Group Priority Scheduling*

Simon S. Lam and Geoffrey G. Xie
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{lam,xie}@cs.utexas.edu

Abstract

For many applications, the end-to-end delay of an application-specific data unit is a more important performance measure than the end-to-end delays of individual packets within a network. From this observation, we propose the idea of group scheduling. Specifically, consecutive packet arrivals in a flow are partitioned into groups, and the same deadline (called group priority) is assigned to every packet in a group. In this paper, we first present an end-to-end delay guarantee theorem for a network of guaranteed-deadline (GD) servers. The theorem can be instantiated to obtain end-to-end delay bounds for a variety of source control mechanisms and GD servers. We then specialize the delay guarantee theorem to group scheduling for a subclass of GD servers. We work out a detailed example to demonstrate how to use group scheduling in a particular class of networks. The advantages of group scheduling are discussed and illustrated with empirical results from simulation experiments.

1 Introduction

Consider packet-switching networks delivering packets from sources to destinations. A flow is modeled as a sequence of packets. Some flows traversing the network are said to be *guaranteed*, because the network ensures an end-to-end delay upper bound for each packet in these flows. Conceptually, a network provides end-to-end delay bounds by implementing each communication channel as a guaranteed-deadline (*GD*) *server*. At a GD server, each packet arrival from a guaranteed flow is given a *deadline* (also called *priority*), and the server ensures that the packet departs by the deadline. Many GD service disciplines have been proposed [1, 2, 3, 4, 9, 13].

For most applications, a flow is generated by its source as a sequence of messages, each of which is then segmented and transported as a sequence of packets by the network.[1] To these applications, the end-to-end delay of a message is a more important performance measure than the end-to-end delay of a packet. This observation motivated the idea of *group scheduling*, namely: packets arriving at a GD server from a guaranteed flow are partitioned into groups. The same deadline (called *group priority*) is assigned to every packet in the same group.

Group scheduling has two advantages. First the priority of a flow changes less frequently, i.e., from one group to the next rather than from one packet to the next. Hence, the channel scheduler's work in updating its priority data structure (e.g., a heap) would be much reduced for large groups. (An empirical investigation quantifying this reduction can be found in [11].) Second, group scheduling offers more flexible deadlines; consequently, channel schedulers can better cope with temporary overloads.

The balance of this paper is organized as follows. In Section 2, we introduce the class of GD servers and prove an end-to-end delay guarantee theorem. The theorem can be instantiated to obtain end-to-end delay bounds for a variety of source control mechanisms and GD servers; in particular, different GD servers can coexist in the same end-to-end path. With the theorem, the problem of deriving an end-to-end upper bound for a guaranteed flow is reduced to a set of single-node problems.

In Section 3, we introduce and develop the concept of group scheduling. We prove a relaxed deadline theorem for a subclass of GD servers, called the *priority subclass*. The delay guarantee theorem in Section 2 is then specialized to group scheduling for a subclass of GD servers.

In Section 4, we work out a detailed example for a particular class of networks [7]. We derive end-to-end delay bounds for messages (called bursts in [7]), and illustrate how to choose group sizes such that the end-to-end delays of messages are unaffected by group scheduling.

In Section 5, we present empirical results from simulation experiments for MPEG video flows. We show that group scheduling, aside from reducing the channel scheduler's work, has another advantage, i.e., when some channels in a network are severely overbooked (heavily utilized), the relaxed deadlines of group scheduling actually improve the statistical performance (loss rates, delays, and queue sizes) of the network.

*Research supported in part by National Science Foundation grants no. NCR-9004464 and NCR-9506048.

[1] The terms *packet* and *message* are interpreted generally herein. For examples, a packet may be an IP datagram or an ATM cell. A message may be a file or a video picture.

2 End-to-End Delay Guarantee

Consider a packet switching network in which each packet is of variable, bounded length (in number of bits). Each communication channel in the network is statistically shared and will be referred to as a *server*.

We will focus upon a flow, say f, which is a sequence of packets. Packets in the flow traverse a path of $K+2$ nodes. Node 0 is the source of the flow, and node $K+1$ the destination. Nodes 1 through K are servers. The network is to provide an end-to-end delay guarantee to the flow. Such a flow will be called a guaranteed flow. (We do not care whether or not the network also provides delay guarantees to other flows that statistically share the same servers.)

Packets in flow f traverse the path in FIFO order. Specifically, the ordering of packets in flow f is preserved at every node along the path.

Notation for server k

β_k	a nonnegative time constant associated with node k (seconds)
$\tau_{k,k+1}$	channel propagation time from node k to node $k+1$ (seconds); each channel is assumed to be reliable and FIFO
α_k	$= \beta_k + \tau_{k,k+1}$ (seconds)
s_k^{max}	maximum packet size at node k (bits)
C_k	transmission rate of channel from node k to node $k+1$ (bits/second)
p	an arbitrary packet served (the packet may belong to any flow)

Notation for flow f

i, j	indices of flow f's sequence of packets
$A_k^f(i)$	arrival time of packet i at node k (time when last bit of packet arrives)
$P_k^f(i)$	packet-dependent component of packet i's deadline at node k
$L_k^f(i)$	departure time of packet i at node k (time when last bit of packet leaves)
$s^f(i)$	size of packet i (bits)

$A_k^f(i)$, $P_k^f(i)$, and $L_k^f(i)$, for $i \geq 1$, are positive real numbers. Indices i and j are positive integers. Additionally, we also use m, n, and l as positive integers whose meanings depend upon context.

2.1 Guaranteed-deadline servers

A GD server provides the following service to each guaranteed flow f it serves:

- packets in flow f depart in FIFO order
- server ensures that the departure time of packet i is bounded as follows:

$$L_k^f(i) \leq P_k^f(i) + \beta_k \tag{1}$$

where the deadline on the right hand side has two components: (i) a packet-dependent component, $P_k^f(i)$, which depends on packet i (its arrival time, flow, length, priority, etc.), and (ii) a nonnegative constant, β_k. Note that each component may vary from one server to another, and β_k may be zero.

The function $P_k^f(\cdot)$ is not yet specified. Any function may be used so long as for a guaranteed flow, the server can ensure that, for all i, packet i departs by its deadline.

Many service disciplines in the literature belong to this class. They differ in packet deadlines, scheduling algorithms, and admission control conditions. Some examples and references are given in Section 2.4.

2.2 Delay guarantee theorem

Consider a guaranteed flow f traversing the path from node 0 to node $K+1$. Nodes 1 to K are GD servers (different service disciplines may be used at different nodes). The following lemma is immediate from the definition of α_k.

Lemma 1. For packet $i = 1, 2, \ldots$ in flow f and node $k = 1, 2, \ldots, K$, the arrival time of packet i at node $k+1$ is bounded as follows:

$$A_{k+1}^f(i) \leq P_k^f(i) + \alpha_k \tag{2}$$

We next present a general delay guarantee formula for flow f. The formula makes use of *reference clock values* at nodes 1 to K, which are described below.

