{\sum_{i=1}^{N} F_i}\right)^M \tag{4.4}$$

Because each level has different configuration of average delay value, we need include this effect in to our delay score function. The average delay value would affect the overall flows sizes transmitted through the particular level. So, from (4.4), given d_i is the average delay in DiffServ level *i* and d_b is the average delay in DiffServ level 1, we formulate the function as follows:

$$D_i = \left(\frac{F_i}{\sum_{i=1}^{N} F_i \Big/ (1 + \frac{d_i}{d_b})}\right)^M \tag{4.5}$$

Another thing worth mentioning is the average delay value has to be proportionally scaled, because the nature of unit we use to represent the average delay is quite large, that is in milliseconds. If we do not scale down the average delay value, not only we might end up an infinitesimally small delay score, which makes the score harder to analyze, but we are also going to lose the purpose of scaling down the flows sizes mentioned earlier.

Since we are interested in the overall delay score information for the whole DiffServ level, from (4.5), we average the total of delay score of each level.

$$Ds = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{F_i}{\sum_{i=1}^{N} F_i \Big/ (1 + \frac{d_i}{d_b})}\right)^M \tag{4.6}$$

The average calculation is better than the summation for this particular delay model, because when we sum the delay score of each level of DiffServ, the delay score of the worst level will far outweigh the delay score of the better level. In other words, the calculation will be mainly based on the worst level delay score.

The delay score is weighted based on the end-user experience, using the RDI value. The RDI value is calculated based on table I and the relative importance of the objects given by the end-user.

$$Ds = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{RDI}\left(\frac{F_i}{\sum_{i=1}^{N}F_i \Big/ (1+\frac{d_i}{d_b})}\right)^M$$

where:

- *Ds* is the score function for the overall delay model
- *i* is DiffServ level *i*
- *N* is maximum level of DiffServ, including all sublevels
- *M* is the number of main level of DiffServ
- F_i are total flows sizes that will be transmitted through DiffServ level *i* in kilobytes
- d_i is the average delay in DiffServ level *i*
- d_b is the average delay in DiffServ level 1
- *RDI* is the relative delay priority index

After running several tests, for this particular delay score function, we conclude empirically that the range of RDI value is $1 \geq \text{RDI} \geq 88$, where the smaller the RDI value, the less important the delay, and vice versa.

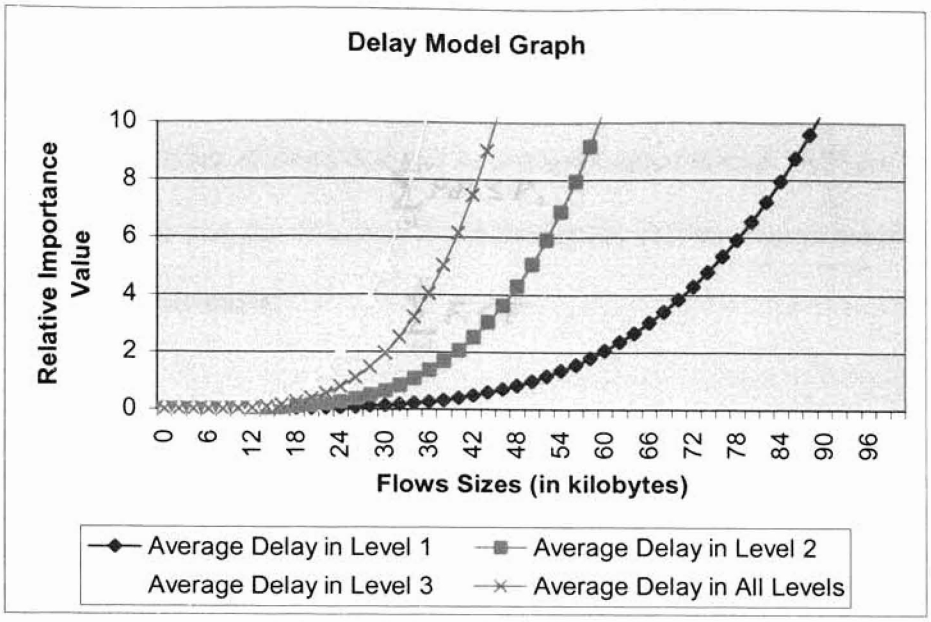


Figure 4. Delay Model Graph

The delay at each DiffServ level is represented as such (figure 4). The average or expected value models the overall effect of delay. Research is required to identify the relationship between QoS parameters such as loss/ delay and user's perception. Only then can communications systems and protocols be designed from a user's perspective.

The maximum QoS is therefore a sum of the loss and delay scores. In other words, the maximizing QoS will be obtained by minimizing the sum of the delay and the loss scores. For the particular total budget, we want to get the combination that has the lowest score. The lower the score, the better the QoS.

$$\max QoS = \min(Ls + Ds)$$

subject to:

$$\sum_{i=1}^{N} F_i P_i \leq P,$$

$$\sum_{i=1}^{N} F_i \leq F$$

where:

- P is total budget of the user
- F is total flows sizes to be transmitted in kilobytes
- F_i are total flows sizes that will be transmitted through DiffServ level i in kilobytes
- P_i are the prices of the DiffServ level i
- N is the maximum level in DiffServ
- P, F, F_i, P_i and N are element of integer

One heuristic implementation to solve this minimization problem is to use a binary search algorithm.

Mapping Mechanism Algorithm

Mapping a flow to one or more DiffServ level consists of two steps:

- Irrespective of the number of packets, consider all packets as belonging to the same flow and determine the DiffServ distribution that will yield the minimum score and satisfy the budget. The output of this step will indicate how many packets can be accommodated at each DiffServ level.

- Given the number of flows that can be accommodated in each DiffServ level, the next step is to map the distribute flows among the DiffServ levels based on the relative priority index.

We use binary search to reduce the search space for finding the optimum combination that will yield the minimum score (maximum QoS) and satisfy the budget (first step above). The algorithm does not distinguish between flows. The algorithm is briefly outlined below and consists of two stages.

Stage I

The algorithm begins by assigning all flows to the highest DiffServ level that the budget permits. If this assignment results in an under-budget assignment, half of the flows are assigned to the next higher DiffServ level. If this combination is over-budget, half of the flows in the higher DiffServ level are put back into the lower level. If the new distribution is still under-budget, half of the remaining flows in the lower level are assigned to the higher level. This procedure is repeated until a two-level DiffServ combination that utilizes the maximum allowable budget is found.

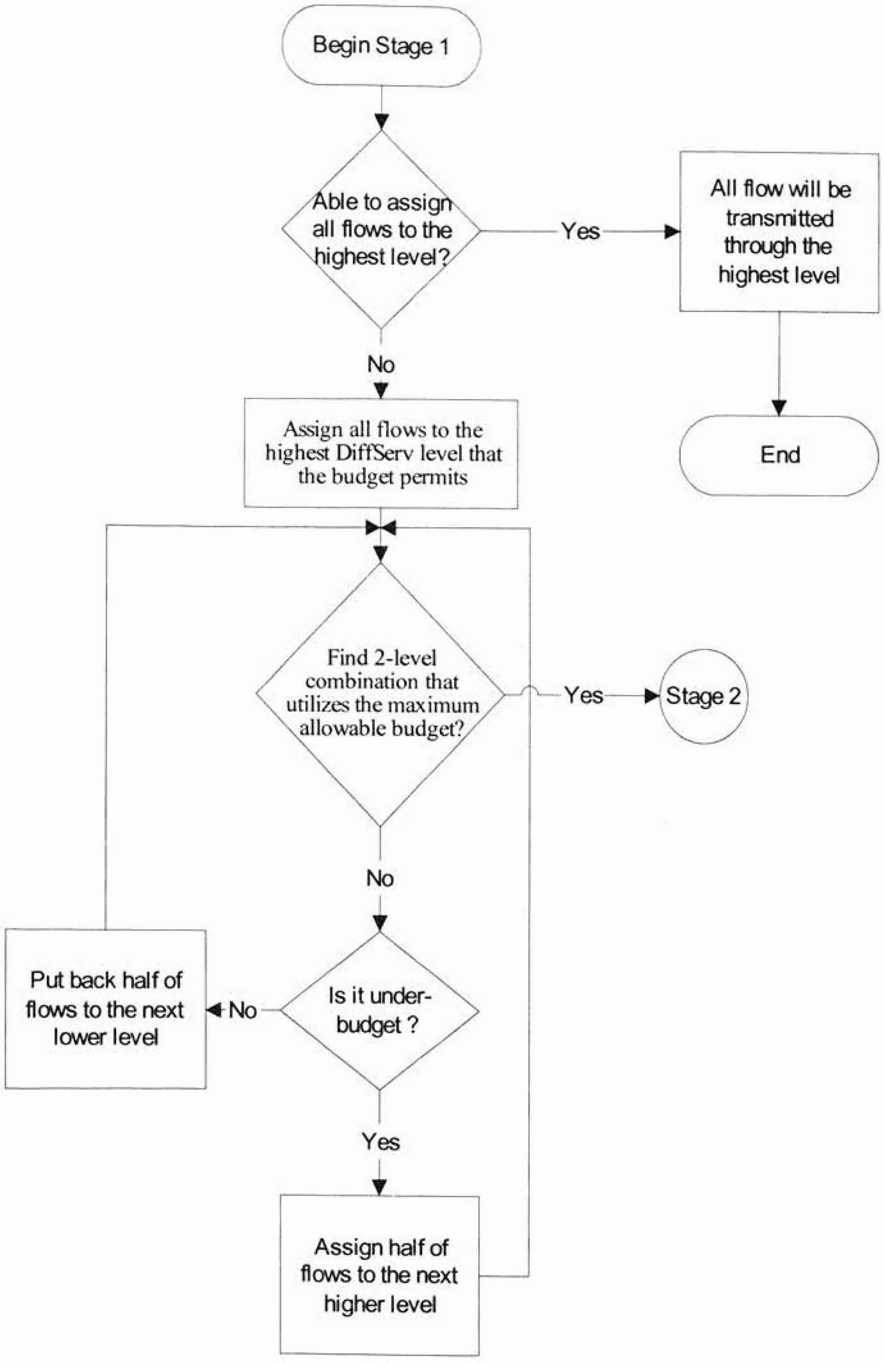

Begin Stage 1
Able to assign all flows to the highest level?
Yes
All flow will be transmitted through the highest level
End
No
Assign all flows to the highest DiffServ level that the budget permits
Find 2-level combination that utilizes the maximum allowable budget?
Yes
Stage 2
No
Is it under-budget ?
No
Put back half of flows to the next lower level
Yes
Assign half of flows to the next higher level

Figure 5. Mapping Mechanism Flowchart Stage 1

Stage II

In the second stage of the algorithm, the flows are distributed across three levels. Half of the flows in the middle level are transferred to the next higher DiffServ level and the other half to the next lower DiffServ level. If the result of this assignment is over budget, then half of the flows in the highest level are assigned into the lowest level. This process is repeated until the mechanism finds an under-budget combination. Next, the number of flows in the highest DiffServ level is not modified, instead the algorithm now adjusts flows sizes in the middle DiffServ level and the lower DiffServ level. Binary search is once again employed to modify the flows sizes in the middle DiffServ level and the lower DiffServ level. When this three-level utilization reaches the maximum budget allowed, the process stops. This optimum combination will be kept in the buffer for future comparison.

The algorithm proceeds to consider the most recent under-budget or over-budget combination. If the current combination is under-budget, the flows sizes in the highest level of the current combination are increased by the flows sizes of half of the flows of the most recent highest level under-budget combination. Alternatively, if the current combination is over-budget, decrease the flows sizes in the highest level by half of the flows at the most recent highest level over-budget combination. The process is repeated until the maximum utilization of the budget.

