7-1-2000

A Study of Dynamic Bandwidth Allocation With Preemption and Degradation For Prioritized Requests

Pranav Dharwadkar
Purdue University School of Electrical and Computer Engineering

Howard Jay Siegel
Purdue University School of Electrical and Computer Engineering

Edwin K. P. Chong
Purdue University School of Electrical and Computer Engineering

A STUDY OF DYNAMIC BANDWIDTH ALLOCATION WITH PREEMPTION AND DEGRADATION FOR PRIORITIZED REQUESTS

PRANAV DHARWADKAR
HOWARD JAY SIEGEL
EDWIN K. P. CHONG

TR-ECE 00-9
JULY 2000

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

A Study of Dynamic Bandwidth Allocation With Preemption and Degradation For Prioritized Requests

Pranav Dharwadkar

Howard Jay Siegel

Edwin K. P. Chong

Purdue University

School of Electrical and Computer Engineering

1285 Electrical Engineering Building

West Lafayette, IN 47907-1285 USA

{dharwadk, hj, echong)@ecn.purdue.edu

July 2000

Purdue University

School of Electrical and Computer Engineering

Technical Report TR-ECE 00-9

2000 • Purdue University • P. Dharwadkar, H. J. Siegel, E. K. P. Chong • TR-ECE 00-9 • 2000

972109

ACKNOWLEDGMENTS

This research was supported by the DARPA/ISO BADD Program and the Office of Naval Research under grant number N00014-97-1-0804 and by the DARPA/ITO AICE Program under contract numbers DABT63-99-C-0010 and DABT63-99-C-0012. Intel and Microsoft donated some of the equipment used in this research.

We would like to thank Prof R. L. Givan for his insightful comments we are grateful to Shoukat Ali, Tracy Braun, Jong-Kook Kim and Amit Naik, who read the manuscript carefully and offered a number of useful suggestions.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Page

Page

ABSTRACT

Bandwidth allocation is a fundamental problem in communication networks where bandwidth needs to be reserved for requests (connections) to guarantee a certain quality of service (QoS) for the request. Guaranteeing QoS to the request means that the user can explicitly specify certain requirements for a request such as bandwidth. The problem of bandwidth allocation is further intensified when the requested bandwidth exceeds the available unused bandwidth and so not all requests can be completely served. This research examines on-line bandwidth allocation, where the decision for acceptance or rejection of the request has to be made when future requests and their arrival statistics are not known. A request can be defined as a flow of information from a source to a destination with a certain amount of bandwidth, a priority level, a utility function that is based on the bandwidth received, and a worth that is based on the utility function and the priority level. The goal of the research is to develop a scheduling heuristic for an overloaded system that attempts to schedule the requests such that the sum of the worths of the requests satisfied in a fixed interval of time is the maximum. The scheduling heuristic can preempt or degrade already scheduled requests. Three different types of utility functions, step, linear, and concave are examined. Other parameters being considered include network loading and the relative weights of the different priority levels.

1. INTRODUCTION

Bandwidth allocation is one of the most important problems in the management of networks that offer a guaranteed bandwidth policy, such as ATM [ATM99]. In such a network, if a user (i.e., an application) wants a guaranteed bandwidth for a communication (also called a request), the user has to reserve, in advance, the amount of bandwidth required. This is in contrast to the current Internet wherein the requests are satisfied with a best-effort policy, i.e., there is no guarantee on the bandwidth received by a user. The advantages of a guaranteed bandwidth policy are many. Examples are as follows.

1. Bounded delay: the delay experienced by a user's request is bounded.
2. Differentiated service: users can expect different levels of quality of service (QoS) based on the amount of money paid and the amount of bandwidth reserved.
3. Simple pricing: a user can be charged for the amount of bandwidth allocated.
4. Fairness: just one user cannot occupy all the bandwidth available.

The major drawback of a guaranteed bandwidth policy is inefficiency; the bandwidth received may not actually be fully utilized Thus, a good bandwidth allocation strategy is essential for such networks.

If the network were to reserve bandwidth for a request and the request does not use all the reserved bandwidth then that would lead to under-utilization of the links. For example, assume the user requests an interactive multimedia session. An interactive multimedia session may involve human interaction due to which there will be periods when the link is not being used, e.g., when the user is reading the information presented. Because the bandwidth was reserved for the user's request, any unused bandwidth (that is not being used by the user's request) may not be used to satisfy other requests, and hence may result in a loss of revenue to the network bandwidth provider. It may be argued that the if the user is willing to pay for the unused bandwidth, there would be no loss of revenue to the network bandwidth provider. Given a choice, a user may not be willing to pay for the unused bandwidth. In this scenario, the network should be capable of dynamically allocating bandwidth as and when the user requires it.

Another case where dynamic bandwidth allocation would be useful is a real-time multimedia session. For example, assume the user requests a video clip. The user may dynamically demand more bandwidth by resizing the window and consequently requesting a higher image resolution or, by fast-forwarding the clip. In this case, if the user is willing to pay for the extra bandwidth required, the network should be able to dynamically allocate more bandwidth to the user. If the network has a fixed bandwidth reservation policy, the user would have to reserve the total bandwidth needed during the session, i.e., the user would have to reserve and pay for the extra bandwidth too. But the user may not use the extra bandwidth for most of the time during the session. Thus, greater flexibility is needed than just reserving a fixed amount of bandwidth. This flexibility can be provided if the network is capable of dynamically allocating bandwidth.

Dynamic bandwidth allocation constitutes a paradigm shift from current bandwidth allocation policies such as reservation of bandwidth. In dynamic bandwidth allocation the users do not reserve the required bandwidth, but dynamically rent the required bandwidth. The network service provider would like to maximize the revenue received by renting bandwidth to different users. Maximization of revenue may involve preempting/degrading existing user requests to satisfy "more valuable" requests that would otherwise be rejected because of lack of available bandwidth. The rationale behind this is as follows. Assume a request has rented some bandwidth and paid some money for the rented bandwidth. If this request is occupying the bandwidth that is needed by a request paying more for the same amount of bandwidth, then it may be beneficial to preempt/degrade the lower paying request in favor of the higher paying request. The goal of this research is to develop a heuristic that will aid in making decisions as to which request should be admitted/rejected, and what bandwidth should be allocated to the request if admitted.

In particular, the objective of this research is to develop a scheduling heuristic for an overloaded system that attempts to schedule the requests such that the sum of the worths of the satisfied requests obtained by the schedule is the maximum. One of the factors affecting the worth of a request is the utility function of the request that is based on the amount of bandwidth received by the request. The utility function of the request depends upon the application generating the request. For example, a file transfer may have a concave utility function because it is not real-time and hence is delay insensitive, i.e., it can tolerate some delay [She95]. Alternatively, a real-time application such as Internet telephony may have step utility function [She95]. The heuristic developed in this research considers requests having three different types of utility functions: step, concave, and linear. Most of the requests that currently exist in the Internet have a utility function that is one of these three types. To the best of the author's knowledge, there is no research reported in the literature with the objective of maximizing the sum of the worths of satisfied requests with these three different types utility functions of the requests.

In a military environment, there may be many warfighters in remote locations requesting information such as terrain maps, enemy locations, and troop movements. Each of these requests for information may have a priority and a utility associated with it that indicates the "worth of the request to the warfighter. For example, if the warfighter receives the information requested after the deadline specified, then it would be of zero worth to the warfighter. If there were many warfighters requesting some information, then it may be possible that not all the requests can be satisfied. Thus, it may be beneficial to maximize the worths of all the requests satisfied. The heuristic developed in this research can be used to allocate the bandwidth to the different warfighter’s requests such that the total worth of all the requests satisfied is the maximum.

This heuristic can be used by Internet service providers (ISP) that provide bandwidth to its clients for some amount of money. The value of a client's request may be the amount of money the client pays for the request, which is a function of the amount of bandwidth the client's request received and the cost per unit bandwidth the client is willing to pay. The ISP would like to maximize the total amount of revenue received by accepting the "more valuable" client requests and rejecting the "less valuable" client requests. The value of a client's request may correspond to the worth of a request in this research. **Thus,** maximizing the total revenue received by the ISP would correspond to maximizing the sum of the **worths** of all the requests satisfied in this research. The ISP can use the heuristic developed in this research to determine the bandwidth allocations to the different client requests such that the sum of the **worths** of the client requests satisfied is maximum, thus maximizing the revenue received.

This dynamic bandwidth allocation heuristic has been developed for scheduling requests to achieve a high aggregated value within a distributed network **infrastructure** envisioned in the Defense Advanced Research Projects Agency (DARPA) Agile Information Control Environment (AICE) program **[AIC98].** The objective of the DARPA AICE program is to enable dynamic management of network resources over distributed and disparate networks (including both military and commercial networks) in accordance with the commander's policy. This policy includes assignment of priority levels to requests and relative weights for the priority levels. AICE consists of four functional layers: a physical networks layer, a MetaNet layer, an Adaptive Information Control (AIC) layer, and an Information Policy Management (IPM) layer. The MetaNet layer interacts with multiple physical networks to provide end-to-end **QoS** differentiable services to the AIC layer for allocation. The AIC layer is responsible for the allocation of the end-to-end resources established by the MetaNet to requests to achieve a high global worth as defined by the **IPM** layer. The bandwidth allocation heuristic in this research has been developed for an AICE-like environment where the AIC has direct knowledge of the state of the underlying network.

Thus, the scheduling heuristic presented here attempts to maximize the sum of the **worths** of the prioritized requests satisfied in an overloaded AICE-like communications environment. It assumes each request has a utility function for the bandwidth received that is concave, linear, or step function. Furthermore, a request's assigned bandwidth may be preempted or degraded by this heuristic. Simulation experiments are conducted to evaluate several variations of the heuristic and compare them to upper bounds and a simple scheduling technique.

The report is organized as follows. The network model and the request model assumed in this research are described in Section 2. In Section 3, the problem that this research attempts to solve and the need for a heuristic are explained. A brief overview of some of the literature related to this work is presented in Section 4. In Section 5, the scheduling heuristic developed in this research is explained and the bounds on the performance of the heuristic are examined in Section 6. The simulation experiments conducted and the results obtained are presented in Sections 7 and 8, respectively. The last section provides a brief summary of this work and also discusses possible future work The pseudo-code for the heuristic is given in Appendices A, B, and C, and the glossary of notation is presented in Appendix D. The C source code for the heuristic is given in [Dha00].

2. DISTRIBUTED COMMUNICATION NETWORK

2.1. Overview

The underlying network model and the request model assumed in this report is discussed in this section. The performance measure for the heuristic is also presented in this section. The network model used in this research is similar to the network models considered in [FeM95] and the Internet [Com95, NAP98]. To explain the assumed network model better, a brief description of these other network models is presented.

2.2. Existing Network Models

2.2.1. Distributed computing

An admission control heuristic for distributed applications (e.g., distributed computing) over an ATM network is described in [FeM95]. The admission control heuristic proposes to allow connections belonging to the same application to share common links to increase utilization. The network model used in [FeM95] is similar to the network model assumed in this research. The model in [FeM95] assumes that there are a set of slave hosts that send their requests to a master host via the same intermediate switch and the same intermediate link. The intermediate switch is assumed to be a high-speed switch that forwards the data from the slave hosts to the master host, but the intermediate link has a fixed capacity and hence is the bottleneck. If the sum of the bandwidths of the requests on the intermediate link exceeds the link bandwidth then some requests may have to be dropped. Thus, it is essential to perform admission control so that the sum of the bandwidths of the requests does not exceed the link bandwidth. A similar scenario exists in the network model assumed in this report, as will be explained in Subsection 2.3.

2.2.2. Current Internet

The original Internet architecture consisted of a single dominant National Science Foundation (NSF) backbone network that supported all the Internet traffic. This architecture underwent a major change fiom the single dominant NSF backbone network to a series of commercial provider owned backbone networks. The commercial providers typically are the ISPs that offer Internet access to their clients such as large corporations, universities, and individual dial-up users. Under these conditions, the backbones had to have some means of exchanging data. To serve this purpose the concept of a network access point (NAP) was introduced. **NAPs** were designated to serve as data interchange points for the ISPs, as shown in Figure 2.1.

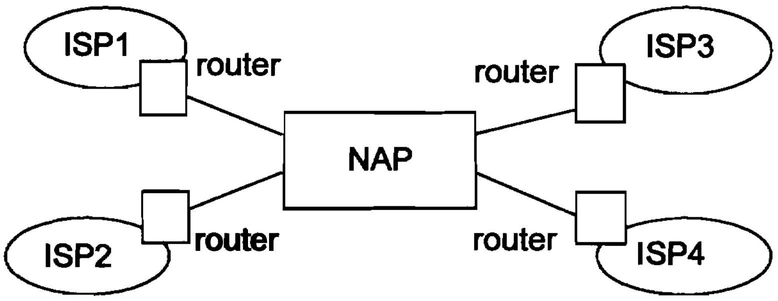

Figure 2.1. Network access point (NAP).

The ISPs send the traffic fiom its clients to the NAP and the NAP then forwards the traffic fiom one ISP to another. The NAP switches (i.e., forwards) data at a very high speed. The ISPs typically have a service level agreement (SLA) with the NAP wherein the ISPs agree to send data at a rate no greater than a predetermined fixed rate. The switching capacity of the NAP is typically very high, and if the ISPs do not violate their SLA, then the NAP is usually not the bottleneck. Thus, the ISP would like to maximize the sum of the "worths" of the requests that it sends to the NAP in accordance with its SLA. The heuristic developed in this research considers a similar problem for the network model shown in Figure 2.2.

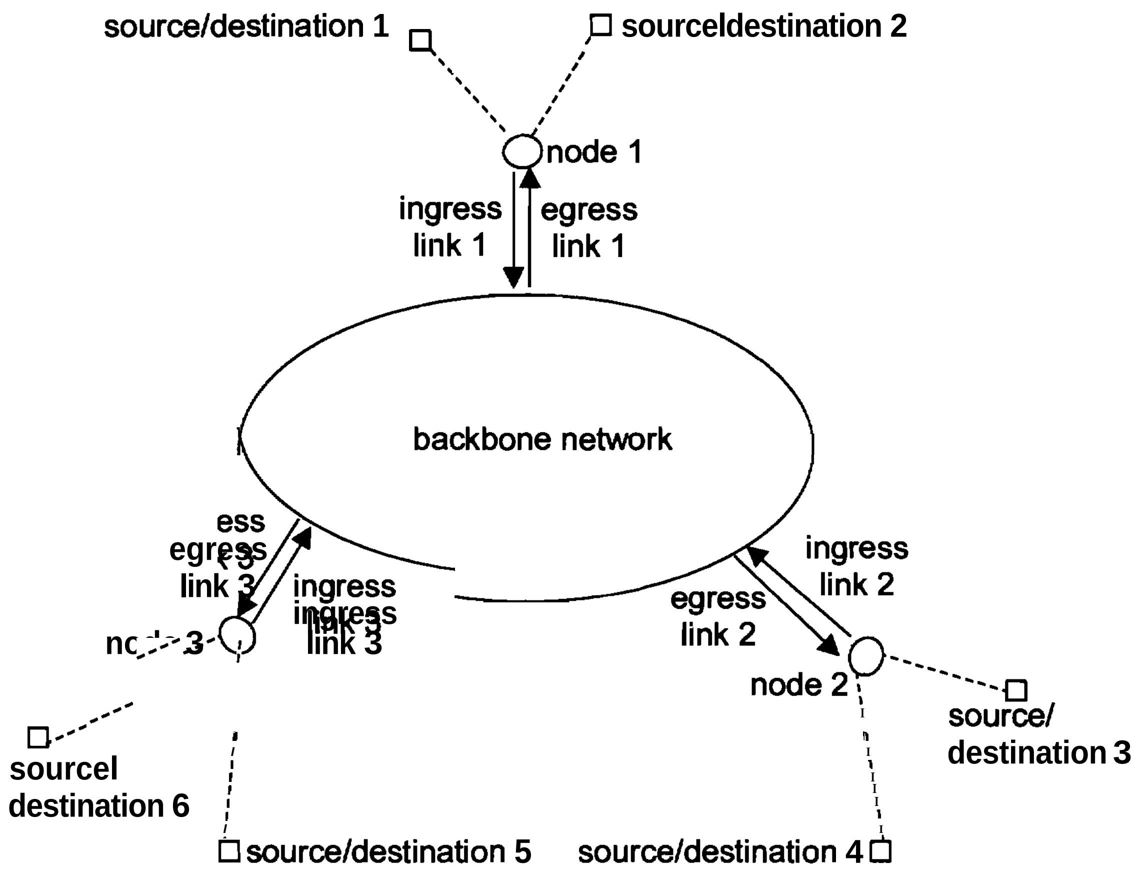

Figure 2.2. Network model.

2.3. Network Model

The underlying network model assumed in this report is shown in Figure 2.2. The sources shown are the applications that generate the requests, where a request can simply be defined as a flow of information from a source to a destination, with a certain amount of bandwidth, a priority level, and a utility that is a function of the amount of bandwidth received. A request is formally defined later in this section. The decision to admit/reject a request is made at the nodes shown in Figure 2.2. If a request is admitted, the amount of bandwidth to be allocated to the request is decided at the nodes.

The nodes provide network ingress and network egress. If a request is admitted, the nodes send the request to the network cloud via the links connecting the node to the network cloud as shown in Figure 2.2. Each node is connected to the backbone network by two unidirectional links. A network ingress link transfers data fiom a node to the backbone network A network egress link transfers data fiom the backbone network to a node. The request would be routed by the backbone network to the destination node via its egress **link,** and then delivered to the final destination. The backbone network can be thought of as a very high-speed switch that forwards the data fiom the ingress links to the egress links. The ingress links and egress links have fixed capacities. It is assumed that if a request can be accommodated by its associated ingress and egress links, then the network can satisfy the request. That is, it is the links that are the system bottlenecks, and not the backbone network Thus, the requests should be scheduled such that the sum of the bandwidths of the requests utilizing a link does not exceed the link bandwidth.

The model assumed here is very similar to the model in **[FeM95],** where the link is the bottleneck and not the switch. If the backbone network in Figure 2.2 is a NAP (or a switch) and the **ingress/egress** nodes in Figure 2.2 are the **ISPs,** then the model assumed here is similar to the current Internet. In the Internet, the ISP would like to maximize the *sum* of the "**worths**" of the requests sent over a link. In the model assumed here, the goal is to maximize the sum of the worths of the requests satisfied such that the sum of the bandwidths of the satisfied requests does not exceed the link bandwidth.

The problem of hierarchical link sharing has been discussed in **[FlJ95],** where a single link has to be shared by multiple organizations with different levels of **QoS** requirements. In **[FlJ95],** a single bottleneck link is considered, and this bottleneck link needs to be shared among different traffic types such as real-time **(e.g.,** steaming audio and video) and non-real-time traffic **(e.g.,** ftp). The network model assumed in this research can be considered to be an extension of the single bottleneck link model in **[FlJ95]** by considering two bottleneck links instead of one.

2.4. Request Model

2.4.1. Request definition

A request is defined as a flow of information from a source node to a destination node with a certain amount of bandwidth, a start time, an end time, a priority level, and a utility function that is based on the amount of bandwidth received. Requests that require a certain amount of bandwidth for some specified duration of time are called session type requests. Assume for a request r_k, i_k is the network ingress link or the in-link, o_k is the network egress link or the out-link, s_k is the start time, e_k is the end time, rb_k is the requested bandwidth, b_k is the current bandwidth, p_k is the priority level, u_k is the utility (a value between 0 and 1 that is a function of the amount of bandwidth received by the request), and w_k is the worth. Thus, the request r_k can be represented by

$$r_k = \{i_k, o_k, rb_k, b_k, s_k, e_k, p_k, u_k, w_k\} .$$

The session of the request is defined to be the time interval from the start time to the end time of the request.

In the military environment where this research can be applied, if the request cannot be allocated its desired bandwidth (rb_k) because of the oversubscribed network or its priority level, then the request may either be allocated degraded bandwidth (determined by the network) or no bandwidth at all. In such a situation only, the requestor may be willing to accept degraded bandwidth rather than have the request rejected. Thus, the bandwidth allocated to the request need not remain fixed for the duration of the request, i.e., the bandwidth allocated to the request can be decreased or increased during the session of the request.

The total utility of a request is calculated based on the amount of the bandwidth that the request received during every time instant (e.g., second) of its session. The bandwidth received by the request at every time instant of its session is denoted by $b_k(t)$.

Then, the total utility of a request r_k is U_k, a value between 0 and 1, where

$$U_k = \left[\left(\sum_{t=s_k}^{e_k} u_k(b_k(t)) \right) \Big/ (e_k - s_k) \right]. \qquad (2.1)$$

When calculating the worth of a request, a weighted priority of the request is used rather than just its priority level. The reason is explained in Subsection 2.4.2. The weighted priority is some function of the priority level of the request. Let this function be denoted by Π. The worth of the request is defined as the weighed priority times the total utility of the request. Thus, the worth of the request is calculated as

$$w_k = \Pi(p_k) \times U_k$$

Therefore, substituting the expression for the total utility of the request from Equation 2.1,

$$w_k = \Pi(p_k) \times \left[\left(\sum_{t=s_k}^{e_k} u_k(b_k(t)) \right) \Big/ (e_k - s_k) \right] \qquad (2.2)$$

This approach to calculating the worth is based on the FISC measure in [KiH00]. The priorities and the utility functions are explained in detail in the following subsections.

2.4.2. Priority

Bandwidth should be allocated to the requests in some order. Intuitively, this ordering should begin with "more important" requests. Some priority must therefore be associated with a request so that an algorithm can evaluate the relative merit of any given request compared to *any* other request. As mentioned earlier, a weighted priority (that is some function of the priority of the request) is used to calculate the worth of the request. The weight of a priority level indicates the relative importance of a priority level to another.

In this research, it is assumed that there are four priority levels, where level i is more important than level j, for $i < j$, $1 \leq i,j \leq 4$. The priority scheme is based on a weighting constant ω, as was used in [ThB00]. The weight of priority level i is:

$$\Pi(i) = \omega^{(4-i)}.$$

Two cases for ω are considered: mode two, when $\omega = 2$, and mode ten, when $\omega = 10$. In mode two, with $\omega = 2$, the weighted priority of priority level one would be eight, and the weighted priority of priority level four would be one. In mode ten, with $\omega = 10$, the weighted priority of priority level one would be 1000, but the weighted priority of priority level four would still be one. Thus, even though the priority levels of the requests remain the same, the relative weighted priorities would change from mode two to mode ten.

The reason for this concept of mode-based weighted priorities in the military context that this work was carried out is as follows. Assume that there are two different modes, a war mode (where $\omega = 10$) and a peace mode (where $\omega = 2$). A request issued by a commander may be assigned a priority of one while a request issued by a private may be assigned a priority of four. Recall that it is assumed that the communication system is overloaded. In peace mode, a priority level one request (with weight $2^3 = 8$) is considered worth more than seven priority level four requests ($7 \times 2^0 = 7$). It may be beneficial to satisfy one priority level one request instead of seven priority level four requests, or nine priority level four requests instead of one priority level one request. But in the war mode, a priority level one request (with weight $10^3 = 1000$) is considered worth more than 999 priority level four requests ($999 \times 10^0 = 999$). Thus, it may be beneficial to satisfy one priority level one request instead of 999 priority level four requests, or 1001 priority level four requests instead of one priority level one request. This effect of change in relative importance of priorities (of the requests) due to change in mode can be captured by the concept of a weighted priority as explained above.

In a commercial network a similar situation may exist. The two modes can be a lightly loaded network (where $\omega = 2$) and a heavily loaded network (where $\omega = 10$). A request issued by the Chief Executive Officer (CEO) of the company may be assigned a priority of one while the request issued by an employee may be assigned a priority of four. In a lightly loaded network, a priority level one request (with weight $2^3 = 8$) is considered worth more than seven priority level four requests (with weight $7 \times 2^0 = 7$). Hence, in a lightly loaded network, the priority level one request may be satisfied instead of seven priority level four requests. But in a heavily loaded network, a priority level one request (with weight $10^3 = 1000$) is considered worth more than 999 priority level four requests (with weight $999 \times 10^0 = 999$). Hence, the priority level one request would be satisfied instead of 999 priority level four requests. This significantly higher relative importance for the priority level one request, in a heavily loaded network, can be achieved by the weighted priority scheme described earlier.

2.4.3. Utility function

The utility of a request is a function of the bandwidth that the request receives during its session. This utility can be any arbitrary function of the bandwidth received, depending upon the application generating the request. Different types of applications could have different needs both in terms of desired bandwidth and ability to operate with less than the desired bandwidth

For example, there may be high-quality multimedia applications that are designed to be transmitted at a fixed bandwidth. For such an application, if the bandwidth requirements are met, the utility obtained is the maximum utility of the request. If the requirements are not met (by not allocating sufficient bandwidth to the application), the utility obtained is zero. Thus, such an application may generate a request that has a step utility function, i.e., if the application gets the bandwidth needed, then its utility would be the maximum utility and if not, zero utility [She95]. The utility function for such a request with a bandwidth requirement of rb_k is illustrated in Figure 2.3.

There may be other applications that are designed to adapt to transmissions with less than the full desired bandwidth. For example, a teleconferencing session may be structured to operate with reduced bandwidth and commensurate reduced quality. Such an application may generate a request that has a linear or a concave utility function [She95]. In addition to the requested bandwidth rb_k, the request may even specify a minimum bandwidth requirement mb_k. If the request is allocated bandwidth less than its minimum bandwidth mb_k, then its utility is zero.