Notation for flow f

$v^f(i)$	a time constant associated with packet i (seconds)
$V_k^f(i)$	reference clock value of packet i at node k

$v^f(i)$ and $V_k^f(i)$ are positive real numbers, for $i \geq 1$. The time constant, $v^f(i)$, can be interpreted as the service time of packet i (excluding waiting time) guaranteed by each node in the path. $V_k^f(i)$ is determined as follows for $i \geq 1$, with $V_k^f(0)$ defined to be 0 :

$$V_k^f(i) = \max\{V_k^f(i-1), A_k^f(i)\} + v^f(i) \tag{3}$$

Thus $V_k^f(i)$ can be thought of as the *expected finishing time* of packet i at node k, and is to be used as a time reference in our delay guarantee formula for flow f. As such, reference clock values are neither computed nor actually implemented by the nodes.

Node k ensures that packet i departs before its deadline, which is $P_k^f(i) + \beta_k$. Therefore, $A_{k+1}^f(i)$ depends upon $P_k^f(i)$ as shown in (2), and $V_{k+1}^f(i)$ depends upon $A_{k+1}^f(i)$ as shown in (3).

A concrete way to interpret $v^f(i)$ is to assume that packet i has been allocated a throughput of $\lambda^f(i)$ bits/second at each node, such that: $v^f(i) = s^f(i)/\lambda^f(i)$. We note that adaptive throughput allocation on a per packet basis is unrealistic in practice. However, adaptive throughput allocation on a per burst basis has been proposed [7], where each burst consists of a number of packets; see Section 4.

Lemma 2. For packet $i = 1, 2, \ldots$ in flow f and node $k = 1, 2, \ldots, K-1$

$$V_{k+1}^f(i) \leq V_k^f(i) + \max_{1 \leq j \leq i} \{v^f(j) + (P_k^f(j) - V_k^f(j))\} + \alpha_k \quad (4)$$

Theorem 1. For packet $i = 1, 2, \ldots$ in flow f, the arrival time of packet i at the destination is bounded as follows:

$$A_{K+1}^f(i) \leq V_1^f(i) + \sum_{k=1}^{K-1} \max_{1 \leq j \leq i} \{v^f(j) + (P_k^f(j) - V_k^f(j))\} + (P_K^f(i) - V_K^f(i)) + \sum_{k=1}^{K} \alpha_k \quad (5)$$

This is our delay guarantee theorem. By definition, the end-to-end delay for packet i is $A_{K+1}^f(i) - A_1^f(i)$. The delay guarantee in (5) can be instantiated to obtain end-to-end delay upper bounds for a variety of source control mechanisms and different service disciplines at nodes 1 through K. Specifically, the delay guarantee in (5) provides an upper bound on the end-to-end delay of packet i if

- a source control mechanism is chosen such that $V_1^f(i) - A_1^f(i)$ at node 1 has a finite upper bound, and
- a GD server is chosen for node k, $1 \leq k \leq K$, such that the term, $P_k^f(j) - V_k^f(j)$, has a finite upper bound.

Note that different service disciplines may be chosen for different nodes, and the term, $P_k^f(j) - V_k^f(j)$, may be positive or negative. With Theorem 1, *the problem of deriving an end-to-end delay upper bound for a guaranteed flow traversing a network path is reduced to a set of single-node problems.*

2.3 Examples of source control

The goal of source control is to upper bound $V_1^f(i) - A_1^f(i)$. A widely used mechanism is leaky bucket control. If the source of flow f is controlled by a leaky bucket with token rate ρ and bucket depth σ, then for all packet i in the flow [5],

$$V_1^f(i) \leq A_1^f(i) + \frac{\sigma}{\rho} \quad (6)$$

To obtain an end-to-end upper bound for flow f, $V_1^f(i)$ is instantiated to $A_1^f(i) + \sigma/\rho$ in the delay guarantee formula of Theorem 1.

Another example of source control is the separation timing constraint between consecutive bursts in a flow [7]; see Section 4 for more details.

2.4 Examples of GD servers

The GD class of servers is general and includes many service disciplines in the literature. There are differences in their $P_k^f(\cdot)$ functions, β_k constants, scheduling algorithms, and admission control conditions. We next discuss four well-known examples.

For a VC server, the P values are virtual clock values computed as follows [13], for all $j \geq 1$,

$$P_k^f(j) = \max\{P_k^f(j-1), A_k^f(j)\} + v^f(j) \quad (7)$$

where $P_k^f(0) = 0$, and $v^f(j)$ is equal to $s^f(j)/\lambda^f(j)$. Under certain admission control conditions, the VC server provides the guaranteed deadline in (1) with $\beta_k = s_k^{max}/C_k$ [10]. From (3) and (7), it is trivial to show that, for all j,

$$P_k^f(j) - V_k^f(j) = 0 \quad (8)$$

For a PGPS server, $P_k^f(j)$ is the *virtual-time finishing time* of packet j. Under certain admission control conditions, the PGPS server provides the guaranteed deadline in (1) with $\beta_k = s_k^{max}/C_k$ [9]. It is shown in [5] that if the server allocates a minimum rate of λ^f to every packet of flow f, such that $v^f(j) = s^f(j)/\lambda^f$, then the following holds for all j,

$$P_k^f(j) - V_k^f(j) \leq 0 \quad (9)$$

For a Delay-EDD server [2], the P values of packets are computed as follows [12], for all $j \geq 1$,

$$P_k^f(j) = \max\{A_k^f(j) + d_k^f, P_k^f(j-1) + v^f\} \quad (10)$$

where $P_k^f(0) = -v^f$, d_k^f is a local delay bound for every packet in flow f, and $v^f(j) = v^f = s^f/\lambda^f$, with s^f and λ^f being the same for all j. If certain schedulablity conditions are met, then a Delay-EDD server provides the guaranteed deadline in (1) with $\beta_k = 0$ [2]. By induction, it is easy to show that for all $j \geq 1$

$$P_k^f(j) - V_k^f(j) = d_k^f - v^f \quad (11)$$

For a leave-in-time server [3], the P values of packets are computed as follows[2] for $j \geq 1$,

$$P_k^f(j) = \max\{A_k^f(j), V_k^f(j-1)\} + d_k^f(j) \quad (12)$$

$$V_k^f(j) = \max\{A_k^f(j), V_k^f(j-1)\} + v^f(j) \quad (13)$$

where $V_k^f(0) = 0$, $d_k^f(j)$ is the local delay bound of packet j, and $v^f(j) = s^f(j)/\lambda^f$, with the reserved rate λ^f the same for every packet in flow f. Under certain admission control conditions, a leave-in-time server provides the guaranteed deadline in (1) with

[2] The packet arrival time, $A_k^f(j)$, should be interpreted as the time when packet i becomes *eligible* in [3].