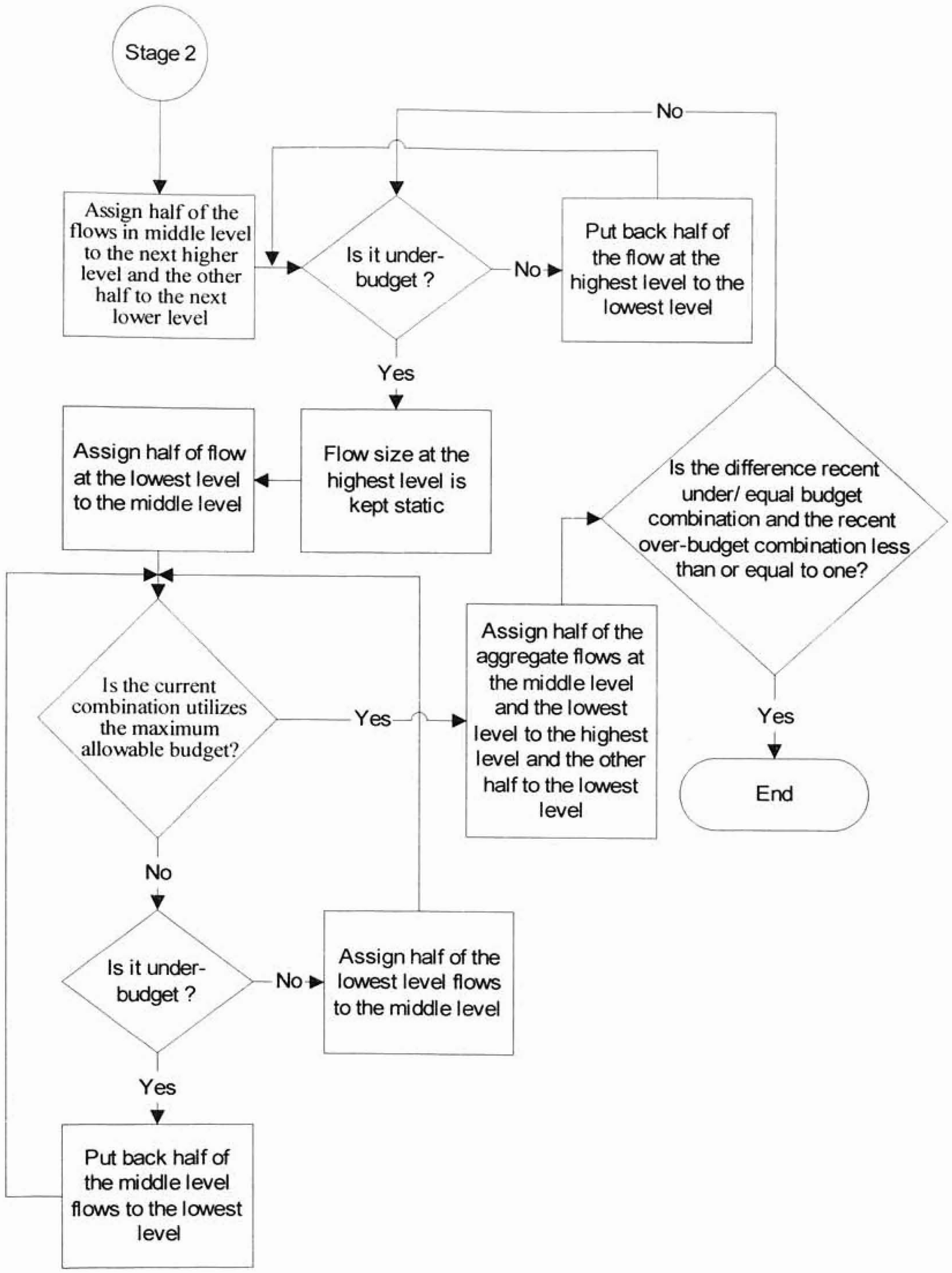
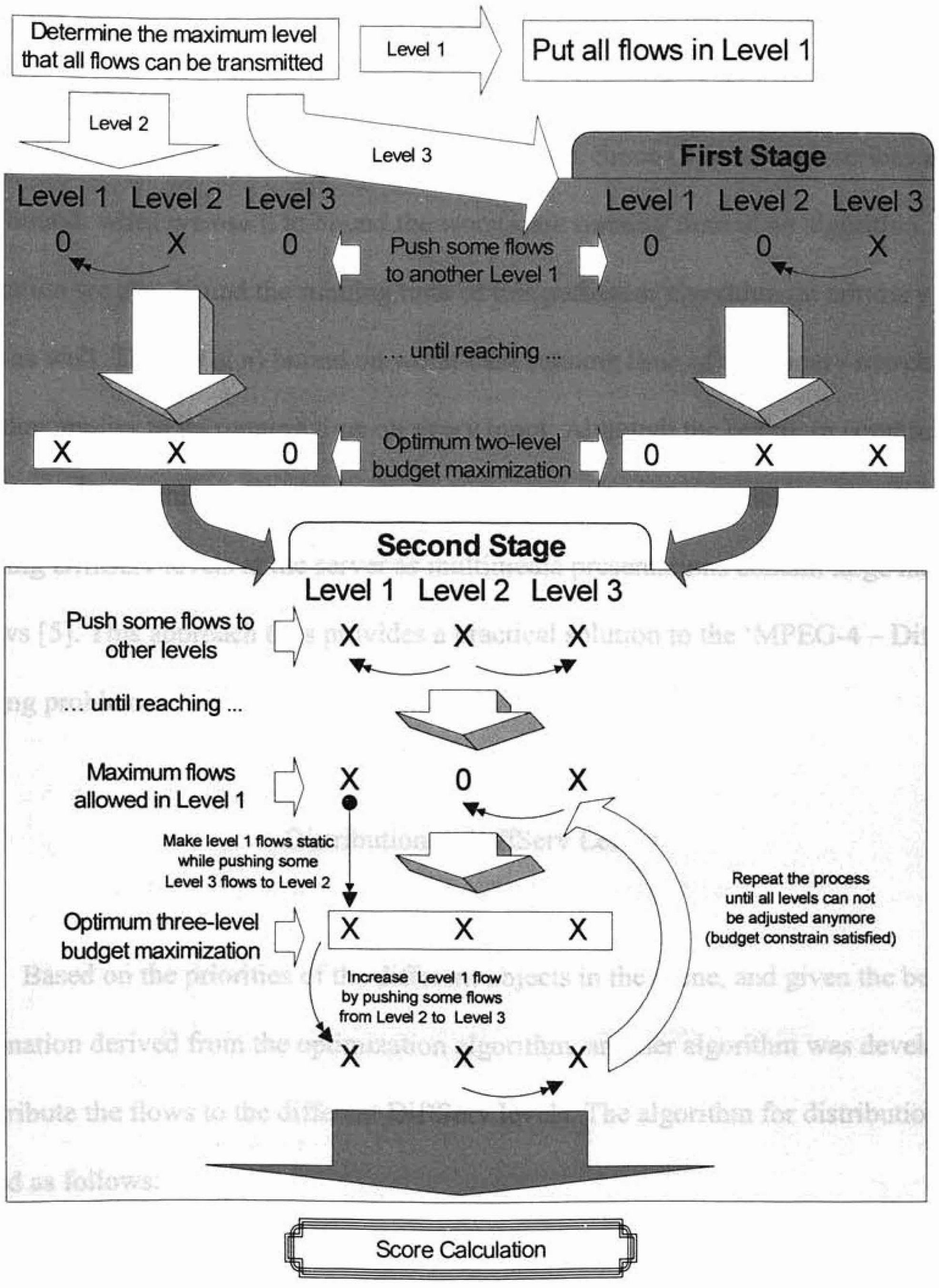

Stage 2
Assign half of the flows in middle level to the next higher level and the other half to the next lower level
Is it under-budget ?
No
Put back half of the flow at the highest level to the lowest level
No
Yes
Flow size at the highest level is kept static
Assign half of flow at the lowest level to the middle level
Is the difference recent under/ equal budget combination and the recent over-budget combination less than or equal to one?
Yes
End
Is the current combination utilizes the maximum allowable budget?
Yes
Assign half of the aggregate flows at the middle level and the lowest level to the highest level and the other half to the lowest level
No
Is it under-budget ?
No
Assign half of the lowest level flows to the middle level
Yes
Put back half of the middle level flows to the lowest level

Figure 6. Mapping Mechanism Flowchart Stage 2

The underlying objective of this algorithm is to transmit as much of the flow through the higher DiffServ levels. The outline algorithm is illustrated in figure 7. The best combination is the one that gives the minimum score and this is defined to be the DiffServ distribution for the given budget.



Determine the maximum level that all flows can be transmitted
Level 1
Put all flows in Level 1
Level 2
Level 3
First Stage
Level 1 Level 2 Level 3
0 X 0
Push some flows to another Level 1
0 0 X
... until reaching ...
X X 0
Optimum two-level budget maximization
0 X X
Second Stage
Level 1 Level 2 Level 3
Push some flows to other levels
X X X
... until reaching ...
Maximum flows allowed in Level 1
X 0 X
Make level 1 flows static while pushing some Level 3 flows to Level 2
Optimum three-level budget maximization
X X X
Repeat the process until all levels can not be adjusted anymore (budget constrain satisfied)
Increase Level 1 flow by pushing some flows from Level 2 to Level 3
X X X
Score Calculation

Figure 7. Complete Mapping Mechanism Illustration

Complexity Analysis

The complexity of this algorithm is O(lg *n*), where *n* is the total flow size. The running time to evaluate the best combination is mainly based on this algorithm, so the complexity to produce the best combination is O(lg *n*). Since O-notation describes an upper bound, when we use it to bound the worst-case running time of an algorithm, by implication we also bound the running time of this particular algorithm on arbitrary inputs as well. Thus O(lg *n*) bound on worst-case running time of this binary search algorithm applies to its running time on every input. Although the optimum combination may not be found, this algorithm results in fast computation time which is essential for assigning DiffServ levels at the server as multimedia presentations contain large numbers of flows [5]. This approach thus provides a practical solution to the 'MPEG-4 – DiffServ' mapping problem.

Distribution to DiffServ Levels

Based on the priorities of the different objects in the scene, and given the best combination derived from the optimization algorithm, another algorithm was developed to distribute the flows to the different DiffServ levels. The algorithm for distribution is defined as follows:

```
lastlevel: the last (or lowest) DiffServ level
lastobject: the last object defined in the particular scene
```

```
buffer(i, o, x): a function which add the x number of flows of object o
                 to DiffServ level i buffer
number(DiffServ): a function which returns the total number of DiffServ
                  level
size(o): a function which returns the number of flows of object o
storage(i): a function which returns the remaining number of flows that
            can be transmitted through DiffServ level i for the
            particular scene
priority(o): a function which returns the priority flag of object o
total(a): a function which returns the total flows of priority a that
          have not been assigned to any DiffServ level

i ← 1 //start at DiffServ level 1
o ← 1 //start with first object

Repeat
{
      if size(o) > 0
      {

            // Put a appropriate amount of packets to the DiffServ
               level buffer to ensure the fairness among the objects

            p ← min(size(o), size(o)/ total(priority(o)) * storage(i))

            buffer(i, o, p)

            storage(i) ← storage(i) - p
            size(o) ← size(o) - p
            total(priority(o)) ← total(priority(o)) - p
      }
      ++o

      if o = lastobject
      {
            i++    // next level
            o ← 1 // start again from the first object
      }

      if i = lastlevel
      {
            // Put all remaining object to the lowest level

            Repeat
            {
                  if size(o) > 0
                        buffer(i, o, size(o))
                  ++o

            } until o = lastobject
      }
} until i = number(DiffServ) AND best(i) ≤ 0
```

The algorithm assigns all objects to the highest levels feasible, with highest priority objects assigned first. If all the highest level objects cannot be assigned to level 1, the objects are distributed proportionally to level 1 based on their sizes.

Suppose there is an audio-visual scene that with the following data sizes and user-oriented preferences (priority):

TABLE III

MPEG-4 OBJECTS AND THEIR PARAMETERS EXAMPLE

Object Type	**Size**	**User-defined Priority**
V_1	15 KB	High
V_2	20 KB	Medium
V_3	30 KB	Low
A	20 KB	High
T	15 KB	Low

The best combination suggested by the binary search algorithm is as follows:

TABLE IV

BEST COMBINATION SUGGESTED

DiffServ Level	**Suggested Size**
Level 1	40 KB
Level 2	20 KB
Level 3	40 KB

Therefore the final distribution is:

TABLE V

FINAL DISTRIBUTION

DiffServ	**Distribution**		
Level 1	V_1 *15 KB*	A *20 KB*	V_2 *5 KB*
Level 2	V_2 *15 KB*	V_3 *3 KB*	T *2 KB*
Level 3	V_3 *27 KB*	T *13 KB*	

CHAPTER V

EXPERIMENTAL RESULTS

We ran several simulations to test the algorithm we presented before. The hardware specifications used for the simulations are listed below:

TABLE VI

COMPUTER SPECIFICATIONS

Processor	AMD™ Athlon Thunderbird 1.1 Ghz
Motherboard	ASUS™ A7V (VIA KT133A Chipset)
Random Access Memory	Corsair Micro™ 512 MB PC-142 SDRAM CL2
SCSI Controller	Adaptec™ AHA-2940UW
Hard Drive	Western Digital™ 9.1 GB WD91 Ultra2 SCSI
Graphics Card	Gainward™ 128 MB DDR GeForce 3 Ti-500
Network Interface Card	Linksys™ USB100TX USB 100 mbit
Operating System	Windows 2000 Advanced Server™

Table VII below shows the prices, delay and loss for one of our simulations. For example, it costs \$4 to transmit 1 KB over DiffServ level 1, and the average delay and loss in level 1 are 100 ms and 1% per KB, respectively. Figure 8 shows the flow allocation across the 3 DiffServ levels with increasing budget using our binary search heuristic algorithm described in Chapter IV. Figure 9 shows the optimum flow allocation, which considers all possible combinations. The binary search heuristic does not consider all possible allocations. However, as figure 10 indicates, the scores, which is measure of the QoS delivered to the user, are almost identical in both cases. This is significant as our ultimate objective is to deliver optimum QoS to end-user. This result therefore confirms the validity of our approach. Even if the budget allows all flows to be transmitted at level 1, some flows are still transmitted at level 2. This is because of our approach to modeling delay, which is as in figure 4. Such an approach does not try to concentrate all the flows in one level. Although most of the flows are in level 1, a spread of flow delays is envisioned as being more beneficial to the user. An alternate model would specify delay in terms of the worst-case delay. Such a model is appropriate for data flows, but not for audio-visual flows where if only a relatively few flows are delayed, the impact on the user is negligible. Yet another approach would be a simple average.

TABLE VII

SIMULATION DATA I

Total flows sizes to be transmitted = 100 KB			
DiffServ	**Price**	**Average Delay**	**Average Loss**
Level 1	\$4 per KB	100 ms	1% per KB
Level 2	\$2 per KB	200 ms	2% per KB
Level 3	\$1 per KB	400 ms	4% per KB

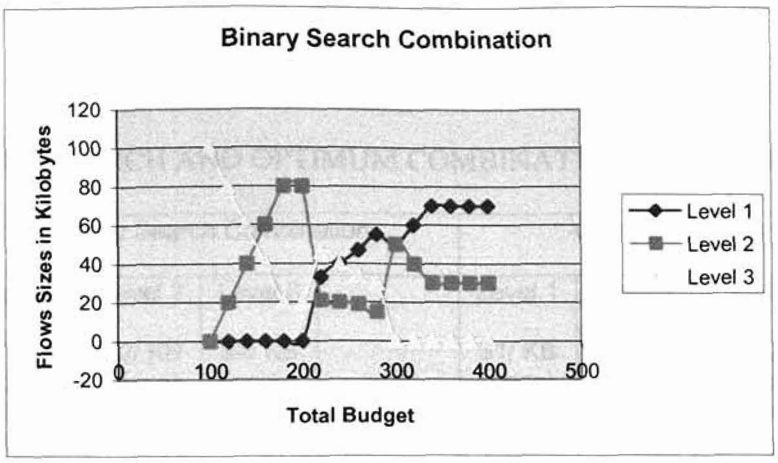


Figure 8. Binary Search Combination Result 1 – Polynomial Delay Function

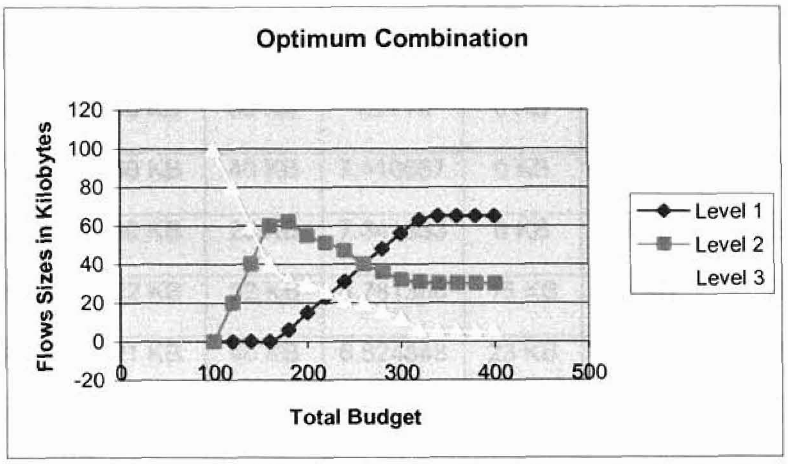


Figure 9. Optimum Combination Result 1 – Polynomial Delay Function

Table VIII shows the flow distribution using the binary search heuristic and the optimum flow distribution. We can see the overall score values of binary search heuristic are closely identical with the optimum combination.