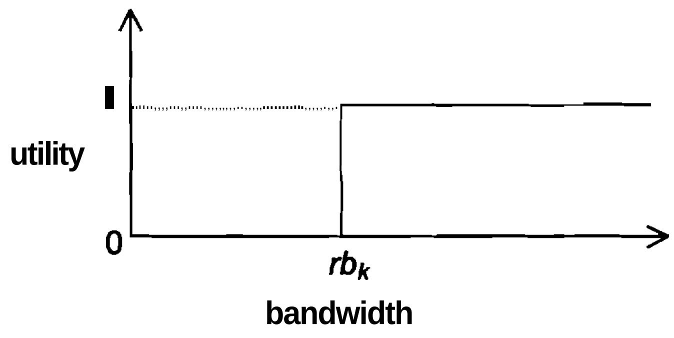

Figure 2.3. Request with a step utility function.

In case of a step utility function, the minimum bandwidth would be the bandwidth requested. The linear and concave utility functions of a request are illustrated in Figure 2.4. Most of the requests that currently exist in the Internet have a utility function that is one of the three types of utility functions considered in this research [She95].

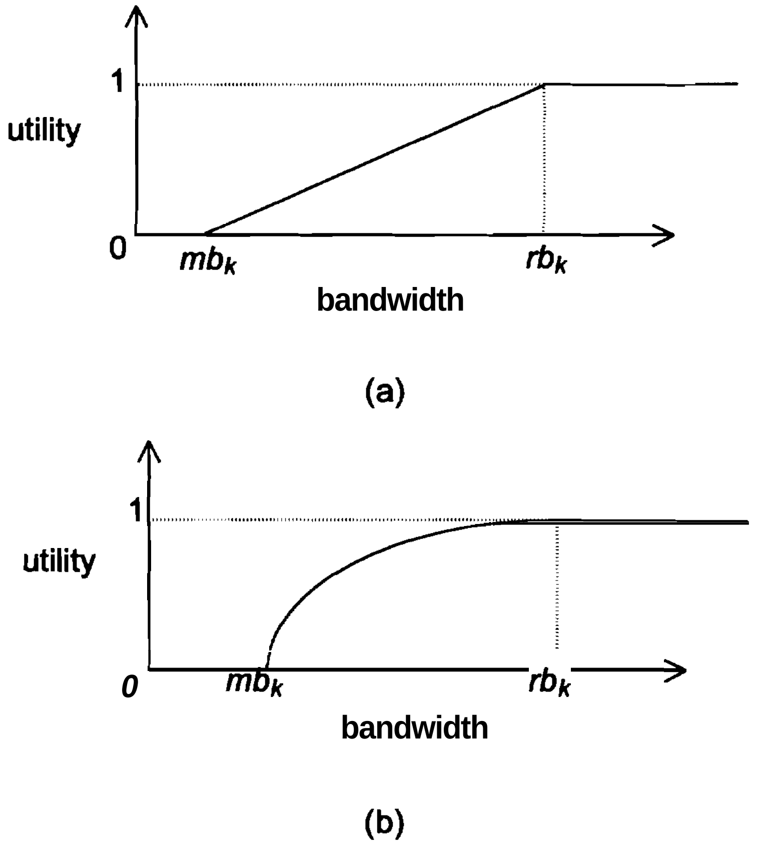

Figure 2.4. Request with a (a) linear utility function, and (b) concave utility function.

2.5. Performance Measure

Let $\underline{S}$ be the set of all the requests that arrive at the nodes over a fixed interval of time. The worth of a request is calculated using Equation **2.2.** If a request in S is satisfied, Equation **2.2** yields the worth of the request, or else Equation **2.2** yields zero. The performance measure assumed in this report is the **sum** of the worths of all the requests in S. This **sum** of the worths of all the requests in S is the total worth $\underline{W}$.

$$W = \sum_{k \in S} w_k \tag{2.3}$$

Substituting the expression for w_k **from** Equation **2.2** in Equation **2.3.**

$$W = \sum_{k \in S} \Pi(p_k) \times \left[\left(\sum_{t=s_k}^{e_k} u_k(b_k(t)) \right) \Big/ (e_k - s_k) \right] \tag{2.4}$$

The goal of this research is to maximize this total worth W. **Other** studies that use the sum of the worths as the performance measure in AICE-like environments include **[ThB00, ThS00, ThT00b].**

2.6. Summary

The network model and the request model assumed in this research have been described in this section. The concept of weighted priorities and how it affects the relative importance of priorities of the requests has also been explained. In this section, the three **different** types of utility **functions** and the applications that can generate such utility **functions** have been discussed. The performance measure for the heuristic, **i.e.,** sum of the worths of all the requests that arrive during a given interval of time, has also been stated. In the next section, the problem that this research attempts to solve and the need for a heuristic are explained.

3. PROBLEM DEFINITION

3.1. Overview

In this section, the problem that this research attempts to solve is presented. The intractability of the problem is argued and the need for a heuristic is explained. The similarities and differences between the bandwidth allocation problem described here and the **fractional** knapsack problem are discussed in Subsection 3.4.

As **defined** earlier in this report, a request is a flow of data **from** a source node to a destination node with a start time, an end time, a certain amount of bandwidth, a priority level, and a utility that is a function of the amount of bandwidth received. The worth of the request is defined as the product of the weighted priority and the utility of the request. The utility function of the request can be a linear, step, or a concave function of the amount of bandwidth the request receives. Recall that the performance of a schedule is determined by W, where

$$W = \sum_{k \in S} \Pi(p_k) \times \left[\left(\sum_{t=s_k}^{e_k} u_k(b_k(t)) \right) \Big/ (e_k - s_k) \right]$$

3.2. Types of Scheduling

Scheduling heuristics can be grouped into two categories: off-line scheduling and on-line scheduling. In the context of this research, an off-line scheduling algorithm would have **knowledge** of all the requests that have arrived in the network **[BrS99].** The off-line scheduling algorithm, as the name suggests, is executed off-line with no time constraints such as, start time of a request. Alternatively, an on-line scheduling algorithm has to make decisions of acceptance/rejection of requests without prior knowledge of the future arrival of requests [MaA99]. In this research, no assumptions are made regarding the future arrival of requests. Thus, the performance of the heuristic developed in this research does not depend upon the arrival pattern of the requests. Because an off-line scheduling heuristic has knowledge of all the requests that have arrived at the nodes, and no time constraints, the value of W obtained by an off-line scheduling heuristic is usually better than that obtained by an on-line scheduling heuristic.

On-line scheduling can be considered to consist of two types: immediate scheduling and batch scheduling [MaA99]. In immediate on-line scheduling, requests are considered for scheduling as soon as they arrive. Alternatively, in batch on-line scheduling, the requests are not considered for scheduling as soon as they arrive, but they are first grouped in batches. These batches are processed, and the processed batch is then considered for scheduling. For example, processing the batch of requests may involve sorting the batch by some measure such as worth per bandwidth desired by the request. The sorted batch may then be scheduled by first scheduling the request with the highest worth per bandwidth requested, then scheduling the request with the next highest worth per bandwidth requested, and so on. In this research, the immediate on-line scheduling problem is considered. This problem is further explained in the next subsection.

3.3. Immediate On-line Scheduling Problem

The network ingress and network egress links, shown in Figure 2.2, have a fixed bandwidth. This fixed amount of link bandwidth needs to be shared among the requests utilizing the link, i.e., the link bandwidth needs to be allocated to the different requests on the link. The problem of bandwidth allocation arises when the total bandwidth required by the requests exceeds the available link bandwidth and not all requests can be satisfied. Such a system where the total bandwidth requested exceeds the available bandwidth is an overloaded system. The problem is further intensified because the decision for acceptance/rejection of the request has to be made on-line, i.e., the future arrival of requests is not known (and no assumptions are made regarding the arrival of requests).

The objective is to maximize W over all the links subject to the constraint that the total bandwidth of the requests satisfied on a link does not exceed the link bandwidth.

The goal of this research is to develop a scheduling heuristic for an overloaded system that attempts to **maximize** W. This research considers preemption as well as degradation of some existing requests to allow more bandwidth to be allocated to new requests to increase the sum of the **worths** of all the requests satisfied during a given interval of time.

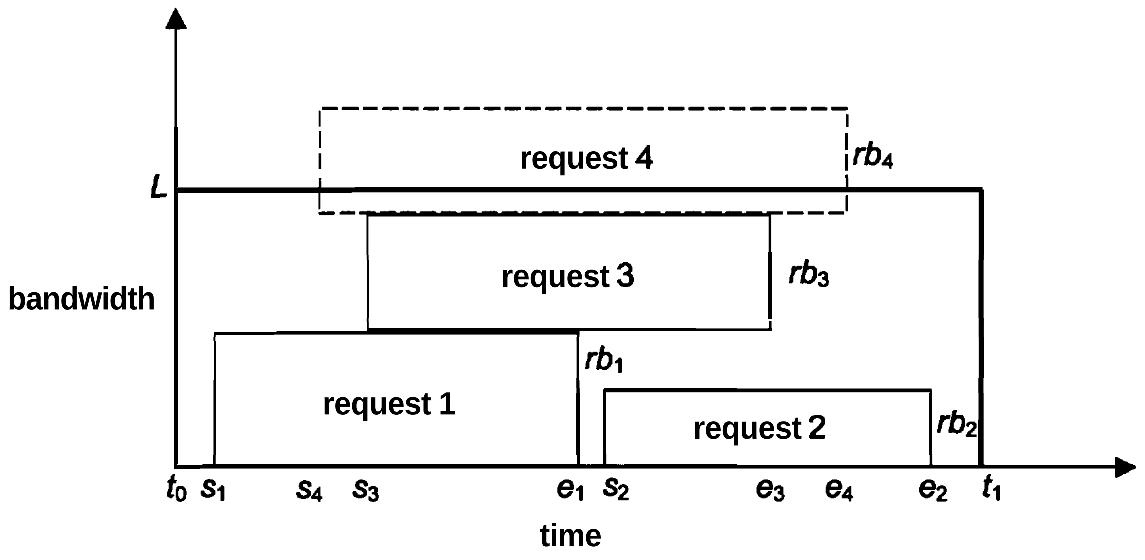

Figure 3.1. A snapshot of the requests satisfied at a link **from** time t_0 to t_1. Request 4 is the new request being considered for scheduling. Each rectangle indicates a request; the width of the rectangle is the duration of the request and the height of the request is the bandwidth required by the request. The height of the outer rectangle is the link bandwidth L. The start and the end times of the requests are indicated on the X-axis.

To explain the problem more clearly, refer to Figure 3.1. Consider that there are a few requests (requests 1, 2, and 3 in Figure 3.1) that have been already been scheduled (i.e., allocated bandwidth during the time interval specified). Let these requests have the same ingress link i and different egress links. Now suppose a new request r_4 (request 4 *in* Figure 3.1) arrives for scheduling at a time before *to.* Request r_4 *has* the same ingress link, a requested bandwidth of rb_4, and a worth of w_4. If the sum of the bandwidths of the requests currently being scheduled and of request r_4 exceeds the link bandwidth (as illustrated in Figure 3.1), then r_4 cannot be satisfied with its desired bandwidth. A decision has to be made whether to admidreject request r_4 and if r_4 is admitted then what bandwidth should be allocated to r_4.

The goal is to maximize W, the problem is how to make a decision such that this goal is achieved. The reason is, because of the on-line nature of the problem, it is always possible to second guess decisions made in the past, i.e., a decision made previously to accept a request may have been wrong because it caused a subsequent request with a higher worth to be rejected. For example, the accepted request may have used up the entire available bandwidth on the link causing a subsequent request of higher worth to be rejected due to lack of available bandwidth. Thus, the on-line nature of the problem leads to a lower W than what an off-line scheduling heuristic that has full knowledge of the arrival of the requests could achieve.

This aspect of the problem leads to the issue of degradation and preemption of existing requests to free up bandwidth to be used by new, "more valuable" requests. For example, if a request with a high worth cannot be satisfied because a request with a lower worth is occupying the available bandwidth then it would be beneficial to preempt/degrade the lower worth request to satisfy the higher worth request. But the issue is deciding which requests should be preempted/degraded, and what should be the amount of degradation if a request is degraded. This research attempts to develop a scheduling heuristic that will help make the above decision.

One method for deciding which requests to preempt/degrade, and the amount by which the bandwidth of a request should be degraded, is exhaustive search. Consider a set of all the existing requests that overlap in time with the new request r_4, i.e., consider all the requests that start or end or both during the <u>session</u> (i.e., time s_4 to e4) of the new request r_4. This set of requests is the set of conflicting requests for r_4's session. In the example shown in Figure 3.1, the conflicting requests would be requests 1, 2, and 3. Let there be $\underline{n}$ such conflicting requests. Many choices exist as to which request should be preempted/degraded. For example, one choice may be to allocate the full bandwidth needed by the new request and preempt one of the existing requests (e.g., r_3). Another choice may be to degrade the bandwidths of some of the existing requests (e.g., r_1 and r_2) and allocate the freed-up bandwidth to the new request. To determine which of these choices would result in maximizing the worths of this set of four requests, all the choices may have to be evaluated. Evaluating these choices may take a huge amount of time, as demonstrated next.

An example of the time taken for an exhaustive search is as follows. For the sake of simplification, assume that the range of bandwidth (required bandwidth − minimum bandwidth) for all the requests is the same. Let this range be m Kbps. Assuming a minimum increment in bandwidth of 1Kbps, each request can have m choices for the amount of bandwidth received. If there are n conflicting requests, and each request can have m choices for the amount of bandwidth received, then the total number of choices to be evaluated are m^n. If m = 100 (a typical value assumed in this research is 1000) and n = 6, the number of choices are 10^{12}. If the time taken for evaluating each choice is 1μs, then time taken for evaluating 10^{12} choices is 11.5 days, which is a huge amount of time. Hence, evaluating all the m^n choices is an infeasible solution for the on-line problem described above. Thus, there is a need for a heuristic that can solve the problem described above.

The immediate on-line heuristic has to make decisions regarding the admission/rejection of requests without prior knowledge about the future arrival of the requests. The heuristic has to make this decision before the start time of the request, i.e., the start time of the request is a constraint for the heuristic. Because of the reasons mentioned above, the immediate on-line heuristic does not perform as well as the off-line scheduling heuristic, which has prior knowledge of all the requests that have arrived, and has no time constraints.

Many heuristics are presented in the literature that consider the on-line scheduling problem. Some of the heuristics only consider preemption and not degradation of bandwidth allocated to the requests. Some heuristics only consider requests with a concave, continuously differentiable type of utility function. To the best of the author's knowledge there is no known heuristic or algorithm presented in the open literature that addresses the above bandwidth allocation problem considering the three different types of utility **functions** of the requests.

3.4. Fractional Knapsack Problem

The bandwidth allocation problem described above is related to the fiactional knapsack problem. However, it will be shown that the problem addressed in this report is more complex.

A fiactional knapsack problem is posed as follows **[CoL90].** A thief robbing a store finds n items; the i^{th} item is worth v_i dollars and weighs g_i pounds, where v_i, g_i are integers. The thief wants to take as valuable load as possible, but he can carry at most G pounds in the knapsack for some integer G. The thief can take **fractions** of the items rather than having to make a binary (0/1) choice for each item. What items should the thief take?

The bandwidth allocation problem considered in this research can be thought of as a **fractional** knapsack problem as follows. **Assume** the items correspond to requests, the worth of an item corresponds to the worth of a request, the weight of an item corresponds to the bandwidth required by a request, and the total weight G corresponds to the **link** bandwidth. In the fiactional knapsack problem the goal is to maximize the worth of the items stolen, while in this research, the goal is to maximize W. Satisfying means that the request **(i.e.,** item) was allocated some bandwidth above the minimum bandwidth required, during its session **(i.e.,** stolen). Thus, the bandwidth allocation problem considered to this problem has been shown to be very similar to a fiactional knapsack problem.

In a fiactional knapsack problem, the thief can maximize the total worth of items stolen as follows. The worth per pound of each item is first calculated. Obeying the greedy strategy, the thief begins by taking as much as possible of the item with the greatest worth per pound. If the supply of that item is exhausted and the thief can still take more, the thief takes as much as possible of the item with the next greatest worth per pound and so forth until the weight limit G is reached.

If the fractional knapsack approach is used for the bandwidth allocation problem described here, then the set of conflicting requests should be sorted by the worth per unit bandwidth. The request with the highest worth per unit bandwidth should be satisfied first, the request with the next highest worth per unit bandwidth should be satisfied next, and so on, until there is no more available bandwidth. The fractional knapsack problem would have to be solved at every time instant, because the set of conflicting requests (items in the store to be stolen) is different at every time instant.

It may appear from the above discussion that the solution to the fractional knapsack problem would yield a solution to the bandwidth allocation problem described in this research. But the solution to a "traditional" fractional knapsack problem would not a solution to the bandwidth allocation problem considered here. The reason is as follows. In a traditional fractional knapsack problem, the function relating the weight of an item to its worth is linear, i.e., if the thief took half of the item, the thief would get half the worth of the item. But in this research the function relating the worth of a request (worth of an item) to the bandwidth required by the request (weight of an item) can be of three types, linear, step, or concave. For example, if the request is allocated half of the bandwidth, the worth obtained may be half the worth (in case of linear utility function), zero worth (in case of step utility function), or 3/4th of the worth (in case of concave utility function). The bandwidth allocation problem described here is sort of a "multi-dimensional" fractional knapsack problem where the function relating the weight of the item and the worth of the item can be a linear, step, or concave function. Hence, the bandwidth allocation problem described here is more complex than the fractional knapsack problem.

3.5. Summary

The two types of scheduling methods, off-line and on-line scheduling were briefly discussed in this section. The bandwidth allocation problem considered in this research is an immediate on-line scheduling problem. In this section, the intractability of this problem has been demonstrated. The infeasibility of an exhaustive search solution and the need for a heuristic has also been presented. Many heuristics have been presented in the literature that are either applicable only for a concave continuously differentiable type utility function, or only consider preemption and not degradation. A summary of some such related work is presented in the next section.

4. RELATED WORK

To the best of the author's knowledge, the dynamic bandwidth allocation problem considering requests with step, concave, or linear utility functions has not been addressed in the literature. The research here also differs from the related work in the ways discussed in this section. The issue of non-preemptive (non-degrading) on-line bandwidth allocation (also referred to as call control) has been addressed in [AwA93, AwB94]. Our research focuses in the use of preemption and degradation for the immediate on-line scheduling problem.

The problem described in [Kel97] is similar to the problem that this research attempts to solve. In [Kel97], the requests are assumed to have utility functions that are strictly concave, i.e., the utility functions are continuous and differentiable. The utility of a request (which is a function of the amount of bandwidth received) in [Kel97] corresponds to the worth of a request in this research. The goal of [Kel97] is to maximize the sum of the utilities of all the requests, such that the total bandwidth allocated to the requests does not exceed the link bandwidth. Because the utility functions of the requests in [Kel97] are strictly concave and differentiable, a theoretical solution using Lagrangian methods is proposed. In the current Internet, there may be many requests that do not have a strictly concave utility function. For example, requests generated by real-time applications such as audio and video may have a step utility function. Our research considers requests having three different types of utility functions: step, concave, and linear. Most of the requests in the current Internet have a utility function that is one of these three types [She95]. In [Kel97], a one-link network model is assumed, i.e., degradations in the bandwidth allocated to the request at are considered at one link. For each request there are two bottleneck links in the network model assumed in our research (the ingress and egress links in Figure 2.2). Hence, this research takes into consideration the case where the request's bandwidth may be degraded at both the ingress and the egress links.

A decentralized market based approach for optimal resource **allocation** is described in [ThTOOa]. The market-based approach offers an alternative to the policy-based approach, where requests are admitted based on the current willingness of the user to pay for the reservation of resources for the request. The market-based approach follows directly **from** research in the field of economics, where similar problems exist when equilibrium needs to be achieved between high demand and low supply. In [ThTOOa], as in our research, the **users'** preferences are summarized by means of their utility functions. The objective of the resource allocation problem in [ThTOOa] is to determine the amount of resources to be allocated to requests such that the sum of the users' utilities is maximized. The market-based approach in [ThTOOa] is a decentralized approach where the users can dynamically change the amount they are willing to pay for the resources requested. That is, it can be thought of as the priority levels of the requests can change. But in our research, the priority levels of the requests are fixed and do not change. Because the uses can dynamically change the amount they are willing to pay, the users' requests may be degraded arbitrarily without following any utility function per se. In our research the users' bandwidth is degraded considering the utility function of the users' request. For example, if a request has a step utility **function,** the request is not degraded by a small amount, it is either preempted or not degraded **at** all. In our research the user can even **specify** a minimum bandwidth requirement. If the users' request is allocated bandwidth less than the minimum bandwidth specified, the utility is **zero.**

In **[BaM98],** the problem of dynamic bandwidth allocation is considered by assuming that every request will have a delay requirement rather than a bandwidth requirement. The objective of **[BaM98]** is to minimize the number of bandwidth allocation changes while **satisfying** the delay requirements (there are no priorities). Heuristics for dynamic bandwidth allocation for the single-source single-destination case and the multiple- source multiple-destination case are presented in **[BaM98].** In our research, the requests are assumed to have a bandwidth requirement and not a delay requirement, and the goal is to satisfy the bandwidth requirements of the requests while maximizing the total worth of all the requests that have arrived in a fixed interval of time.

A class of resource allocation algorithms for scheduling requests to achieve a high aggregated value (i.e., utility) within a distributed network infrastructure is described in [PiWOO]. The utility of a request in [PiWOO] corresponds to the worth of a request in this research. In [PiWOO], a batch of requests, i.e., batch on-line scheduling (as discussed in the Subsection 3.2) is considered, as opposed to the immediate on-line scheduling problem considered in our research. The three heuristics described in [PiWOO] to solve the resource allocation problem are as follows.

1. The baseline no scheduling heuristic, where no scheduling is done and the request is started at the earliest possible start time when enough bandwidth is available.
2. The greedy heuristic, where the request that yields the maximum utility is scheduled first and so on, until the total bandwidth of the link is allocated.
3. The maximum ratio heuristic, where the request that has the maximum utility per bandwidth ratio is scheduled first and so on, until the total bandwidth of the link is allocated.

The two types of requests considered in [PiWOO] are as follows.

1. Bandwidth based requests, where the requests have an earliest start time and latest end time and a firm time duration for which the bandwidth is required. The start time and end time specified are not firm, i.e., the request can start any time after the earliest start time and end before the latest end time.
2. Volume based requests, where the requests specify a bandwidth requirement with an earliest start time and latest end time, but no time duration is specified.

Our research only considers bandwidth type of requests with a firm start and end time (and hence the time duration for which the bandwidth is required is also firm). In contrast to the model for our research, in [PiWOO], the bandwidth used by a request cannot vary with time, and once a request begins transmission it cannot be preempted or degraded.

The issue of dynamic bandwidth allocation for multimedia applications is discussed in [ReR98]. It is argued in [ReR98] that the requests generated by multimedia applications would dynamically demand different bandwidths during a session and the network should have the capability to dynamically reallocate the bandwidth to these requests. The requests in [ReR98] are assumed to have a satisfaction profile that expresses the satisfaction of the user with the bandwidth currently allocated to the user's request. The network dynamically adjusts the bandwidth allocated to the requests based on the bandwidth requirements of the requests and the satisfaction profiles of the requests. A concept called the application's softness that describes the application's tolerance to degradation in bandwidth allocated to its request and sensitivity to delay experienced by its request is presented in [ReR98]. The softness of the application is considered while deciding how much bandwidth needs to be allocated to the application during the length of the session. *Our* research incorporates the softness of the application (i.e., user) in the utility function of the request generated by the user. If the request has a concave utility function then it is tolerant to degradation in bandwidth during the session, but if its utility function is a step function then it is not tolerant to degradation in bandwidth during the session. The worth of a request considered in this research, corresponds to the satisfaction profile of the user's request in [ReR98], because it indicates how much the user is willing to pay for a certain amount of bandwidth. While the goal of [ReR98] is to design a framework that is capable of dynamically allocating bandwidth, the goal of our research is to dynamically allocate bandwidth to the different requests such that the total worth of all the requests satisfied in a fixed interval of time is maximized.

In [FuR97], an on-line Dynamic Search Algorithm (DSA) that dynamically adjusts the resource allocation based on measured QoS parameters such as bandwidth and loss rate is presented. The QoS parameter considered in [FuR97] is the cell loss probability (CLP) of a request. The DSA dynamically adjusts the bandwidth allocated to the requests to satisfy the desired CLP of the request. The goal of the DSA is to adjust the bandwidth so as to provide each request its desired CLP, with the minimum number of bandwidth allocation changes. DSA renegotiates the bandwidth periodically so as to minimize the number of changes in allocation. *Our* research differs from [FuR97] in that the bandwidth of a request is considered as the QoS parameter. In our research, the bandwidth is dynamically adjusted whenever the session of the new request overlaps with the session of an existing request. Furthermore our performance measure is worth (defined in Subsection 2.5) and the number of changes made to the bandwidth allocation is only a secondary concern.

A bandwidth allocation method for elastic traffic is presented in [Low00]. Elastic traffic is defined as traffic that can tolerate some degradations in bandwidth, i.e., the utility function of the traffic is a strictly concave function [Kel97, Low00, She95]. In [Low00], the users are allocated some fixed minimum bandwidth and a random extra amount of bandwidth. The allocations and the prices are adjusted to adapt to resource availability and user demands. Equilibrium is achieved when all the users optimize their worth and demand equals supply for non-free resources such as link bandwidth. The goal is to converge to this equilibrium, and the method proposed is similar to the one proposed in [ThT00a] (described above). *Our* research does not divide the bandwidth allocated into fixed and variable bandwidth, rather it dynamically allocates bandwidths to the requests such that the amount of bandwidth allocated is at least the minimum bandwidth required. In [Low00], the users can change the amount of money paid, i.e., the worth, at their own discretion, while in our research, the network changes the amount of bandwidth it allocates to different users based on the worth (fixed priority level and fixed utility function) of the user's request and the available bandwidth.