$\beta_k = s_k^{max}/C_k$ [3]. Subtracting (13) from (12), we have for all j:

$$P_k^f(j) - V_k^f(j) = d_k^f(j) - v^f(j) \qquad (14)$$

For example, suppose every server in the path of flow f is one of the four GD servers described above. In this case, to obtain an end-to-end delay upper bound for f, we simply replace the term $P_k^f(j) - V_k^f(j)$, for $1 \le k \le K$, in the delay guarantee formula of Theorem 1 by the appropriate term on the right hand side of (8), (9), (11), or (14).

In summary, we have shown how to obtain end-to-end delay upper bounds for source control mechanisms and GD servers that are known. In the next section, we illustrate how to apply the delay guarantee formula in Theorem 1 to a new scheduling idea.

3 Group Scheduling

In this section, we introduce and develop the concept of *group scheduling*. We first prove a theorem about relaxing deadlines for a subclass of GD servers. We then generalize the previous model of a flow, which is a sequence of packets, by adding some structure to the sequence. Such generalization allows the modeling of message segmentation and specification of jitter constraints (see Section 4). We then illustrate how to specialize the delay guarantee theorem to group scheduling for a subclass of GD servers.

3.1 Relaxed deadline theorem

Consider a subclass of GD servers, called the *priority subclass*, with the following additional properties:

- *work conserving*—The server does not idle when there are bits to send.
- *nonpreemptive*—The transmission of a packet cannot be preempted.
- *priority service*—In selecting the next packet to serve, the packet in queue with the smallest deadline is chosen. Ties between packets of different flows are broken arbitrarily, and ties between packets of the same flow are broken by arrival times (to preserve the FIFO property).

Note that each service discipline in the priority subclass is almost completely specified. Only the P functions—$P_k^f(\cdot)$ for all f—remain to be specified. Also, since β_k is a constant, the server can use the P values of packets, rather than their deadlines, as priorities.

The following theorem is about two related systems, an original system and a modified system; we use the term *system* to refer to a particular implementation of a server k in the priority subclass. The arrival times and lengths of packets are the same in each system. The arrival time of an arbitrary packet p at server k is $A_k(p)$. In the original system, the deadline and departure time of packet p are $P_k(p) + \beta_k$ and $L_k(p)$, respectively. In the modified system, the deadline and departure time of packet p are $P'_k(p) + \beta_k$ and $L'_k(p)$, respectively. Furthermore, for all p, it is assumed that $P'_k(p) \ge P_k(p)$; the modified system is said to have *relaxed deadlines* compared to the original system.

Theorem 2. If, for all packet p, the deadline $P_k(p) + \beta_k$ is met in the original system, that is,

$$L_k(p) \le P_k(p) + \beta_k \qquad (15)$$

then, for all packet p, the relaxed deadline $P'_k(p) + \beta_k$ is met in the modified system, that is,

$$L'_k(p) \le P'_k(p) + \beta_k \qquad (16)$$

3.2 Messages and groups

Depending upon whether the packet switching network is an ATM network or an IP network, a packet may represent an ATM cell or an IP datagram. In any case, the end-to-end delay of a packet may not be the desired performance measure for many applications. For example, a video picture (or file) being sent by an application over an IP network may be segmented into a sequence of IP datagrams. The delay incurred to deliver the entire video picture (or file) is much more important to the application than the delays of individual IP datagrams. As another example, an IP datagram encapsulating some email message may be segmented into a sequence of cells for delivery over an ATM network. The delay incurred to deliver the entire email message is more important than the delays of individual cells.

Motivated by these observations, we introduce the concept of a *message* which is a data unit of variable, bounded length (in number of bits). Consider the same network path of $K + 2$ nodes in Section 2. We assume that the source generates a traffic flow as a sequence of messages. Each message is segmented into one or more packets which are sent to the network (namely, node 1 on the path). Subsequently at the destination, each message is reassembled from its packets.

Note that a message is primarily a source-destination concept. We next introduce the concept of a *group* of packets, which is meaningful only to the packet switching network (nodes 1 through K on the path) but not to sources and destinations. Within the network, a guaranteed flow is represented as a sequence of packet groups. For each group, the largest of the group's packet deadlines is chosen to be the group priority. Each packet is scheduled on the basis of its group priority.

Notation for groups. We will use h as the sequence index which identifies a particular group of packets in flow f, and the notation $h(i)$ to denote the group that includes packet i.

The sizes of groups in a flow are parameters to be adaptively controlled for optimizing network performance (see Section 4.2). In the balance of this paper, we will consider the end-to-end delay of a message to be the performance measure of interest. For this reason, group size is chosen to be less than or equal to message size (number of packets). Specifically, a message's sequence of packets is partitioned into one or more groups

of packets. This model subsumes two special cases: (i) one-packet-per-group, which is the same as the previous flow model, and (ii) one-message-per-group, i.e., all packets of a message constitute a group.

We can also specify group sizes such that a group may consist of the packets of multiple messages. Such large groups would be appropriate for messages that are actually segments of some application-specific data unit, and the end-to-end delay of the application data unit is the performance measure of interest.

3.3 Advantages of group scheduling

Consider a group of consecutive packet arrivals from flow f to node k. For each packet i in the group, its deadline is supposed to be $P_k^f(i) + \beta_k$. With group scheduling, however, the largest deadline in the group is assigned to all packets of the group. Such deadline will be referred to as the group deadline or *group priority*. Note that all except one packet in the group have relaxed deadlines.

Group scheduling has two advantages. First, within a network node the channel scheduler's work is much reduced. This is because the scheduler needs to update its priority data structure whenever the priority of a flow changes. With group scheduling, a flow's priority changes only once per group rather than once per packet; see [11]. Second, we discovered that the flexibility of relaxed deadlines results in better statistical performance (i.e., delay, queue size, and loss probability) for networks where some channels are heavily utilized. We will provide some empirical results to support this claim in Section 5.

3.4 End-to-end delay guarantee

The delay guarantee in Theorem 1 can be specialized to group scheduling. First, we need to specify the service discipline at each network node on the path. In the balance of this paper, we will consider servers in the priority subclass that use virtual clock values as priorities [13]. More specifically, the virtual clock values of a flow f are computed assuming that packet i in the flow is allocated a thoughput of $\lambda^f(i)$ bits/second at each server on the path, and that some admission control mechanism ensures that the capacity of each server is not exceeded [10].

From Section 2.4, we see that the virtual clock value of packet i in flow f at server k is equal to its reference clock value $V_k^f(i)$. Under group scheduling, the packet-dependent deadline of i, $P_k^f(i)$, is assigned the virtual clock value of the last arrival in group $h(i)$. Thus, at node k, for $k = 1, 2, \ldots, K$, we have[3]

$$P_k^f(i) - V_k^f(i) \leq \sum_{n \in h(i), n \neq i} v^f(n) \qquad (17)$$

[3]It is assumed that the interarrival time between packets i and $i+1$ in the same group is less than or equal to $s^f(i)/\lambda^f(i)$. A somewhat weaker jitter constraint on packet arrival times can also be used; see Section 4.