TABLE VIII

BINARY SEARCH AND OPTIMUM COMBINATIONS COMPARISON

Total Budget	Binary Search Combination				Optimum Combination			
	Level 1 $1/ KB (price) 100 ms (delay) 1% (loss)	Level 2 $2/ KB (price) 200 ms (delay) 2% (loss)	Level 3 $4/ KB (price) 400 ms (delay) 4% (loss)	Score (*Ls* + *Ds*)	Level 1 $1/ KB (price) 100 ms (delay) 1% (loss)	Level 2 $2/ KB (price) 200 ms (delay) 2% (loss)	Level 3 $4/ KB (price) 400 ms (delay) 4% (loss)	Score (*Ls* + *Ds*)
$100	0 KB	0 KB	100 KB	45.66667	0 KB	0 KB	100 KB	45.66667
$120	0 KB	20 KB	80 KB	25.00533	0 KB	20 KB	80 KB	25.00533
$140	0 KB	40 KB	60 KB	12.776	0 KB	40 KB	60 KB	12.776
$160	0 KB	60 KB	40 KB	7.410667	0 KB	60 KB	40 KB	7.410667
$180	0 KB	80 KB	20 KB	7.341333	6 KB	62 KB	32 KB	6.090861
$200	26 KB	22 KB	52 KB	8.781368	15 KB	55 KB	30 KB	5.081375
$220	33 KB	21 KB	46 KB	6.824848	23 KB	51 KB	26 KB	4.248638
$240	40 KB	20 KB	40 KB	5.309333	31 KB	47 KB	22 KB	3.587516
$260	47 KB	19 KB	34 KB	4.186259	40 KB	40 KB	20 KB	3.08
$280	55 KB	15 KB	30 KB	3.649042	48 KB	36 KB	16 KB	2.725483
$300	50 KB	50 KB	0 KB	2.958333	56 KB	32 KB	12 KB	2.515221
$320	60 KB	40 KB	0 KB	2.552000	63 KB	31 KB	6 KB	2.433911
$340	70 KB	30 KB	0 KB	2.457667	65 KB	30 KB	5 KB	2.430542
$360	70 KB	30 KB	0 KB	2.457667	65 KB	30 KB	5 KB	2.430542
$380	70 KB	30 KB	0 KB	2.457667	65 KB	30 KB	5 KB	2.430542
$400	70 KB	30 KB	0 KB	2.457667	65 KB	30 KB	5 KB	2.430542

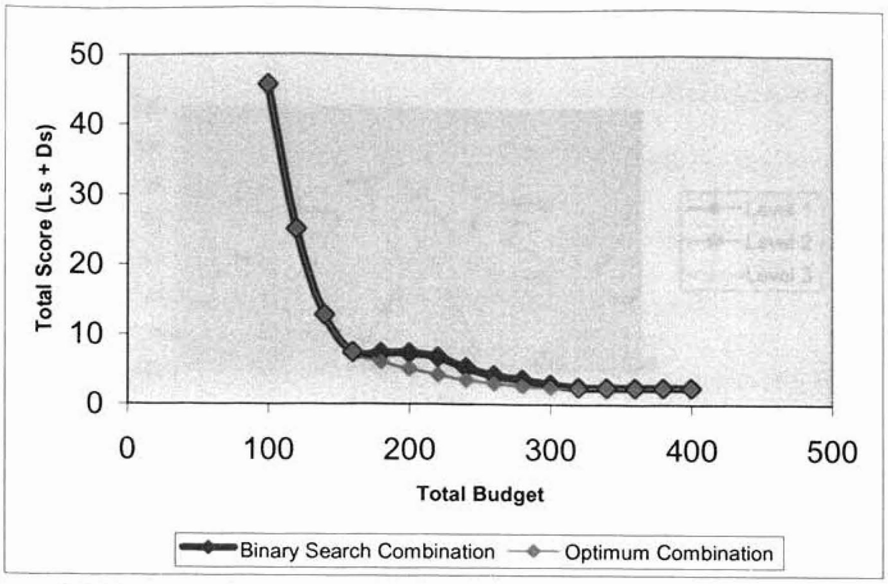


Figure 10. Scores: Binary Search Combination vs. Optimum Combination 1

Other results are shown below in figures 11 and 12 with a different value for average loss and average delay.

TABLE IX

SIMULATION DATA II

Total flows sizes to be transmitted = 100 KB			
DiffServ	**Price**	**Average Delay**	**Average Loss**
Level 1	$4 per KB	100 ms	1% per KB
Level 2	$2 per KB	200 ms	2% per KB
Level 3	$1 per KB	700 ms	7% per KB

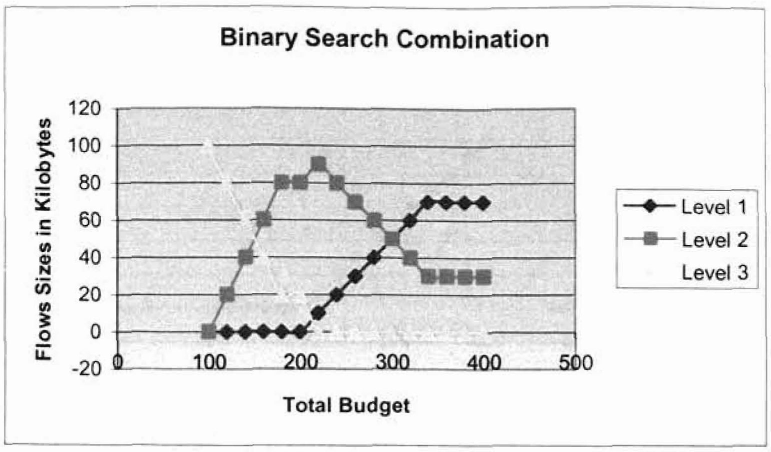


Figure 11. Binary Search Combination Result 2 – Polynomial Delay Function

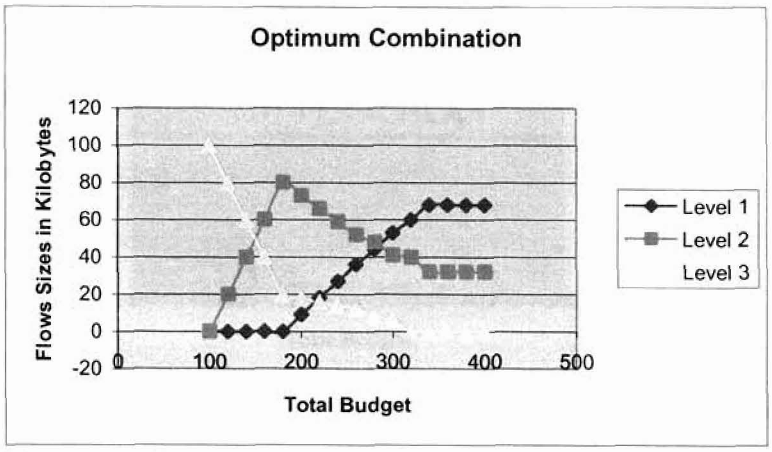


Figure 12. Optimum Combination Result 2 – Polynomial Delay Function

An alternative model where both delay and loss are captured as linear functions was also simulated (figures 13 and 14). The simulation showed that the binary heuristic algorithm and the optimum allocation produced identical results, that is, identical scores and identical allocation of flows to the different DiffServ levels. This simulation also showed that budget permitting, the entire transmission takes place at the highest DiffServ level 1.

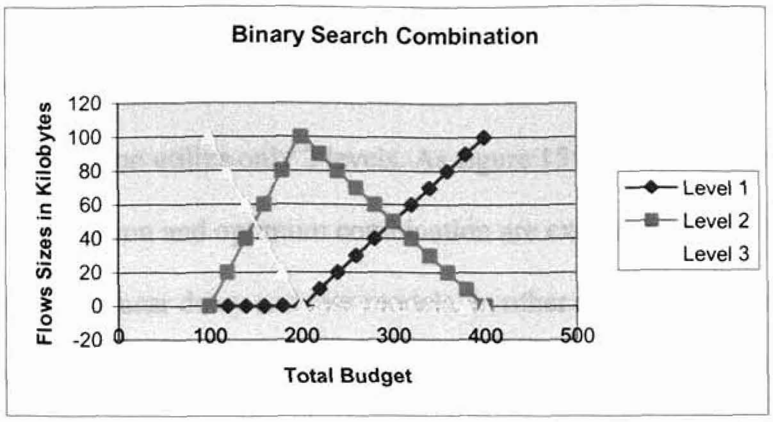


Figure 13. Binary Search Combination Result – Linear Delay Function

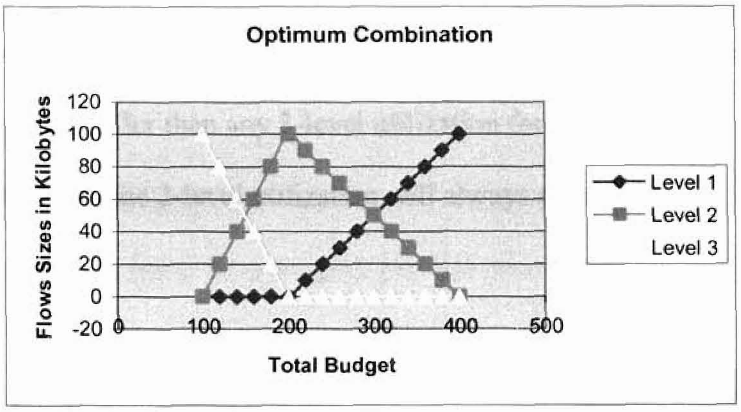


Figure 14. Optimum Combination Result - Linear Delay Function

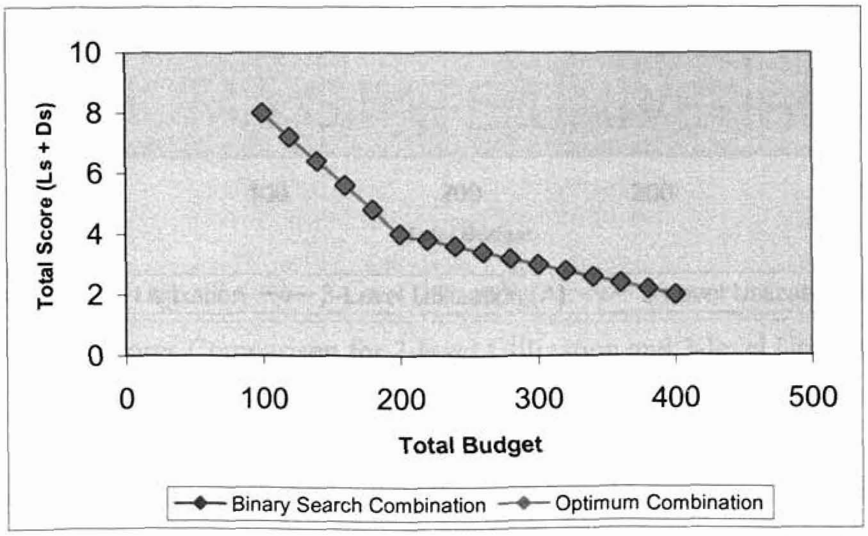


Figure 15. Scores: Binary Search Combination vs. Optimum Combination 2

From figures 13 and 14, we can see that both the binary search combination and the optimum combination utilize only 2 levels. As figure 15 shows, the score for both the binary search combination and optimum combination are exactly identical. This is the mapping obtained for linear delay and loss models. In other words, when loss and delay are modeled as linear functions, the result is a 2-level utilization, whereas if delay is modeled as polynomial function, the resulting allocation is a 3-level utilization.

Figure 16 is generated using the binary search algorithm. It suggests, when both delay and loss are modeled as linear functions, the total score (Ls + Ds) for 2-level utilization is always smaller than any 3-level utilization for all possible total budgets in the graph, which means the 2-level utilization will always maximize QoS.

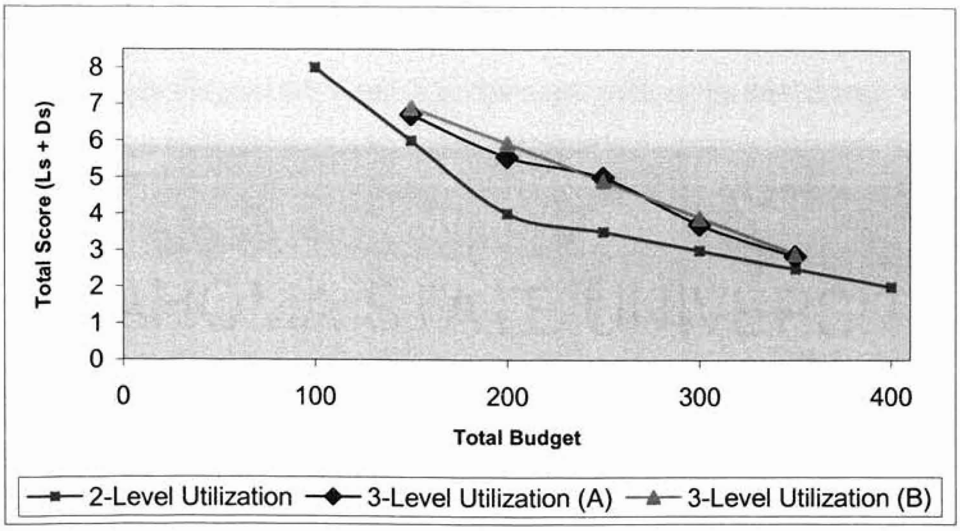


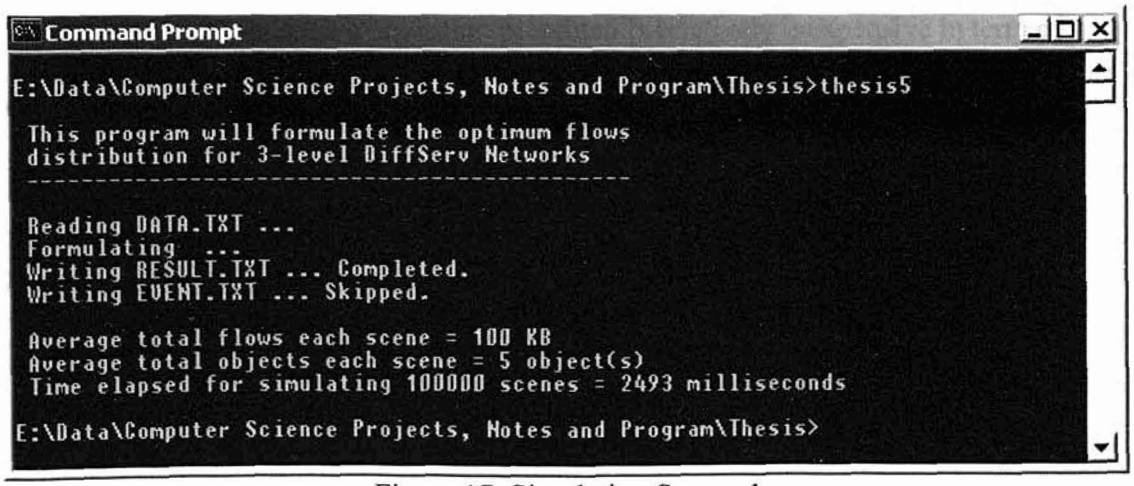
Figure 16. Scores Comparison for 2-level Utilization and 3-level Utilization

This phenomenon occurs because of the model used to represent delay and loss. As shown by these experiments, functions used to model QoS parameters have an impact on the total QoS. The impact of QoS parameters on user's perception has not been reported in the literature. It will not be possible to provide a precise mapping mechanism to the different DiffServ levels such that user's perception is enhanced until the perceptual impact of deteriorating QoS on the user has been studied in detail.

After running many tests, using different values of prices, average loss, average delay and number of flows, in most cases, the combination of binary search algorithm approach is almost identical or exactly identical to the optimum combination. Therefore, we conclude empirically that the mapping algorithm is valid.

The experiments also measured the time consumed for the mapping algorithm in order to process the raw artificial data and produce the combination.