A bandwidth allocation scheme with preemption is described in [BaC99]. The scheme in [BaC99] proposes that to decide which requests to reject/preempt, the duration of the request and the time for which the request has been in session should be considered, completely ignoring the bandwidth requirement of the request. In particular, a request with a very large bandwidth requirement may be preempted to accommodate a request with a longer duration and a smaller bandwidth requirement. The research in [BaC99] presents different algorithms such as the left-right algorithm that implements the compromise between the need to hold on to requests that have been running for the longest amount of time (thus, capitalizing on the work done) and the need to hold on to requests that will run for the longest time in the future (thus, guaranteeing future work). The algorithms presented in [BaC99] though surprisingly simple seem to ac'hieve good results. Our research differs from [BaC99] in that it allows for degradation of' bandwidth allocated to requests as well as preemption of requests. Thus, the heuristic presented in our research has to explore more choices when deciding whether to admit/reject the request. Furthermore, in [BaC99], requests do not have priority levels.

A brief overview of some of the literature related to this work was presented in this section. The scheduling heuristic developed in this research is presented in the next section.

5. SCHEDULING HEURISTIC

5.1. Overview

In this research, the network is simulated using a resource allocation table and this is explained in detail in Subsection 5.2. In Subsection 5.3, the marginal worth, **i.e.,** the change in the worth of the concave or linear request due to a unit change in the bandwidth allocated to the request, is described. An example of overlapping requests is illustrated in Subsection 5.4. In Subsection 5.5 the scheduling heuristic: is discussed. The design of the scheduling heuristic is explained step by step. The issues encountered while designing the heuristic and the corresponding design decisions made are stated. The heuristic is summarized at the end of Subsection 5.5, while the detailed pseudo-code is presented in Appendices A, B, and C.

5.2. Network Simulator

In this research, table is used to record information about the requests that were admitted. This table, called a resource allocation table (RAT) records information such as the ingress and egress links utilized by the request, the bandwidth allocated to the request, the start time, and the end time of the request. The RAT essentially simulates a network. The available bandwidth at each link and at every instant of time can be determined from the RAT. When a new request arrives at an ingress node, the RAT is examined to determine whether there is enough bandwidth available to satisfy the new request at every time instant of its session, both at its ingress and egress links. If there is sufficient bandwidth available at every instant of time during the new request's session, then the information about the request is added to the RAT and the request is considered as satisfied. If sufficient bandwidth is not available then the scheduling heuristic is invoked. The heuristic performs computations to determine which bandwidths of the existing requests should be degraded and by what amount, if at all. If the new request can be satisfied with an increase in the total worth of satisfied requests, then the information about the new request is recorded in the RAT.

5.3. Marginal Worth

As mentioned earlier, the marginal worth is the change in the worth of a concave or linear request due to a unit change in the bandwidth allocated to the request. It is calculated as follows. The derivative of the utility function of the request is called the marginal utility of the request. The marginal utility is essentially the slope of the utility function of the request. The marginal utility indicates the change in the utility of the request due to a unit change in the bandwidth allocated to the request. The worth of a request is the product of the weighted priority and the utility of the request (Equation 2.2). Thus, the derivative of the worth of the request is the product of the weighted priority and the marginal utility of the request. The derivative of the worth of the request is the marginal worth of the request.

The marginal worth of the request may change depending upon the amount of bandwidth received in case of a request with a concave utility function. In a concave utility function, the utility obtained per bandwidth depends on the amount of bandwidth received. Because the worth depends upon the utility function, the worth changes as the amount of bandwidth received changes. In Figure 5.1, the different marginal worths of a request depending upon the bandwidth received are shown. The marginal worth of the request with a concave utility function increases as the bandwidth allocated to the request is decreased (not decreased below mb_k).

For a linear utility function, marginal utility is just the slope when the bandwidth is between mb_k and rb_k. When the bandwidth is below mb_k, then the utility is zero and hence the marginal utility is zero.

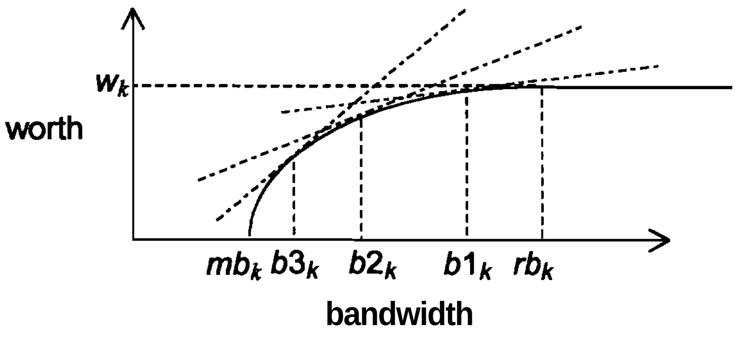

Figure 5.1. Different marginal worths of a request with a concave utility function.

The utility functions of the requests can also be non-differentiable functions such as step functions. In this research, the marginal worth of a request with a step utility function is considered to be the ratio of the worth of the request (w_k) and the desired

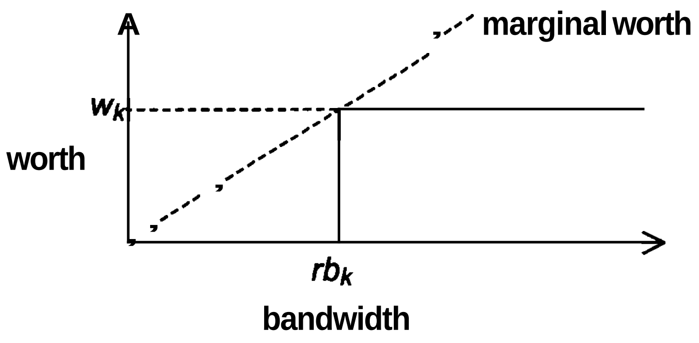

Figure 5.2. Marginal worth of a request with a step utility function.

bandwidth (rb_k) of the request as illustrated in Figure. 5.2. For a step function, marginal worth does not represent the change in worth for a unit loss of bandwidth. The way a step function's marginal worth is used is explained in Subsection 5.5.

5.4. Overlapping Requests

A request r_1 overlaps with request r_2 if: r_1 ends during r_2's session (i.e., from s_2 to e_2), r_1 starts during r_2's session, or r_1 starts and ends during r_2's session. These three cases are illustrated in Figure 5.3. Consider a new request that arrives to be scheduled.

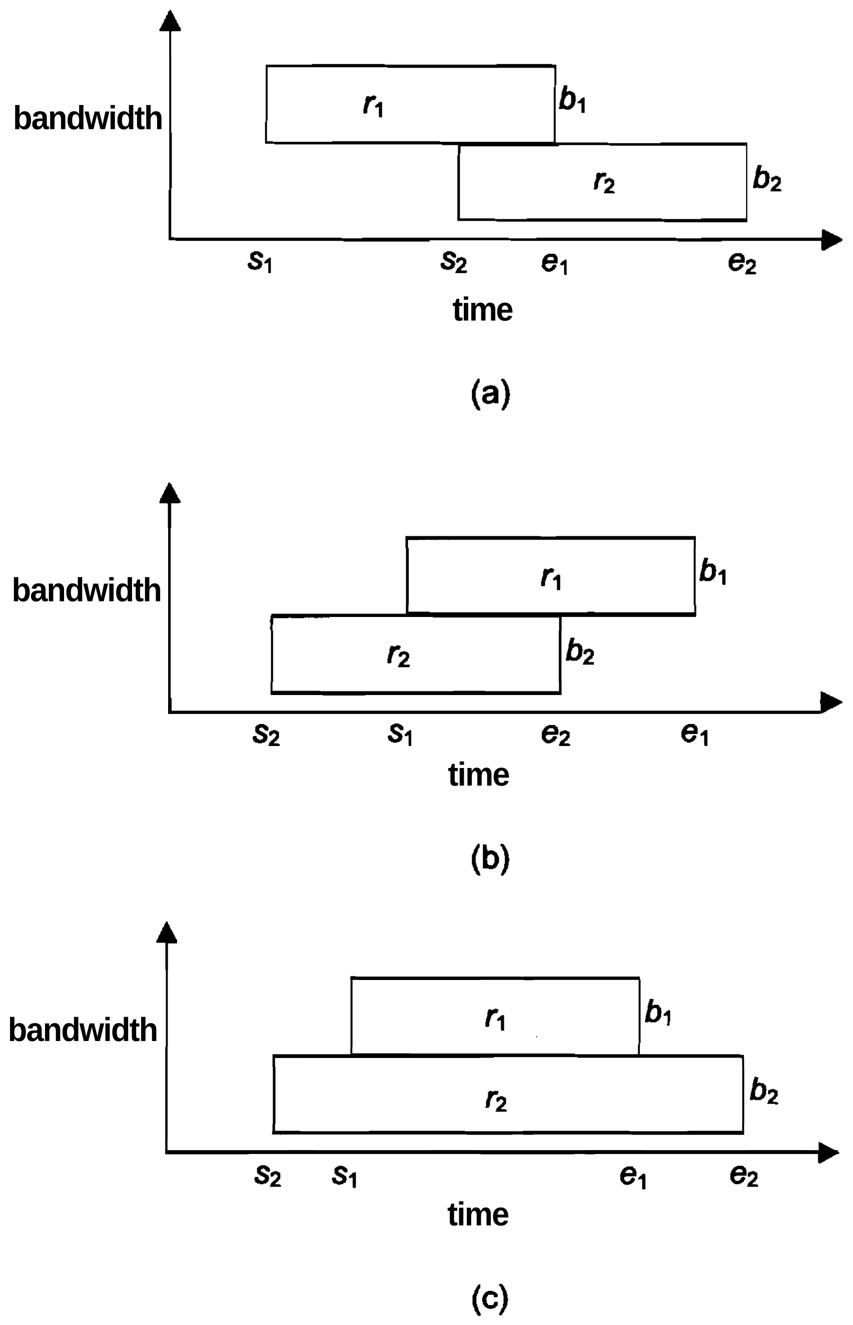

Figure 5.3. Requests r_1 and r_2 overlapping in time. (a) r_1 ends during r_2's session. (b) r_1 starts during r_2's session. (c) r_1 starts and ends during r_2's session.

The set of requests that overlap with the new request is the set of conflicting requests (and includes the new request). If a request is not in the set of conflicting requests, then degrading/preempting that request would not free up bandwidth for the new request. This is because, if two requests do not overlap in time (as shown in Figure 5.3), then these two requests do not compete for the same bandwidth. Hence, the set of conflicting requests is considered when deciding which requests should be degraded/preempted to make bandwidth available for the new request.

In Figure 5.4, the set of conflicting requests for the new request r_4 consists of request r_1 at time s_4, requests r_1 and r_3 at time s_3, and requests r_2 and r_3 at time s_2. Hence the set of conflicting requests is different at times s_4, s_3, and s_2 and may change at every instant of time, depending upon how many existing requests (i.e., already scheduled requests) overlap with the new request.

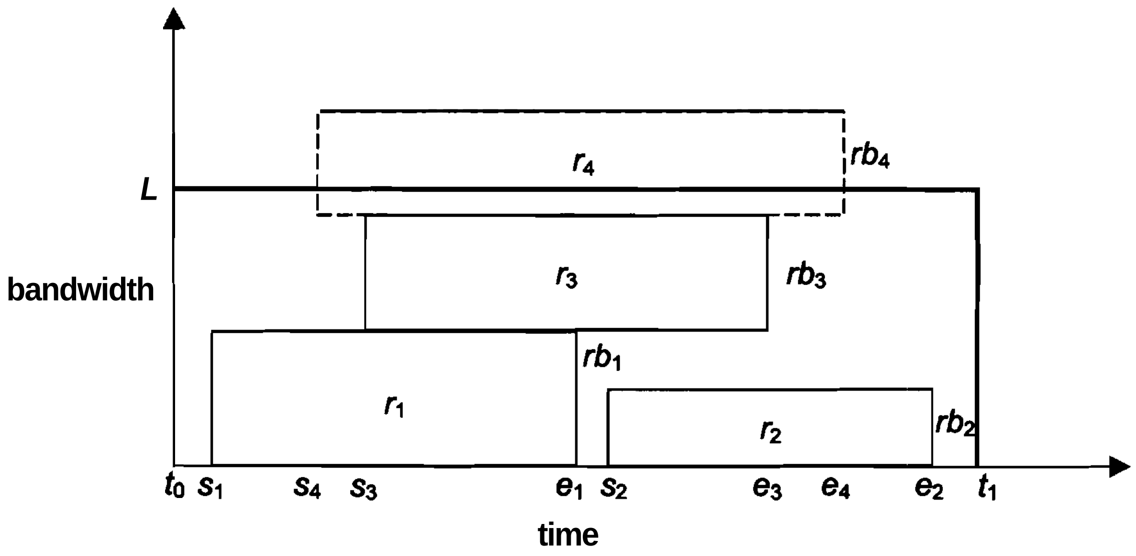

Figure 5.4. A snapshot of the requests satisfied at a link from time t_0 to t_1. Request r_4 is the new request being considered for scheduling. The rectangle indicates the request; the width of the rectangle is the duration of the request and the height of the rectangle is the bandwidth required by the request. The height of the outer rectangle is the link bandwidth L. The start and the end times of the requests are indicated on the X-axis.

5.5. Immediate On-line Scheduling Heuristic

In this research, the requests are assumed to have some time difference between their arrival time and their start time. This time difference is called the lead. time. The lead time is introduced because the heuristic takes a finite amount of time to schedule a request. Hence, if the heuristic cannot schedule the request before its start time (ie., the time difference between the arrival time and start time of the request is not sufficient for the heuristic to schedule the request) the request should be rejected immediately and not considered for scheduling.

When a new request (e.g., r_4 in Figure 5.4) is considered for scheduling (refer to the main module of the pseudo-code in Appendix A), the difference between the start time and the current time is compared with the lead time. If the difference is less than the lead time, the request is rejected. Otherwise, the RAT is queried to determine whether the request can be satisfied at every time instant of the request's session. If the request can be satisfied with its desired bandwidth at both its ingress and egress links, then the information about the request is recorded in the RAT and the request is admitted. If the request cannot be satisfied, the scheduling heuristic is invoked.

The goal of the heuristic is to allocate the bandwidth to the different requests such that the total worth W is maximized. The methodology of the heuristic is to degrade/preempt the bandwidth allocated to existing requests, so that the freed up bandwidth (due to degradation/preemption of existing requests) can be allocated to the new request, such that the total worth W is maximized. The problem is to determine which existing requests' bandwidth should be degraded/preempted, and if degraded, by what amount, and what bandwidth should be allocated to the new request:. The new request may not be allocated the full bandwidth desired, depending upon its relative worth as compared to the other requests. For example, a maximum total worth of satisfied requests might be obtained when the bandwidths allocated to the existing requests are not degraded but the new request is allocated bandwidth less than its desired bandwidth. This may be because the marginal worth of the new request is less than the marginal worth of the existing requests.

The main idea of the heuristic is that a request whose marginal worth is the smallest should be the first request whose bandwidth is **degraded/preempted** to accommodate the new request. Then the bandwidth of the request with the next higher marginal worth is degraded, and so on, until the new request can be satisfied either with desired or degraded bandwidth. Requests are considered in increasing order of marginal worth because if the bandwidth of the request whose marginal worth is the smallest is degraded by some amount, then the change in total worth for that amount of degradation in bandwidth is the least. This is because if the bandwidth of a request, whose marginal worth is not the smallest, is degraded by some amount, the resulting change in the total worth of satisfied requests would obviously be higher as compared to the change in total worth due to degradation of bandwidth of a request with the least marginal worth. Thus, at any instant of time, the bandwidth of the request whose marginal worth is the least is degraded. This is the crux of the heuristic. Only the bandwidths of the requests conflicting with the new request are **degraded/preempted** to accommodate the new request, as explained in Subsection 5.4.

At every instant of time, the set of conflicting requests is determined (refer to the **schedule** module of the pseudo-code in Appendix A). This set of requests conflicting with the new request, and the new request, are assigned to an array ***R*** **size** ***n*****.** The marginal worth of every request in this set of conflicting requests (array R) is calculated. This array is sorted in the decreasing order of marginal worth **(i.e.,** the request in. $R[n]$ has the least marginal worth) and the bandwidth of the request with the smallest marginal worth is degraded.

If the request **has** a concave utility function, the marginal worth of the request increases as the bandwidth allocated to the request is decreased (not decreased below the minimum bandwidth of the request mb_k). Hence, the bandwidth of' a request with a concave utility function is degraded until the point when the marginal worth of that request is no longer the least (explained later in this subsection). This request is then inserted in the array in the correct position in the order of decreasing marginal worth. The request whose marginal worth is now the least should be degraded until its marginal worth is no longer the least. This continues until the bandwidth released by the degradations of the bandwidth's of existing requests is sufficient to **satisfy** the new request with desired or degraded bandwidth at that instant of time.

Because the new request is included in the array *R,* the new request is also considered for degradationlpreemption. That is if the marginal worth of the new request is the least at any point in time, then its bandwidth should be degraded. Thus, the new request would be allocated the degraded bandwidth and not the **full** bandwidth desired. In case the new request is preempted, then that indicates that it cannot be satisfied.

For example, consider a set of conflicting requests containing r_1, r2, and the new request. Assume r_1 and r_2 have concave and linear utility functions, respectively, as illustrated in Figure 5.5. Further assume that the marginal worth of the new request is greater than the marginal **worths** of r_1 and r_2. **This** set of conflicting requests is assigned to array R. R is sorted in decreasing order of marginal worth as explained earlier. When r_1 is allocated bandwidth rb_1, the marginal worth of request r_1 is the least. **Hence,** the bandwidth of request r_1 would be degraded first. The bandwidth of request r_1 is degraded repeatedly until its marginal worth is no longer the least. Once the bandwidth allocated to r_1 has been degraded to $b1_1$, the marginal worth of r_2 becomes slightly greater than marginal worth of request r_1. The marginal worth of request r_2 is now the least and hence r_2 should now be degraded. The request r_1 is inserted in the array R in the correct position in the order of decreasing marginal worth. This continues until the bandwidth **freed** up (due to degradations of existing requests) is sufficient to satisfy the new request with desired or degraded bandwidth. The bandwidth $b1_1$ at which the marginal worth of request r_1 becomes less than the marginal worth of request r_2 can be exactly determined because the marginal worths of the requests can be pre-computed when the requests arrive at the ingress node. Thus, the amounts by which the bandwidth of requests r_1 and r_2 need to be degraded may be determined exactly.

The set of conflicting requests may change at every time instant because of some requests ending, and some other requests starting during the new request's session. For example, in Figure 5.4, at time s_4, the set of requests conflicting with the new request r_4 consists of request r_1 only. At time s_3, the set of conflicting requests consists of requests

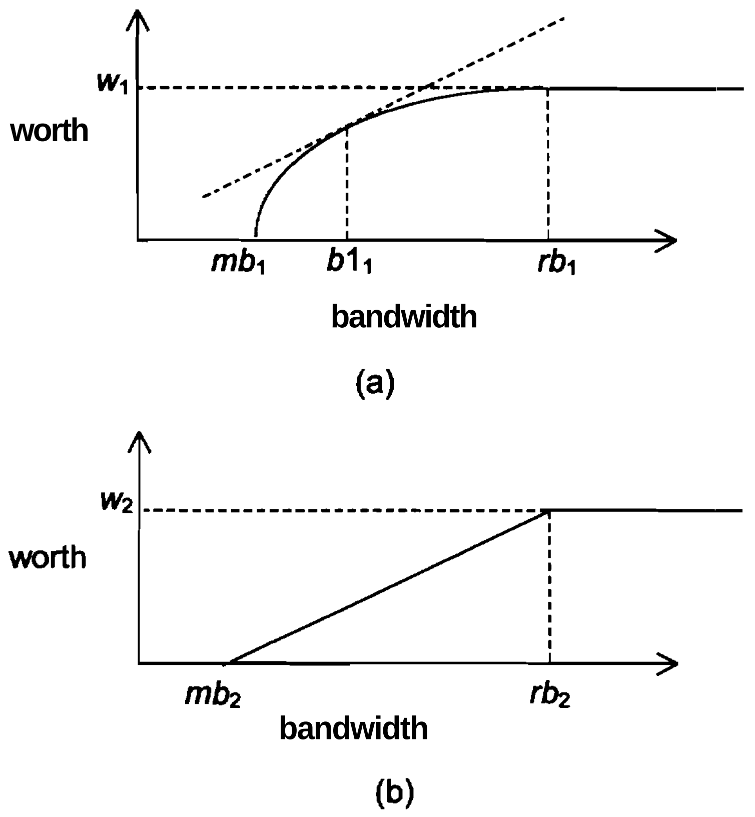

Figure 5.5. Marginal **worths** of (a) request r_1 with a concave utility function, and (b) request r_2 with a linear utility **function.**

r_1 and r_3. At time s_2, the set consists of requests r_2 and r_3. Thus, the above process may need to be repeated at every time instant of the request's session. But if the above process were repeated at every time instant of the request then the heuristic would take too long.

The heuristic is modified as follows (refer to the **find–next–event** module of the pseudo-code in Appendix A). The set of conflicting requests changes only when any request in that set ends, or some other existing request (not in the! set) begins. The heuristic needs to be executed only when the set of conflicting requests changes. An event is defined as the time instant when the set of conflicting requests changes. Thus, the event would be the next time instant after the earliest end time of the requests in the set or the earliest start time of some other request (not in the set), whichever is earlier. In Figure 5.4, the events would be at times s_4, the next time instant **after** e_1, s_2, and the next time instant after e3. The heuristic is executed at every event during the new request's session. For the Figure 5.4 example, the heuristic would be executed at the times mentioned above. When a request is degraded, the bandwidth allocated to the request is degraded for the time interval fiom the current event to the next event.

The heuristic would have to be executed at both the ingress link and the egress link because the set of conflicting requests is different for the links. But, bandwidth allocated to the request should be identical at both the links, i.e., if the request is allocated a certain amount of bandwidth at the ingress link, it should be allocated the same bandwidth at the egress link too. The heuristic is executed to calculate the amount the degradations to the bandwidth of the new request at the ingress link. The new request, with the degraded amount of bandwidth is then considered at the egress link, to determine whether the request can be satisfied with its already degraded bandwidth, or further degradations are needed. If the request is further degraded at the egress link, then the degradations are reflected back in the bandwidth allocated to the request at the ingress link. If the bandwidth allocated to a request was degraded at the new request's ingress link due to the new request, then the bandwidth allocated to the request is degraded at its other (ingress or egress) link too, i.e., the bandwidth of the request is degraded at both its links. This is done for all the requests that were degraded due to the new request.

Whenever a request is preempted, if the bandwidth freed up is more than the needed bandwidth, excess bandwidth is available. The excess bandwidth is redistributed to the other requests in decreasing order of marginal worth, starting with $R[1]$. If a request has a concave utility function, its marginal worth decreases as the bandwidth allocated to it increases. Hence, the request with a concave utility function is allocated bandwidth until its marginal worth is no longer the largest or is new current bandwidth equals its desired bandwidth. It is then reinserted in R in the correct order based on its marginal worth. If a request has a linear utility function, it is allocated bandwidth until its new current bandwidth equals its desired bandwidth. If a request has a step utility function, it is not allocated any bandwidth (because its current bandwidth equals its desired bandwidth). This is continued until all the excess bandwidth has been redistributed or all the other requests are at their desired bandwidth. When calculating the worth due to redistribution of excess bandwidth, the collective increase in worth due to the increase in the bandwidths allocated to requests is calculated.

Once it is determined that the request can be satisfied at the ingress and egress links, the change in total worth due to degradation of bandwidths of existing requests and addition of worth due to redistribution of excess bandwidth and the new request is calculated. If this change in the total worth is more than zero, then there is an increase in worth obtained by **satisfying** the new request. The new request is admitted and the degradations/preemptions calculated for existing requests are implemented. If the change in the total worth is less than zero, then there is no increase in worth obtained by satisfying the new request. Hence, the new request is rejected and the calculated degradations to bandwidths of existing requests are ignored (not implemented). The bandwidths of the requests that were degraded by the new request are restored and also the requests that were preempted by the new requests are restored (allocated their original bandwidth).

As suggested by the example above, the requests with concave, linear, and step utility functions are degraded differently (refer to the **degrade** module of the pseudo-code in Appendix A). The bandwidth allocated to a request with a concave utility function is degraded in steps (of unit size), because the marginal worth of the request changes with every unit change in the bandwidth allocated (refer to the **degrade–concave** module of the pseudo-code in Appendix A). The **bandwidth** allocated to the request is degraded in steps until the marginal worth of the request is no longer the least (as explained in the example above). The concave request is then reinserted in the correct position in R in the order of decreasing marginal worth.

The bandwidth allocated to a request with a linear utility function is degraded as follows (refer to **degrade–linear** module of the pseudo-code in Appendix A).

1. If the amount of bandwidth needed (amount of degradation) is less than the difference between the current bandwidth allocated to the request and the minimum bandwidth of the request, the request with the linear utility function is degraded by the amount of bandwidth needed.
2. If the amount of bandwidth needed is greater than the difference between the current bandwidth allocated to the request and the minimum bandwidth of the request, the request with the linear utility function is preempted.