Corollary 1. For group scheduling using virtual clock values, the end-to-end delay guarantee is:

$$A_{K+1}^f(i) \leq V_1^f(i) + (K-1) \max_{1 \leq j \leq i} \{v_g^f(j)\} + (v_g^f(i) - v^f(i)) + \sum_{k=1}^{K} \alpha_k \qquad (18)$$

where for $1 \leq j \leq i$, $v_g^f(j) = \sum_{n \in h(j)} v^f(n)$ and $\alpha_k = (s_k^{max}/C_k) + \tau_{k,k+1}$.

4 A Detailed Example

We next illustrate how to apply group scheduling and the delay guarantee in Corollary 1 to a particular class of networks [7], where a guaranteed flow is generated as a sequence of bursts. We will assume that all packets have a fixed size. End-to-end delay bounds are derived for the path of $K+2$ nodes in Section 2. In this section, we focus upon a particular guaranteed flow, f, and will omit the superscript f in the following notation for clarity.

Notation for bursts

(m,l)	the lth packet in the mth burst of flow f
$A(m,l)$	arrival time of packet (m,l) at a node
$D(m,l)$	end-to-end delay of packet (m,l) (from arrival at node 1 to arrival at node $K+1$)
b_m	size of burst m (packets)
δ_m	maximum duration of burst m (seconds)
λ_m	$= b_m/\delta_m$, average rate of burst m (packets/second)
g_m	$\leq b_m$, group size chosen for burst m (packets)

For burst m, the group size g_m is chosen at the network entrance such that $g_m \leq b_m$. The burst's sequence of packets is partitioned into groups such that each group consists of g_m packets except for the last group whose size may be smaller.

A burst is the same as a message in Section 3.2 but with additional properties. Specifically, the sequence of bursts is required to satisfy the following when its packets arrive at the network entrance (node 1):

Flow Specification

- The first packet of burst m carries information[4] on λ_m, b_m, and g_m.
- Packets in burst m satisfy a *jitter* timing constraint, namely: for $l = 1, 2, \ldots, b_m$,

$$0 \leq A(m,l) - A(m,1) \leq \frac{l-1}{\lambda_m} \qquad (19)$$

- Bursts in the flow satisfy a *separation* timing constraint, namely: for $m \geq 1$,

$$A(m+1,1) - A(m,1) \geq \delta_m \qquad (20)$$

[4]This set of parameter values is implementation-dependent.

The information carried in the first packet of each burst allows every server in the path to allocate a reserved rate to the flow adaptively on a per burst basis. The group size used in scheduling also changes from burst to burst.

The jitter timing constraint requires that packets of the same burst arrive at the network entrance within some bounded duration. Note that without this requirement, it would be impossible for the network to provide an end-to-end delay upper bound to the burst. This is a rather weak constraint and can be easily satisfied if the packets of a burst are derived from the same application data unit at the source. The jitter timing constraint can be exploited to compute virtual clock values very efficiently for each flow at a server [7]; specifically, the main steps of the computation are performed only once per burst.

The separation timing constraint is a sufficient condition for a VC priority server to allocate reserved rates to flows adaptively on a per burst basis and provide guaranteed deadlines [10]. The constraint also ensures that each *active* flow contains at most one active burst—a server can make use of this information to check that its capacity has not been depleted by allocations to flows. Lastly, the constraint is a source control mechanism that ensures that at node 1 the difference between the virtual clock value and arrival time of $(m,1)$, the first packet of burst m, is upper bounded by $1/\lambda_m$ for all m.

To ensure that the jitter and separation timing constraints are preserved when the packets of a flow arrive at node k, for $k = 2,\ldots,K$, the packets pass through a flow regulator before they arrive at the queue of the server. If a packet arrives ahead of schedule, it is delayed by the flow regulator to the extent that the packet is ahead of its deadline; such delays increase the end-to-end delay lower bound for packets, but do not impact the end-to-end delay guarantee of Corollary 1. Note that Corollary 1 is applicable because the server at node k (excluding the flow regulator) is work conserving.

4.1 Delay bounds

If the channel capacity, for every channel on the path, is not exceeded by the aggregate reserved rate of active flows [10], a tight end-to-end delay upper bound for the first packet of every burst can be derived as a special case of Corollary 1.

Corollary 2.

$$D(m,1) \le \frac{g_m}{\lambda_m} + (K-1)\max_{1\le n\le m}\{\frac{g_n}{\lambda_n}\} + \sum_{k=1}^{K}\alpha_k \qquad (21)$$

Corollary 2 generalizes a theorem in [7] for the special case of individual priority (that is, $g_m = 1$ for all m). A tight end-to-end delay lower bound is the following:

$$D(m,1) \ge (K-1)\frac{g_m}{\lambda_m} + \sum_{k=1}^{K}\alpha_k \qquad (22)$$

Since flow regulators preserve the jitter timing constraint for each burst in a guaranteed flow [7], the delay of packet (m,l) is bounded as follows:

$$D(m,l) \le D(m,1) + \frac{l}{\lambda_m} \qquad (23)$$

The end-to-end delay of burst m, denoted by D_m, measured from the time when packet $(m,1)$ arrives at node 1 to the time when packet (m,b_m) arrives at node $K+1$, is bounded as follows:

$$D_m \le D(m,1) + \frac{b_m}{\lambda_m} = D(m,1) + \delta_m \qquad (24)$$

4.2 How to determine group sizes

The source of a guaranteed flow negotiates with the network to agree upon QoS parameter values, which determine flow characteristics and service guarantees. (For a commercial network, the cost of flow delivery would depend upon these negotiated values.) In this example, we consider the following QoS parameters.

λ_{max}	maximum rate to be reserved for a burst ($\lambda_m \le \lambda_{max}$ for all m), to be guaranteed by source
δ_{max}	maximum burst duration ($\delta_m \le \delta_{max}$), to be guaranteed by source
D_{max}	maximum end-to-end delay of any burst in flow, to be guaranteed by network

Note that λ_m is an average rate determined by b_m and δ_m. Thus, to conform to the negotiated value of λ_{max}, it is sufficient that the source controls its burst sizes, such that, for all m,

$$\frac{b_m}{\delta_m} \le \lambda_{max} \qquad (25)$$

If the flow conforms to Flow Specification at its network entrance, the network will ensure that burst delays do not exceed D_{max}. The negotiated values of D_{max} and δ_{max} are used to determine the group sizes for bursts, as described below.