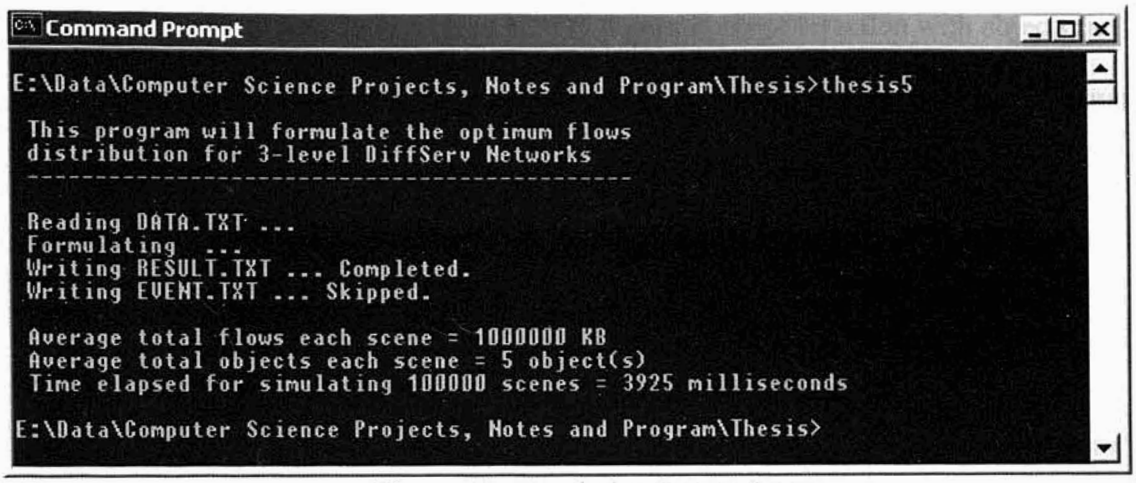
To compute 100,000 audio-visual scenes continuously with an average of 5 objects each scene (total flows sizes roughly 100 KB each scene) takes 2,493 milliseconds.



```
E:\Data\Computer Science Projects, Notes and Program\Thesis>thesis5

 This program will formulate the optimum flows
 distribution for 3-level DiffServ Networks
 ------------------------------------------

 Reading DATA.TXT ...
 Formulating  ...
 Writing RESULT.TXT ... Completed.
 Writing EVENT.TXT ... Skipped.

 Average total flows each scene = 100 KB
 Average total objects each scene = 5 object(s)
 Time elapsed for simulating 100000 scenes = 2493 milliseconds

E:\Data\Computer Science Projects, Notes and Program\Thesis>
```

Figure 17. Simulation Screen 1

Using the same parameters as above, we increase the flows sizes by about 10,000 times, so the total flows sizes now are roughly about 1,000,000 KB for each scene. The time consumed to compute 100,000 scenes continuously is 3,925 milliseconds.



```
E:\Data\Computer Science Projects, Notes and Program\Thesis>thesis5

 This program will formulate the optimum flows
 distribution for 3-level DiffServ Networks
 ------------------------------------------

 Reading DATA.TXT ...
 Formulating  ...
 Writing RESULT.TXT ... Completed.
 Writing EVENT.TXT ... Skipped.

 Average total flows each scene = 1000000 KB
 Average total objects each scene = 5 object(s)
 Time elapsed for simulating 100000 scenes = 3925 milliseconds

E:\Data\Computer Science Projects, Notes and Program\Thesis>
```

Figure 18. Simulation Screen 2

It is worth noting that the time consumed includes two I/O operations (reading the artificial data from the hard drive and store the result back to the hard drive). Therefore, we can say that the mapping algorithm presented is relatively inexpensive in term of computational time.

CHAPTER VI

CONCLUSION AND FUTURE WORK

Audio-visual standards such as MPEG-4 permit user interaction with objects in the audio-visual scene and still picture standards such as JPEG-2000 permit specification of 'regions of interest'. In this thesis, we have proposed a mapping to assign user-prioritized audio-visual objects to DiffServ levels such that QoS is maximized within given budget constraints. This mapping is a two-phase process. In the first phase the relative priorities of QoS parameters are derived and in the second phase, flows are assigned to DiffServ levels based on the required QoS.

The ultimate objective of the mapping is to provide the user with optimum quality. The basic idea is to transmit user specified higher priority objects at higher DiffServ levels, thus benefiting the user. This approach is also applicable to JPEG-2000 images.

The validity of our approach has been confirmed by experimental results. Therefore, the mapping algorithm presented provides a fast heuristic approach, which has been shown to be empirically legitimate for user-oriented object-based multimedia communication over DiffServ networks.

Scenes in a video scene may change resulting in different objects for a different scene. Even if the objects are identical, their sizes may change (zooming in for example). The work reported here does not take into account the changes in the number and types of objects caused by scene changes or the changes in data sizes of objects caused by different views, zooming, etc.

Other future work will focus on the following aspects: the extension of the proposed approach to meet the constraints caused by network congestion, more analytical experiments to understand the impact of QoS from a user's perspective, and extending the mapping mechanism to deal with the multiple users.

REFERENCES

1. R. T. Apteker, J. A. Fisher, V. S. Kisimov and H. Neishlos, "Video Acceptability and Frame Rate," *IEEE Multimedia*, vol. 2, no.3, Fall 1995, pp. 32-40.

2. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, IETF, December 1998.

3. R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, IETF, June 1994.

4. R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource Reservation Protocol (RSVP)," RFC 2205, IETF, September 1997.

5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to Algorithms," *The MIT Press*, 2002, pp. 26-27.

6. G. Ghinea and J. P. Thomas, "QoS Impact on User Perception and Understanding of Multimedia Video Clips," *Proceedings of the ACM Multimedia Conference '98*, 1998.

7. Ghinea and J. P. Thomas, "Crossing the Man-Machine Divide: A Mapping based on Empirical Results," *The Journal of VLSI Signal Processing - Systems for Signal, Image, and Video Technology*, vol. 29, no. 1-2, August/ September 2001, pp. 139-147.

8. ISO/IEC JTC1/SC29/WG11 (MPEG), website: http://www.cselt.it/mpeg.

9. JPEG2000 Part 2 Final Committee Draft, ISO/IECJTC1/SC20 WG1 N2000, December 2000.

10. G. Kühne, and C. Kuhmünch, "Transmitting MPEG-4 Video Streams over the Internet: Problems and Solutions," *ACM Multimedia '99 (Part 2)*, October 1999, pp.135-138.

11. H. Lee, H. Kwon, and Y. Nemoto, "Guaranteeing Multiple QoSs in Differentiated Services Internet," *IEEE Transaction on Multimedia*, June 2002, pp. 233 – 238.

12. A.G. Malamos, E.N. Malamas, T.A. Varvarigou and S.R. Ahuja, "A Model for Availability of Quality of Service in Distributed Multimedia System," *Multimedia Tools and Applications 16*, 2002, pp. 207-230.

13. K. Nichols, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," RFC 2638, IETF, July 1999.

14. A. Puri, and A. Eleftheriadis, "MPEG-4: an Object-based Multimedia Coding Standard Supporting Mobile Applications," Mobile Networks and Applications 3, 1998, pp. 5-32.

15. J. Shin, J. Kim, and C.-C. J. Kuo, "Content-based Packet Video Forwarding Mechanism in Differentiated Service Networks," *International Packet Video Workshop*, Sardinia, Italy, May 2000.

16. J. Shin, J. Kim, and C.-C. J. Kuo, "Quality-of-Service Mapping Mechanism for Packet Video in Differentiated Services Network," *IEEE Trans. on Multimedia*, vol. 3, no. 2, June 2001, pp. 219-231.

17. J. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, IETF, September 1997.

18. M. Zitterbart, "A Model for Flexible and High Performance Communication Sub-systems", *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 1, January 1993, pp. 507-518.

SELECTED BIBLIOGRAPHY

Cisco White Paper, "Implementing DiffServ for End-to-End Quality of Service," *Cisco IOS Release 12.1(5)T*, 2001.

D. Q. Hai, and S. T. Vuong, "Dynamic-Distributed Differentiated Service for Multimedia Applications," *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2000, pp. 586-594.

H. Lee, H. Kwon, and Y. Nemoto, "Guaranteeing Multiple QoSs in Differentiated Services Internet," *Seventh IEEE International Conference on Parallel and Distributed Systems*, July 2000, pp. 233-238.

Z. Jiang, and L. Kleinrock, "An Adaptive Network Prefetch Scheme," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, April 1998, pp. 358-368.

A. van der Schaaf, and J. A. L. Arellano, "On balancing Multiple Video Streams with Distributed QoS Control in Mobile Communications," *2nd International Symposium of Mobile Multimedia Systems & Applications*, November 2000, pp. 126-133.