If the bandwidth allocated to a request with a step utility function is degraded by a small amount, then the utility of the request would be zero and hence the worth is zero (refer to Figure 5.2). Hence, the bandwidth allocated to a request with a step utility function cannot be degraded by a small amount of bandwidth; it should either be preempted or not degraded at all.

Preemption of a request with a step utility function may not be desirable in some cases. For example, assume that for a new request to be satisfied, the amount by which an existing request needs to be degraded is A. Further, assume that the marginal worth of a request with a step function is the least and hence the request may have to be preempted to satisfy the new request. Thus, $R[n]$ contains the request with a step utility fiinction and the requests higher up in the list may have step, linear, or concave utility functions.

If A is greater than the bandwidth rb_k of the request with a step utility function, then the request with a step utility function can be preempted (refer to lines 6-12 of **degrade** module of the pseudo-code in Appendix A). The reason is as follows. The bandwidth needed is more than the bandwidth allocated to the request with a step utility function. The request has the least marginal worth and hence it is being considered for degradation/preemption. Because the bandwidth needed is more than the bandwidth allocated to the request (with the step utility function), it does not matter whether its utility function is step or not, because, its marginal worth does not change (as the marginal worth of a request with a concave utility function changes). For all practical purposes, the request can be thought of as a request with a linear utility function. If the bandwidth needed is more than the bandwidth currently allocated to a request with a linear utility function, the request is preempted. Similarly, if the bandwidth needed is more than the bandwidth rb_k of a request with a step utility function, the request can be preempted.

If A is less than rb_k, then depending upon the ratio of A and rb_k, a decision needs to be made whether the request k with a step utility function should be preempted or the request with the next higher marginal worth should be considered. The reason is as follows.

If A is large (e.g., *90* percent of rb_k), then it may be **beneficial** to preempt the request with a step utility function. Although there will be some unused bandwidth (ten percent) because of the preemption, it may be possible to reallocate this unused bandwidth to other requests that have been previously degraded.

In contrast, if A is very small (e.g., ten percent of rb_k), then it may not be beneficial to preempt the request with the step utility function. This is because it may not be possible to reallocate all of the unused bandwidth to the other requests. Also, the bandwidth needed, A, may be obtainable **from** other requests (e.g., the request with the next higher marginal worth), by losing less worth than the worth lost by preempting the request with a step utility function.

For example, **if the** request $R[n-1]$ has a concave or a linear utility function, and if the loss of worth by degrading $R[n-1]$ by A is less than the worth lost by preempting the request with the step utility function (i.e., $R[n]$) then it may be beneficial to degrade $R[n-1]$ by A rather than preempting $R[n]$. Alternatively, if $R[n-1]$ has a step utility function, the bandwidth of $R[n-1]$ may be smaller than the bandwidth of $R[n]$, and preempting $R[n-1]$ may result in a smaller loss of worth than preempting $R[n]$. Thus, **degrading/preempting** the request with the next higher marginal worth may result is a smaller loss of worth than preempting $R[n]$ and hence $R[n-1]$ may be considered. The cases discussed above are some of the possibilities that exist. Three variations in the heuristic have been developed that consider the different possibilities.

In the first variation for degrading a step **function** called the *50%* variation (module **degrade–step–50** of the pseudo code in Appendix A), the ratio of A and the bandwidth of $R[n]$ is compared to *0.5.* If the bandwidth needed is more than *50%* of the bandwidth of the request $R[n]$, the request in $R[n]$ is preempted. If the bandwidth needed is less than *50%* of the bandwidth of $R[n]$, the request with the next higher marginal worth is considered, i.e., $R[n-1]$. If the loss of worth obtained by **degrading/preempting** the request $R[n-1]$ is more than the worth of $R[n]$, the request in $R[n]$ should be preempted. If the loss of worth obtained by **degrading/preempting** $R[n-1]$ is less than the worth of $R[n]$, then the request in $R[n]$ is not preempted, and the request $R[n-1]$ is considered for **degradation/preemption.**

The reason why only $R[n-1]$ is considered is as follows. If the heuristic decided not to preempt $R[n]$, it would have to degradelpreempt some of the requests $R[0]$ to $R[n-1]$ in the list to **satisfy** the new request. The marginal worth of $R[n-1]$ is the least as compared to the marginal **worths** of requests $R[0]$ to $R[n-1]$ (because the list is sorted in decreasing order of marginal worth). Degrading $R[n-1]$ by a unit amount of bandwidth would result in the least loss of worth as compared to the loss of worth by degrading other requests in the list ($R[0]$ to $R[n-2]$) by a unit amount of bandwidth, but possibly not when degrading by A. Hence, the loss of worth obtained by **degrading/preempting** $R[n-1]$ by the amount of bandwidth needed is only an estimate of the amount of worth that will actually be lost if $R[n]$ is not preempted and requests $R[0]$ to $R[n-1]$ are considered for degradation. This estimate of the loss of worth is used for comparison with the worth of the step **function** to make a decision of whether to preempt the request in $R[n]$, or degradelpreempt the request in $R[n-1]$. This is a heuristic approach to estimating the loss of worth. An exhaustive search may be employed to find the actual loss of worth, but the time complexity of an exhaustive search is too high (described in Subsection 3.3). Future work will attempt to determine better approaches to estimate the loss of worth.

When calculating the loss of worth due to **degradation/preemption** of $R[n-1]$, it is implicitly assumed that the all the bandwidth required is obtained **from** $R[n-1]$. **But** when the requests are actually degraded, all the bandwidth need not be obtained by degrading $R[n-1]$ alone; other requests may be degraded too. For example, assume that the request in $R[n-1]$ has a concave utility **function.** It will be degraded in steps of unit **size** (refer to **degrade–concave** module of the pseudo-code in Appendix A), until its marginal worth is no longer the least. That is, all the bandwidth needed is not obtained by degrading $R[n-1]$ alone, but other requests ($R[n-2]$ to $R[1]$) may be **degraded/preempted** too. In this section, the expression $\underline{\text{degrade}\ R[n-1 \rightarrow 1]}$ is used to indicate that requests; $R[n-1]$ to $R[1]$ are considered for degradation in decreasing order of marginal worth.

In summary, the loss of worth is calculated based only on $R[n-1]$ when making a decision whether to preempt request $R[n]$ or not. Once it is decided that the request $R[n]$ is not to be preempted, the request $R[n-1]$ is degraded as it normally would be. Thus, the actual loss of worth may be less than the loss of worth estimated by degrading/preempting $R[n-1]$ alone.

The bandwidth needed may be more than the bandwidth of request $R[n-1]$, and hence if it is decided not to preempt $R[n]$, request $R[n-1]$ will be preempted and other requests will have to be preempted. In this case the actual loss of worth incurred due to preemption of $R[n-1]$ and degradation/preemption of other requests $R[0]$ to $R[n-2]$ is more than the loss of worth as estimated above. If the actual loss of worth is to be calculated, then an exhaustive search may have to be employed which takes a huge amount of time. Hence a trade-off between speed and accuracy is achieved by considering only the $R[n-1]$th request for degradation/preemption. In the 25/75 variation, two thresholds are considered and both $R[n-1]$ and $R[n-2]$ are considered for degradation/preemption.

In the second variation for degrading a step function called 25/75 (module **degrade–step–25/75** of the pseudo-code in Appendix B), the ratio of A and rb_k is compared with two thresholds, 0.25 and 0.75. The following three cases arise.

1. If A is more than 75% of bandwidth of $R[n]$, the request in $R[n]$ is preempted. This is because a large amount of bandwidth is needed as signified by the fact that the bandwidth needed is more than 75% of the bandwidth of the request in $R[n]$. Because of the large amount of bandwidth needed there is a low probability that the loss of worth due to the degradation/preemption of requests besides $R[n]$ is less than worth of $R[n]$. This is because it may require preempting requests higher up in the list (for example $R[n-1]$ and $R[n-2]$) to obtain the large amount of bandwidth needed. Thus, even though the bandwidth obtained by preemption is more than the bandwidth needed, the loss of worth may be still less as compared to degrading/preempting requests higher up in the list. Hence, in this case

the request in $R[n]$ is preempted. Furthermore, unused bandwidth may be reallocated to already scheduled requests with less than their requested bandwidths.

2. If A is less than 25% of bandwidth of $R[n]$, that indicates a small amount of bandwidth is needed. Hence the request with the next higher marginal worth is considered. Because the bandwidth needed is a small amount, there is a high probability that the loss of worth due to degradation of request $R[n-1]$ will be less than the worth of request $R[n]$. Hence, degrade $R[n-1 \rightarrow 1]$.
3. If the amount of bandwidth needed is between 25% and 7.5% of the bandwidth of $R[n]$, then the loss of worth by **degrading/preempting** the requests $R[n-1]$ and $R[n-2]$ is considered. If the loss of worth is higher than the worth of request $R[n]$, the request is $R[n]$ is preempted. But if'the loss of worth is less than the worth of request $R[n]$, then that indicates that it would be better to degradelpreempt the requests $R[n-1]$ and $R[n-2]$ (if need be) instead of preempting request $R[n]$. Thus, degrade $R[n-1 \rightarrow 1]$.

When calculating the loss of worth due to **degradation/preemption** of requests $R[n-1]$ and $R[n-2]$, a set of conflicting requests consisting of two elements $R[n-1]$ and $R[n-2]$ is generated. The marginal worths of the requests are calculated and the bandwidth allocated to the requests is degraded based on the marginal worths of the requests, just like it is done in a typical situation, but only two requests $R[n-1]$ and $R[n-2]$ are considered. Here again, only when making a decision whether to preempt request $R[n]$ or not, the requests $R[n-1]$ and $R[n-2]$ are **degraded/preempted** to calculate the loss of worth. Once it is decided that the request $R[n]$ is not to be preempted, the requests $R[n-1]$ and $R[n-2]$ are degraded as they normally would be.

If the amount of bandwidth needed is between 25% to 75% of the bandwidth of $R[n]$, then it indicates that a large amount of bandwidth is needed. The probability that this amount of bandwidth can be obtained by **degrading/preempting** $R[n-1]$ alone is low, compared to the A < 25% case. Hence, two requests $R[n-1]$ and $R[n-2]$ are considered instead of only considering $R[n-1]$ (as in the 25% case and the 50% variation). Future work may consider more than two requests.

The bandwidth needed may be more than the bandwidths of $R[n-1]$ and $R[n-2]$. In this case too (as explained earlier for the 50% variation), the loss of worth as estimated above may be less than the actual loss of worth. Here again a trade-off is achieved between speed and accuracy.

The 25/75 variation attempts to obtain a high W value than the 50% variation by using two thresholds. However, on average, its execution time is larger. As will be shown in Subsection 8.2, its performance is comparable to that of the 50% variation.

The third variation for degrading a step function is called redistribute (module **degrade–step–redist** of the pseudo-code in Appendix C). Here, the worth obtained due to redistribution of excess bandwidth (i.e., bandwidth of $R[n]$ $-$ A) is considered. When a request is preempted and the bandwidth obtained due to preemption is more than the bandwidth needed, then excess amount of bandwidth can be reallocated to the other existing requests. Hence, the net loss of worth of the request $R[n]$ due to preemption is the worth of the request minus the worth obtained by reallocating the excess bandwidth to other requests. If the net loss of worth of $R[n]$ is less than or equal to zero, the request $R[n]$ is preempted. This is because the request $R[n]$ can be preempted and the excess bandwidth can be redistributed to other requests achieving an increase in worth. Otherwise the following cases arise.

1. If the bandwidth needed is less than the bandwidth of $R[n-1]$, the net loss of worth of $R[n]$ is compared to the loss of worth due to degradation of $R[n-1]$. If the request in $R[n-1]$ has a step utility function, the net loss of worth is considered as above. If the loss of worth by preempting $R[n]$ is less than the loss of worth due to degrading/preempting $R[n-1]$, then $R[n]$ is preempted. Otherwise, degrade $R[n-1 \rightarrow 1]$.
2. If the bandwidth needed is more than the bandwidth of $R[n-1]$, a heuristic similar to the 50% variation is employed. If the bandwidth needed is more than 50% of the bandwidth of $R[n]$, then request in $R[n]$ is preempted. Otherwise, if the loss of worth due to preemption of $R[n-1]$ is greater than or

equal to the net loss of worth of $R[n]$, then $R[n]$ is preempted. If the net loss of worth of $R[n]$ is more, then degrade $R[n-1 \rightarrow 1]$.

The redistribute variation takes a significant amount of time as compared to the 50% variation and the 25/75 variation. However, it attempts to incorporate into the decision process the impact of the reallocation of the excess bandwidth. A more detailed explanation of the heuristic and the variations is given in the pseudo-code in Appendices A, B, and C.

The heuristic and its variations proposed in this research use a greedy approach. In a greedy approach there is always a concern that the approach may lead to a local optimum (sum of worths being maximum at a single time instant) instead of a global optimum (total sum of worths being maximum over a given interval of time). Also, when making a decision whether to admit/reject a request, just comparing the new worth (after degradation of new and existing requests) and the old worth (before degradation of new and existing requests) may not yield good results. For example, assume a priority level four request is preempted to free up bandwidth for a priority level three request because the priority level three request gives more worth than the priority level four request. But the priority level three request may itself be preempted for a priority level one request that arrives later in time. The bandwidth freed up due to preemption of the priority level three request may be sufficient to satisfy both the priority level one requests and the priority level four request. Thus, it would have be better to reject the priority level three request, and admit the priority level one and priority level four request, if it is 'known that the priority level one request would arrive later. The scheduling heuristic is an immediate on-line scheduling heuristic and future arrival of requests is not known. Hence, situations like the one described above may occur.

The approach used in this research to deal with this issue is to introduce some randomness in the decision making. This random factor is called the globalization factor (GF). The GF is used as follows. A random number between 0 and 1 is generated. If the total worth of the requests after degradation of requests is more than the worth of the requests before the degradation, and the random number generated is more than the globalization factor, the new request is admitted and the degradations/preemptions calculated for existing requests are implemented. Otherwise the new request is rejected and the calculated degradations to the bandwidths of existing requests are ignored (not implemented). The values of globalization factor experimented with in this research were 5%, 10%, 15%, 20%, and 25%. The results of these experiments will be discussed in Section 8. The globalization factor introduces a random factor in the decision making and this random factor prevents decisions being made solely on the basis of difference in new and old worth. The globalization factor approach is only one approach that was experimented with in this report. There may be other approaches too and the future work will attempt to determine other approaches as a solution to the issue mentioned above.

The full heuristic is presented as follows. When a new request r_k amves at the node, the RAT is checked to determine whether the request can be satisfied at its ingress link. If the request can be satisfied, the information about the request is recorded and the request is admitted. If the request cannot be satisfied, the scheduling heuristic is invoked. The scheduling heuristic considers the request on a link by link basis. It first considers the ingress link. At the start time of the request r_k, a set of requests conflicting with r_k is determined. The new request is also added to this set. This set is sorted in decreasing order by the marginal worths of the requests. The request at the end of the set is the request with the least marginal worth. This request is degraded until its marginal worth is no longer the least. The heuristic handles requests with step utility functions with one of the three variations described above. The set is sorted again as before. This is continued until the new request can be satisfied at that instant of time. Satisfying a request at an instant of time means that the sum of the bandwidths of the requests in the set should not exceed the link bandwidth at that instant of time.

The next event, at which a request from the current set ends or a new request from among the existing requests begins, is determined. At the next event the scheduling heuristic is executed, to determine whether the new request can be satisfied at that instant too. This is continued until the end time of the new request is reached. If the request could be satisfied at every event, then the heuristic is executed at the new request's egress link. Any additional degradations to the new request at its egress link are reflected back at its ingress link too and the bandwidth finally allocated to the request is the identical at its ingress and at its egress links. The change in the total worth, due to degradations in bandwidths of existing requests and addition of the worth of the new request, is calculated. If the total change of worth is greater than zero, and a random number generated between 0 and 1 is greater than the globalization factor, the request is admitted and the degradations/preemptions calculated for existing requests are implemented. Otherwise the new request is rejected and the calculated degradations to the bandwidths of existing requests are ignored (not implemented). The detailed pseudo-code of the heuristic is presented in Appendices A, B, and C. The pseudo-code may be referred to for further details.

5.6. Summary

The concept of marginal worths and overlapping requests was explained in this section. The heuristic and its three different variations were discussed. A conceptual overview of the heuristic was described in this section and the detailed pseudo-code is presented in the Appendices A, B, and C. The concept of globalization factor and the reason why it was introduced has also been explained. The performance of the heuristic is compared to the complete sharing policy and the upper bounds described in the next section.

6. PERFORMANCE COMPARISONS

6.1. Overview

The heuristic and the different variations developed in this research were discussed in the last section. The **performance** of the heuristic is compared to a simple scheduling technique and three upper bounds presented in this **section**. As per Equation 2.4, the performance measure of the system is the *sum* of the worths (W) of the requests satisfied over a given interval of time. The interval of time, over which the **sum** of the worths of the requests is calculated, is called the <u>simulation time</u>. The concept of simulation time and the method for calculating the simulation time is explained in greater detail in the next section.

6.2. Simple Scheduling Technique

In this research, a simple scheduling technique is used to compare the performance of heuristic at the lower end and is based on the <u>complete **sharing** policy</u> **(<u>cs</u>)** **[BoM98].** In the complete sharing policy the scheduling heuristic is not invoked. When a new request arrives at an ingress node, RAT (resource allocation table, discussed in Section 5) is checked to determine whether there is enough bandwidth available to **satisfy** the new request's **full** bandwidth rb_k at its ingress and egress links. If there is sufficient bandwidth available, the new request is admitted and information about the request is stored in the RAT. If the request cannot be satisfied, the request is rejected. The sum of the worths of all the requests satisfied in this manner, over a given interval of time is calculated. This sum is compared with the sum of the worths of the requests satisfied by the heuristic in Section 8.

When calculating the sum of the worths of the requests satisfied over a given interval of time for the complete sharing policy, the requests are satisfied only with desired bandwidth and not degraded bandwidth. The utility functions of the requests are not considered and hence requests with concave, linear, and step utility functions are treated alike. The priority level of a request is not considered when making a decision whether to admit/reject the request. If sufficient bandwidth is available to satisfy the request, the request is admitted or else it is rejected. The bandwidths allocated to existing (already scheduled) requests are not degraded/preempted to satisfy requests with a higher priority and hence higher worth.

The scheduling heuristic considers degradation/preemption of existing requests to allocate more bandwidth to requests with a higher worth. The heuristic also considers the utility functions of the requests. The scheduling heuristic degrades/preempts the requests differently based on the utility functions of the requests, thus attempting to maximize the worth W.

6.3. Upper Bounds

Three upper bounds were considered in this research. The first upper bound is the sum of the worths of all the requests that have arrived in the network during the simulation time. This bound is a loose upper bound in that it may be unachievable and optimistic.

Two tighter upper bounds, an ingress upper bound and an egress upper bound, have been considered in this research. For the calculation of both the bounds, the full knowledge of all the requests that have arrived in the network is assumed.

The ingress upper bound is calculated as follows. For ingress link i, a list of all the requests that utilize the ingress link i during the simulation time is considered. The number of bits that each request needs to transmit is obtained by multiplying the full bandwidth desired by the request and the duration for which the bandwidth is desired.

The worth per bit of each request is calculated by dividing the full worth of the request by the number of bits that each request needs to transmit. All these requests in the list are then sorted in decreasing order of worth per bit, i.e., the first request in the sorted list would have the highest worth per bit.

The interval of time over which the sum of worths of the requests is calculated is the simulation time. All the requests in the list would have started (and probably ended) during this simulation time. The maximum number of bits that can possibly be transmitted during this simulation time is the product of the link bandwidth (155 Mbps) and the simulation time. The requests are satisfied in the decreasing order of worth per bit (i.e., the request with the highest worth per bit is satisfied first, the request with the next highest worth per bit is satisfied next and so on) until the number of bits satisfied equals the maximum number of bits that can possibly be transmitted during the simulation time (discussed further in Section 7). The sum of the worths of all the requests that could be satisfied in the manner described above, on the link i, is calculated.

The process is repeated for all the ingress links in the system The total sum of worths obtained by summing up the worths of the requests satisfied on each link, is an upper bound on the performance of the heuristic.

For the upper bound calculation described above, any congestion at the egress link is not considered (and hence it is called the ingress upper bound). That is, if a request is allocated some bandwidth at the ingress link it is assumed for the ingress upper bound calculation that it is allocated the same bandwidth at the egress link too, without any degradation. Because the calculation of the upper bound assumes full prior knowledge of all the requests that have arrived in the network, it can sort the requests in decreasing order of worth per bit and satisfy only those requests that yield the maximum worth per bit. For the upper bound calculation, the start and stop times of the requests are ignored and only the bits of the request are considered.

In the scheduling heuristic, the request may be degraded at the ingress and at the egress link. Also, the heuristic makes decisions regarding admission/rejection of a request on-line, without prior knowledge of future arrival of requests. Furthermore, the scheduling heuristic also considers start and stop times of the requests. Thus, for the reasons mentioned above, the upper bound may be an **unachievable** upper bound on the performance of the heuristic. However, the heuristic allows **degradation/preemption,** whereas the ingress upper bound does not. **Thus,** this is an upper bound on an optimal schedule if no **degradation/preemption** is allowed.

The **egress upper** bound is similar to the ingress upper bound described above. In the egress upper bound, the same process described above is conducted for the egress links and congestion at the ingress links is not considered.

The reason why an egress upper bound is needed in addition to the ingress upper bound is as follows. For example, assume that all the requests have **different** ingress links but the same egress link. If the ingress and egress upper bounds were computed, the ingress upper bound would yield a comparatively loose upper bound because the degradations at the egress links are not considered. But the egress **upper** bound would yield a tighter upper bound because the degradations at the egress links are considered. Hence, the egress upper bound would give a more accurate estimate of the achievable performance in the situation described above.

6.4. Summary

Three upper bounds and one simple scheduling technique used for comparison with the performance of the scheduling heuristic were presented in this section. The simulation experiments conducted in this research are explained in the next section. The **different** parameters considered and the values assumed in the **experiments** are also described in the next section.

7. SIMULATION EXPERIMENTS

7.1. Overview

The simulation experiments conducted in this research are described in this section. Loading of the network, weighting of priority levels, and the globalization factor are considered. Motivation for the values that were selected for these parameters is also given.

7.2. Parameters

Without loss of generality, a link bandwidth of 155 Mbps (OC3) was assumed for the ingress and egress links in Figure 2.2. The link bandwidth of 155 Mbps is the typical link bandwidth needed by large corporations and ISPs [NAP98]. In a military environment, such high bandwidth links may exist between the different military headquarters spread across the country (or the full world). A high bandwidth link may also exist between the military headquarters and the satellite base stations. For the sake of simplicity, the link bandwidths of all the ingress and egress links shown in Figure 2.2 are assumed to be the same. The link bandwidths may be different and these different link bandwidths can be considered by the heuristic. Again for the sake of simplicity, the number of links was chosen to be fifteen. The number of links in the network does not affect the operation of the heuristic.

The bandwidth required by a request depends upon the application generating the request. For example, applications such as audio conferencing, streaming audio (CD quality), and low quality video transmission [MiM96] may require a bandwidth of 500

Kbps (Kilobits per second). Applications such as high quality MPEG video may require a bandwidth of 10 Mbps (Megabits per second) [ReR95]. The requests in this research were assumed to have a bandwidth requirement in the range of 500 Kbps to 10Mbps. In this research, a uniform random distribution was used to generate the bandwidth requirements of the requests in the range of 500 Kbps to 10 Mbps. Recall that the requests were assumed to have four priority levels (1, 2, 3, and 4). Two mode values ω (the weighting constant that decides the weights of the priority levels) were assumed: mode two and mode ten. The weighting function for a priority level i, as explained in Subsection 2.4.2, was assumed to be

$$\Pi(i) = \omega^{(4-i)}.$$

The utility functions of the requests were assumed to be of three types: concave, linear, and step. The utility of the request, as determined by the utility function, is a value between 0 and 1. If a request with a desired bandwidth of rb_k and a minimum bandwidth of mb_k is allocated bandwidth $b_k(t)$ at time t, then the concave utility function of a request is given by

$$u_k(b_k(t)) = \left[\frac{1 - e^{-\beta(b_k(t) - mb_k)}}{1 - e^{-\beta(rb_k - mb_k)}}\right]. \tag{7.1}$$

The linear utility function of such a request is given by

$$u_k(b_k(t)) = \left[\frac{b_k(t) - mb_k}{rb_k - mb_k}\right]. \tag{7.2}$$

The step utility function of a request with desired bandwidth rb_k (desired bandwidth rb_k = minimum bandwidth mb_k) is given by

$$\begin{aligned} u_k(b_k(t)) &= 1 \text{ if } b_k(t) \geq rb_k \qquad (7.3) \\ &= 0 \text{ if } b_k(t) < rb_k. \end{aligned}$$

In this research, the minimum bandwidth (mb_k) desired by a request (in case of requests with linear and concave utility functions) is assumed to be in the range of 25% to 75% of the required bandwidth of the request (rb_k), i.e., the bandwidth of a request can be degraded by an amount equaling 25% to 75% of its required bandwidth.

The values of β (see Equation 7.1) for the concave utility functions assumed for the simulation experiments were determined based on the average slope of the linear utility functions of the requests. The values of β were determined to be 0.0004, 0.000835, and 0.0015. The concave utility functions of requests with these β values are illustrated in Figure 7.1.