We first derive a *uniform upper bound* on $D(m,1)$ for all m. Let $m*$ denote the index of the *slowest* burst in the flow, that is, for all m,

$$\lambda_{m*} \le \lambda_m \qquad (26)$$

Suppose each burst is allocated a reserved rate equal to its average rate. For the special case of individual priority ($g_m = 1$ for all m) the slowest burst in the flow determines the uniform upper bound of $D(m,1)$, which is:

$$D(m,1) \le \frac{K}{\lambda_{m*}} + \sum_{k=1}^{K}\alpha_k \qquad (27)$$

For the general case of group priority, if g_{m*} is chosen to be 1 and g_m a positive integer such that

$$\frac{g_m}{\lambda_m} \le \frac{1}{\lambda_{m*}} \qquad (28)$$

it is easy to observe, from (21), that the same uniform upper bound in (27) applies. This observation suggests that subject to (28), *group scheduling can be used without increasing the worst-case delay of any burst in the flow.* To illustrate the potential benefit of group scheduling, consider interframe-encoded pictures in a video flow, which have very large size fluctuations, e.g., for MPEG sequences studied in [6], an I picture is up to 30 times the size of a B picture. From (28), g_m can be as large as 30 for an I picture. Thus we see that for such video traffic, the frequency of priority changes for the flow can be significantly decreased, which reduces the work of the channel scheduler. (We believe that reducing the channel scheduler's work is essential to high-speed packet switch design.)

We next consider the QoS parameter, D_{max}. In order for the network to provide the bound D_{max} to every burst, the reserved rates of bursts in the flow must be lower bounded to avoid having a burst that travels too slowly. Specifically, the minimum reserved rate for a burst should be

$$\lambda_{min} = K/(D_{max} - \delta_{max} - \sum_{k=1}^{K} \alpha_k) \qquad (29)$$

Note that if λ_{min} is larger than λ_{max}, there is a conflict between the negotiated values of λ_{max} and D_{max}. A renegotiation between source and network would be required. Suppose that this is not true. To derive a condition for determining group sizes for bursts, we consider two possible scenarios.

First, one or more bursts in the flow may be so slow that the uniform upper bound is very large, in fact, it is larger than the value of D_{max} negotiated between source and network. To ensure that the uniform upper bound is less than D_{max}, each burst in the flow must be allocated a reserved rate not less than λ_{min} at each server, i.e., the reserved rate of burst m is chosen to be $\max\{\lambda_m, \lambda_{min}\}$.

Second, the value of D_{max} negotiated between source and network is larger than the uniform delay upper bound. In this case, a group size larger than 1 may be used for scheduling even for the slowest burst in the flow. This group size, denoted by g_{min}, is chosen to be a positive integer such that the following holds:

$$\frac{g_{min}}{\lambda_{m*}} \leq \frac{1}{\lambda_{min}} \qquad (30)$$

In this case, the condition for selecting group sizes for bursts can be relaxed from (28) to the following:

$$\frac{g_m}{\lambda_m} \leq \frac{g_{min}}{\lambda_{m*}} \qquad (31)$$

From the above discussion, the general condition for selecting the group sizes of bursts is:

$$g_m \leq \min\{b_m, \lfloor \frac{\lambda_m}{\lambda_{min}} \rfloor\} \qquad (32)$$

which follows from (30) and (31), and the requirement, $g_m \leq b_m$.

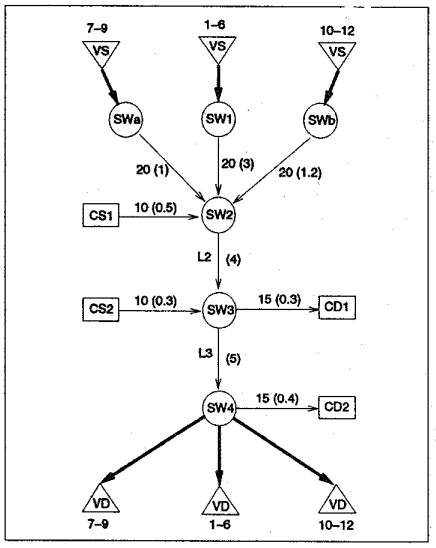

Figure 1: Simulated network.

5 Empirical Results

The end-to-end delay upper bound in (24) is provided by the network to every burst in flow f and is referred to as a *deterministic bound.* It is based upon the assumption that the capacity of every channel in the path has not been exceeded by the aggregate reserved rate allocated to active flows [10]. For many multimedia applications, however, *statistical* guarantees are acceptable, such as: 99% of the pictures in a video sequence are delivered with delays less than the upper bound. We designed a set of experiments to investigate the extent to which the channels in a network path can be *overbooked* before some bursts encounter delays larger than the bound in (24).

From the same experiments, we discovered that group scheduling, aside from reducing the scheduler's work, has another advantage. Specifically, when some channels in a network path are severely overbooked (heavily utilized), group scheduling improves the network's statistical performance (i.e., loss rates, end-to-end delays, queue sizes). This is because group scheduling offers more flexible deadlines for scheduling packets. Consequently, schedulers can better cope with temporary overloads.

5.1 Network configuration

The experiments were conducted using a discrete-event simulator from [7]. The network simulated is illustrated in Figure 1. There are six switches labeled SW. Each switch has a buffer pool for 1200 packets, which is shared by all video flows.

Each thin arrow in Figure 1 represents a channel, which (except for L2 and L3) is labeled by its capacity in megabits per second (Mbps). The channel propagation delay, in milliseconds (ms), is shown in parentheses. Channels L2 and L3 have the same capacity C. The value of C can be changed from one experiment to another.

MPEG sequence	g_{min} = 1		g_{min} = 2		g_{min} = 4	
	max	ave	max	ave	max	ave
Terminator	27	7.1	55	14.4	110	29.3
ParentsSon	16	3.4	32	7.3	64	14.6
RedsNightmare	40	7.3	81	14.5	162	29.4
Student	5	2.0	10	4.8	20	9.4
Driving	51	9.3	102	20.0	205	36.7
Airwolf 2	23	5.4	47	11.2	94	23.0
Simpsons I	18	5.5	37	11.5	74	23.5
Canyon	9	2.6	18	5.9	36	10.8
FlowerGarden	9	2.8	19	6.2	38	13.3
UnderSiege	12	2.8	24	6.1	48	11.6
StarTrek II	4	1.6	8	3.7	16	7.8
Energizer	14	3.6	28	7.9	56	15.0

Table 1: Group sizes for three g_{min} values

Each thick arrow represents a set of channels, one for each video flow. Each such channel has a capacity larger than λ_{max} of the flow it carries; the capacity varies from 10 to 15 Mbps, and the propagation delay also varies.