APPENDIX A

SOURCE CODE OF MAPPING MECHANISM ALGORITHM IN C LANGUAGE

```
/*****************************************************************************
 * Program Name: MAPMECH.C                                                   *
 *                                                                           *
 * Author: Karyanta Purne                                                    *
 *                                                                           *
 * Description: Simulation of the mapping mechanism described in this thesis *
 *                                                                           *
 * Remark: There are very minimum internal documentation about this program  *
 *         as the documentation is strictly not required for this purpose.   *
 *         The algorithm and flowchart of this program is described in detail *
 *         in Chapter IV of this thesis.                                     *
 *         However if you have any question about this program, you can write *
 *         to karyanta@hotmail.com. I will gladly answer any question you have *
 *****************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
#include<math.h>
#include<time.h>

#define MAXSTORAGE 1024
#define TIMESIZE 256

#define VDEL 2
#define ADEL 2
#define TDEL 1
#define VLOS 1
#define ALOS 3
#define TLOS 3

void initialize(void);
void level1Process(int);
void firstStageLevel2(int);
void firstStageLevel3(int);
void secondStage(int, int, int, int, int);
void scoreEvaluation(int);
void store(struct objectInfo *i, struct objectInfo **start, struct objectInfo **last);
void storeLevel1(struct level1Info *i, struct level1Info **level1Start, struct level1Info
**level1Last);
void storeLevel2(struct level2Info *i, struct level2Info **level2Start, struct level2Info
**level2Last);
void storeLevel3(struct level3Info *i, struct level3Info **level3Start, struct level3Info
**level3Last);
void view(void);
void viewDist(void);
void distribution(void);

int totalBudget, i, j, max;
double minScore;
int level1Price, level2Price, level3Price;
float level1Delay, level2Delay, level3Delay;
float level1Loss, level2Loss, level3Loss;
int level1Packet, level2Packet, level3Packet;
int level1Store[MAXSTORAGE], level2Store[MAXSTORAGE], level3Store[MAXSTORAGE],
priceStore[MAXSTORAGE];
int bestLevel1, bestLevel2, bestLevel3, bestPrice;
int totalHigh, totalMed, totalLow;
int DELWg, LOSWg;
double videoDEL, videoLOS, audioDEL, audioLOS, textDEL, textLOS;
double bestDEL, worstDEL, bestLOS, worstLOS, denomDEL, denomLOS, QoPDEL, QoPLOS;
double delay[MAXSTORAGE], loss[MAXSTORAGE], score[MAXSTORAGE];
int videoCount, audioCount, textCount, prioritySum;
int video[MAXSTORAGE], audio[MAXSTORAGE], text[MAXSTORAGE];
int vPriority[MAXSTORAGE], aPriority[MAXSTORAGE], tPriority[MAXSTORAGE];
char* getTime(void);
FILE *eventFile, *dataFile, *resultFile;

struct objectInfo {
        int type;
        int no;
        int size;
        int priority;
        struct objectInfo *next;
};

struct level1Info {
        int type;
        int no;
        int size;
        int priority;
        struct level1Info *next;
};

struct level2Info {
        int type;
        int no;
        int size;
        int priority;
        struct level2Info *next;
};

struct level3Info {
        int type;
        int no;
        int size;
        int priority;
        struct level3Info *next;
};

struct objectInfo *start;
struct objectInfo *last;
struct level1Info *level1Start;
struct level1Info *level1Last;
struct level2Info *level2Start;
struct level2Info *level2Last;
struct level3Info *level3Start;
struct level3Info *level3Last;

void initialize(void)
{
        start = last = NULL;
        level1Start = level1Last = NULL;
        level2Start = level2Last = NULL;
        level3Start = level3Last = NULL;
        *level1Store = NULL;
        *level2Store = NULL;
        *level3Store = NULL;
        *priceStore = NULL;
        *video = NULL;
        *audio = NULL;
        *text = NULL;
        *vPriority = NULL;
        *aPriority = NULL;
        *tPriority = NULL;
        totalHigh = totalMed = totalLow = 0;
        videoDEL = videoLOS = audioDEL = audioLOS = textDEL = textLOS = 0;
        bestDEL = worstDEL = bestLOS = worstLOS = denomDEL = denomLOS = QoPDEL = QoPLOS =
0;
```

```
        prioritySum = 0;
        videoCount = audioCount = textCount = totalBudget = i = j = max = 0;
}

void level1Process(int totalPacket)
{
        int totalPrice = 0;

        totalPrice = totalPacket * level1Price;

        max = totalPrice;
        priceStore[i] = totalPrice;
        level1Store[i] = totalPacket;
        level2Store[i] = 0;
        level3Store[i] = 0;

        fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d", level1Packet,
level2Packet, level3Packet, totalPrice);
}

void secondStage(int totalPacket, int overBudget2, int underBudget2, int
tempLevel1Packet, int tempLevel3Packet)
{
        int totalPrice = 0, overBudget = 0, underBudget;
        int tempLevel2Packet = 0;

        do
        {
                totalPrice = (tempLevel1Packet*level1Price) +
(tempLevel2Packet*level2Price) + (tempLevel3Packet*level3Price);
                fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d",
tempLevel1Packet, tempLevel2Packet, tempLevel3Packet, totalPrice);

                if((totalPrice > max) && (totalPrice <= totalBudget))
                {
                        i = 0;
                        max = totalPrice;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }

                else if(totalPrice == max)
                {
                        i = i + 1;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }


                if(totalPrice >= totalBudget)
                {
                        overBudget2 = tempLevel1Packet;
                        tempLevel1Packet = (tempLevel1Packet + underBudget2) /2;

                        if(tempLevel1Packet >= totalPacket)
                                tempLevel1Packet = totalPacket - 1;

                        tempLevel2Packet = 0;
                        tempLevel3Packet = totalPacket - tempLevel1Packet -
tempLevel2Packet;
                }

                else if (totalPrice < totalBudget)
                {
                        tempLevel2Packet = tempLevel3Packet/2;
                        tempLevel3Packet = totalPacket - tempLevel1Packet -
tempLevel2Packet;
                        overBudget = 0;
                        underBudget = 0;

                        do
                        {
                                totalPrice = (tempLevel1Packet*level1Price) +
(tempLevel2Packet*level2Price) + (tempLevel3Packet*level3Price);
                                fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d",
tempLevel1Packet, tempLevel2Packet, tempLevel3Packet, totalPrice);

                                if((totalPrice > max) && (totalPrice <= totalBudget))
                                {
                                        i = 0;
                                        max = totalPrice;
                                        priceStore[i] = totalPrice;
                                        level1Store[i] = tempLevel1Packet;
                                        level2Store[i] = tempLevel2Packet;
                                        level3Store[i] = tempLevel3Packet;
                                }

                                else if(totalPrice == max)
                                {
                                        i = i + 1;
                                        priceStore[i] = totalPrice;
                                        level1Store[i] = tempLevel1Packet;
                                        level2Store[i] = tempLevel2Packet;
                                        level3Store[i] = tempLevel3Packet;
                                }

                                if(((tempLevel3Packet - overBudget) <= 1) &&
((tempLevel2Packet - underBudget) <= 1))
                                        break;


                                if(totalPrice < totalBudget)
                                {
                                        underBudget = tempLevel2Packet;
                                        tempLevel2Packet = tempLevel2Packet +
(abs(overBudget - tempLevel3Packet) /2);
                                        tempLevel3Packet = totalPacket - tempLevel1Packet -
tempLevel2Packet;
                                }

                                else if (totalPrice > totalBudget)
                                {
                                        overBudget = tempLevel3Packet;
                                        tempLevel3Packet = tempLevel3Packet +
(abs(tempLevel2Packet - underBudget) /2);
                                        tempLevel2Packet = totalPacket - tempLevel1Packet -
tempLevel3Packet;
                                }
                        } while (totalPrice != totalBudget);

                        underBudget2 = tempLevel1Packet;
                        tempLevel1Packet = (tempLevel1Packet + overBudget2) /2;

                        if(tempLevel1Packet >= totalPacket)
                                tempLevel1Packet = totalPacket - 1;

                        tempLevel2Packet = 0;
                        tempLevel3Packet = totalPacket - tempLevel1Packet -
tempLevel2Packet;
                }
        } while((overBudget2 - underBudget2) > 1);
}

void firstStageLevel2(int totalPacket)
{
        int totalPrice = 0, overBudget = 0, underBudget = 0;
        int tempLevel1Packet = 0, tempLevel2Packet = 0, tempLevel3Packet = 0;
        int underBudget2 = 0, overBudget2 = 0;

        totalPrice = totalPacket*level2Price;
        fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d", tempLevel1Packet,
totalPacket, tempLevel3Packet, totalPrice);

        max = totalPrice;
        priceStore[i] = totalPrice;
        level1Store[i] = tempLevel1Packet;
        level2Store[i] = totalPacket;
        level3Store[i] = tempLevel3Packet;

        tempLevel2Packet = totalPacket/2;
        tempLevel1Packet = totalPacket - tempLevel2Packet;
        overBudget = totalPacket;
        underBudget = 0;

        do
        {
                totalPrice = (tempLevel2Packet*level2Price) +
(tempLevel1Packet*level1Price);
                fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d",
tempLevel1Packet, tempLevel2Packet, tempLevel3Packet, totalPrice);

                if((totalPrice > max) && (totalPrice <= totalBudget))
                {
                        i = 0;
                        max = totalPrice;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }

                else if(totalPrice == max)
                {
                        i = i + 1;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }

                if(((tempLevel2Packet - overBudget) <= 1) && ((tempLevel1Packet -
underBudget) <= 1))
                        break;

                if(totalPrice < totalBudget)
                {
                        underBudget = tempLevel1Packet;
                        tempLevel1Packet = tempLevel1Packet + (abs(overBudget -
tempLevel2Packet) /2);

                        if(tempLevel1Packet >= totalPacket)
                                tempLevel1Packet = totalPacket - 1;

                        tempLevel2Packet = totalPacket - tempLevel1Packet;
                }

                else if (totalPrice > totalBudget)
                {
                        overBudget = tempLevel2Packet;
                        tempLevel2Packet = tempLevel2Packet + (abs(tempLevel1Packet -
underBudget) /2);

                        if(tempLevel2Packet >= totalPacket)
                                tempLevel2Packet = totalPacket - 1;

                        tempLevel1Packet = totalPacket - tempLevel2Packet;
                }
        } while (totalPrice != totalBudget);

        overBudget2 = totalPacket;
        underBudget2 = tempLevel1Packet;
        tempLevel1Packet = tempLevel1Packet + ((totalPacket - underBudget) / 2);

        if(tempLevel1Packet >= totalPacket)
                tempLevel1Packet = totalPacket - 1;

        tempLevel3Packet = totalPacket - tempLevel1Packet;

        secondStage(totalPacket, overBudget2, underBudget2, tempLevel1Packet,
tempLevel3Packet);
}

void firstStageLevel3 (int totalPacket)
{
        int totalPrice = 0, overBudget = 0, underBudget = 0;
        int tempLevel1Packet = 0, tempLevel2Packet = 0, tempLevel3Packet = 0;
        int underBudget2 = 0, overBudget2 = 0;

        totalPrice = totalPacket*level3Price;
        fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d", tempLevel1Packet,
tempLevel2Packet, totalPacket, totalPrice);

        max = totalPrice;
        priceStore[i] = totalPrice;
        level1Store[i] = tempLevel1Packet;
        level2Store[i] = tempLevel2Packet;
        level3Store[i] = totalPacket;

        if(totalPrice == totalBudget)
                return;

        tempLevel3Packet = totalPacket/2;
        tempLevel2Packet = totalPacket - tempLevel3Packet;
        overBudget = totalPacket;
        underBudget = 0;

        do
        {
                totalPrice = (tempLevel3Packet*level3Price) +
(tempLevel2Packet*level2Price);
                fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t $%d",
tempLevel1Packet, tempLevel2Packet, tempLevel3Packet, totalPrice);

                if((totalPrice > max) && (totalPrice <= totalBudget))
                {
                        i = 0;
                        max = totalPrice;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }

                else if(totalPrice == max)
                {
                        i = i + 1;
                        priceStore[i] = totalPrice;
                        level1Store[i] = tempLevel1Packet;
                        level2Store[i] = tempLevel2Packet;
                        level3Store[i] = tempLevel3Packet;
                }

                if(((tempLevel3Packet - overBudget) <= 1) && ((tempLevel2Packet -
underBudget) <= 1))
                        break;

                if(totalPrice < totalBudget)
                {
                        underBudget = tempLevel2Packet;

                        tempLevel2Packet = tempLevel2Packet + (abs(overBudget -
tempLevel3Packet) /2);

                        if(tempLevel2Packet >= totalPacket)
                                tempLevel2Packet = totalPacket - 1;

                        tempLevel3Packet = totalPacket - tempLevel2Packet;
                }

                else if (totalPrice > totalBudget)
                {
                        overBudget = tempLevel3Packet;

                        tempLevel3Packet = tempLevel3Packet + (abs(tempLevel2Packet -
underBudget) /2);

                        if(tempLevel3Packet >= totalPacket)
                                tempLevel3Packet = totalPacket - 1;

                        tempLevel2Packet = totalPacket - tempLevel3Packet;
                }
        } while (totalPrice != totalBudget);

        overBudget2 = totalPacket;
        underBudget2 = tempLevel1Packet;
        tempLevel3Packet = tempLevel3Packet + (underBudget / 2);

        if(tempLevel3Packet >= totalPacket)
                tempLevel3Packet = totalPacket - 1;

        tempLevel1Packet = totalPacket - tempLevel3Packet;

        secondStage(totalPacket, overBudget2, underBudget2, tempLevel1Packet,
tempLevel3Packet);
}

void scoreEvaluation(int totalPacket)
{
        double denom, lossWg, delayWg, avgDelay;
        double x, losLevel1, losLevel2, losLevel3, delLevel1, delLevel2, delLevel3;

        lossWg = 1;
        delayWg = 1;

        for(j = 0; j <= i; j++)
        {
                delLevel1 =
pow(((double)level1Store[j]/((double)totalPacket/(1+(((double)level1Delay/(double)level1D
elay))))), (3));
                delLevel2 =
pow(((double)level2Store[j]/((double)totalPacket/(1+(((double)level2Delay/(double)level1D
elay))))), (3));
                delLevel3 =
pow(((double)level3Store[j]/((double)totalPacket/(1+(((double)level3Delay/(double)level1D
elay))))), (3));

                losLevel1 = level1Store[j] * level1Loss;
                losLevel2 = level2Store[j] * level2Loss;
                losLevel3 = level3Store[j] * level3Loss;

                loss[j] = (losLevel1/ LOSWg) + (losLevel2/ LOSWg) + (losLevel3/ LOSWg);
                delay[j] = ((delLevel1/ DELWg) + (delLevel2/ DELWg) + (delLevel3/
DELWg))/3;

                score[j] = delay[j] + loss[j];
        }
}
```

```
void storeLevel1(struct level1Info *i, struct level1Info **level1Start, struct level1Info
**level1Last)
{
        struct level1Info *old, *p;

        p = *level1Start;

        if(!*level1Last)
        {
                i->next = NULL;
                *level1Last = i;
                *level1Start = i;
                return;
        }

        old = NULL;

        while(p)
        {

                if(p->priority >= i->priority)
                {
                        old = p;
                        p = p->next;
                }
                else
                {
                        if(old)
                        {
                                old->next = i;
                                i->next = p;
                                return;
                        }

                        i->next = p;
                        *level1Start = i;
                        return;
                }
        }
        (*level1Last)->next = i;
        i->next = NULL;
        *level1Last = i;
}

void storeLevel2(struct level2Info *i, struct level2Info **level2Start, struct level2Info
**level2Last)
{
        struct level2Info *old, *p;

        p = *level2Start;

        if(!*level2Last)
        {
                i->next = NULL;
                *level2Last = i;
                *level2Start = i;
                return;
        }

        old = NULL;

        while(p)
        {
                if(p->priority >= i->priority)
                {
                        old = p;
                        p = p->next;
                }
                else
                {
                        if(old)
                        {
                                old->next = i;
                                i->next = p;
                                return;
                        }

                        i->next = p;
                        *level2Start = i;
                        return;
                }
        }
        (*level2Last)->next = i;
        i->next = NULL;
        *level2Last = i;
}

void storeLevel3(struct level3Info *i, struct level3Info **level3Start, struct level3Info
**level3Last)
{
        struct level3Info *old, *p;

        p = *level3Start;

        if(!*level3Last)
        {
                i->next = NULL;
                *level3Last = i;
                *level3Start = i;
                return;
        }

        old = NULL;

        while(p)
        {
                if(p->priority >= i->priority)
                {
                        old = p;
                        p = p->next;
                }
                else
                {
                        if(old)
                        {
                                old->next = i;
                                i->next = p;
                                return;
                        }

                        i->next = p;
                        *level3Start = i;
                        return;
                }
        }
        (*level3Last)->next = i;
        i->next = NULL;
        *level3Last = i;
}


void store(struct objectInfo *i, struct objectInfo **start, struct objectInfo **last)
{
        struct objectInfo *old, *p;

        p = *start;

        if(!*last)
        {
                i->next = NULL;
                *last = i;
                *start = i;
                return;
        }

        old = NULL;

        while(p)
        {
                if(p->priority >= i->priority)
                {
                        old = p;
                        p = p->next;
                }
                else
                {
                        if(old)
                        {
                                old->next = i;
                                i->next = p;
                                return;
                        }

                        i->next = p;
                        *start = i;
                        return;
                }
        }
        (*last)->next = i;
        i->next = NULL;
        *last = i;
}

void view(void)

{
        struct objectInfo *info;

        info = start;

        while(info)
        {
                printf("\n %d", info->type);
                printf("\t%d", info->no);
                printf("\t%d", info->size);
                printf("\t%d", info->priority);

                info = info->next;
        }

}

void viewDist(void)
{
        struct level1Info *tempLevel1Info;
        struct level2Info *tempLevel2Info;
        struct level3Info *tempLevel3Info;

        tempLevel1Info = level1Start;
        tempLevel2Info = level2Start;
        tempLevel3Info = level3Start;

        fprintf(eventFile, "\n\nDiffServ distribution for the particular %d object(s)
scene:\n", videoCount+audioCount+textCount);
        fprintf(eventFile,
"============================================================\n");

        fprintf(eventFile, "DiffServ Level 1 Distribution:\n");
        while(tempLevel1Info)
        {
                if(tempLevel1Info->type == 1)
                        fprintf(eventFile, " V");
                else if(tempLevel1Info->type == 2)
                        fprintf(eventFile, " A");
                else if(tempLevel1Info->type == 3)
                        fprintf(eventFile, " T");

                fprintf(eventFile, "%d", tempLevel1Info->no);


                if(tempLevel1Info->priority == 3)
                        fprintf(eventFile, "(high)=");
                else if(tempLevel1Info->priority == 2)
                        fprintf(eventFile, "(med)=");
                else if(tempLevel1Info->priority == 1)
                        fprintf(eventFile, "(low)=");

                fprintf(eventFile, " %d KB    ", tempLevel1Info->size);

                tempLevel1Info = tempLevel1Info->next;
        }

        fprintf(eventFile, "\n\nDiffServ Level 2 Distribution:\n");
        while(tempLevel2Info)
        {
                if(tempLevel2Info->type == 1)
                        fprintf(eventFile, " V");
                else if(tempLevel2Info->type == 2)
                        fprintf(eventFile, " A");
                else if(tempLevel2Info->type == 3)
                        fprintf(eventFile, " T");

                fprintf(eventFile, "%d", tempLevel2Info->no);

                if(tempLevel2Info->priority == 3)
                        fprintf(eventFile, "(high)=");
                else if(tempLevel2Info->priority == 2)
                        fprintf(eventFile, "(med)=");
                else if(tempLevel2Info->priority == 1)
                        fprintf(eventFile, "(low)=");


                fprintf(eventFile, " %d KB    ", tempLevel2Info->size);

                tempLevel2Info = tempLevel2Info->next;
        }

        fprintf(eventFile, "\n\nDiffServ Level 3 Distribution:\n");
        while(tempLevel3Info)
        {
                if(tempLevel3Info->type == 1)
                        fprintf(eventFile, " V");
                else if(tempLevel3Info->type == 2)
                        fprintf(eventFile, " A");
                else if(tempLevel3Info->type == 3)
                        fprintf(eventFile, " T");

                fprintf(eventFile, "%d", tempLevel3Info->no);

                if(tempLevel3Info->priority == 3)
                        fprintf(eventFile, "(high)=");
                else if(tempLevel3Info->priority == 2)
                        fprintf(eventFile, "(med)=");
                else if(tempLevel3Info->priority == 1)
                        fprintf(eventFile, "(low)=");

                fprintf(eventFile, " %d KB    ", tempLevel3Info->size);

                tempLevel3Info = tempLevel3Info->next;
        }
        fprintf(eventFile, "\n");
}
```

```
void distribution(void)
{
        int size = 0;
        int level1Size, level2Size, level3Size;
        float temp;

        struct objectInfo *info;
        struct level1Info *tempLevel1Info;
        struct level2Info *tempLevel2Info;
        struct level3Info *tempLevel3Info;

        level1Size = bestLevel1;
        level2Size = bestLevel2;
        level3Size = bestLevel3;
        info = start;

        while(level1Size > 0)
        {
                if(info->priority == 3)
                        size = totalHigh;
                else if (info->priority == 2)
                        size = totalMed;
                else if (info->priority == 1)
                        size = totalLow;

                if(info->size > 0)
                {
                        tempLevel1Info = (struct level1Info *)malloc(sizeof(struct
level1Info));

                        tempLevel1Info->type = info->type;
                        tempLevel1Info->no = info->no;
                        tempLevel1Info->priority = info->priority;

                        temp = info->size;

                        if((temp/ size * level1Size) < temp)
                                tempLevel1Info->size = ceil(temp/ size * level1Size);
                        else
                                tempLevel1Info->size = temp;

                        level1Size = level1Size - tempLevel1Info->size;
                        info->size = info->size - tempLevel1Info->size;

                        storeLevel1(tempLevel1Info, &level1Start, &level1Last);

                        if(info->priority == 3)
                                totalHigh = totalHigh - tempLevel1Info->size;
                        else if(info->priority == 2)
                                totalMed = totalMed - tempLevel1Info->size;
                        else if(info->priority == 1)
                                totalLow = totalLow - tempLevel1Info->size;


                }
                info = info->next;

                if(!info)
                        break;
        }

        info = start;
        while(level2Size > 0)
        {
                if(info->priority == 3)
                        size = totalHigh;
                else if (info->priority == 2)
                        size = totalMed;
                else if (info->priority == 1)
                        size = totalLow;
                if(info->size > 0)
                {
                        tempLevel2Info = (struct level2Info *)malloc(sizeof(struct
level2Info));

                        tempLevel2Info->type = info->type;
                        tempLevel2Info->no = info->no;
                        tempLevel2Info->priority = info->priority;

                        temp = info->size;

                        if((temp/ size * level2Size) < temp)
                                tempLevel2Info->size = ceil(temp/ size * level2Size);
                        else
                                tempLevel2Info->size = temp;

                        level2Size = level2Size - tempLevel2Info->size;
                        info->size = info->size - tempLevel2Info->size;

                        storeLevel2(tempLevel2Info, &level2Start, &level2Last);
                        if(info->priority == 3)
                                totalHigh = totalHigh - tempLevel2Info->size;
                        else if(info->priority == 2)
                                totalMed = totalMed - tempLevel2Info->size;
                        else if(info->priority == 1)
                                totalLow = totalLow - tempLevel2Info->size;
                }

                info = info->next;

                if(!info)
                        break;
        }

        info = start;
        while(info > 0)
        {
                if(info->size > 0)
                {
                        tempLevel3Info = (struct level3Info *)malloc(sizeof(struct
level3Info));

                        tempLevel3Info->type = info->type;
                        tempLevel3Info->no = info->no;
                        tempLevel3Info->priority = info->priority;
                        tempLevel3Info->size = info->size;

                        storeLevel3(tempLevel3Info, &level3Start, &level3Last);
                }
                info = info->next;
        }
}

void main(void)
{
        int totalPacket = 0, tempBudget, lowestPrice;
        int dataCount = 0;
        double packetCount = 0, objectCount = 0;
        float xx;
        struct objectInfo *info;
        time_t beginTime;
        time_t endTime;

        start = last = NULL;

        printf("\n This program will formulate the optimum flows");
        printf("\n distribution for 3-level DiffServ Networks");
        printf("\n ------------------------------------------");

        printf("\n\n Reading DATA.TXT ...");
        printf("\n Formulating  ...");
        beginTime = time(NULL);

        if((dataFile = fopen("DATA.TXT", "r")) == NULL)
        {
                printf("\n Can't read file DATA.TXT. Possible cause: File Not Exist.\n");
                exit(0);
        }

        if((resultFile = fopen("RESULT.TXT", "a")) == NULL)
        {
                printf("\n Can't create file RESULT.TXT. Possible cause: File Not
Exist.\n");
                exit(0);
        }

        if((eventFile = fopen("EVENT.TXT", "w")) == NULL)
        {
                printf("\n Can't create/ write to file EVENT.TXT. Possible cause:
Permission Denied.\n");
                exit(0);
        }

        while(!feof(dataFile))
        {
                initialize();

                fscanf(dataFile, "%d %d %d", &videoCount, &audioCount, &textCount);
                for(j=0; j<videoCount;j++)
                {
                        fscanf(dataFile, "%d %d", &video[j], &vPriority[j]);
                        videoDEL = videoDEL + (vPriority[j] * 2);
                        videoLOS = videoLOS + (vPriority[j] * 1);
                        prioritySum = prioritySum + vPriority[j];

                        if(vPriority[j] == 3)
                                totalHigh = totalHigh + video[j];
                        else if(vPriority[j] == 2)
                                totalMed = totalMed + video[j];
                        else
                                totalLow = totalLow + video[j];

                        info = (struct objectInfo *)malloc(sizeof(struct objectInfo));

                        info->type = 1;
                        info->no = j+1;
                        info->size = video[j];
                        info->priority = vPriority[j];

                        store(info, &start, &last);
                }



                for(j=0; j<audioCount;j++)
                {
                        fscanf(dataFile, "%d %d", &audio[j], &aPriority[j]);
                        audioDEL = audioDEL + (aPriority[j] * 2);
                        audioLOS = audioLOS + (aPriority[j] * 3);
                        prioritySum = prioritySum + aPriority[j];


                        if(aPriority[j] == 3)
                                totalHigh = totalHigh + audio[j];
                        else if(aPriority[j] == 2)
                                totalMed = totalMed + audio[j];
                        else
                                totalLow = totalLow + audio[j];


                        info = (struct objectInfo *)malloc(sizeof(struct objectInfo));
                        info->type = 2;
                        info->no = j+1;
                        info->size = audio[j];
                        info->priority = aPriority[j];

                        store(info, &start, &last);
                }

                for(j=0; j<textCount;j++)
                {
                        fscanf(dataFile, "%d %d", &text[j], &tPriority[j]);
                        textDEL = textDEL + (tPriority[j] * 1);
                        textLOS = textLOS + (tPriority[j] * 3);
                        prioritySum = prioritySum + tPriority[j];


                        if(tPriority[j] == 3)
                                totalHigh = totalHigh + text[j];
                        else if(tPriority[j] == 2)
                                totalMed = totalMed + text[j];
                        else
                                totalLow = totalLow + text[j];


                        info = (struct objectInfo *)malloc(sizeof(struct objectInfo));

                        info->type = 3;
                        info->no = j+1;
                        info->size = text[j];
                        info->priority = tPriority[j];

                        store(info, &start, &last);
                }

                bestDEL = ((double)videoCount * 6 + (double)audioCount * 6 +
(double)textCount * 1) / ((double)videoCount * 3 + (double)audioCount * 3 +
(double)textCount * 1);
                worstDEL = ((double)videoCount * 2 + (double)audioCount * 2 +
(double)textCount * 3) / ((double)videoCount * 1 + (double)audioCount * 1 +
(double)textCount * 3);

                bestLOS = ((double)videoCount * 1 + (double)audioCount * 9 +
(double)textCount * 9) / ((double)videoCount * 1 + (double)audioCount * 3 +
(double)textCount * 3);
                worstLOS = ((double)videoCount * 3 + (double)audioCount * 3 +
(double)audioCount * 3) / ((double)videoCount * 3 + (double)audioCount * 1 +
(double)textCount * 1);

                denomDEL = (bestDEL/ worstDEL - 1) / 88;
                denomLOS = (bestLOS/ worstLOS - 1) / 25;

                QoPDEL = ((videoDEL + audioDEL + textDEL) / prioritySum)/ worstDEL;
                QoPLOS = ((videoLOS + audioLOS + textLOS) / prioritySum)/ worstLOS;


                DELWg = ((QoPDEL - 1) / denomDEL) + 1;
                LOSWg = ((QoPLOS - 1) / denomLOS) + 1;

                fscanf(dataFile, "%d %d %d %f %f %f %f %f %f %d %d", &level1Price,
&level2Price, &level3Price, &level1Delay, &level2Delay, &level3Delay, &level1Loss,
&level2Loss, &level3Loss, &totalPacket, &totalBudget);

                fprintf(eventFile, "\n\nThe binary search algorithm for the follwing
criteria:");
                fprintf(eventFile, "\n\nPrice per kilobytes: Level 1 = $%d, Level 2 = $%d,
Level 3 = $%d", level1Price, level2Price, level3Price);
                fprintf(eventFile, "\nTotal flows sizes: %d KB, total budget = $%d",
totalPacket, totalBudget);
                fprintf(eventFile,
"\n=============================================================");
                fprintf(eventFile, "\nLevel 1\t\tLevel 2\t\tLevel 3\t\tPrice");
```

```
		fprintf(eventFile, "\n-----------------------------------------------------
-----");

		if((totalPacket*level1Price) <= totalBudget)
			level1Process(totalPacket);

		else if((totalPacket*level2Price) <= totalBudget)
			firstStageLevel2(totalPacket);

		else if((totalPacket*level3Price) <= totalBudget)
			firstStageLevel3(totalPacket);

		else
		{
			printf("\n Your budget too low");
			//fprintf(eventFile, "\nYour budget too low. Try again with
different budget");
			exit(0);
		}

		fprintf(eventFile, "\n\nThe best combination candidate(s) with score
evaluation:");
		fprintf(eventFile,
"\n=========================================================================================
===");
		fprintf(eventFile, "\nLevel 1\t\tLevel 2\t\tLevel 3\t\t Delay\t\t
Loss\t\tScore");
		fprintf(eventFile, "\n-----------------------------------------------------
-----------------------------------");

		scoreEvaluation(totalPacket);

		for(j = 0; j <= i; j++)
		{
			fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t%lf\t%lf\t%lf",
level1Store[j], level2Store[j], level3Store[j], delay[j], loss[j], score[j]);

			if((minScore > score[j]) || (minScore == 0))
			{
				minScore = score[j];
				bestLevel1 = level1Store[j];
				bestLevel2 = level2Store[j];
				bestLevel3 = level3Store[j];
				bestPrice = priceStore[j];
			}
		}

		fprintf(eventFile, "\n\nThe BEST combination suggested:");
		fprintf(eventFile,
"\n============================================================");
		fprintf(eventFile, "\nLevel 1\t\tLevel 2\t\tLevel 3\t\tScore");
		fprintf(eventFile, "\n-----------------------------------------------------
-----");
		fprintf(eventFile, "\n %d KB\t\t %d KB\t\t %d KB\t\t%lf", bestLevel1,
bestLevel2, bestLevel3, minScore);

		fprintf(resultFile, "%d, %d, %d, %d, %d, %d\n", totalBudget, totalPacket,
bestLevel1, bestLevel2, bestLevel3, bestPrice);

		distribution();
		viewDist();
		dataCount++;
		packetCount = packetCount + totalPacket;
		objectCount = objectCount + videoCount + audioCount + textCount;
	}

	endTime = time(NULL);
	fclose(resultFile);
	printf("\n Writing RESULT.TXT ... Completed.");
	fclose(eventFile);
	printf("\n Writing EVENT.TXT ... Completed.");
```

```
        printf("\n\n Average total flows each scene = %.01f KB", packetCount/dataCount);
        printf("\n Average total objects each scene = %.01f object(s)",
objectCount/dataCount);
        printf("\n Time elapsed for simulating %d scenes = %u milliseconds\n", dataCount,
clock());

        fclose(dataFile);
}
```