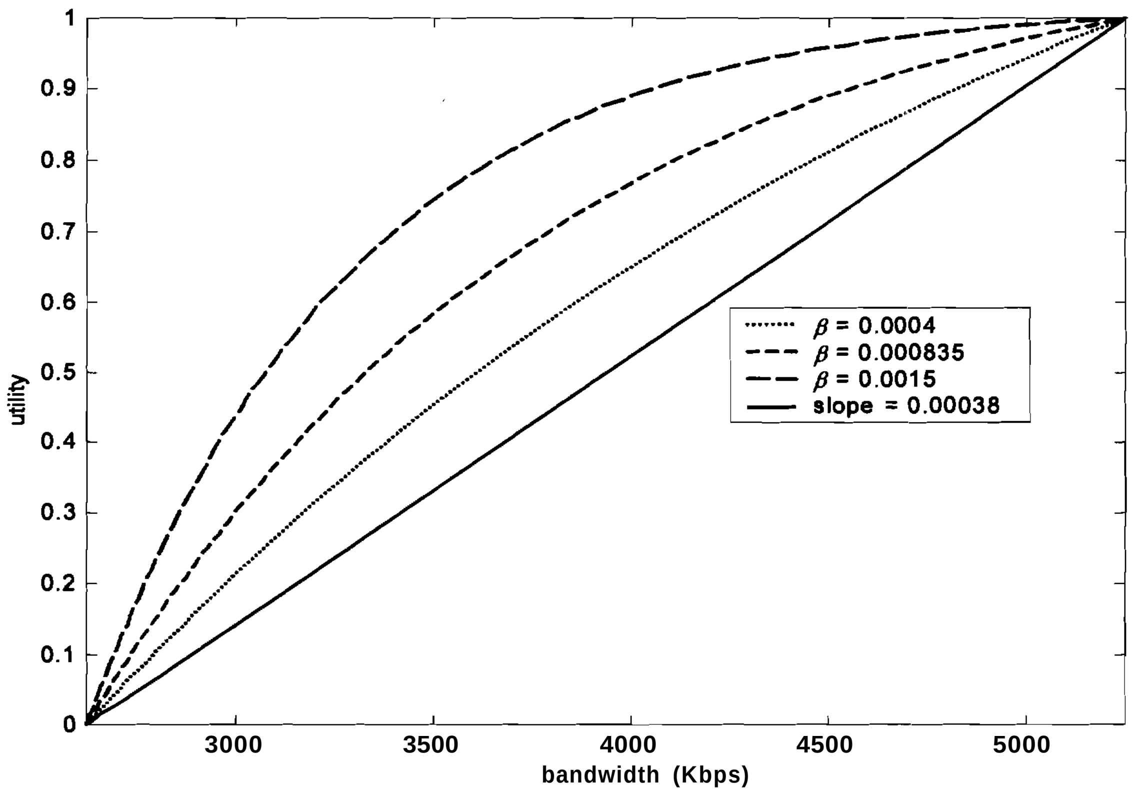

Figure 7.1. Concave and linear utility **functions** of the requests **with** a **minimum** bandwidth of 2625 Kbps (average minimum bandwidth) and required bandwidth of 5250 Kbps (average required bandwidth).

These β values were chosen to ensure that the bandwidth allocated to a request with a concave utility function is not degraded always when compared to a request with a linear utility function. Referring to Figure 7.1, when the bandwidth allocated to the request with a concave utility function of β = 0.004 is 3900 Kbps (half of the average requested bandwidth of the request), the marginal worth of the request will be more than the average marginal worth of a request with a linear utility function. Hence, on average the bandwidth allocated to a request with a linear utility function will now be degraded instead of the request with the concave utility function (the heuristic degrades the request with the least marginal worth). Initially, based on averages, when the bandwidth allocated to a request with a concave utility function is the required bandwidth of the request, the marginal worth of the request is smaller than the marginal worth of a linear utility function. But when the bandwidth allocated to the request is reduced to half (approximately) of its required bandwidth, the marginal worth of the request with a concave utility function becomes more than a request with a linear utility function and hence the bandwidth allocated to a request with a linear utility function will be degraded. The average required bandwidth of a request with a step function is equal to the average maximum bandwidth of request with a concave or a linear utility function, while the minimum bandwidth of the step function is zero.

In the simulation studies discussed in Section 8, the types of the utility functions of the requests were randomly selected as one of step, linear, and each of three concave functions. Thus, there are approximately 20% of each type.

Ethernet traffic and World Wide Web traffic has been shown to be self-similar in [CrB96, LeT94]. Self-similarity is the property associated with an object when its appearance remains unchanged regardless of the scale at which it is viewed. In case of objects such as Ethernet traffic, a long-range dependence is observed [LeT94], i.e., values at any instant are typically non-negligibly correlated with values at all future instants.

A distribution is heavy-tailed if

$$P[X > x] \sim x^{-\alpha} \text{ as } x \to \infty, \text{ where } 0 < \alpha < 2.$$

It has been shown in [PaK96] that the traffic in the Internet is self-similar because the files and data transferred follow a heavy-tailed distribution. Considering this fact, the session durations of the requests considered in the simulation experiments were assumed to follow heavy-tailed distribution. That is, the random distribution used to generate the session durations of the requests (i.e., the start time to the end time of the request) was assumed to be heavy-tailed [PaK96]. One of the simplest heavy-tailed distributions is the

Pareto distribution. The probability density **function** (PDF) of the Pareto distribution is given by

$$f(x) = \alpha k^{\alpha} x^{\alpha-1}$$

where $\alpha, k > 0$, and $x \geq$ k. The cumulative distribution has the form

$$F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}.$$

The parameter k represents the smallest possible value of the random variable. If a ≤ 2 the distribution has infinite variance; if a ≤ 1 then the distribution also has infinite mean. As a decreases, a large proportion of the probability mass is present in the tail of the distribution. In practical terms, if the session duration follows a heavy-tailed distribution, then extremely large session durations can be generated with non-negligible probability. In the simulations here, although the session duration was assumed to follow a Pareto distribution, the session durations were truncated at some high value for practical considerations. The reason is that in a realistic **network/system**, there will never be a request that has infinite duration. Hence, considering a realistic environment, the session duration was truncated at some high value. Although the random distribution used to generate the session durations is not a true heavy-tailed distribution, it is based on a heavy-tailed distribution that is truncated for practical considerations.

The degree of self-similarity of a series (or **traffic**) is expressed by the Hurst parameter *H*. The Hurst parameter H is related to the parameter a in the Pareto distribution as H = (3 – a) / 2 **[PaK96].** The typical values of H were shown to be in the range of 0.7 to 0.8 for the World Wide Web traffic **[Fel00, CrB97, PaK96].** From the values of H, the typical values of a were calculated to be from 1.4 to 1.6. The a value assumed in the simulation experiments was 1.5. The request durations were assumed to be in the range of 2 minutes to 60 minutes. A typical voice application would require bandwidth for the duration of a few minutes. Applications such as video **conferencing,** streaming videos for movies, or lecture broadcasts (over the Internet) would typically require session duration of over 30 minutes. The session durations of 2 minutes to 60 minutes assumed in this research encompasses all these different session duration requirements. Thus, the parameter k in the Pareto distribution has a value of 2 minutes or 120 seconds. The mean of the Pareto distribution is $\alpha k/(\alpha - 1)$. For the values of k and a assumed in the simulation experiments, the mean of the Pareto distribution is 6 minutes. The session duration is truncated at 60 minutes (ten times the mean), **i.e.,** only session durations between 2 minutes to 60 minutes are generated.

In the simulation experiments the arrival sequence of the requests is assumed to be correspond to a Poisson arrival sequence. An arrival sequence of requests is said to follow the Poisson arrival sequence with rate $1/\lambda$ (units) if the inter-arrival time between the requests follows an exponential distribution with mean h. The higher the mean λ of the exponential distribution used to generate the inter-arrival time between requests, the lower the rate of the Poisson arrival sequence will be. The method of determining the arrival rate of $1/\lambda$ is explained later in this subsection.

The lead time assumed in this research was 2 minutes to 2 hours. A minimum lead time of 2 minutes is only an estimate of the worst case performance of the heuristic (maximum time taken by the heuristic) and does not reflect actual execution times of the heuristic. An upper limit of 2 hours (7200) seconds was assumed to prevent requests **from** requesting bandwidth many hours (or days) in advance. This reduces the time for the simulation experiments and does not in any way affect the performance of the heuristic.

The simulation experiments in this research are conducted for a time interval corresponding to the earliest start time of all the requests until the time when N requests end. The number N chosen for the simulation experiments was 2000. **Thus,** the time interval is the time interval between the earliest start time (of all the requests) and the time when 2000 requests end. The start and end times for a simulation experiment are called the simulation **start** and simulation **end** times, respectively. Because the simulation end time corresponds to the time when N requests end **(i.e.,** Nth end time), there may be many requests which begin before the simulation end time but do not end before the simulation end time. These requests are also considered for the simulation experiments. **Thus,** it should be noted that the time interval between the earliest start time (of all the requests) and the time when N requests end is considered for the simulation experiments and not a fixed number of requests. The number of requests considered during the simulation time interval (the time difference between the simulation end time and simulation start time) could be more than N. (This number depends upon the "arrival rate" of the requests and this is explained. later in this section). F denotes the total number of requests that start before the simulation end time (i.e., start time of the request is before the simulation end time). N of these requests end before or at the simulation time while $(F - N)$ requests start before the simulation end time but end after the simulation end time.

The reason why the above method is chosen is as follows. The "steady state" performance of a heuristic should be used for evaluating performance of the heuristic. If a fixed number of requests are considered for the simulation experiments as opposed to a fixed interval of time, the system is not in steady state for the duration of the simulation time interval. For example, if 2000 requests are considered (and not a fixed time interval), at the time instant before the simulation end time, the 2000th request will be the only request being scheduled, i.e., it is not competing for bandwidth with any other request. Hence, there are some "transient" effects during the start of the simulation and during the end of the simulation. Alternatively, when a fixed interval of time is considered for the simulation experiments, even at the simulation end time there are many requests overlapping in time and hence competing for the available bandwidth. Thus, the performance of the heuristic in the steady state is determined.

The performance measure for the system is the sum of the worths of the requests satisfied during this simulation time interval. Recall the total sum. of worths W is calculated as (explained in Subsection 2.5):

$$W = \sum_{k \in S} \Pi(p_k) \times \left[\left(\sum_{t=s_k}^{e_k} u_k(b_k(t)) \right) \Big/ (e_k - s_k) \right].$$

The set S over which the total sum of worths is calculated consists of these F requests. The worth of a request is calculated as the worth obtained during the simulation time interval. Hence, for requests that begin during the simulation time interval and do not end before (or at) the simulation end time, the worth is calculated as the worth obtained due to bandwidth allocated to the request during the simulation time interval. The worth obtained due to bandwidth allocated to the request after the simulation time interval is not considered. This can be thought of as if a request does not end during the simulation time interval, it is truncated at the simulation end time. It should be noted that the heuristic does not truncate the requests. It considers the entire duration of the request while scheduling the request. But when the total worth of all the requests satisfied during a fixed interval of time is calculated, only the worth of a request obtained due to the bandwidth allocated to it during the simulation time interval is considered.

The upper bounds and the complete sharing policy only consider all the requests in S. In the calculation of the upper bounds, any request that begins during the simulation time, but ends afterwards, receives prorated worth (i.e., the percentage of its worth that corresponds to the percentage of its bits that are received before the simulation time ends). Similarly, for the complete sharing policy, the prorated worth of such a request is considered if it received full bandwidth during the simulation time.

The three different parameters that were varied to observe the performance of the heuristic are as follows.

1. Loading factor (lf): The loading factor indicates the amount of "load" that is placed on the network. The loading factor is defined as the ratio of total number of bits that needs to be allocated during a fixed interval of time and the maximum number of bits that can possibly be allocated during that fixed interval of time (simulation time).

   The loading factor is calculated as follows. The desired bandwidth of each of the F requests is converted into bits by multiplying the desired bandwidth with the time duration for which the bandwidth is desired. Only the bits of a request that need to be allocated before the simulation end time are considered. The bits that need to be allocated after the simulation end time are neglected (i.e., the requests are truncated as explained above). The total number of bits that need to be allocated during the simulation time interval is called the offered load. The offered load indicates the amount of bandwidth desired by the requests during the simulation time interval. Thus,

$$\text{offered load} = \sum_{k=1}^{F} \left[rb_k \times (s_k - e_k) \right]. \tag{7.4}$$

The maximum number of bits that can be allocated during the simulation time interval is calculated by multiplying the link bandwidth L by the number of ingress links and the simulation time interval. This number is the maximum number of allocable bits for the simulation time interval. The simulation time interval is denoted by sim_time and the total number of ingress links in the network is M. Thus,

$$\text{maximum allocable bits} = (L \times M \times \text{sim\_time}) . \qquad (7.5)$$

The loading factor is the ratio of the offered load and the maximum allocable bits.

$$\text{loading factor} = \text{offered load} / \text{maximum allocable bits}. \qquad (7.6)$$

Substituting the Equations 7.4 and 7.5 in 7.6 the loading factor is calculated as

$$\text{loading factor} = \frac{\sum_{k=1}^{F} rb_k \times (s_k - e_k)}{L \times M \times \text{sim\_time}} \qquad (7.7)$$

The two loading factors considered in this research are 0.7 and 1.2. As mentioned earlier, the loading factor indicates the load placed on the network. A loading factor of 0.7 represents a moderately loaded network situation when not all requests can be satisfied. (A typically loaded network would have a loading factor of under 0.5.) A loading factor of 1.2 indicates the load on the network during periods of heavy congestion and/or periods of burst in traffic. For example, in the Internet, a burst in traffic may occur when a large number of packets arrive at a router and the router cannot route all these packets. Thus, variation of the loading factor explores the performance of the heuristic in different overloaded network conditions.

The arrival rate indicates the number of requests that arrive (in the network) per second. The loading factor depends upon the arrival rate of the requests, the average duration of the requests, and the average desired bandwidth of the request. Increasing the arrival rate increases the loading factor, while keeping the average values of the duration and the desired bandwidth of the request the same. Because the loading factor was considered as the parameter to be varied, the arrival rate is adjusted so that a loading factor of 0.7 and 1.2 is achieved. The arrival rates were adjusted by trial and error to determine what arrival rate would result in a loading factor of 0.7 and 1.2.

2. Mode value: Two mode values, $\omega = 2$ and $\omega = 10$, were considered in this research. The concept of mode values and weighted priorities was explained in Subsection 2.4.2. Variation in the mode values would show the performance of the heuristic in different situational modes. The mode values of two and ten were used in a manner similar to [ThB00].
3. Globalization factor (GF): Values of the globalization factor experimented with (in addition to 0%) were 5%, 10%, 15%, and 25%. A globalization factor of 5% gave the best results and hence the 5% globalization factor was used for the simulation experiments (in addition to 0%).

For the scenario where lf = 1.2, ω = 1.2, and GF = 0%, simulation experiments were conducted to find the number of experiments that yields a 95% confidence interval of less than ± 5% of the mean of the sum of the worths of the requests satisfied. Each experiment involved F (–2000) requests. A set of 20 different experiments, 40, and 60 were examined. It was found that 20 experiments were sufficient, as discussed further in Section 8.

7.3. Summary

The values of the different parameters that were considered in this research such as, the bandwidth desired by the requests, the session duration of the requests, and the random distribution used to generate the requests, have been presented in this section. The loading factor, the mode value, and the globalization factor that were varied and the reason why these parameters were varied also have been discussed in this section. The results of the simulation experiments are discussed in the next section.

8. RESULTS OF SIMULATION EXPERIMENTS

8.1. Overview

The simulation experiments conducted in this research were discussed in the previous section. The parameters varied for the experiments and the values of the parameters assumed were also described in the previous section. The results of the simulation experiments are presented in this section. The performance of the scheduling heuristic under different loading conditions, different mode values, and different values of the globalization factor is discussed in this section. The trends displayed as these parameters are varied are also explained in this section.

8.2. Evaluation of Simulations

Twenty, 40, and 60 randomly generated experiments were conducted for the lf = 1.2, $\omega = 10$, and GF = 0% scenario and for the 50% heuristic variation as was discussed in Section 7. The total sum of worths obtained was averaged over these 20, 40, and 60 test cases. This average sum of worths, along with the confidence intervals is plotted in Figure 8.1. The size of the 95% confidence interval reduces as the number of experiments increases from 20 to 40 to 60. The 95% confidence interval reduced from $\pm$ 1.56% of the mean for 20 experiments, to $\pm$ 1.07% for 40 experiments, to $\pm$ 0.86% for 60 experiments. Because the size of the 95% confidence interval for 20 experiments was within the desired range of $\pm$ 5%, 20 was chosen as the number of experiments for all the remaining simulation experiments.

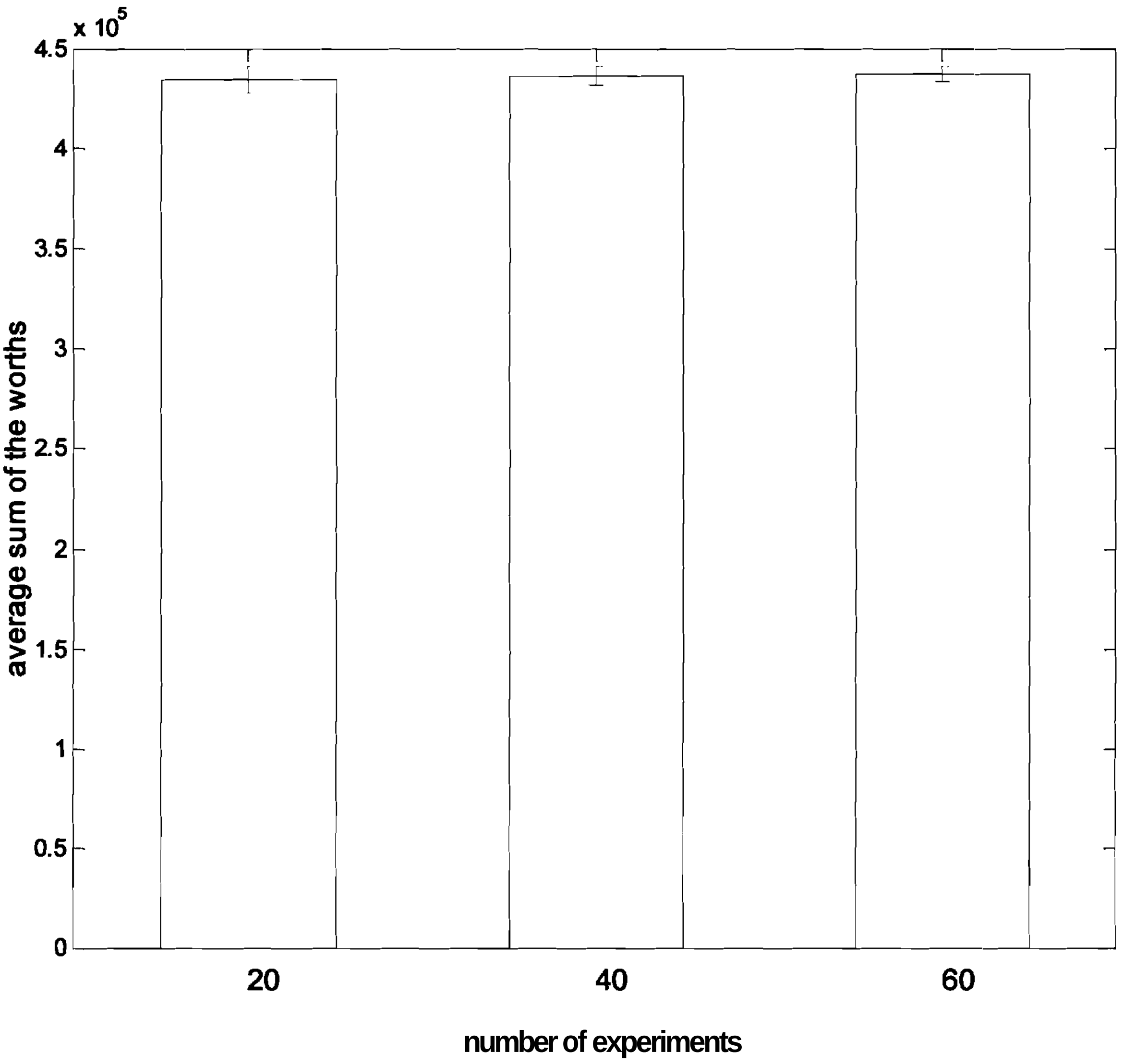

Figure 8.1. Confidence intervals of 20, 40, and 60 experiments with a mode value ω = 10, loading factor = 1.2, and GF = 0% for the 50% variation of the heuristic.

The three parameters varied were the loading factor, the mode, value and the GF. The two values of the loading factor examined were 0.7 and 1.2. The mode values examined were value $\omega = 2$ and $\omega = 10$. The GF values examined were 0% and 5%. Thus, eight different cases were examined.

The average sum of the worths (over 20 experiments) obtained by complete sharing policy, the heuristic variations, and the upper bounds for these eight cases are shown in Figures 8.2 to 8.9. The 95% confidence intervals are also given. The general trends that can be observed are as follows. The three heuristic variations perform comparably as shown. As the mode value ω is increased fiom 2 to 10, the average sum of worths obtained increases because the worths of the requests of priority levels 1 to 3 are much higher.

As the loading factor is increased fiom 0.7 to 1.2, the number of requests considered for the simulation experiments increases, leading to an increase in the number of preemptions/degradations of requests (a preemption of a new request is a rejection of that request). This results in a decrease in the number of requests being satisfied. Hence, the average sum of the worths obtained decreases as the loading factor is increased from 0.7 to 1.2.

The performance improvement of the heuristic variations over the complete sharing policy varies. As the loading factor is increased fiom 0.7 to 1.2, the performance improvement of the heuristic variations over the complete sharing policy increases. For the scenario where, the loading factor is 1.2, $\omega = 10$, and GF = 0%, the heuristic variations are approximately 90% better than the complete sharing policy.

The globalization factor did not achieve any significant improvement in the performance of the heuristic. Even though several values were experimented with, it might be the case that a GF value not experimented with in this research might achieve a significant improvement in performance. It may also be the case that the GF should not be a constant, but should rather be a function of the change in worth (after degradation of bandwidths of existing requests and addition of the new request), the priority levels of the requests under consideration, and so on. Future work may determine the values of GF and parameters that GF depends upon to achieve a significant improvement in performance.

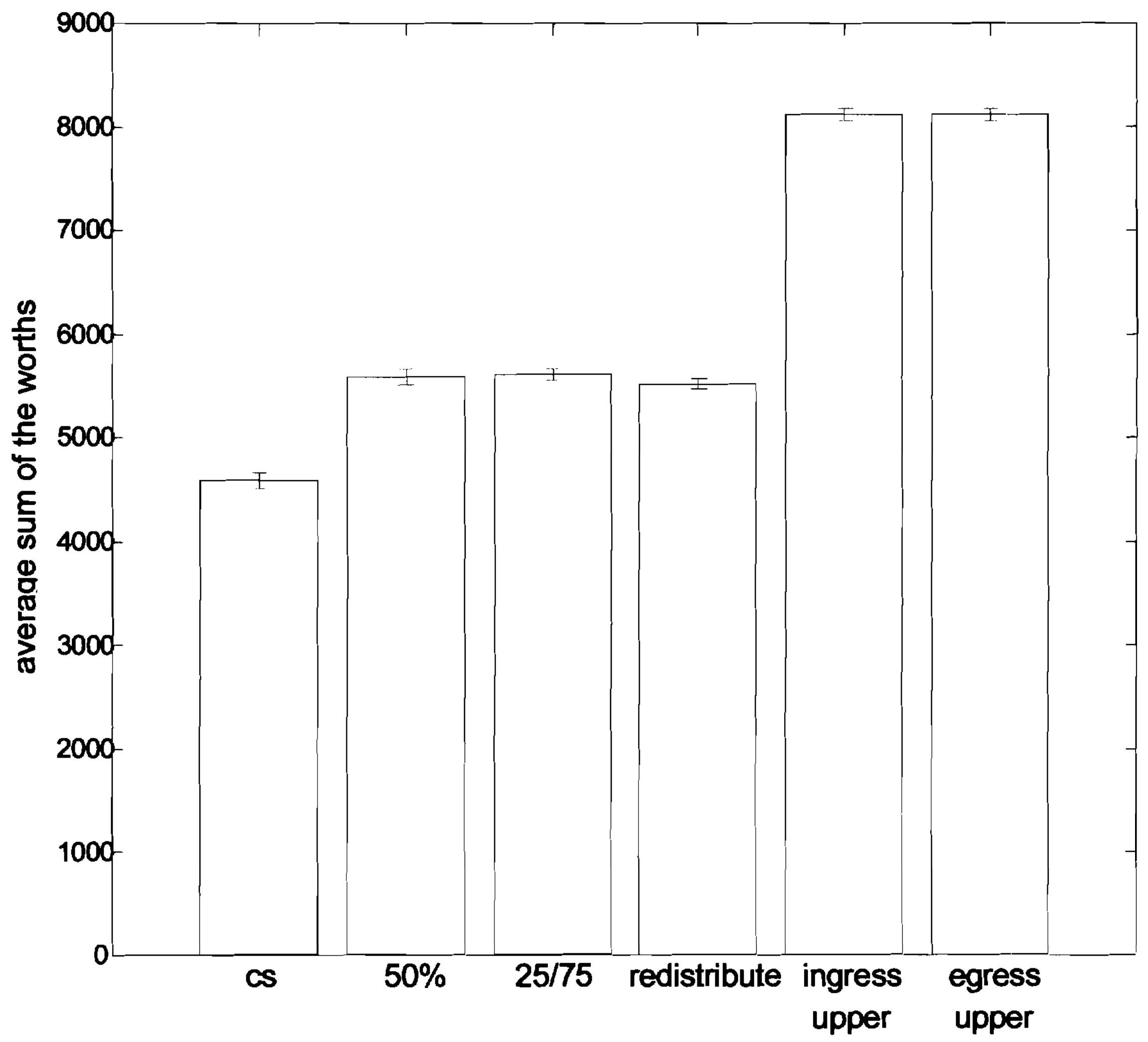

Figure 8.2. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 2$, loading factor = 0.7, and GF = 0% averaged over 20 experiments. The loose upper bound is 8514.