5.2 Video flows and ABR traffic

The simulated network carries twelve video flows, as well as some ABR traffic. In Figure 1, the source of each video flow is labeled VS, and the destination VD. The video flows travel from their sources through three different switches (SW1, SWa, SWb) to SW2. From there, they all travel through SW3 and SW4 to their destinations. The video flows were generated using traces obtained from MPEG video sequences.[5]

Pictures are represented as bursts in Section 4. The average rate of a picture is computed as follows. Each packet is 53 bytes long with a 48-byte payload. Let b_m be the number of packets needed to carry the bits of picture m. The average rate of picture m is $53 \times 8 \times b_m \times 30$ bits per second, where we have used 1/30 second as δ_m for all m. For most of our experiments (all of the performance results illustrated in this section), the packets in a burst were generated with a fixed interpacket gap.

We did not try to identify values for the QoS parameters, λ_{max} and D_{max}, appropriate for a particular multimedia application. We used average rates of the largest and smallest pictures in the sequence as values for λ_{max} and λ_{min}, respectively.

The group size for each picture in a video sequence was calculated to be the largest integer that satisfies the inequality in (31); we experimented with several values of g_{min}. The maximum and average group sizes for each of the twelve video sequences are shown in Table 1.

In addition to the video flows, the network carried two ABR traffic flows: a flow from CS1 to CD1 via L2, and the other from CS2 to CD2 via L3. Each was a Poisson source whose rate was set to be between 0.20 and 0.21 of the capacity C of channel L2 (also L3) for each experiment.

For L2 and L3, 0.2 of the channel capacity C was allocated to ABR traffic by assigning virtual clock values to ABR packets as priorities [7]. Whenever there was nothing to send from the video flow queues, the entire channel capacity was available to ABR traffic.

We ran each experiment for 10 seconds of simulated time. About 300 pictures were delivered for each video flow. Three of the MPEG sequences were not long enough, and their traces were wrapped around.

5.3 Overbooking to increase utilization

A source can misbehave and generate packets at a rate higher than the reserved rate of its flow. Such behavior will cause its own packets to incur large delays. However, it will not affect the deterministic delay bounds provided by the network to other flows, so long as, for every channel, the channel capacity is not exceeded by the aggregate reserved rate allocated to active flows.

We designed a set of experiments to evaluate network performance when the channel capacity at L2 and L3 is intentionally overbooked by not implementing any admission control mechanism based upon the maximum reserved rate λ_{max} of each flow. Since the reserved rate of a flow changes from burst to burst, it is possible that the largest pictures of all video flows are served by L2 (or L3) at the same time. The sum of λ_{max} over all twelve video flows is equal to 49.77 Mbps. Since only 0.8 of the channel capacity C is allocated to video flows, to ensure that the channel capacity of L2 (L3) is not exceeded, we must have C = 62.21 Mbps. We refer to this case as 0% overbooking.

In the experiments described below, we actually used a value of C smaller than 62.21 Mbps. If C is chosen for an experiment such that 49.77 Mbps *exceeds* $0.8C$ by n%, the channels in the experiment are said to be n% overbooked. The experiment is referred to as *n% overbooking.*

5.4 Channel utilization and loss rate

The objective of overbooking is to increase channel utilization. We performed a series of experiments from 32% to 208% overbooking. Figure 2 shows that the utilization of channel L2 increases almost linearly with overbooking. Three cases of group priority were investigated, for g_{min} equal to 1, 2, and 4. At 208% overbooking, the channel utilization was 0.958 for group priority with g_{min} = 2 and 0.952 for individual priority. The utilization for individual priority was smaller because some packets were dropped (due to buffer overflow).

In Figure 3, we show the percentage of packets dropped due to buffer overflow at L2 which has space for 1200 packets shared by all twelve video flows. Note that the loss rate was zero for group priority with g_{min} equal to 2 or 4. It was fairly low for the other two cases, considering that the channel utilization exceeded 0.95.

5.5 Impact on delay bound

We measured the sum of reserved rates of *active* flows as a function of time, and compared it with the channel capacity of L2 (L3). For the experiments at 32% overbooking, the channel capacity was not exceeded by the aggregate reserved rate of active flows at any time. But for experiments at 76% overbooking, and higher, the channel capacity was exceeded frequently.

[5] See [8] for a more detailed description of the experiments and empirical results.

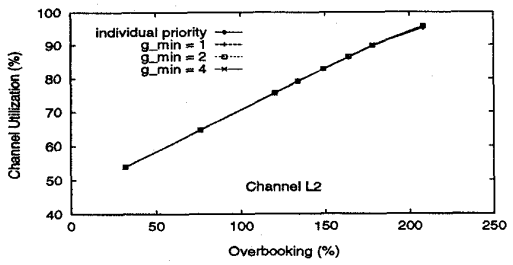

Figure 2: Channel utilization versus overbooking

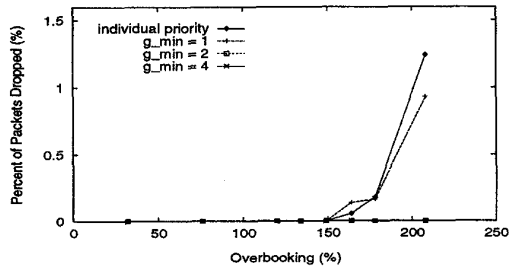

Figure 3: Packet loss rate versus overbooking

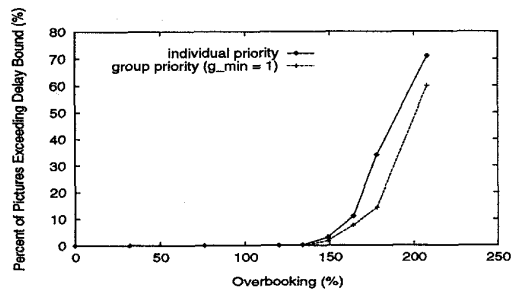

Figure 4: Impact of overbooking on delay bound

The delays of individual pictures (bursts) were measured and compared to the upper bound given by (24) for each video sequence. The results were plotted in Figure 4 for two cases: (i) individual priority and (ii) group priority with $g_{min} = 1$. These two cases have the same delay upper bound for each video sequence, determined by the slowest picture in the sequence.

As illustrated in Figure 4, the delay bounds held for all pictures in all video sequences up to 120% overbooking. At 134% overbooking, the delays of a small number of pictures (less than 1%) exceeded their bounds. In all experiments, the fraction of pictures violating their delay bounds was smaller for group priority (with $g_{min} = 1$) than for individual priority.