APPENDIX B

CONTENT SAMPLE OF 'DATA.TXT' FILE

The program will read the data from the DATA.TXT file to be processed. The content format is:

```
{total video objects} {total audio objects} {total text object}

{video object 1 flow sizes} {priority}
...
{video object n flow sizes} {priority}

{audio object flow sizes} {priority}

{text object flow sizes} {priority}

{level 1 price} {level 2 price} {level 3 price} {average level 1 delay}
      {average level 2 delay} {average level 3 delay} {average level 1
      loss} {average level 2 loss} {average level 3 loss} {total flows
      sizes} {budget}
```

Sample of two continuous sets of data:

```
3 1 1
20 2
15 1
20 3
15 1
30 3
4 2 1 100 110 400 0.01 0.02 0.07 100 275
3 1 1
15 3
20 2
30 1
20 3
15 1
4 2 1 100 200 400 0.01 0.02 0.04 100 250
```

APPENDIX C

CONTENT SAMPLE OF 'RESULT.TXT' FILE

After the data has been processed, the result file is created. The content format of the 'RESULT.TXT' file is:

```
{budget}, {total flows sizes}, {flows sizes suggested to be transmitted
      through level 1}, {flows sizes suggested to be transmitted
      through level 2}, {flows sizes suggested to be transmitted
      through level 3}, {total price}
```

Sample result from data defined in Appendix B:

```
275, 100, 53, 16, 31, 275
250, 100, 37, 39, 24, 250
```

APPENDIX D

CONTENT SAMPLE OF 'EVENT.TXT' FILE

'EVENT.TXT' file is created to examine the important steps of the program from the beginning to result production of the algorithm described in Chapter IV. The content and format of the event file are self-explanatory. Below is the sample of event file created from processing data defined in Appendix B:

```
The binary search algorithm for the following criteria:

Price per kilobytes: Level 1 = $4, Level 2 = $2, Level 3 = $1
Total flows sizes: 100 KB, total budget = $275
=================================================================
Level 1             Level 2             Level 3             Price
---------------------------------------------------------------
 0 KB                100 KB              0 KB                $200
 50 KB               50 KB               0 KB                $300
 25 KB               75 KB               0 KB                $250
 37 KB               63 KB               0 KB                $274
 43 KB               57 KB               0 KB                $286
 40 KB               60 KB               0 KB                $280
 39 KB               61 KB               0 KB                $278
 38 KB               62 KB               0 KB                $276
 69 KB               0 KB                31 KB               $307
 53 KB               0 KB                47 KB               $259
 53 KB               23 KB               24 KB               $282
 53 KB               12 KB               35 KB               $271
 53 KB               17 KB               30 KB               $276
 53 KB               15 KB               32 KB               $274
 53 KB               16 KB               31 KB               $275
 61 KB               0 KB                39 KB               $283
 57 KB               0 KB                43 KB               $271
 57 KB               21 KB               22 KB               $292
 57 KB               11 KB               32 KB               $282
 57 KB               6 KB                37 KB               $277
 57 KB               3 KB                40 KB               $274
 57 KB               4 KB                39 KB               $275
 59 KB               0 KB                41 KB               $277
 58 KB               0 KB                42 KB               $274
 58 KB               21 KB               21 KB               $295
 58 KB               11 KB               31 KB               $285
 58 KB               6 KB                36 KB               $280
 58 KB               3 KB                39 KB               $277
 58 KB              2 KB               40 KB              $276
 58 KB              1 KB               41 KB              $275

The best combination candidate(s) with score evaluation:
=================================================================
Level 1      Level 2     Level 3      Delay        Loss        Score
-----------------------------------------------------------------
 53 KB       16 KB        31 KB      0.050029    0.251667    0.301695
 57 KB       4 KB         39 KB      0.089869    0.281667    0.371535
 58 KB       1 KB         41 KB      0.102788    0.289167    0.391955

The BEST combination suggested:
====================================================
Level 1            Level 2            Level 3            Score
----------------------------------------------------
 53 KB              16 KB              31 KB              0.301695

DiffServ distribution for the particular 5 object(s) scene:
=======================================================
DiffServ Level 1 Distribution:
 V3(high)= 20 KB    T1(high)= 30 KB    V1(med)= 3 KB

DiffServ Level 2 Distribution:
 V1(med)= 16 KB

DiffServ Level 3 Distribution:
 V1(med)= 1 KB    V2(low)= 15 KB    A1(low)= 15 KB


The binary search algorithm for the following criteria:

Price per kilobytes: Level 1 = $4, Level 2 = $2, Level 3 = $1
Total flows sizes: 100 KB, total budget = $250
==========================================================
Level 1            Level 2            Level 3            Price
----------------------------------------------------
 0 KB               100 KB             0 KB               $200
 50 KB              50 KB              0 KB               $300
 25 KB              75 KB              0 KB               $250
 75 KB              0 KB               25 KB              $325
 50 KB              0 KB               50 KB              $250
 37 KB              0 KB               63 KB              $211
 37 KB              31 KB              32 KB              $242
 37 KB              47 KB              16 KB              $258
 37 KB              39 KB              24 KB              $250
 43 KB              0 KB               57 KB              $229
 43 KB              28 KB              29 KB              $257
 43 KB              14 KB              43 KB              $243
 43 KB              21 KB              36 KB              $250
 46 KB              0 KB               54 KB              $238
 46 KB              27 KB              27 KB              $265
 46 KB              14 KB              40 KB              $252
 46 KB              7 KB               47 KB              $245
 46 KB              10 KB              44 KB              $248
 46 KB              12 KB              42 KB              $250
 48 KB              0 KB               52 KB              $244
 48 KB              26 KB              26 KB              $270
```

```
48 KB              13 KB              39 KB              $257
48 KB              7 KB               45 KB              $251
48 KB              4 KB               48 KB              $248
48 KB              5 KB               47 KB              $249
48 KB              6 KB               46 KB              $250
49 KB              0 KB               51 KB              $247
49 KB              25 KB              26 KB              $272
49 KB              13 KB              38 KB              $260
49 KB              7 KB               44 KB              $254
49 KB              4 KB               47 KB              $251
49 KB              2 KB               49 KB              $249
49 KB              3 KB               48 KB              $250

The best combination candidate(s) with score evaluation:
======================================================================
Level 1      Level 2      Level 3      Delay        Loss         Score
----------------------------------------------------------------------
 25 KB       75 KB        0 KB         0.046247     0.145833     0.192081
 50 KB       0 KB         50 KB        0.066767     0.208333     0.275100
 37 KB       39 KB        24 KB        0.014999     0.175833     0.190833
 43 KB       21 KB        36 KB        0.026980     0.190833     0.217814
 46 KB       12 KB        42 KB        0.040507     0.198333     0.238841
 48 KB       6 KB         46 KB        0.052440     0.203333     0.255773
 49 KB       3 KB         48 KB        0.059301     0.205833     0.265134

The BEST combination suggested:
========================================================
Level 1            Level 2            Level 3            Score
--------------------------------------------------------
 37 KB              39 KB              24 KB              0.190833

DiffServ distribution for the particular 5 object(s) scene:
=========================================================
DiffServ Level 1 Distribution:
 V1(high)= 15 KB    A1(high)= 20 KB    V2(med)= 2 KB

DiffServ Level 2 Distribution:
 V2(med)= 18 KB    V3(low)= 14 KB    T1(low)= 4 KB

DiffServ Level 3 Distribution:
 V3(low)= 16 KB    T1(low)= 11 KB
```

APPENDIX E

PAPER SUBMISSION

Based on this thesis, a paper has been accepted for presentation at the IEEE International Conference on Networking and will be published in the proceedings. The full reference and information about the conference is described in the publication page.