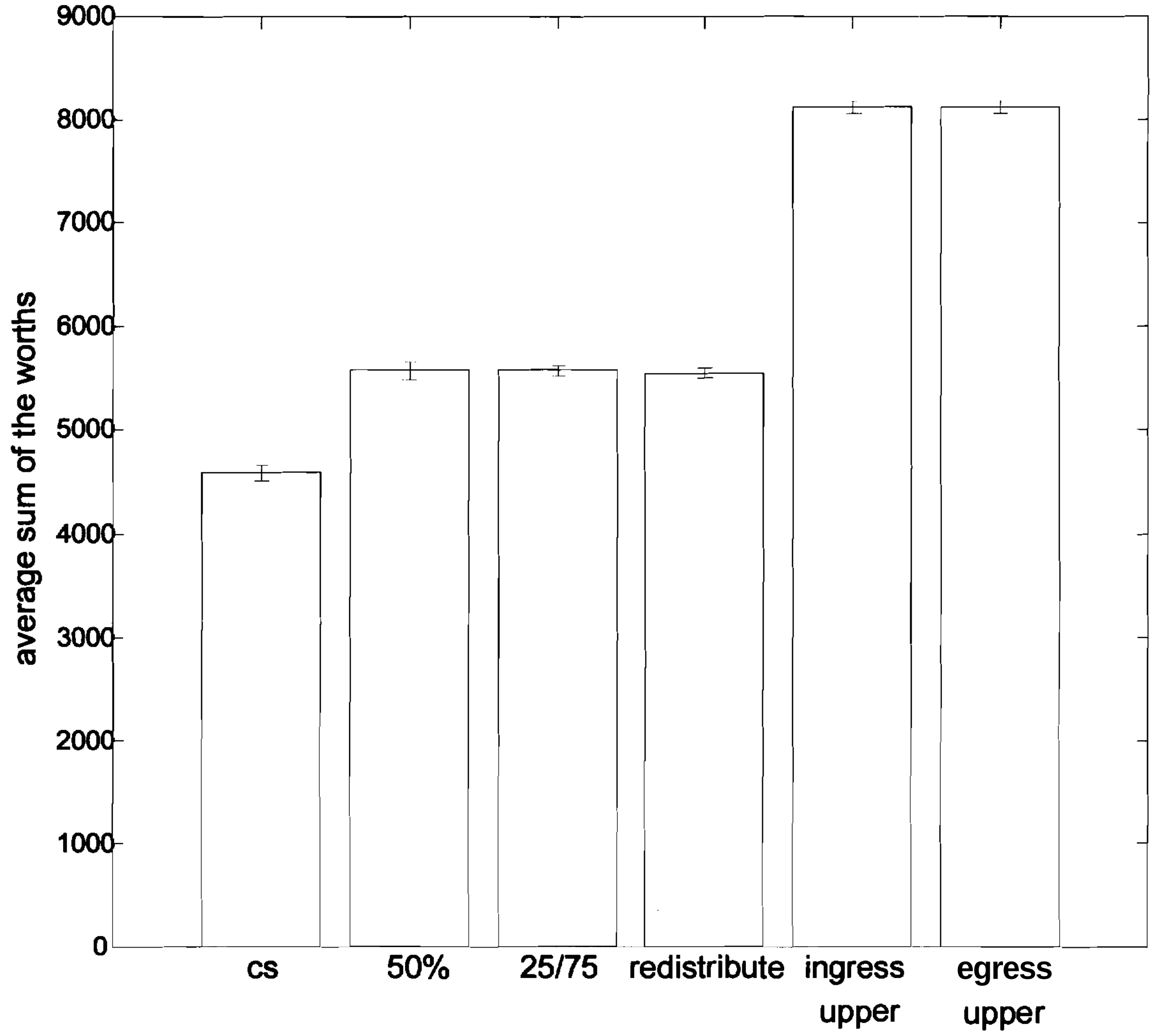

Figure 8.3. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 2$, loading factor = 0.7, and GF = 5% averaged over 20 experiments. The loose upper bound is 8514.

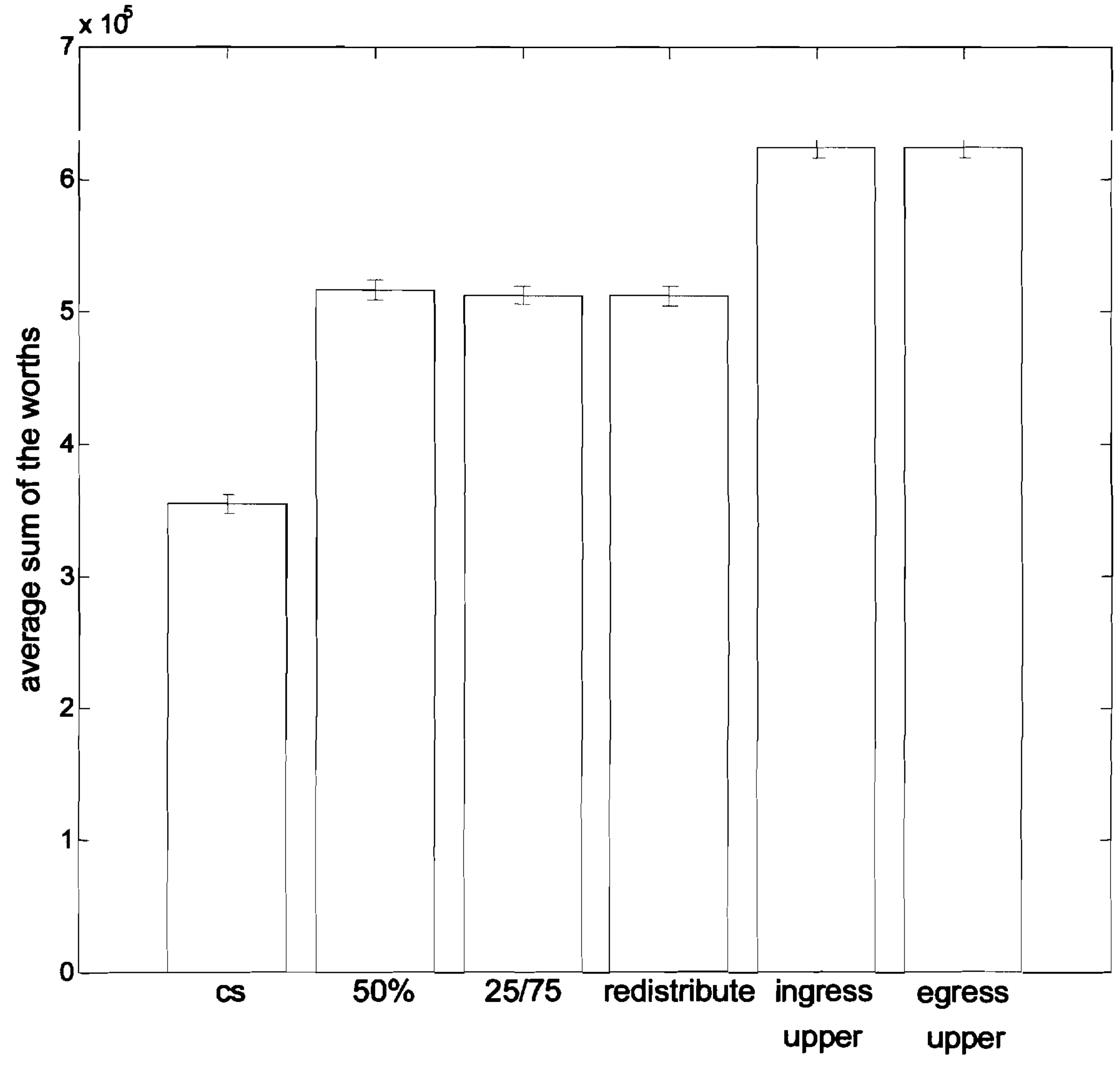

Figure 8.4. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 10$, loading factor = 0.7, and GF = 0% averaged over 20 experiments. The loose upper bound is 632741.

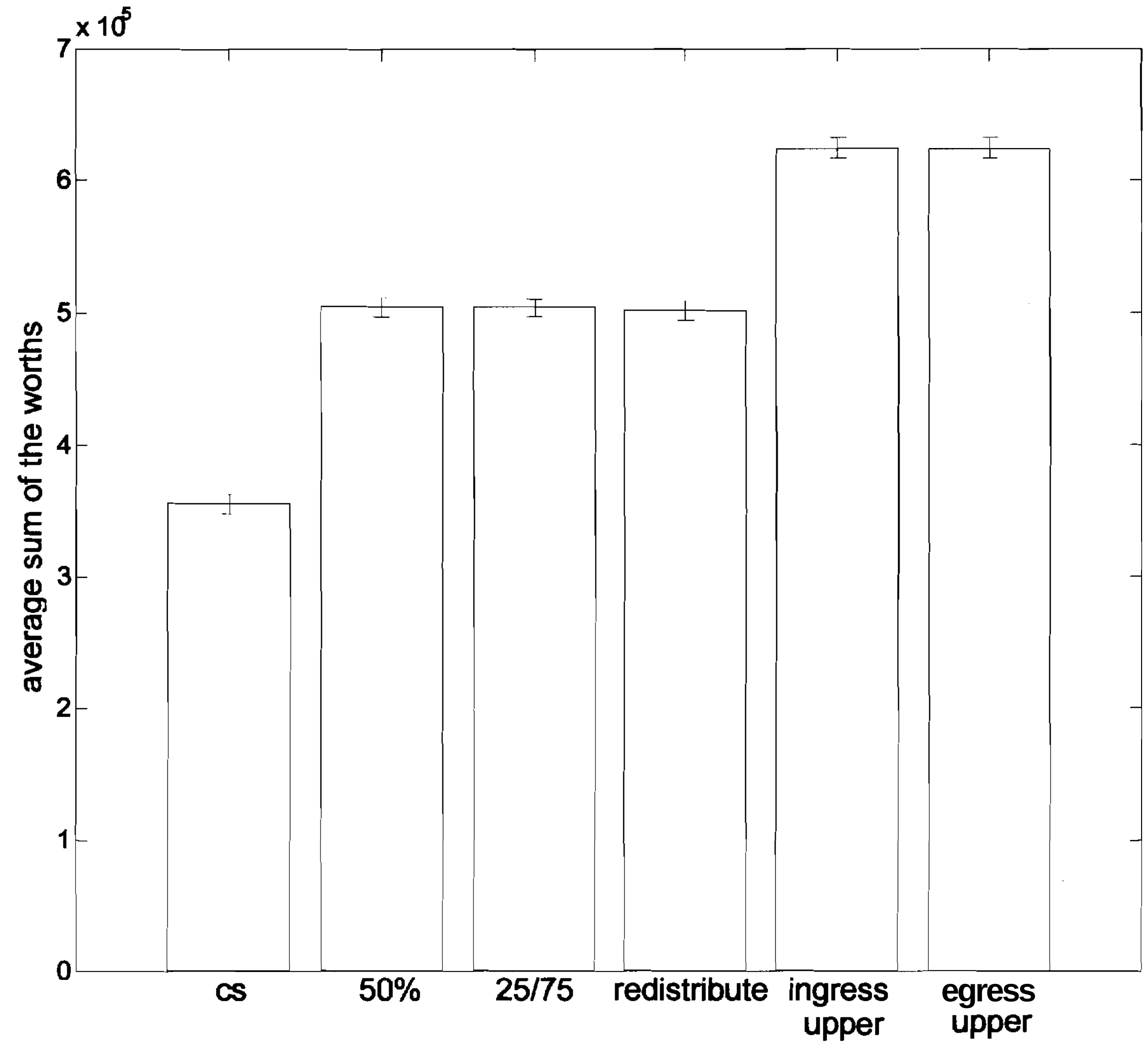

Figure 8.5. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 10$, loading factor = 0.7, and GF = 5% averaged over 20 experiments. The loose upper bound is 632741.

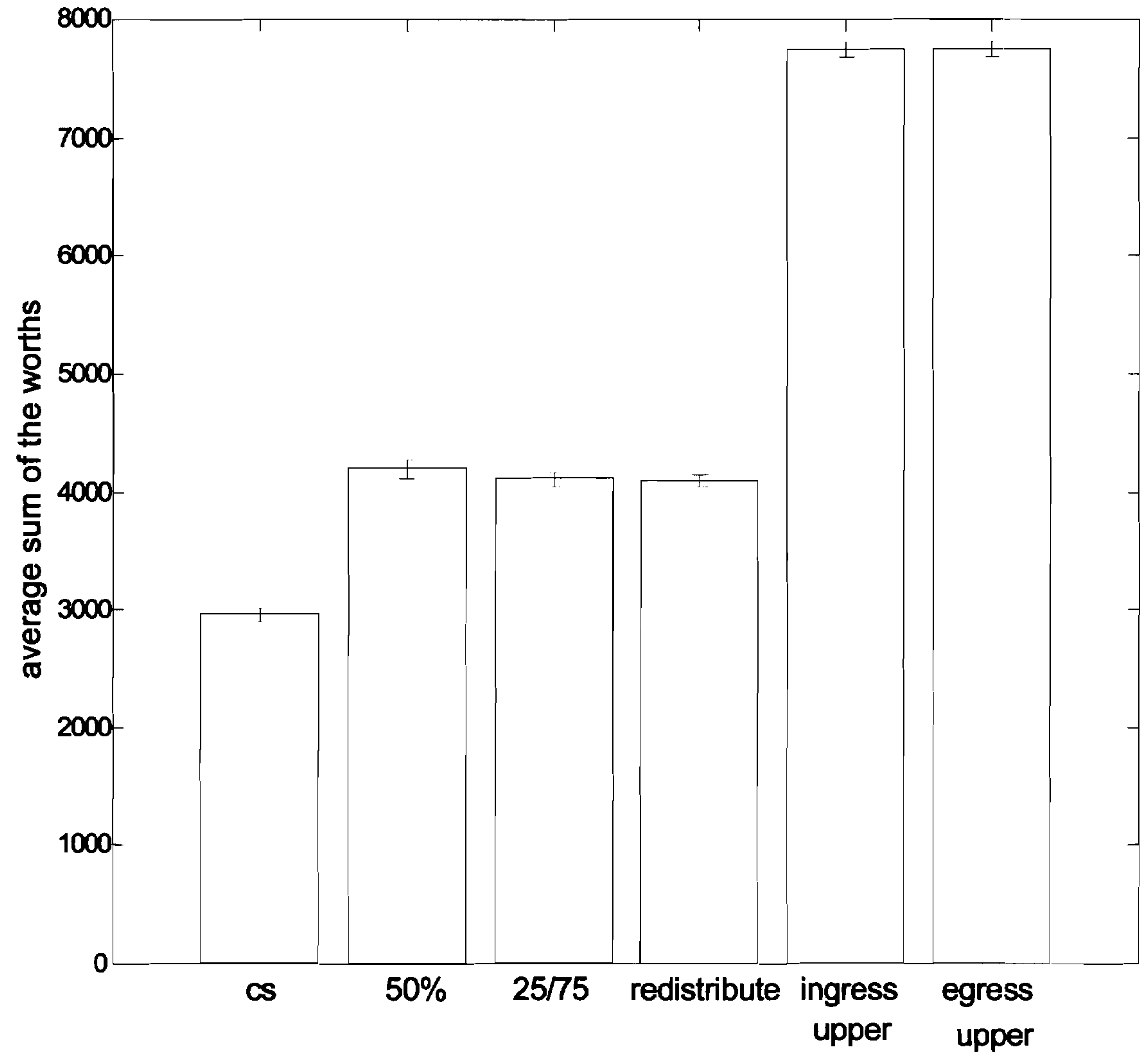

Figure 8.6. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 2$, loading factor = 1.2, and GF = 0% averaged over 20 experiments. The loose upper bound is 9417.

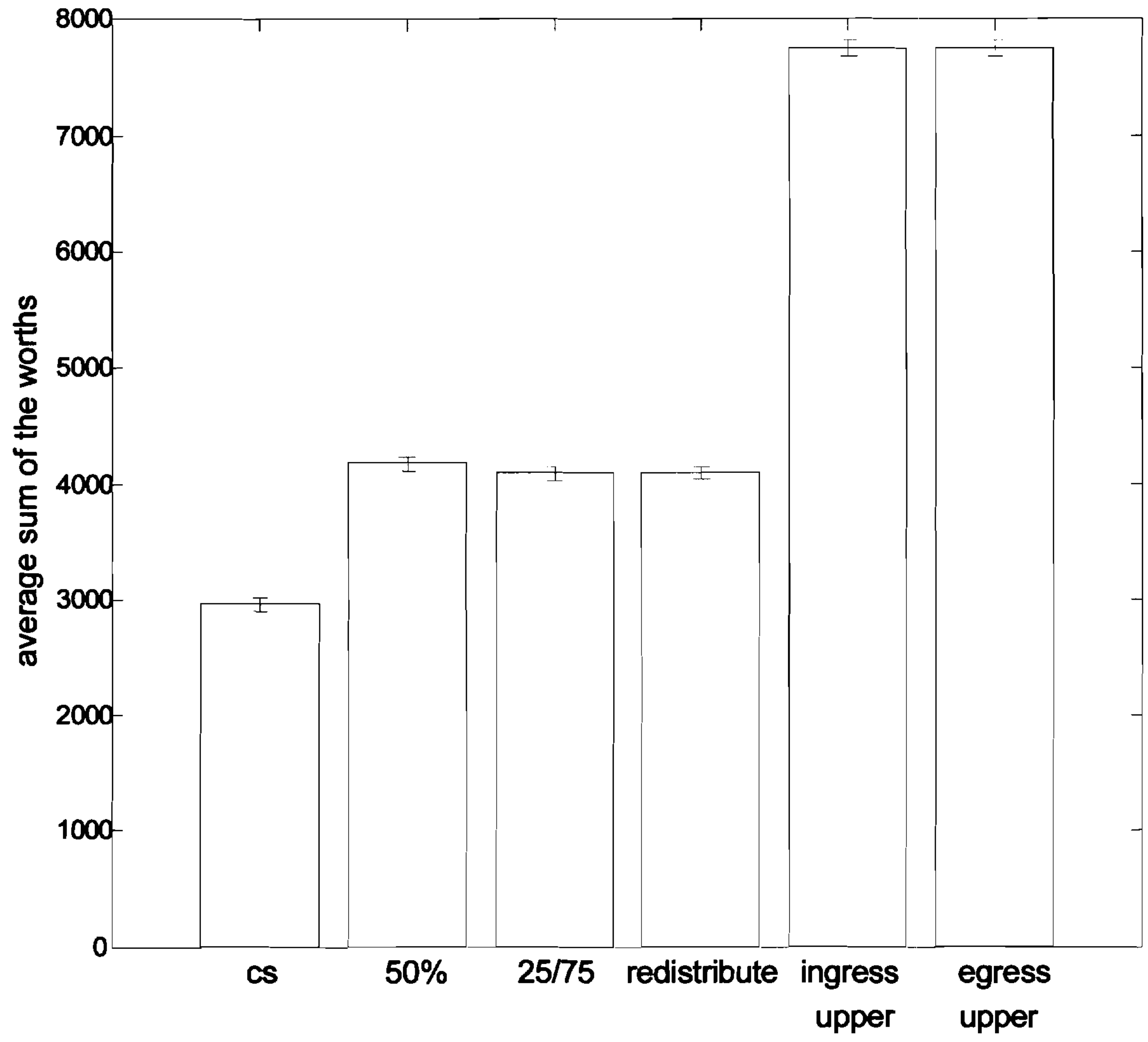

Figure 8.7. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 2$, loading factor = 1.2, and GF = 5% averaged over 20 experiments. The loose upper bound is 9417.

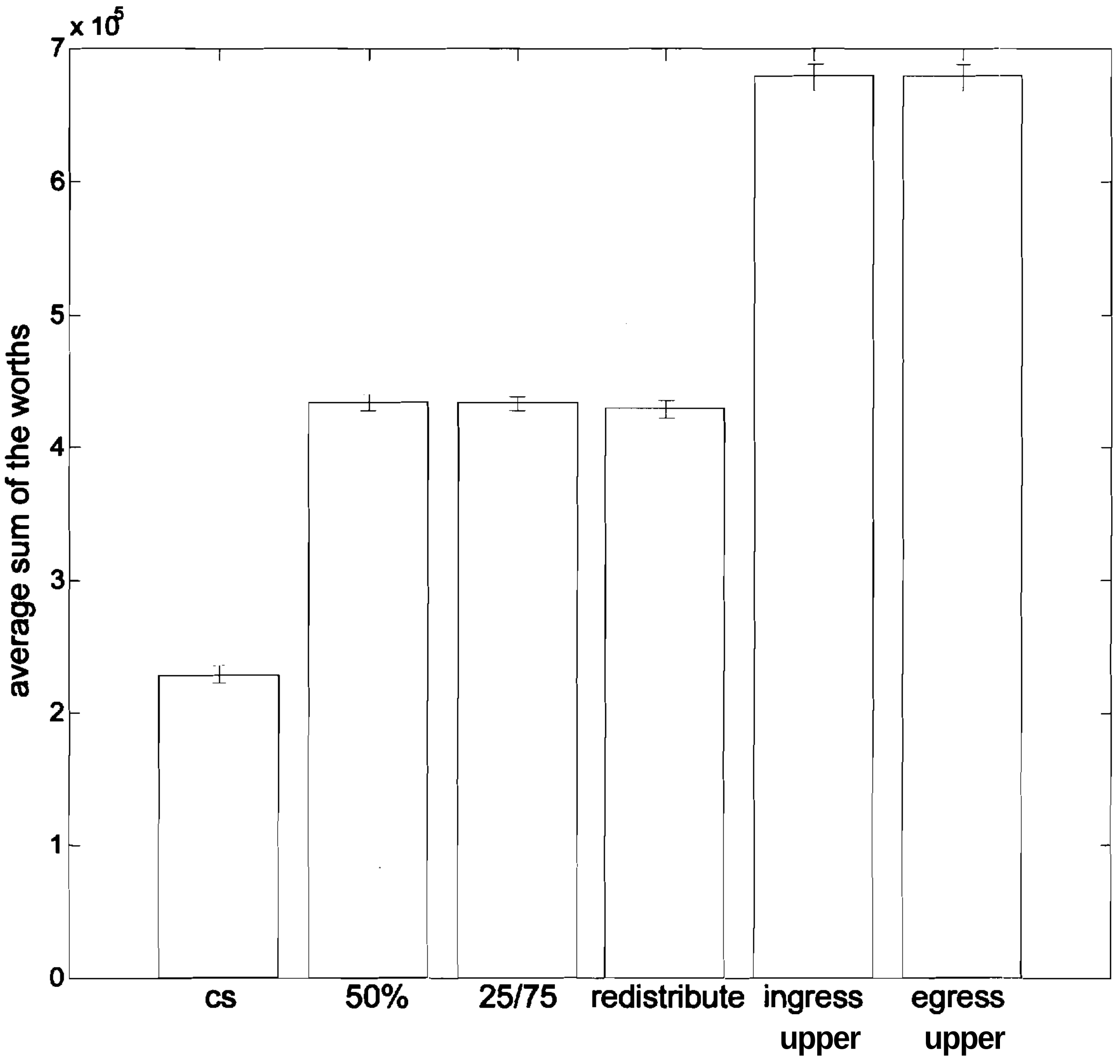

Figure 8.8. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 10$, loading factor = 1.2, and GF = 0% averaged over 20 experiments. The loose upper bound is 700152.