5.6 End-to-end delay performance

We observed that the delay performance of group priority with $g_{min} = 1$ was slightly better than individual priority in almost all experiments. However, group priority with g_{min} equal to 2 and 4 performed better than individual priority only when the network was heavily loaded. At 164% overbooking, all three cases of group

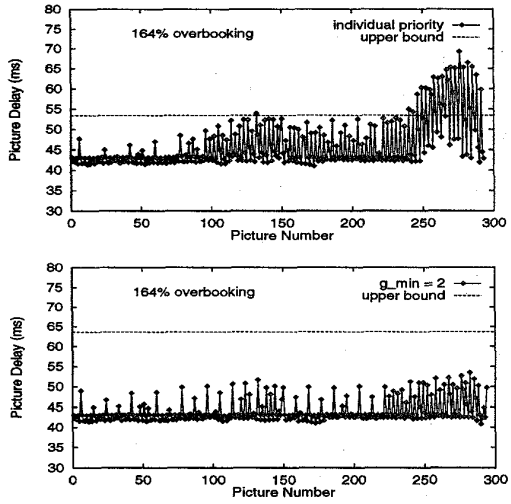

Figure 5: End-to-end picture delays of Energizer sequence

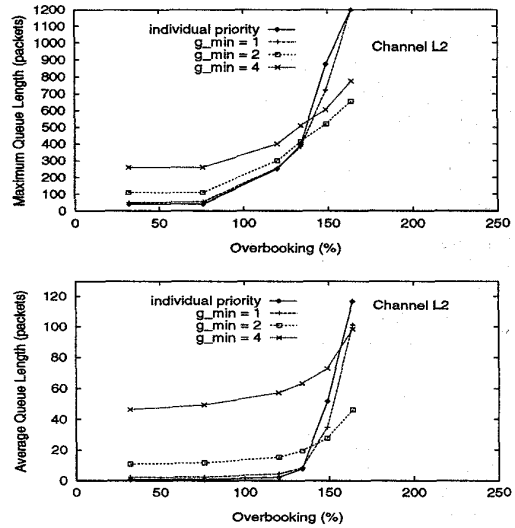

Figure 6: Video queue length versus overbooking

priority had better performance than individual priority (see [8]). In Figure 5, we show the end-to-end picture delays of the Energizer video sequence at 164% overbooking, for individual priority and group priority with g_{min} equal to 2.

5.7 Queue sizes

In Figure 6, we show the maximum and average video queue length at L2 versus overbooking, where *video queue length* denotes the aggregate size of all twelve video flow queues. In Figure 7, we show the video

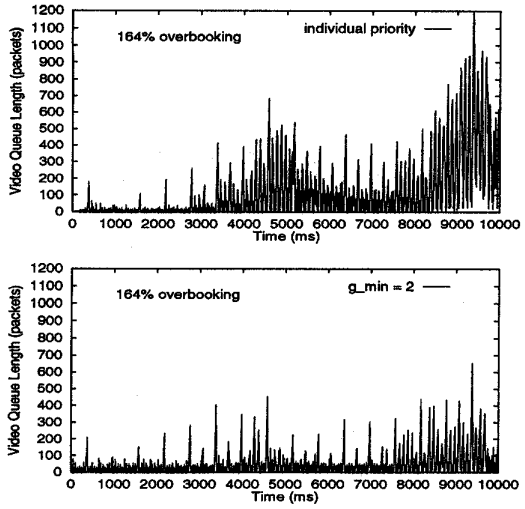

Figure 7: Video queue length at L2 over time

queue length at L2 as a function of time at 164% overbooking, for individual priority and group priority with $g_{min} = 2$.

6 Conclusions

We introduced the class of GD servers and proved an end-to-end delay guarantee theorem. The theorem can be instantiated to obtain end-to-end delay bounds for a variety of source control mechanisms and GD servers; in particular, different GD servers can coexist in the same end-to-end path. With the theorem, the problem of deriving an end-to-end delay upper bound for a guaranteed flow is reduced to a set of single-node problems.

We then introduced and developed the concept of group scheduling. We proved a relaxed deadline theorem for the priority subclass of GD servers. The delay guarantee theorem is then specialized to group scheduling for a subclass of GD servers.

We worked out a detailed example for a particular class of networks [7]. We derived end-to-end delay bounds for bursts (messages), and illustrated how to choose group sizes such that the end-to-end delays of bursts are unaffected by group scheduling.

Group scheduling has two advantages. First the priority of a flow changes less frequently, i.e., from one group to the next rather than from one packet to the next. Hence, the channel scheduler's work in updating its priority data structure (e.g., a heap) would be much reduced for large groups. (An empirical investigation quantifying this reduction can be found in [11].) Second, group scheduling offers more flexible deadlines; consequently, channel schedulers can better cope with temporary overloads.

References

[1] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM '89*, pages 3–12, August 1989.

[2] Domenico Ferrari and Dinesh Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, pages 368–379, April 1990.

[3] Norival R. Figueira and Joseph Pasquale. Leave-in-time: A new service discipline for real-time communications in a packet-switching network. In *Proceedings of ACM SIGCOMM '95*, pages 207–218, August 1995.

[4] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for high speed applications. In *Proceedings of IEEE INFOCOM '94*, pages 636–646, March 1994.

[5] Pawan Goyal, Simon S. Lam, and Harrick M. Vin. Determining end-to-end delay bounds in heterogeneous networks. In *Proceedings of NOSSDAV*, Durham, New Hampshire, April 1995.

[6] Simon S. Lam, Simon Chow, and David K.Y. Yau. An algorithm for lossless smoothing of MPEG video. In *Proceedings of ACM SIGCOMM '94*, pages 281–293, London, England, August 1994.

[7] Simon S. Lam and Geoffrey G. Xie. Burst Scheduling: Architecture and algorithm for switching packet video. Technical Report TR-94-20, University of Texas at Austin, Austin, Texas, July 1994. An abbreviated version in *Proceedings of IEEE INFOCOM '95*, April 1995.

[8] Simon S. Lam and Geoffrey G. Xie. Group priority scheduling. Technical Report TR-95-28, University of Texas at Austin, Austin, Texas, July 1995; revised, January 1996.

[9] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Trans. on Networking*, pages 344–357, June 1993.

[10] Geoffrey G. Xie and Simon S. Lam. Delay guarantee of Virtual Clock server. Technical Report TR-94-24, University of Texas at Austin, Austin, Texas, October 1994. To appear in *IEEE/ACM Trans. Networking*, December 1995.

[11] Geoffrey G. Xie and Simon S. Lam. An efficient scheduler for real-time traffic. Technical Report TR-95-29, University of Texas at Austin, Austin, Texas, July 1995. Available from *http://net.cs.utexas.edu/users/~lam/NRL/*.

[12] Hui Zhang and Srinivasan Keshav. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM '91*, pages 113–121, August 1991.

[13] Lixia Zhang. VirtualClock: A new traffic control algorithm for packet switching networks. In *Proceedings of ACM SIGCOMM '90*, pages 19–29, August 1990.

Appendix

Proof of Lemma 2. We use induction on i.

Base case. From (3), for $i = 1$,

$$V^f_{k+1}(1) = A^f_{k+1}(1) + v^f(1) \quad (33)$$

$$\{by\ Lemma\ 1\}$$

$$\leq P^f_k(1) + \alpha_k + v^f(1) \quad (34)$$

$$= V^f_k(1) + v^f(1) + (P^f_k(1) - V^f_k(1)) + \alpha_k \quad (35)$$

Thus the inequality in (4) holds for $i = 1$.