Below is the full submitted paper:

USER-ORIENTED OBJECT-BASED MULTIMEDIA COMMUNICATIONS OVER DIFFSERV NETWORKS

KARYANTA PURNE AND J. P. THOMAS

Department of Computer Science
Oklahoma State University
700 N. Greenwood Ave., Tulsa, OK 74106, USA
Email: jpt@okstate.edu

MPEG-4 permits specification and composition of media objects in an audio-visual scene and still picture standards such as JPEG-2000 permits specification of regions of interest. We propose a two-stage mapping mechanism to assign user-oriented prioritized audio-visual objects to DiffServ levels such that 'Quality of Service' (QoS) is maximized within given budget constraints. Experimental results confirm the validity of this approach.

1 Introduction

Internet multimedia applications have very diverse requirements and the current best-effort model is inadequate for multimedia. Network services to support QoS are necessary to overcome the current best-effort infrastructure limited capabilities. The idea is to let the network provide a different level of assurance in terms of network QoS parameters within its resource capacity.

The Internet Engineering Task Force (IETF) has proposed two models, which are Integrated Services (IntServ) [11] with the Resource Reservation Protocol (RSVP) [3] and the Differentiated Services (DiffServ) [2]. The Integrated Services (IntServ) focuses on individual packet [11]. Each packet can request a specific level of service from the router; the router will grant or reject the requests based on availability and capacity of resources. IntServ has major problems in terms of scalability and manageability [10]. In the DiffServ model, resources are allocated differently for various aggregated traffic flows[1] based on a set of bits. The basic idea is to support a set of traffic class, e.g.: a premium service (PS), which support low loss and delay/ jitter and an assured service (AS), which provides better than best-effort but without guarantee [9]. The DiffServ model provides a simpler approach, which handles well the scalability and manageability problems for the core routers.

In this paper, the concept of 'media objects' as specified in MPEG-4 or JPEG-2000 'regions of interest' (ROI) is used to optimize the users' experiences of multimedia transmitted over DiffServ networks. ROI coding [7] enables a non-uniform distribution of the image quality between a selected region (the ROI) and

[1] A flow is stream of packets with common source address, destination address and port number.

the rest of the image (background). An ROI is a region of the image that is expected to have a better quality than the rest at any decoding bit-rate.

An MPEG-4 audio-visual scene is composed of several media objects, organized in a hierarchical fashion. At the leaves of the hierarchy, we find primitive objects, such as still images, video objects and audio objects. MPEG-4 also provides content-based access to individual objects in a scene. We assume that the MPEG-4 objects can be prioritized by the user as defined by the MPEG-4 standards [6]. However, although the MPEG-4 standards specify interaction with the objects, the MPEG-4 products currently available do not permit user prioritization of objects. We therefore envisage this research as applying to a future MPEG-4 player.

2 Objectives of the Study

DiffServ is not envisioned as a free service. The user has to budget for the video transmission and in this research the budget is specified per size of data to be transmitted. The budget is a varying function of time. Each DiffServ level provides a different QoS and each level will therefore carry a different cost. In our 3-level DiffServ model, level 1 which is the most expensive provides the best QoS. Level 3 provides the minimum QoS and is therefore the cheapest. Constrains to be considered when assigning each flow to the appropriate level of DiffServ network include available budget, price of each DiffServ level, flow delay and loss provided by each DiffServ level.

It is unrealistic to ask the average non-technical user to specify the importance of flows and expect him/ her to supply the optimum number of flows to be transmitted in each DiffServ level within the available budget. However, the user can be expected to provide some form of an indicator of the relative importance of the different objects presented in a multimedia scene. For example, in a news scene, from a user's perspective the audio object and the newscaster object will have high priority whereas the background object will have a lower priority. Ideally, all the objects will be transmitted at the highest QoS. However, this is not feasible at all times unless the user has a large budget. Hence, given the user-defined priority of objects in a scene, the average flow delay and loss for each level of DiffServ and the price of each level of DiffServ, a mechanism is needed to optimally and efficiently map and distribute all the incoming flows to different levels in the DiffServ network with respect to total available budget constraints while achieving maximum QoS.

The delay and loss model affects the mapping to DiffServ levels. In an ideal world, delay and loss should be modeled by the impact on the user. However, the literature does not report on the influence of delay and loss from a user's perspective. In this work, we use a polynomial and a linear model for delay; and a linear model for loss.

3 Mapping Architecture

To assign user's prioritized objects to appropriate DiffServ levels involves two mapping phases. The first mapping phase assigns weights to QoS parameters based on users' preferences. In this phase, flows associated with an object are labeled with a relative weighting for each QoS parameter. The weighting is an indication of the importance of the indicated QoS parameter. For example, delay is more critical for a

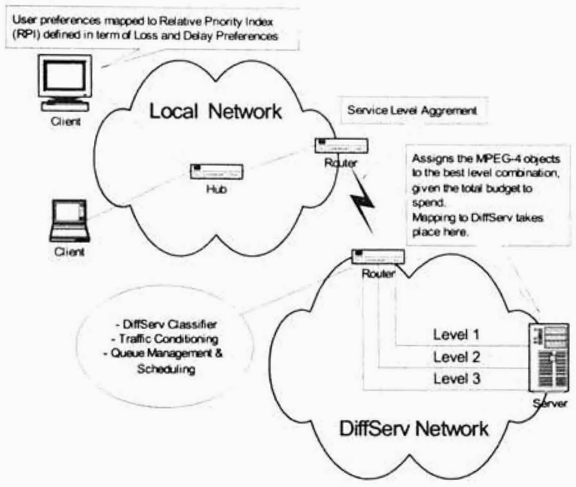

User preferences mapped to Relative Priority Index (RPI) defined in term of Loss and Delay Preferences
Client
Local Network
Service Level Aggrement
Router
Hub
Assigns the MPEG-4 objects to the best level combination, given the total budget to spend.
Mapping to DiffServ takes place here.
Client
Router
- DiffServ Classifier
- Traffic Conditioning
- Queue Management & Scheduling
Level 1
Level 2
Level 3
Server
DiffServ Network

Figure 1. Overall QoS Mapping General Framework in DiffServ Network

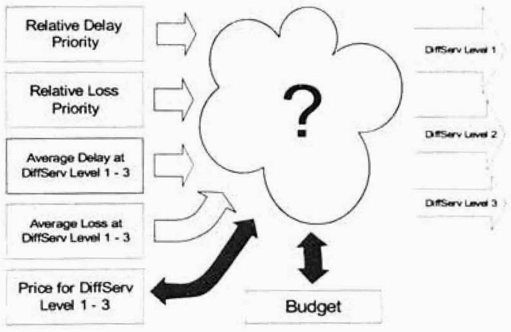

Relative Delay Priority
Relative Loss Priority
Average Delay at DiffServ Level 1 - 3
Average Loss at DiffServ Level 1 - 3
Price for DiffServ Level 1 - 3
?
Budget
DiffServ Level 1
DiffServ Level 2
DiffServ Level 3

Figure 2. Mapping Mechanism Illustration

flow with a relative delay weighting of 10 than for one with a relative delay weighting of 5. In Figure 1, the user specifies his/ her objects preferences (or relative importance of objects) at the client machine. These preferences are mapped to weighted QoS parameters (delay and loss). At the server, the second mapping phase assigns labeled flows based on their QoS weightings to the appropriate DiffServ level given budgetary constraints (figure 2).

3.1 Mapping Model

Shin, et. al. [10] have proposed the QoS mapping framework for packet video in DiffServ network. They propose an adaptive forwarding mechanism based on the Weighted Fair Queuing Algorithm. Very little work has been reported on the impact of varying QoS on user's perception of multimedia. Apteker, et. al. [1] showed that the relationship between frame loss and user satisfaction with multimedia is not a linear relationship. Ghinea and Thomas extended this work by examining the effect of varying QoS on user's perception of multimedia [4]. Moreover, Ghinea and Thomas have proposed a mechanism for approximating a users' perception of multimedia [5]. They called the term 'Quality of Perception' (QoP) and have proposed a mapping from user QoP to network QoS. In their approach, the authors define the user's experience of a multimedia as dependent on the primacy of video, audio and text as the carriers of information. Table 1 shows the relative importance of QoS parameters for each media [12]. For example, the table shows that bit error rate is of low importance to video whereas delay is of medium importance, indicating that bit error rate are more tolerable than delay in video transmission.

Table 1. Conversion matrix linking QoP to QoS

QoP To QoS Mapping		**QoP**		
		Video	**Audio**	**Text**
QoS	Bit error rate	Low	Low	Low
	Delay	Medium	Medium	Low
	Jitter	Medium	Low	Medium
	Segment Loss	Low	High	High
	Segment Order	High	Medium	Medium

In this work, we consider using only delay (DEL) and loss (LOSS) priority index of QoS. The mathematical representation of QoP to QoS mapping is defined as follows [5]:

$$QoP_{IT} \cong \frac{1}{V+A+T}[(V\alpha_{V,DEL} + A\alpha_{A,DEL} + T\alpha_{T,DEL}) * DEL + (V\alpha_{V,LOSS} + A\alpha_{A,LOSS} + T\alpha_{T,LOSS}) * LOSS]$$

where $\boldsymbol{QoP_{IT}}$ is Quality of Perception of Information Transfer; $\boldsymbol{V}$, $\boldsymbol{A}$ and $\boldsymbol{T}$ are relative importance of the video, audio and textual components as conveyors of information; ***DEL*** and ***LOSS*** are actual run-time values of the considered network parameters; $\alpha_{V,DEL}$ ... $\alpha_{V,LOSS}$ are relative priority values of QoP to QoS mapping and they correspond to the elements of table 1 (for instance, $\alpha_{V,DEL}$ = medium while $\alpha_{V,LOSS}$ = high).

However, since we deal with the multiple objects in a MPEG-4 scene, each individual object contributes to the overall QoP of the video scene. Therefore for a scene composed of multiple video objects:

$$QoP_{IT} \cong \frac{1}{V_1+V_2+\cdots+V_N+A+T}[(V_1\alpha_{V,DEL}+V_2\alpha_{V,DEL}+\cdots+V_N\alpha_{V,DEL}+A\alpha_{A,DEL}+T\alpha_{T,DEL})*DEL$$
$$+(V_1\alpha_{V,LOSS}+V_2\alpha_{V,LOSS}+\cdots+V_N\alpha_{V,LOSS}+A\alpha_{A,LOSS}+T\alpha_{T,LOSS})*LOSS] \quad (1)$$

Consider the following example: there are 5 objects V_1, V_2, V_3, A, and T in a MPEG-4 scene (3 video objects, 1 audio object and 1 text object). User prioritizes the objects of this particular MPEG-4 scene as follows:

Table 2. MPEG-4 Objects Priority Example

Objects	V_1	V_2	V_3	A	T
Priority	High	Medium	Low	High	Low

$$QoP_{IT} \cong (\frac{6}{10}V_1+\frac{4}{10}V_2+\frac{2}{10}V_3+\frac{6}{10}A+\frac{1}{10}T)*DEL+(\frac{3}{10}V_1+\frac{2}{10}V_2+\frac{1}{10}V_3+\frac{9}{10}A+\frac{3}{10}T)*LOSS$$

The weightings associated with object indicate the relative loss priority index (RLI) and the relative delay priority index (RDI) of each object from a user's perspective. For example, the delay weighting of object V_1 (6/10) is greater than for object V_2 (4/10).

Given user's preference for individual objects in a scene, the relative loss priority and delay priority values can thus be derived from the formula above. The next step is to find an efficient way to map each flow to different DiffServ levels.

The objective of mapping each flow to a DiffServ level while satisfying the budgetary constraints is to maximize QoS. Maximum quality is obtained when loss and delay are minimized. This can be represented as:

$$\max QoS = \min(f(\sum_{i=1}^{N} RLI_q \bullet l_{i(q)}, \sum_{i=1}^{N} RDI_q \bullet d_{i(q)})) \quad (2)$$

where $\boldsymbol{RLI_q}$ and $\boldsymbol{RDI_q}$ denote relative loss and delay priorities for packet $\boldsymbol{q}$; $\boldsymbol{l_{i(q)}}$ and $\boldsymbol{d_{i(q)}}$ denote loss and delay at the DiffServ level $\boldsymbol{i}$ for packet $\boldsymbol{q}$.

Values of $l_{i(q)}$ and $d_{i(q)}$ are dependent upon the network. The price constraint:

$$\sum_{i=1}^{N} P_{i(q)} \leq P$$

where $P_{i(q)}$ denotes the price of DiffServ level i for flow size q; P denotes the total price or budget of the user.

It is expected that the prices of the DiffServ level will not remain constant for the duration of the transmission. For instance, the prices of all level of DiffServ will be higher at peak times than off-peak times.

The audio-visual scene over DiffServ environment is therefore highly dynamic, due to the huge number of flows being transmitted and the varying price over time. Moreover, as scenes change rapidly, the mapping cannot be considered as a one-off computation; instead the mapping is continuous process for the duration of transmission. Therefore rather than finding an optimum, but computationally expensive solution to this non-linear optimization problem, this work focuses on finding a fast heuristic approach that provides a solution that is close to the optimum.

3.2 Representing Loss and Delay

QoS parameters should be modeled from a user's perspective. The representation of QoS parameters as linear, linear-exponential, and other functions is reported in [8].