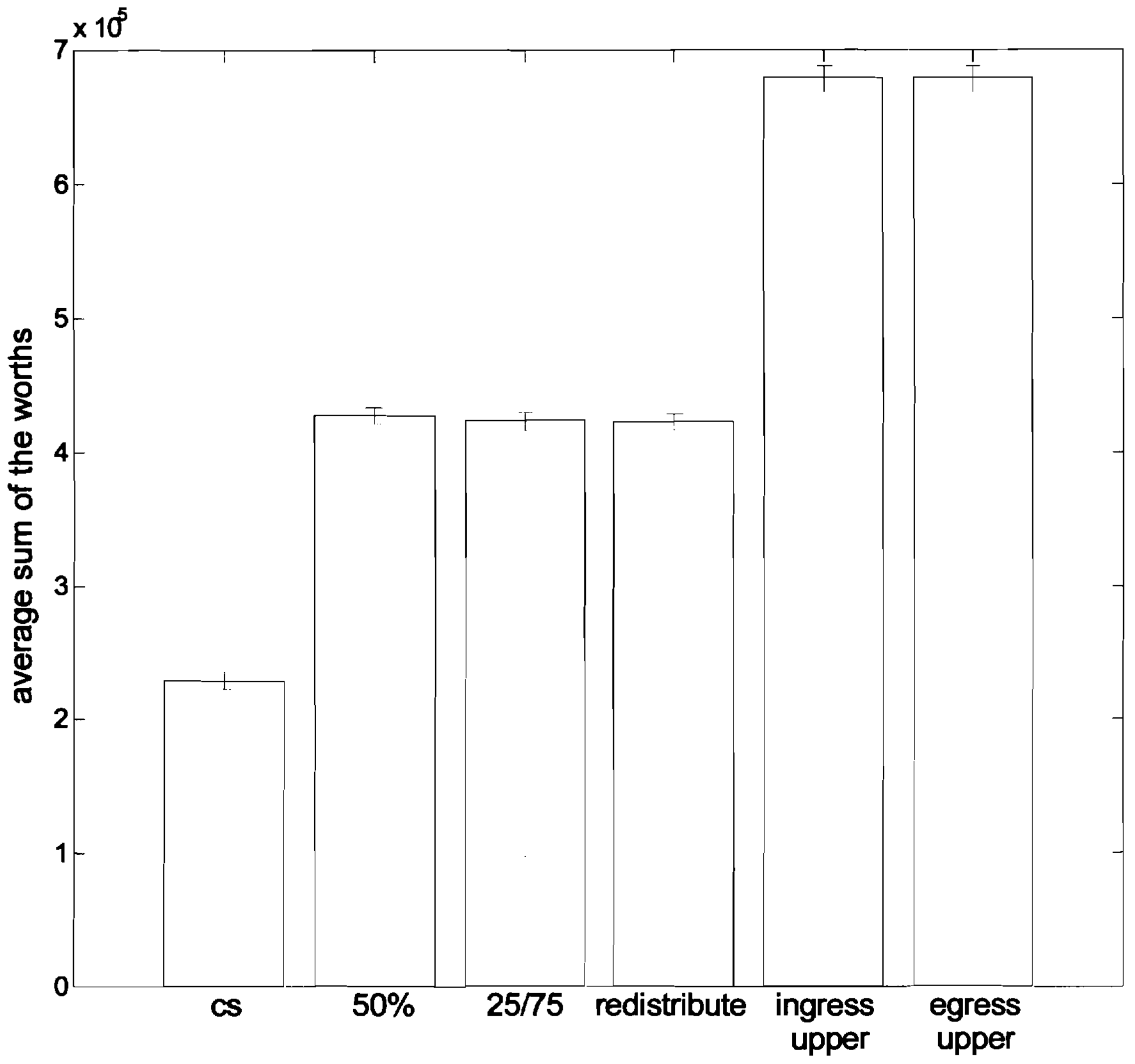

Figure 8.9. Comparison of performance of heuristic variations with the complete sharing policy (cs) and upper bounds for $\omega = 10$, loading factor = 1.2, and GF = 5% averaged over 20 experiments. The loose upper bound is 700152.

The average number of priority level 1, 2, 3, and 4 requests satisfied by the heuristic variations for the eight cases of ω, lf, GF are shown in Figures 8.10 to 8.13. The general trends that can be observed are as follows.

As expected, the number of priority level 1 requests satisfied in mode ten is higher than that satisfied in mode two. This is because of the greater relative difference in the weighted priorities of the requests in mode ten as compared to mode two. In mode two, the scheduling heuristic may satisfy eight priority level four requests instead of one priority level one request. But in mode ten, the scheduling heuristic would have to satisfy 1000 priority level four requests instead of one priority level one request. Thus, the number of priority level one requests satisfied in mode ten is higher than that satisfied in mode two. At the same time, the number of priority level four requests satisfied in mode ten is less than that satisfied in mode two.

The three heuristic variations achieve nearly the same total sum of worths (W) averaged over 20 experiments. But as is apparent from Table 8.1 and Table 8.2, the heuristics behave differently in terms of number of preemptions/degradations of requests with different utility functions. The general trend that can be observed in Table 8.1 and 8.2 is that the number of degradations/preemptions of requests with concave and step utility functions is the highest for the 50% variation the least for 25/75. The number of preemptions/degradations of requests with linear utility functions is the highest for the 25/75 variation, and the lowest for the redistribute variation. The globalization factor has no significant effect on the number of preemptions/degradations of requests with the different types of utility functions.

In 50% variation, the bandwidth needed is compared to one threshold (i.e., 50% of the bandwidth of the request with the step utility function) when making a preemption decision. In contrast, the 25/75 variation compares the bandwidth needed with two thresholds (25% and 75% of the bandwidth of the request with the step utility function) and hence reduces the number of preemptions of requests with step utility functions.

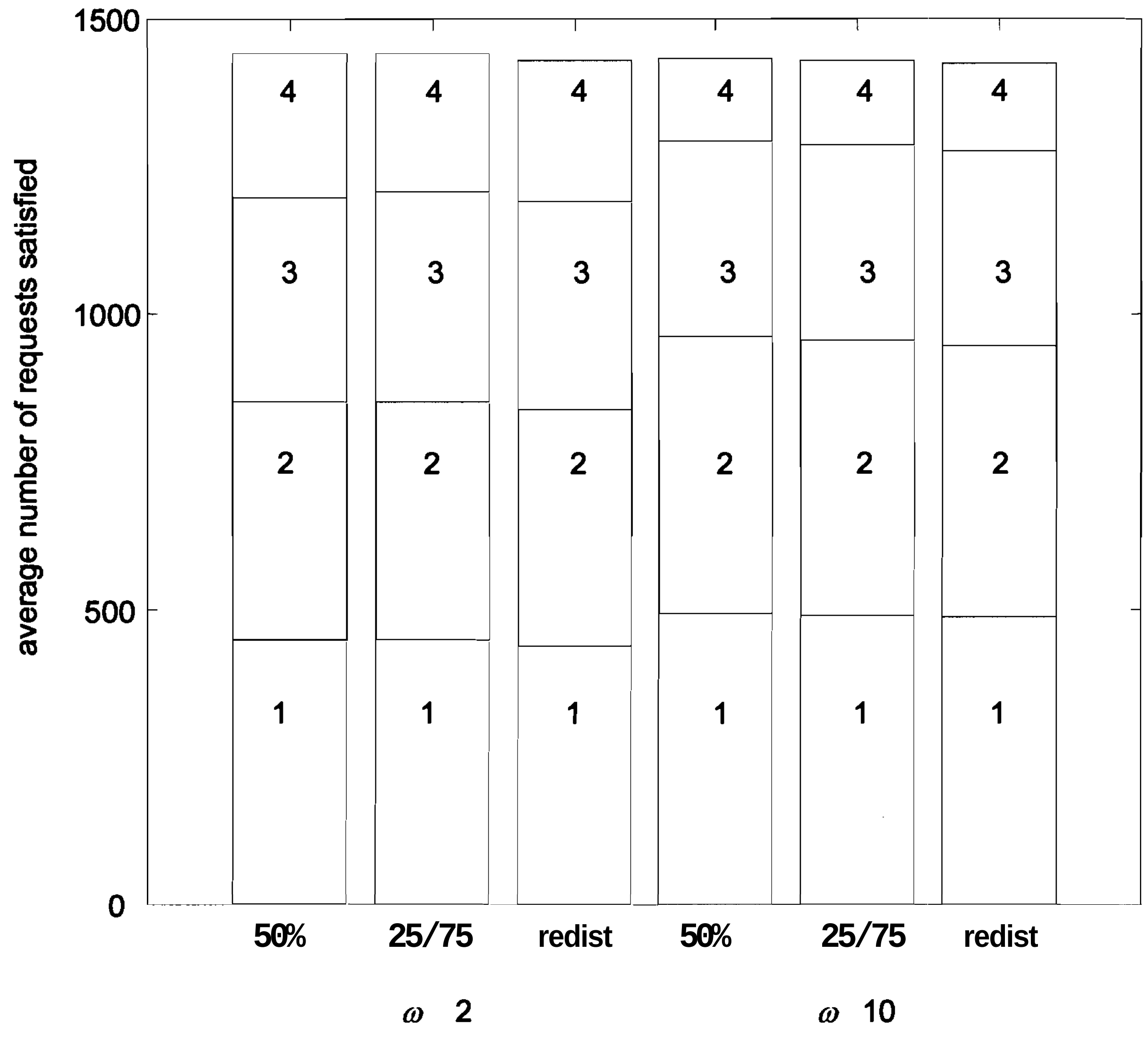

Figure 8.10. Comparison of the average number of requests of priority level 1, 2, 3, and 4 satisfied by the heuristic variations in mode value w = 2 and w = 10 with loading factor = 0.7 and GF = 0%.

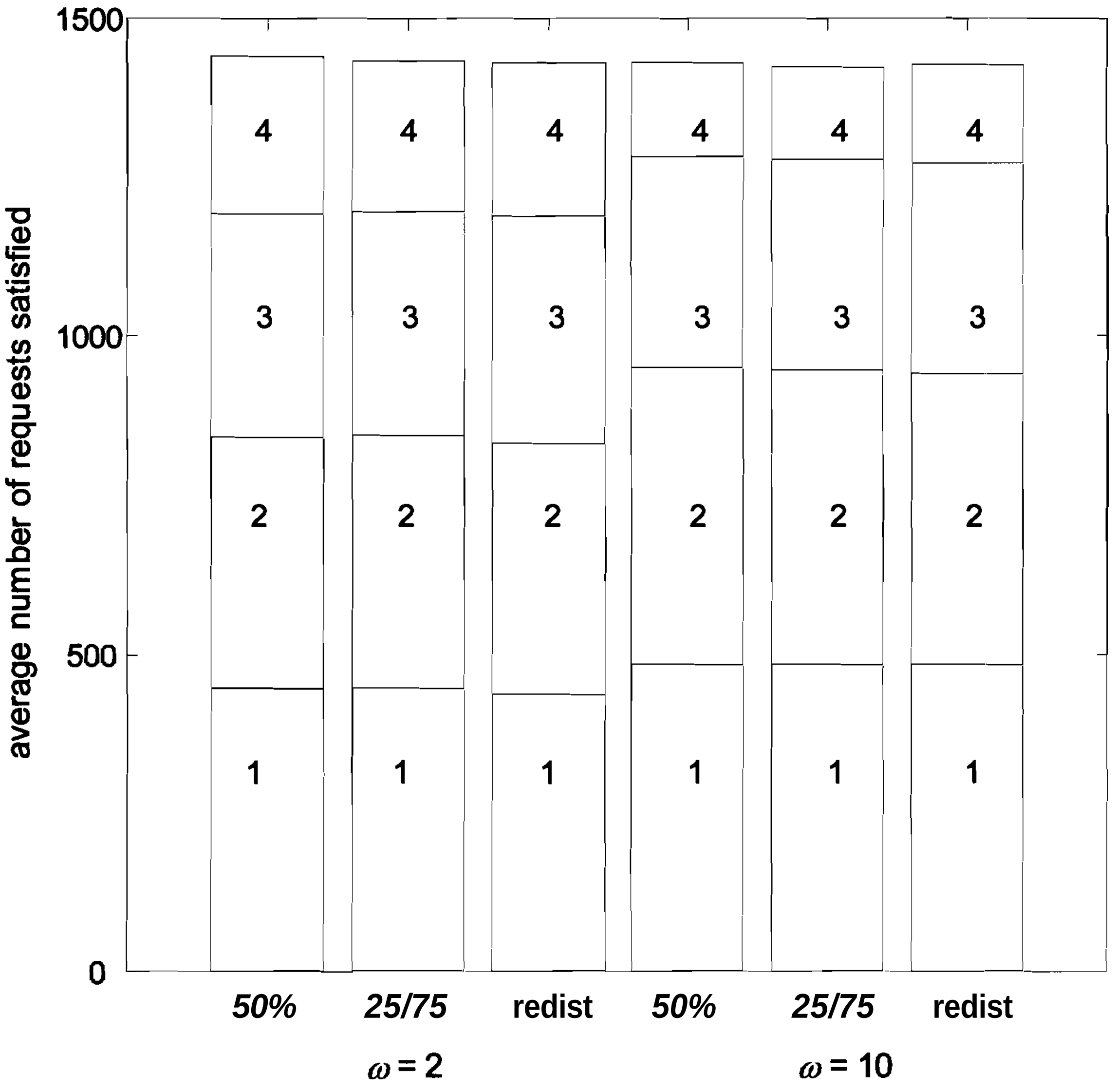

Figure 8.11. Comparison of the average number of requests of priority level 1, 2, 3, and 4 satisfied by the heuristic variations in mode value $\omega = 2$ and $\omega = 10$ with loading factor = 0.7 and GF = 5%.

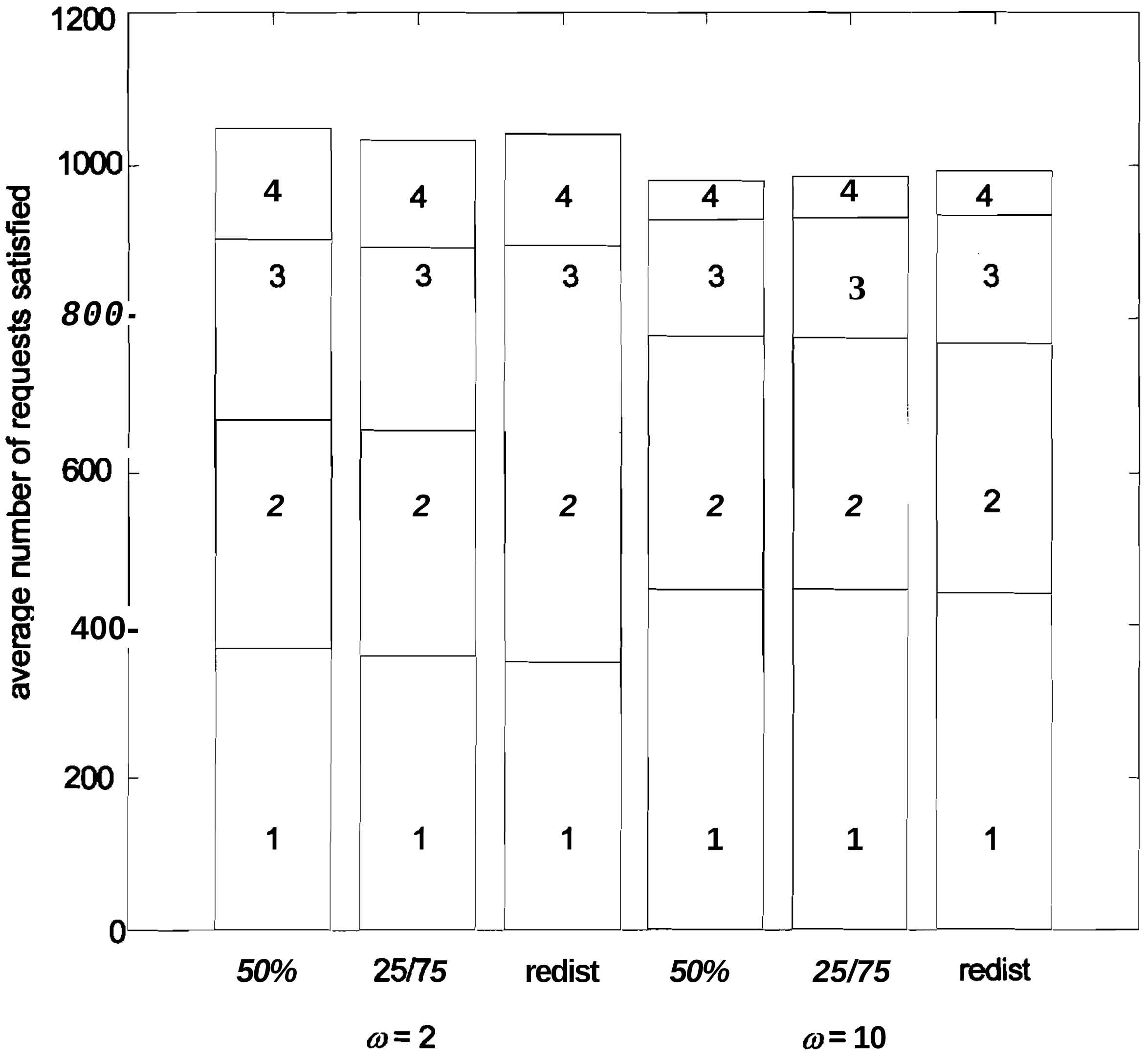

Figure 8.12. Comparison of the average number of requests of priority level 1, 2, 3, and 4 satisfied by the heuristic variations in mode value $\omega = 2$ and $\omega = 10$ with loading factor = 1.2 and GF = 0%.

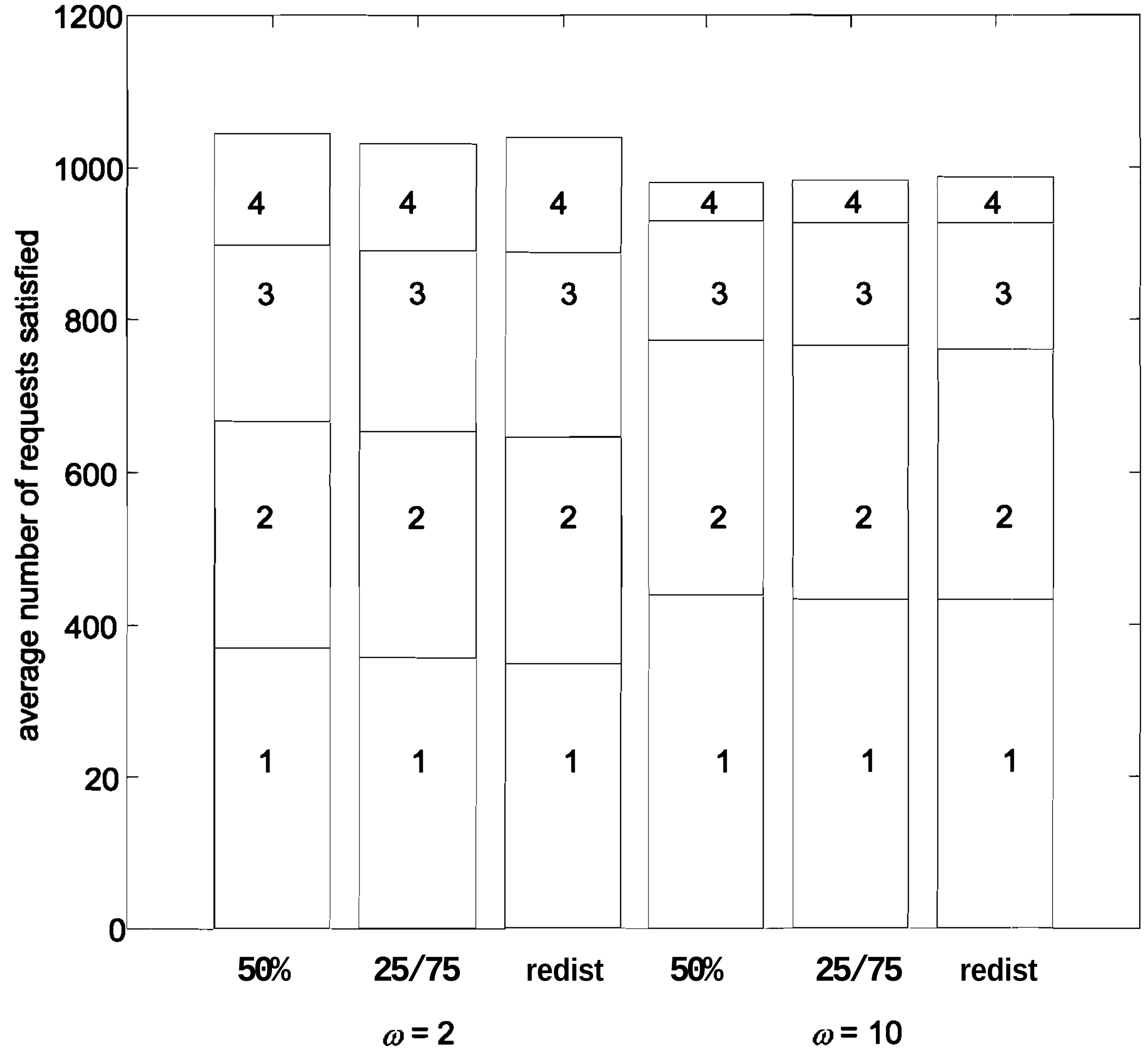

Figure 8.13. Comparison of the average number of requests of priority level 1, 2, 3, and 4 satisfied by the heuristic variations in mode value w = 2 and w = 10 with loading factor = 1.2 and GF = 5%.

The number of priority level one requests satisfied by the redistribute variation is slightly less than that satisfied by the 50% and the 25/75 variation. But, the overall *sum* of the **worths** of the satisfied requests is approximately equal to that obtained by the other two variations. This can be attributed to the fact that the redistribute variation actually considers the worth of redistribution, **i.e.,** it may preempt a higher priority request if the excess bandwidth (bandwidth obtained by preemption of request **minus** the bandwidth needed) can be redistributed among other requests with an increase in total worth. The 50% and the 25/75 variation of the heuristic do not consider the worth of redistribution when making decisions regarding preemption of requests with step utility functions. At the same time, the redistribute variation achieves approximately the same overall worth, with less number of degradations/preemptions of requests as compared to the 50% variation

In general, the number of **degradations/preemptions** of requests with concave utility **functions** is much higher than the number of degradations/preemptions of requests with linear utility functions. This is because, when a request with a linear utility function is degraded, either all the bandwidth needed is obtained **from** the request, or the request is preempted. Hence, when a linear utility **function** is **degraded/preempted,** there is only a single change in bandwidth. In contrast, the concave utility **function** is degraded in steps (of unit size). **After** degrading the request with the concave utility **function** by some units of bandwidth, the marginal worth of the request may no longer be the smallest. Hence, some other request may be degraded. However, the request with concave utility **function** may be degraded again for the scheduling event. **Thus,** a request with a concave utility **function** may be degraded multiple times at each scheduling event.

A request may be degraded by **different** amounts at each scheduling event with every total change in bandwidth allocation to the request at a event being counted as one degradation. The total number of degradations of a request is the *sum* **of** the degradations at all scheduling events.

The execution times for the heuristic variations were calculated for one simulation experiment with a mode value of $\omega = 10$, **lf** = 1.2, and GF = 0%. The 50% variation took the least amount of time: 52 ms per request. The 25/75 variation compares the bandwidth

Table 8.1. Number of degradations and preemptions for requests with step (S), linear (L), and concave (C) utility functions for loading factors (lf) of 0.7 and 1.2, GF = 0%, and mode values $\varpi = 2$ and $\varpi = 10$.

	50%			25/75			redistribute		
	L	C	**S**	**L**	C	**S**	**L**	C	**S**
If = 0.7 **ϖ = 2** GF = 0%	194	80689	174	217	51643	131	169	60610	152
If = 0.7 **ϖ =** 10 GF = 0%	333	26770	187	353	19511	156	289	22328	176
If = 1.2 0 = 2 GF = 0%	461	187398	314	485	116646	253	352	154320	273
If = 1.2 **ϖ =** 10 GF = 0%	634	50728	336	667	39864	313	608	49778	329

Table 8.2. Number of degradations and preemptions for requests with step (S), linear (L), and concave (C) utility functions for loading factors (lf) of 0.7 and 1.2, GF = 5%, and mode values $\omega = 2$ and $\omega = 10$.

	50%			25175			redistribute		
	L	C	S	L	C	S	L	C	S
lf = 0.7 $\omega = 2$ GF = 5%	196	75761	177	213	51963	124	163	57007	149
lf = 0.7 $\omega = 10$ GF = 5%	325	27039	188	350	18988	158	278	25102	171
lf = 1.2 $\omega = 2$ GF = 5%	481	198868	322	487	115597	254	465	154357	320
lf = 1.2 $\omega = 10$ GF = 5%	617	49968	331	662	39740	296	595	51067	331

needed to two thresholds (in one case of the variation), and hence takes slightly more time: 75 ms per request. The redistribute variation actually calculates the worth due to redistribution of excess bandwidth to other requests and hence it takes more time than the other two variations. The execution time of the redistribute variation is 250 ms per request.

8.3. Summary

The results of the simulation experiments conducted in this research were presented in this section. The performance of the heuristic variations has been compared to the simple scheduling technique (complete sharing policy) and upper bounds. The globalization factor did not achieve any significant improvement in the performance of the heuristic variations. The conclusions and suggestions for future work will be discussed in the next section.

9. CONCLUSIONS AND FUTURE WORK

Bandwidth allocation is an important problem in current networks in view of the different types of applications using the same network and each application having different quality of service requirements. In dynamic bandwidth allocation, the users do not reserve (with a guarantee) fixed amounts of bandwidth, but are dynamically allocated the bandwidth. Heuristic variations were developed that attempt to schedule the requests such that the total worth of satisfied requests over a given interval of time is the maximum.

Three different heuristic variations were developed in this research. The different parameters considered were the network loading, a globalization factor, and the relative weights of the different priority levels. Three different types of utility functions of the requests, step, linear, and concave, were considered. The performance of three heuristic variations were shown and compared to the three upper bounds and one simple scheduling technique (based on the complete sharing policy). The results presented showed that the three variations perform comparably to each other. Although the total sum of the worths of the satisfied requests obtained by the three heuristic variations are similar, the heuristic variations degrade/preempt the requests differently.

The 50% variation of the heuristic achieves a high total sum of' worth of satisfied requests, but it also has the maximum number of preemptions of requests with step utility function and has the maximum number of degradation/preemptions of requests with concave utility function. The 25/75 heuristic variation reduces the number of degradations/preemptions for requests with concave utility functions and also reduces the the number of preemptions for the step utility function while achieving a comparable total sum of worths satisfied. But the number of preemptions/degradations of requests with linear utility functions is higher than the corresponding number for the 50% and the redistribute variation of the heuristic. In contrast, the redistribute variation of the heuristic has least number of degradations/preemptions of requests with a linear utility function, and comparable number of degradations/preemptions of requests with step and concave utility functions. The performance of the redistribute function is comparable to both the 50% and the 25/75 variation. The 50% variation has the smallest execution time. All variations improve on the complete sharing policy (as much as 90%).