Inductive step. Assume that the inequality in (4) holds for $1 \leq i \leq n$. A proof that the inequality holds for $i = n + 1$ follows.

From (3), for $i = n+1$,

$$V^f_{k+1}(n+1) = \max\{V^f_{k+1}(n), A^f_{k+1}(n+1)\} + v^f(n+1) \quad (36)$$

- *Case 1.* $V^f_{k+1}(n) \geq A^f_{k+1}(n+1)$:

$$\begin{aligned} V^f_{k+1}(n+1) &= V^f_{k+1}(n) + v^f(n+1) && (37)\\ &\quad \{\textit{by induction hypothesis}\}\\ &\leq V^f_k(n) + \max_{1\leq j\leq n}\{v^f(j) + (P^f_k(j) - V^f_k(j))\} + \\ &\quad \alpha_k + v^f(n+1) && (38)\\ &\quad \{\textit{by (3) and FIFO property}\}\\ &\leq V^f_k(n+1) + \\ &\quad \max_{1\leq j\leq n}\{v^f(j) + (P^f_k(j) - V^f_k(j))\} + \alpha_k && (39)\\ &\leq V^f_k(n+1) + \\ &\quad \max_{1\leq j\leq n+1}\{v^f(j) + (P^f_k(j) - V^f_k(j))\} + \alpha_k && (40) \end{aligned}$$

- *Case 2.* $V^f_{k+1}(n) < A^f_{k+1}(n+1)$:

$$\begin{aligned} V^f_{k+1}(n+1) &= A^f_{k+1}(n+1) + v^f(n+1) && (41)\\ &\quad \{\textit{by Lemma 1}\}\\ &\leq P^f_k(n+1) + \alpha_k + v^f(n+1) && (42)\\ &= V^f_k(n+1) + \alpha_k + v^f(n+1) + \\ &\quad P^f_k(n+1) - V^f_k(n+1) && (43)\\ &\leq V^f_k(n+1) + \\ &\quad \max_{1\leq j\leq n+1}\{v^f(j) + (P^f_k(j) - V^f_k(j))\} + \alpha_k && (44) \end{aligned}$$

Proof of Theorem 1. From Lemma 1,

$$\begin{aligned} A^f_{K+1}(i) &\leq P^f_K(i) + \alpha_K && (45)\\ &= V^f_K(i) + (P^f_K(i) - V^f_K(i)) + \alpha_K && (46)\\ &\quad \{\textit{applying Lemma 2}\}\\ &\leq V^f_{K-1}(i) + \max_{1\leq j\leq i}\{v^f(j) + (P^f_{K-1}(j) - V^f_{K-1}(j))\} + \\ &\quad \alpha_{K-1} + (P^f_K(i) - V^f_K(i)) + \alpha_K && (47)\\ &\quad \{\textit{applying Lemma 2 repeatedly}\}\\ &\leq V^f_1(i) + \sum_{k=1}^{K-1}\max_{1\leq j\leq i}\{v^f(j) + (P^f_k(j) - V^f_k(j))\} + \\ &\quad (P^f_K(i) - V^f_K(i)) + \sum_{k=1}^{K}\alpha_k && (48) \end{aligned}$$

Proof of Theorem 2. The arrival times and lengths of packets are the same in both systems. Since the server is work conserving, each busy period begins and ends at the same time in both systems. Without any loss of generality, it suffices to consider an arbitrary busy period. Each packet in the busy period is identified by its index m in the sequence of departures in the original system. The sequence indices are $1, 2, \ldots, m_1 - 1, m_1, \ldots, m_2, \ldots$. Indices m_1 and m_2 denote two special packets in the sequence, which are introduced below.

The theorem is proved in two parts: (i) We prove that the theorem holds for the special case of a modified system that has exactly one packet in the busy period, say m_1, with a relaxed deadline, i.e., $P_k(m_1) < P'_k(m_1)$, and $\forall m \neq m_1$, $P'_k(m) = P_k(m)$. (ii) The theorem in general is proved by induction on the set of packets in the modified system with relaxed deadlines.

A proof of part (ii) is trivial and is omitted. A proof of part (i) follows. First, we note that if packets in the modified system depart in the same order as they depart in the original system, part (i) holds because (from work-conserving and nonpreemptive assumptions)

$$L'_k(m) = L_k(m) \leq P_k(m) + \beta_k \leq P'_k(m) + \beta_k \quad (49)$$

Otherwise, there is a reordering of the departure sequence due to packet m_1 having a relaxed deadline. Suppose packet m_2 is served behind m_1 in the original system ($m_2 \geq m_1 + 1$ in the original departure sequence) but that m_1 is served immediately after m_2 in the modified system.

For all $m \neq m_1$, we have (from work-conserving and nonpreemptive assumptions)

$$L'_k(m) \leq L_k(m) \leq P_k(m) + \beta_k = P'_k(m) + \beta_k \quad (50)$$

For packet m_1, we have (from work-conserving and nonpreemptive assumptions)

$$\begin{aligned} L'_k(m_1) &= L'_k(m_2) + \frac{s(m_1)}{C_k} = L_k(m_2)\\ &\leq P_k(m_2) + \beta_k = P'_k(m_2) + \beta_k\\ &\leq P'_k(m_1) + \beta_k && (51) \end{aligned}$$

The last inequality follows from $P'_k(m_2) \leq P'_k(m_1)$; otherwise, there would be a contradiction as follows. Suppose $P'_k(m_2) > P'_k(m_1)$. Since m_2 is served ahead of m_1 in the modified system, m_1 must have arrived too late to be selected ahead of m_2 on the basis of a smaller deadline, i.e.,

$$A_k(m_1) > L'_k(m_2^{pre}) \quad (52)$$

where m_2^{pre} denotes the packet served immediately ahead of m_2 in the modified system. There are two possibilities for the identity of packet m_2^{pre}: (i) m_2^{pre} is $m_1 - 1$, in which case, $L'_k(m_2^{pre}) = L_k(m_1 - 1)$, and (ii) $m_2^{pre} \geq m_1 + 1$, in which case,

$$\begin{aligned} L'_k(m_2^{pre}) \geq L'_k(m_1 + 1) &= L_k(m_1 + 1) - (s(m_1)/C_k)\\ &\geq L_k(m_1 - 1) && (53) \end{aligned}$$

Combining the two cases, we have

$$A_k(m_1) > L'_k(m_2^{pre}) \geq L_k(m_1 - 1) \quad (54)$$

Note that the inequality, $A_k(m_1) > L_k(m_1 - 1)$, contradicts the assumption that m_1 is served immediately after $m_1 - 1$ in the original system. Therefore, $L'_k(m_1) \leq P'_k(m_1) + \beta_k$ follows from (51).