We model loss as a linear function. Loss is a summation function on the loss incurred at the different DiffServ levels (figure 3). The assumption here is that the impact of loss on the user is proportional. The score is a measure of the total loss. So, we define the loss model as follows:

$$Ls = \sum_{i=1}^{N} \frac{1}{RLI} F_i \cdot l_i$$

where Ls is the score function for loss model; i is DiffServ level i; N is the maximum level in DiffServ, including all sublevels; F_i are flows sizes that will be transmitted through DiffServ level i in kilobytes; l_i is the average loss in DiffServ level i; RLI is the relative loss priority index.

The RLI is the weighting system for this loss model. It is calculated based on the table 1 and the relative importance of the objects given by the end-user. The smaller the RLI value, the less important the loss, and vice versa.

Delay is modeled as a polynomial function. The impact of a small number of delayed flows on the user is negligible as the human eye will not perceive the delay. However, the effect of a large number of flow delays will be discernible to the user resulting in an unsatisfactory multimedia experience to the user. We model the delay function as follows:

$$Ds = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{RDI} \left(\frac{F_i}{\sum_{i=1}^{N} F_i \Big/ (1 + \frac{d_i}{d_b})} \right)^{M}$$

where ***Ds*** is the score function for delay model; ***i*** is DiffServ level ***i***; ***N*** is maximum level of DiffServ, including all sublevels; ***M*** is the number of main level of DiffServ; ***F_i*** are total flows sizes that will be transmitted through DiffServ level ***i*** in kilobytes; ***d_i*** is the average delay in DiffServ level ***i***; ***d_b*** is the average delay in DiffServ level 1; ***RDI*** is the relative delay priority index.

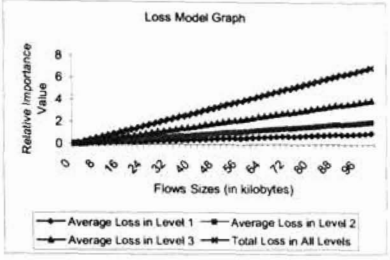


Figure 3. Representation of Loss

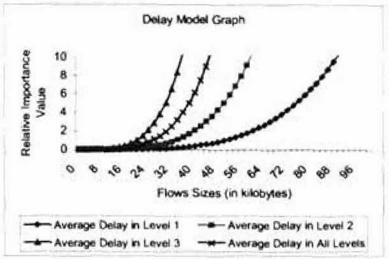


Figure 4. Representation of Delay

Using the number of levels in DiffServ as the exponent gives the same growth rate at each level of DiffServ with different base number. Thus, this will give the "range" effect to the graph for different level of DiffServ (see figure 4). It is worth noting that there is a minor scalability problem with this model. That is, the model cannot accommodate large number of DiffServ levels. One way to approach the scalability issue is to say that there are 3 main levels of DiffServ with 5 sublevels of each main level of DiffServ.

To avoid a combinatorial explosion, since we do not have the maximum number of flows sizes, we need to scale down the sizes of the flows proportionally. Practically, we can divide the flows sizes in the particular level with the maximum sizes that can be transmitted through that level.

Because each level has different configuration of average delay value, we need include this effect in to our delay score function. The average delay value would affect the overall flows sizes transmitted through the particular level.

Since we are interested on the overall delay score information for the whole DiffServ level, we average the total of delay score of each level.

The delay score is weighted based on the end-user's experience, using the RDI value. The smaller the RDI value, the less important the delay. The RDI value is calculated based on the table 1 and the relative importance of the objects given by the end-user.

The delay at each DiffServ level is represented as such (figure 4). The average or expected value models the overall effect of delay. Research is required to identify the relationship between QoS parameters such as loss/delay and user perception. Only then can communications systems and protocols be designed from a user's perspective.

The maximum QoS is therefore a sum of the loss and delay scores (see equation 2). In other words, the maximizing QoS will be obtained by minimizing the sum of the delay and the loss scores. For the particular total budget, we want to get the combination that has the lowest score.

$$\max QoS = \min(Ls + Ds) \tag{3}$$

subject to:

$$\sum_{i=1}^{N} F_i P_i \le P; \sum_{i=1}^{N} F_i \le F$$

where $\boldsymbol{P}$ is total budget of the user; $\boldsymbol{F}$ is total flows sizes to be transmitted in kilobytes; $\boldsymbol{F_i}$ are flows sizes that will be transmitted through DiffServ level $\boldsymbol{i}$ in kilobytes; $\boldsymbol{P_i}$ are the prices of the DiffServ level $\boldsymbol{i}$; $\boldsymbol{N}$ is the maximum level in DiffServ; $\boldsymbol{P}$, $\boldsymbol{F}$, $\boldsymbol{F_i}$, $\boldsymbol{P_i}$ and $\boldsymbol{N}$ are element of integer

4 Binary Search Algorithm

Mapping a flow to one or more DiffServ level consists of two steps:

- Irrespective of the number of flows, consider all flows as belonging to the same flow and determine the DiffServ distribution that will yield the minimum score and satisfy the budget.
- Given the number of flows that can be accommodated in each DiffServ level, the next step is to map the distribute flows among the DiffServ levels based on the relative priority index. This second aspect of the mapping is beyond the scope of this paper.

We use binary search to reduce the search space for finding the optimum combination that will yield the minimum score (maximum QoS) and satisfy the budget (first step above). The algorithm does not distinguish between flows. The algorithm is briefly outlined below and consists of two stages.

Stage I

The algorithm begins by assigning all flows to the highest DiffServ level that the budget permits. If this assignment results in an under-budget assignment, half of the flows are assigned to the next higher DiffServ level. If this combination is over-budget, half of the flows in the higher DiffServ level are put back into the lower level. If the new distribution is still under-budget, half of the remaining flows in the lower level are assigned to the higher level. This procedure is repeated until a two-level DiffServ combination that utilizes the maximum allowable budget is found.

Stage II

In the second stage of the algorithm, the flows are distributed across three levels. Half of the flows in the middle level are transferred to the next higher DiffServ level and the other half to the next lower DiffServ level. If the result of this assignment is over-budget, then half of the flows in the highest level are assigned into the lowest level. This process is repeated until the mechanism finds an under-budget combination. Next, the number of flows in the highest DiffServ level is not modified, instead the algorithm now adjusts flows sizes in the middle DiffServ level and the lower DiffServ level. Binary search is once again employed to modify the flows sizes in the middle DiffServ level and the lower DiffServ level. When this three-level utilization reaches the maximum budget allowed, the process stops.

The algorithm proceeds to consider the most recent under-budget or over-budget combination. If the current combination is under-budget, the flows sizes in the highest level of the current combination are increased by the flow size of half of the flows of the most recent highest level under-budget combination. Alternatively, if the current combination is over-budget, decrease the flows sizes in the highest level by half of the flows at the most recent highest level over-budget combination. The process is repeated until the maximum utilization of the budget.

The underlying objective of this algorithm is to transmit as much of the flows through the higher DiffServ levels. The best combination is the one that gives the minimum score and this is defined to be the DiffServ distribution for the given budget. The complexity of this binary search algorithm is $O(lg\ n)$, where n is the total flows sizes. Since the running time to evaluate the best combination is mainly based on this algorithm, then complexity to produce the best combination is $O(lg\ n)$. Although the optimum combination may not be found, this algorithm results in fast computation time which is essential for assigning DiffServ levels as multimedia presentations contain large numbers of flows.

5 Experiments

Table 3 below shows the prices, delay and loss for one of our simulations. Figures 5 and 6 show how flows are distributed across different DiffServ levels with increasing budget using heuristic and optimum allocation respectively. As the binary heuristic algorithm does not consider all possible combinations, the sizes of the flows vary more than the optimum combination. However, as figure 7 indicates, the scores, which is measure of the QoS delivered to the users, are very similar in both cases, confining the validity of the approach. Even if the budget allows all flows to be transmitted at level 1, some flows are still transmitted at level 2. This is because of our approach to modeling delay, which is as in figure 4. Such as approach does not try to concentrate all the flows in one level. Although most of the flows are in level 1, a spread of flow delays is envisioned as being more beneficial to the user.

Table 3. Simulation Data

DiffServ Levels	Price	Average Delay	Average Loss
Level 1	$4 per KB	100 ms	1% per KB
Level 2	$2 per KB	200 ms	2% per KB
Level 3	$1 per KB	400 ms	4% per KB

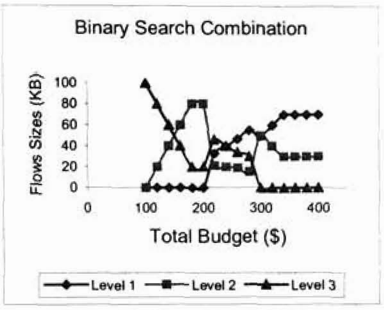


Figure 5. Binary Search Results

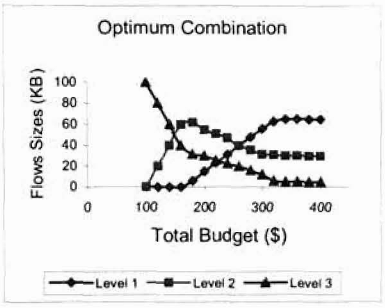


Figure 6. Optimum Combination

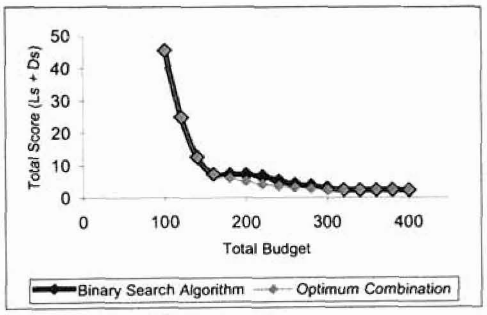


Figure 7. Scores: Binary vs. Optimum

An alternative model where both delay and loss are captured as linear functions was also simulated (figures 8 and 9). The simulation showed that the binary heuristic algorithm and the optimum allocation produced identical results, that is, identical scores and identical allocation of flows to the different DiffServ levels. This simulation also showed that budget permitting, the entire transmission takes place at the highest DiffServ level 1.

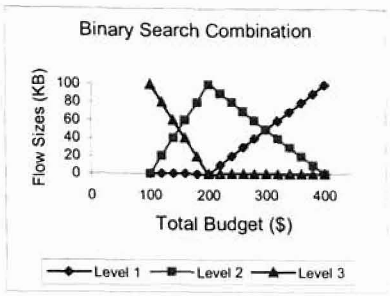


Figure 8. Binary Search Result

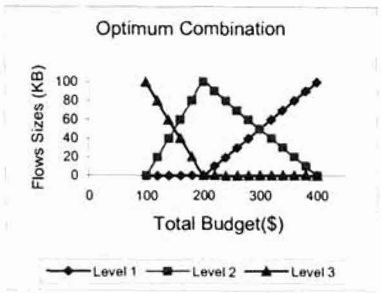


Figure 9. Optimum Combination

From figures 8 and 9, we can see that the both binary search combination and the optimum combination utilizes only 2 levels. This is the mapping of the linear delay and loss models. As figure 10 suggests the total score (Ls + Ds) for 2-level utilization is always smaller than any 3-level utilization for all possible total budget in the graph, which means the 2-level utilization will always maximize QoS.

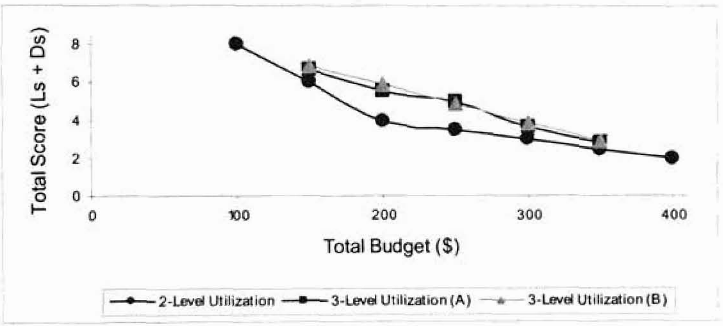


Figure 10. Scores Comparison for 2-level Utilization and 3-level Utilization

This phenomenon occurs because of the model used to represent delay and loss. The impact of delay and loss on user perception has not been reported in the literature. It will not be possible to provide a precise mapping mechanism to the different DiffServ levels such that user perception is enhanced until the perceptual impact of deteriorating QoS on the user has been studied in detail.

6 Conclusions

In this paper, we have proposed a mapping to assign user prioritized audio-visual objects to DiffServ levels such that QoS is maximized within given budget constraints. The ultimate objective of the mapping is to provide the user with optimum quality. The basic idea is to transmit higher priority objects at higher DiffServ levels, thus benefiting the user. This paper shows that the models to capture delay and loss will impact on DiffServ distribution. However scenes in a video may change resulting in different objects for a different scene. Even if the objects are identical, their sizes may change (zooming in for example). The work reported here does not take into account the changes in the number and types of objects caused by scene changes or the changes in data sizes of objects caused by different views, zooming, etc. The approach proposed in this paper can be extended to meet the constraints caused by network congestion. Future work will focus on these aspects.

References

1. R. T. Apteker, J. A. Fisher, V. S. Kisimov and H. Neishlos, "Video Acceptability and Frame Rate," *IEEE Multimedia*, vol. 2, no. 3, Fall 1995, pp. 32-40.
2. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, IETF, December 1998.
3. R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource Reservation Protocol (RSVP)," RFC 2205, IETF, September 1997.
4. G. Ghinea and J. P. Thomas, "QoS Impact on User Perception and Understanding of Multimedia Video Clips," *Proc. ACM Multimedia Conference '98*, 1998.
5. Ghinea and J. P. Thomas, "Crossing the Man-Machine Divide: A Mapping based on Empirical Results," *The Journal of VLSI Signal Processing - Systems for Signal, Image, and Video Technology*, vol. 29, no 1-2, August/ September 2001, pp. 139-147.
6. ISO/IEC JTC1/SC29/WG11 (MPEG), website: http://www.cselt.it/mpeg.
7. JPEG2000 Part 2 Final Committee Draft, ISO/IECJTC1/SC20 WG1 N2000, December 2000.
8. A.G. Malamos, E.N. Malamas, T.A. Varvarigou and S.R. Ahuja, "A Model for Availability of Quality of Service in Distributed Multimedia System," *Multimedia Tools and Applications*, 16, 2002, pp. 207-230.
9. K. Nichols, V. Jacobson and L. Zhang, "A Two-Bit Differentiated Services Architecture for the Internet," RFC 2638, IETF, July 1999.
10. J. Shin, J. Kim, and C.-C. J. Kuo, "Quality-of-Service Mapping Mechanism for Packet Video in Differentiated Services Network," *IEEE Trans. on Multimedia*, vol. 3, no. 2., June 2001, pp. 219-231.
11. J. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, IETF, September 1997.
12. M. Zitterbart, "A Model for Flexible and High Performance Communication Sub-systems", *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 1, January 1993, pp. 507-518.

VITA 2

Karyanta Purne

Candidate for the Degree of

Master of Science

Thesis: USER-ORIENTED QUALITY OF SERVICE MAPPING MECHANISM FOR MPEG-4 IN DIFFERENTIATED SERVICE NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Jakarta, Indonesia, on May 1, 1975, the son of I Wajan Purne and Jap Pauw Joeng.

Education: Graduated from Tarsisius I High School, Jakarta, Indonesia in June 1993; received Bachelor of Science degree in Business Administration, major in Economics, minor in International Business from Oklahoma State University, Stillwater, Oklahoma, in May 1998. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in August 2002.