Future work will consider determining the values of the globalization factor that will result in a significant improvement in the performance of the heuristic. As discussed earlier, in the heuristic variations, only the request one position above the request with a step utility function is used to estimate the loss of worth. Future work will consider looking at more requests to calculate the loss of worth. Future work will also consider utility functions of requests other than step, linear, and concave, e.g., a multi-point step function (for requests with layered encoded data [Sha92]).

The heuristic variations developed first consider the new request's ingress link and then the new request's egress link to calculate the degradations of existing and new requests. Future work may consider both the links simultaneously when calculating the degradations of the existing and new requests.

Whether a request has already begun transmitting or how soon the request will complete transmitting is not considered when choosing among requests to preempt. Future work may consider these factors.

LIST OF REFERENCES

[AIC98] "Agile Information Control Environment proposers information package," BAA 98-26, May 1998, http://webext2.darpa.mil/iso/aice/AICE98-26.htm.

[ATM99] ATM forum, ATM forum traffic management specification version 4.1, Mar. 1999, http://www.atmforum.com/atmforum/specs/approvedhtml/af-tm-0121.000.pdf

[AwA93] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive of on-line routing," ***34th IEEE Symposium on Foundations of Computer science,*** Nov. 1993, pp. 32-40.

[AwB94] B. Awerbuch, Y. Azar, A. Fiat, and A. Rosen, "Competitive non-preemptive call control," ***5th ACM-SIAM Symposium on Discrete Algorithms,*** Jan. 1994, pp. 312-320.

[BaC99] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, "Bandwidth allocation with preemption," ***SIAM Journal on Computing,*** Vol. 28, No. 5, May 1999, pp. 1806-1828.

[BaM98] A. Bar-Noy, Y. Mansour, and B. Schieber, "Competitive dynamic bandwidth allocation," ***7th ACM Symposium on Principles of Distributed Computing,*** June 1998, pp. 31-39.

[BoM98] S. C. Borst and D. Mitra, "Virtual partitioning for robust resource sharing: Computational techniques for heterogeneous traffic," ***IEEE Journal on Selected Areas in Communications,*** Vol. 16, No. 5, June 1998, pp. 668-678.

[BrS99] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogenous computing systems," ***8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99),*** Apr. 1999, pp. 15-29.

[CoL90] T. Cormen, C. Leiserson, and R. Rivest, ***Introduction to Algorithms,*** MIT Press, Cambridge, MA, 1990.

[Com95] D. E. Comer, *Internetworking* with *TCP/IP*. Vol. I: Principles, Protocols, and Architecture, Third Edition, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1995.

[CrB97] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic:Evidence and possible causes," *IEEE/ACM* Transactions on Networking, Vol. 5, No. 6, Dec. 1997, pp. 835-846.

[Dha00] P. N. Dharwadkar, *Dynamic* Bandwidth Allocation with Preemption and Degradation of Prioritized Requests, Master's Thesis, School of Electrical and Computer Engineering, Purdue University, 2000.

[Fel00] A. Feldman, "Characteristics of TCP connection arrivals," in Self Similar Network Traffic and *Performance* Evaluation, K. Park and W. Willinger, John Wiley and Sons, Inc., 2000, pp. 367-399.

[FeM95] J. Fernandez and M. Mutka, "Model and call admission control for distributed applications with correlated bursty traffic," *ACM/IEEE* Supercomputing Conference (Supercomputing *'95)*, Vol. 1, Dec. 1995, pp. 92-113.

[FlJ95] S. Floyd and V. Jacobson, "Link-sharing and resource management for packet networks," *IEEE/ACM* Transactions on *Networking*, Vol. 3, No. 4, Aug. 1995, pp. 365-386.

[FuR97] E. Fulp and D. Reeves, "On-line dynamic bandwidth allocation," IEEE International Conference on *Network* Protocols, Oct. 1997, pp. 134-141.

[Kel97] F. P. Kelly, "Charging and rate control for elastic traffic," European Transactions on Telecommunications and Related *Technologies*, Vol. 8, No. 1, Jan. 1997, pp. 33-37.

[KiH00] J. K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. John, C. Irvine, T. Levin, N. W. Porter, V. Prasanna, and R. F. Freund, "A QoS performance measure framework for distributed heterogeneous networks," 8th *Euromicro* Workshop on Parallel and Distributed Processing (PDP *2000)*, Jan. 2000, pp. 18-27.

[Low00] S. Low, "Equilibrium bandwidth and buffer allocations for elastic traffics," *IEEE/ACM* Transactions on *Networking*, 2000, to appear.

[LeT94] W. E. Lelan, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM* Transactions on *Networking*, Vol. 2, No. 1, Feb. 1994, pp. 1-15.

[MaA99] M. Maheshwaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, "Dynamic mapping of a class of independent tasks onto heterogenous computing systems," Journal of Parallel and Distributed Computing, Vol. 59, No. 2, Nov. 1999, pp. 107-131.

[McJ96] S. McCanne and V. Jacobson, "Receiver-driven layered multicast," ACM *SIGCOMM'96*, Aug. 1996, pp. 117-130.

[MiM96] D. Mitra, J. Morrison, and K. Ramakrishnan, "ATM network design and optimization: A multirate loss network framework," *IEEE/ACM* Transactions on Networking, Vol. 4, No. 4, Aug. 1996, pp. 531-543.

[NAP98] Chicago NAP, Jan. 1998, http://nap.aads.net/main.html.

[PaK96] K. Park, G. Kim, and M. Crovella, "On the relationship between file sizes, transport protocols, and self-similar traffic," International Conference on Network Protocols *(ICNP '96)*, Oct. 1996, pp. 171-180.

[PiW00] C. Piatko, I. Wang, and S. Khuller, "Resource allocation for agile information control," manuscript in preparation, Johns Hopkins University, Applied Physics Lab, MD, 2000.

[ReR95] D. Reininger, G. Ramamurthy, and D. Raychaudhuri, "VBR MPEG video coding with dynamic bandwidth renegotiation," IEEE International Conference on Communications *(ICC '95)*, Vol. 3, No. 1, June 1995, pp. 1773-1777.

[ReR98] D. Reininger, D. Raychaudhuri, and M. Ott, "Dynamic quality of service framework for video in broadband networks," IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, Vol. 12, No. 6, Nov. 1998, pp. 22-34.

[Sha92] N. Sbacham, "Multipoint communication by hierarchically encoded data," IEEE *INFOCOM* '92, May 1992, pp. 2107-2114.

[She95] S. Shenker, "Fundamental design issues for the future Internet," IEEE Journal Selected Areas Communication, Vol. 13, No. 7, Sep. 1995, pp. 1176-1188.

[ThB00] M. D. Theys, N. B. Beck, H. J. Siegel, and M. Jurczyk, "Evaluation of expanded heuristics in a heterogeneous distributed data staging network," 9th Heterogeneous Computing Workshop *(HC*W*2000)*, May 2000, pp. 75-89.

[ThS00] M. D. Theys, H. J. Siegel, and E. K. P. Chong, "Heuristics for scheduling data requests using collective communications in a distributed communication network," Journal of Parallel and Distributed Computing, Special Issue on Routing in Computer and Communication Systems, to appear.

[ThT00a] P. Thomas, D. Teneketzis, and J. K. MacKie-Mason, A Market-Based Approach to Optimal Resource Allocation in Integrated-Services *Connection-*Oriented Networks, Technical Report, CGR-99-03, Systems Science and Engineering Division, Department of Electrical Engineering and Computer Science, University of Michigan, Mar. 1999, 40 pp.

[ThT00b] M. D. Theys, M. Tan, N. Beck, H. J. Siegel, and M. Jurczyk, "A mathematical model and scheduling heuristics for satisfying prioritized data requests in an oversubscribed communication network," IEEE Transactions on Parallel and Distributed Systems, to appear 2000.

APPENDIX A. PSEUDO-CODE FOR THE “50% HEURISTIC"

```
main

 1.  begin
 2.  if ( current–time $\geq$ (newreq_starttime – lead–time) )
         /*lead–time needed for executing heuristic = 120 secs*/
 3.      request cannot be scheduled before start time
 4.      reject request
 5.  else
 6.      check resource allocation table (RAT) to determine
         whether there is available bandwidth
 7.      if (available bandwidth $\geq$ newreq_bandwidth) at
        every time instant of the new request's session at source and
        destination links
 8.          request can be satisfied
 9.          admit request
10.              store the request in the RAT
11.      else
12.          call schedule(newreq)
13.  end
```

schedule(newreq)

```
1.  begin
2.  total–old–worth = sum of worths of scheduled requests
3.  link = newreq_ingresslink
4.  L = link–bandwidth
5.  check–at–time = newreq_starttime
6.  while (check–at–time $\le$ newreq_endtime)
7.      find set of requests conflicting with newreq at time check–at–time
8.      assign these requests to array R   /*numbered 1 to n – 1 */
9.      let number of conflicting requests be n – 1
10.     R[n] = newreq    /*add newreq to the array R */
11.     B = total bandwidth of requests in R
12.     bwneeded = B – L
13.     next–check–at–time = function_find_next_event(link, check–at–time)
14.     while (bwneeded > 0)
15.         order the requests in R by decreasing marginal worth
                    /* last request is the request with least marginal worth*/
16.         amount–bw–deg = degrade(R,n,bwneeded)
                                        /* degrade the request in R[n]*/
17.         bwneeded = (bwneeded – amount–bw–deg)
18.     end while
19.     if (bwneeded < 0)
20.         reallocate(– bwneeded)
        /* request satisfied at one time interval, check at next time interval */
21.     check–at–time = next–check–at–time
22. end while
23. for any request degraded at this link
24.     if this link was the request's ingress link
25.         degrade that request at its egress link by the same amount.
26.     else
27.         degrade that request at its ingress link by the same amount.
28. repeat steps 4-27 with link = newreq_egresslink
29. let modified–worth be worth of scheduled requests (including newreq)
    after executing steps 3-28                     /* explained in text */
30. if (modified–worth > total–old–worth)
31.     admit new request
32. else
33.     reject new request
34.     restore to all requests the bandwidths degraded/preempted during this
        call
35. end
```

degrade(*R*, n, bwneeded)

```
 1.  begin
 2.  if utility function of R[n] = linear function
 3.      amt_bw_deg = degrade_linear(R, n, bwneeded)
 4.  if utility function of R[n] = concave function
 5.      amt_bw_deg = degrade_concave(R, n, bwneeded)
 6.  if (utility function of R[n] = step fun    on)
 7.      if (bwneeded ≥ bandwidth of R[n])
 8.         preempt request in R[n]
 9.         amt_bw_deg = bandwidth of R[n]
10.      else
11.         amt_bw_deg = degrade_step_50(R, bwneeded)
12.  return amt_bw_deg
13.  end
```

cti

```
degrade_step_50%(R, bwneeded)

/* 50% variation of the heuristic is being used *I
I* R[n] is a request with a step utility function */

1.   begin
2.   if (bwneeded ≥ (0.5 x bandwidth of R[n]))
3.       preempt request in R[n]
4.       amt_bw_degraded = bandwidth of R[n]
5.   else
6.       I* bandwidth needed (bwneeded) < 0.5 x bandwidth of R[n], check
         whether the loss of worth due to degradation1preemption of R[n -1] is
         more than the worth of the request R[n] */
7.       if worth of R[n] < loss_of_worth(R, (n-1), bwneeded)
8.           preempt the request in R[n]
9.           amt_bw_degraded = bandwidth of R[n]
10.      else
11.          if (utility function of R[n-1] = linear function)
12.              amt-bw-deg = degrade_linear(R, n-1,bwneeded)
13.          if (utility function of R[n-1] = concave function)
14.              amt-bw-deg = degrade_concave(R, n -1, bwneeded)
15.          if (utility function of R[n-1] = step function)
16.              preempt request in R[n-1]
17.              amt-bw-deg = bandwidth of R[n-1]
18.  return amt-bw-deg
19.  end
```

find_next_event(link, check–at–time)

```
1.  for every existing–request utilizing link do
2.      find the earliest start time of an existing request that is later than the
        check–at–time and earlier than the end time of the new request
3.      find the earliest end time of an existing request that is later than the
        check–at–time and earlier than the end time of the new request
4.  if new request is the request ending the earliest and no existing request
    begins before or at newreq_endtime
5.      next–event = newreq_endtime + 1
6.      return next–event
7.  else
8.      assign the earliest start time to temp–stime
9.      assign the earliest end time to temp–etime
10.     temp–etime = temp–etime + 1
11.     if (temp–stirne > temp–etime)
12.         next–event = temp–etime
13.     else
14.         next–event = temp–stime
15.     return next–event
16. end
```

reallocate(bw)

/* bw is the bandwidth that is to be reallocated to the requests*/

```
1.    begin
2.    request–num = 0
3.    while (bw > 0 and request–num ≤ n)
4.        if (R[request_num]'s utility function = step function )
5.            request–num = request–num + 1
6.            continue (go to step 3)
7.        if (R[request_num]'s utility function = linear function)
8.            if (R[request_num]_allocated_ bandwidth <
              R[request_num]_desired_bandwidth )
9.                if ((R[request_num]_desired_bandwidth –
              R[request_num]_allocated_bandwidth) ≥ bw)
10.                   R[request_num]_allocated_bandwidth =
                      R[request_num]_allocated_bandwidth + bw
11.                   return
12.               else
13.                   R[request_num]_allocated_bandwidth =
                      R[request_num]_desired_bandwidth
14.                   bw = bw – (R[request_num]_desired_bandwidth –
                                R[request_num]_allocated_bandwidth)
15.                   request–num = request–num + 1
16.                   continue (go to step 3)
17.           else
18.               request–num = request–num + 1
19.               continue (go to step 3)
20.       while((R[request_num]'s marginal worth is the highest) and (bw > 0)
                and (R[request_num]_allocated_bandwidth <
                    R[request_num]_desired_bandwidth) )
21.           R[request_num]_allocated_bandwidth =
              R[request_num]_allocated_bandwidth + 1
22.           bw = bw – 1
23.       end while
24.       if (bw > 0)
25.           reinsert R[request_num] in the proper order in the list
26.           request–num = 0
27.   end while
28.   return
29.   end
```

```
degrade–linear(R, x, bwneeded)

1.    begin
2.    if (bwneeded < (current bandwidth of R[x] – minimum bandwidth of R[x]) )
3.         degrade bandwidth allocated to R[x] by amount = bwneeded
           /* bandwidth of a request is degraded until the next event */
4.         amt_bw_degraded = bwneeded
5.     else
6.         preempt request in R[x]
7.         amt_bw_degraded = bandwidth of R[x]
8.     return amt_bw_deg
9.     end
```

```
degrade_concave(R, x, bwneeded)

1.     begin
2.     amt_bw_deg = 0
3.     while (R[x]_marginal worth ≤ R[x –1]_marginal_worth)
4.          degrade bandwidth allocated to R[x] by unit amount
            /* bandwidth of a request is degraded until the next event */
                                       /* unit amount is 1Kbps*/
5.          amt_bw_deg = amt_bw_deg + 1
6.          R[x]_marginal worth = new marginal worth
                                       /*after degradation by unit amount */
7.     end while
8.     retum amt_bw_deg
9.     end
```

loss_of_worth(bwneeded, R, reqnum)

```
1.    begin
2.    if (bwneeded < (current bandwidth of R[reqnum] -
                         minimum bandwidth of R[reqnum]))
3.        loss–worth = (worth of R[reqnum] at current bandwidth -
                  worth of R[reqnum] at (current bandwidth–bwneeded))
4.    else
5.        loss–worth = worth of R[reqnum] at current bandwidth
6.    return loss–worth
7.    end
```

APPENDIX B. PSEUDO-CODE FOR "25/75 HEURISTIC"

Same as the 50% heuristic in Appendix A, except:

(a) step 11, of degrade(R, n, bwneeded) is:

11. amt_bw_deg = degrade_step_25/75(*R*, bwneeded)

(b) the routines degrade_step_25/75(*R*, bwneeded) and loss_of_worth_2(bwneeded, R, reqnum) are used (see the following pages).

degrade_step_25/75(*R*, bwneeded)

/* 25/75 variation of the heuristic is being used */
/* *R*[*n*] is a request with a step utility function */

```
1.  begin
2.  if (bwneeded ≥ (0.75 x bandwidth of R[n]))
3.      preempt request in R[n]
4.      amt_bw_deg = bandwidth of R[n]
5.  else
6.      if (bwneeded ≤ (0.25 x bandwidth of R[n]))
7.      /* bandwidth needed (bwneeded) ≤ 0.25 x bandwidth of R[n], check
        whether the loss of worth due to degradation/preemption of R[n -1]
        is more than the worth of the request R[n] */
8.          if worth of R[n] < loss_of_worth(bwneeded, R, (n-1) )
9.              preempt the request in R[n]
10.             amt-bw-deg = bandwidth of R[n]
11.         else
12.             if utility function of R[n-1] = linear function
13.                 amt_bw_deg = degrade_linear(R, n-1, bwneeded)
14.             if utility function of R[n-1] = concave function
15.                 amt-bw-deg = degrade_concave(R, n-1, bwneeded)
16.             if utility function of R[n-1] = step function
17.                 preempt request in R[n-1]
18.                 amt-bw-deg = bandwidth of R[n-1]
19.     else
        /* bwneeded is more than 25% and less than 75% of bandwidth of
          R[n] */
20.         if worth of R[n] < loss_of_worth_2(R, (n-1), bwneeded)
21.             preempt the request in R[n]
22.             amt-bw-deg = bandwidth of R[n]
23.         else
24.             if utility function of R[n-1] = linear function
25.                 amt_bw_deg = degrade_linear(R, n-1, bwneeded)
26.             if utility function of R[n-1] = concave function
27.                 amt-bw-deg = degrade_concave(R, n-1, bwneeded)
28.             if utility function of R[n-1] = step function
29.                 preempt request in R[n-1]
30.                 amt_bw_deg = bandwidth of R[n-1]
31. return amt-bw-deg
32. end
```

```
loss_of_worth_2(bwneeded, R, reqnum)

1.    begin
2.    if bwneeded ≥ R[reqnum]_current_bandwidth +
                        R[reqnum–1]_current_bandwidth
        /* bwneeded is greater than the sum of bandwidths of R[reqnum] and
           R[reqnum–1], so loss of worth will be the current worths of R[reqnum]
           and R[reqnum–1] */
3.      loss–worth = worth of R[reqnum] at current bandwidth +
                        worth of R[reqnum–1] at current bandwidth
4.    else
        /* bwneeded is less than sum of the bandwidths of R[reqnum] and
        R[reqnum–1], calculate the loss of worth by considering a set of
        conflicting requests R' consisting of requests R[reqnum] and
        R[reqnum–1] and executing the steps 13-20 of the schedule function
        for R'. find the degraded bandwidths of R[reqnum] and R[reqnum–1]
        and calculate the loss of worth due to degradation */
5.      R[reqnum]_original_bw = R[reqnum]_current_bandwidth
6.      R[reqnum–1]_original_bw = R[reqnum–1]_current_bandwidth
7.      generate a set of conflicting requests R' consisting of requests
        R[reqnum] and R[reqnum–1] only
8.      repeat steps 13-20 of "schedule" with the set R' instead of R
9.      loss–worth = (worth of R[reqnum] at R[reqnum]_original_bandwidth –
                        worth of R[reqnum] at R[reqnum]_current_bandwidth) +
                       (worth of R[reqnum–1] at R[reqnum–1]_original_bandwidth
                        – worth of R[reqnum] at R[reqnum]_current_bandwidth)
10.    return loss–worth
11.   end
```

APPENDIX C. PSEUDO-CODE FOR "REDISTRIBUTE HEURISTIC"

Same as the 50% heuristic in Appendix A, except:

(a) step 11, of degrade(R, n, bwneeded) is:

```
11.    amt_bw_deg = degrade_step_25/75(R, bwneeded)
```

(c) the routines degrade_step_redistribute(*R*, bwneeded) and worth_of_redistribution(bw) are used (see the following pages).

```
degrade_step_redistribute(R, bwneeded)
/* redistribute variation of the heuristic is being used */
/* R[n] is a request with a step utility function */

 1.  begin
 2.  if ( (worth of request R[n]) $\leq$ worth_of_redistribution(bandwidth of R[n] –
                                       bwneeded) )
 3.      preempt request in R[n]
 4.      amt–bw–deg = bandwidth of R[n]
 5.  else
     /* worth of request R[n]) $\geq$ worth_of_redistribution(bandwidth of R[n] –
                                       bwneeded) */
 6.      if (bwneeded < bandwidth of R[n–1])
 7.          if ( (worth of request R[n] – worth_of_redistribution(bandwidth of
             R[n] – bwneeded) ) ) $\leq$ (worth of R[n–1] at current bandwidth –
             worth of R[n–1] at (current bandwidth – bwneeded) ) )
 8.              preempt request in R[n]
 9.              amt–bw–deg = bandwidth of R[n]
10.          else
11.              if utility function of R[n–1] = linear function
12.                  amt_bw_deg = degrade_linear(R, n–1, bwneeded)
13.              if utility function of R[n–1] = concave function
14.                  amt–bw–deg = degrade_concave(R, n–1, bwneeded)
15.              if utility function of R[n–1] = step function
16.                  preempt request in R[n–1]
17.                  amt–bw–deg = bandwidth of R[n–1]
18.      else
         /* bwneeded $\geq$ bandwidth of R[n–1]) */
19.          if (bwneeded $\geq$ (0.5 x bandwidth of R[n]))
20.              preempt request in R[n]
21.              amt–bw–deg = bandwidth of R[n]
22.          else
23.          /* bandwidth needed (bwneeded) < 0.5 x bandwidth of R[n], check
             whether the loss of worth due to degradation1preemption of R[n –1]
             is more than the worth of the request R[n] */
24.              if ( (worth of R[n] – worth_of_redistribution(bandwidth of R[n] –
                     bwneeded)) < loss_of_worth(bwneeded, R, (n–1) ) )
25.                  preempt the request in R[n]
26.                  amt_bw_deg = bandwidth of R[n]
27.              else
28.                  preempt R[n–1]
29.                  amt–bw–deg = bandwidth of R[n–1]
30.  return amt–bw–deg
31.  end
```

worth_of_redistribution(bw)

```
1.  begin
2.  request-num = 0
3.  inc-worth = 0
4.  while (bw > 0 and request-num ≤ n)
5.      if (R[request_num]'s utility function = step function )
6.          request-num = request-num + 1
7.          continue /* go to step 4 */
8.      if (R[request_num]'s utility function = linear function)
9.          if (R[request_num]_allocated_ bandwidth <
               R[request_num]_desired_bandwidth )
10.            if ((R[request_num]_desired_bandwidth -
               R[request_num]_allocated_bandwidth) ≥ bw)
11.                inc-worth = inc-worth + worth of R[request_num] at
                   (R[request_num]_current_bandwidth + bw) -
                   worth of R[request_num] at current bandwidth
12.                return inc-worth
13.            else
14.                inc-worth = inc-worth + worth of R[request_num] at desired
                   bandwidth - worth of R[request_num] at current bandwidth
15.                bw = bw - (R[request_num]_desired_bandwidth -
                                R[request_num]_current_bandwidth)
16.                request-num = request-num + 1
17.                continue /* go to step 4 */
18.         else
19.             request-num = request-num + 1
20.             continue  /* go to step 4 */
21.     while((R[request_num]'s marginal worth is the highest) and (bw > 0)
               and (R[request_num]_current_bandwidth <
                     R[request_num]_desired-bandwidth))
22.         R[request_num]_current_bandwidth =
            R[request_num]_current_bandwidth + 1
23.         inc-worth = inc-worth + (worth of  R[request_num] at current
                           bandwidth +1) - worth of R[request_num] at current
                           bandwidth)
24.         bw = bw - 1
25.     end while
26.     if (bw > 0)
27.         reinsert R[request_num] in the proper order in the list
28.         request-num = 0
29. end while
30. return
31. end
```

APPENDIX D. GLOSSARY OF NOTATION

r_k	a request arriving at the ingress node
i_k	ingress link **of** the request r_k
o_k	egress link of the request r_k
s_k	start time of the request r_k
d_k	end time of the request r_k
$b_k(t)$	bandwidth received by the request r_k at time t
session of a request	$(d_k - s_k)$
u_k	utility of the request r_k, which **is** a function of the bandwidth received by the request r_k
U_k	total utility of **the** request r_k obtained by summing the utilities at every time instant of the request's session
w_k	worth of the request r_k that is the product of the weighted priority and the total utility of the request
W	worth of all the requests satisfied in a given interval of time; W is the performance measure of the system
rb_k	requested bandwidth of the request r_k

mb_k	minimum bandwidth required by the request r_k
p_k	priority level of the request r_k, $1 \leq p_k \leq 4$
ϖ	weighting constant, which depends upon the mode; in mode two $\varpi = 2$, in mode ten $\varpi = 10$
$\Pi(p_k)$	weighting function for a priority level that depends upon the mode value.
sim_time	simulation time over which the sum of the worths of the requests satisfied is calculated
loading factor	indicates the amount of "load" that is placed on the network.
arrival rate λ	arrival rate of requests modelled as a Poisson arrival sequence; λ depends upon the loading factor
lead time	time between the arrival time and the start time of the request
bwneeded	bandwidth still needed to be obtained by degradationlpreemption of some request(s)
GF	globalization factor, a factor used to to introduce randomness in decision making