On Scheduling Real-Time Traffic under Controlled Load Service in an Integrated Services Internet

Hongyuan Shi and Harish Sethu

Abstract: **The controlled load service defined within the IETF's Integrated Services architecture for quality-of-service (QoS) in the Internet requires source nodes to regulate their traffic while the network, in combination with an admission control strategy, provides a guarantee of performance equivalent to that achieved in a lightly loaded network. Packets sent in violation of the traffic contract are marked so that the network may assign them a lower priority in the use of bandwidth and buffer resources. In this paper, we define the requirements of a scheduler serving packets belonging to the controlled load service and present a novel scheduler that exactly achieves these requirements. In this set of requirements, besides efficiency and throughput, we include an additional important requirement to bound the additional delay of unmarked packets caused due to the transmission of marked packets while dropping as few marked packets as possible. Without such a bound, unmarked packets that are in compliance with the traffic contract are not likely to experience delays consistent with that in a lightly loaded network. For any given desired bound α on this additional delay, we present the CL(α) scheduler which achieves the bound while also achieving a per-packet work complexity of $O(1)$ with respect to the number of flows. We provide an analytical proof of these properties of the CL(α) scheduler, and we also verify this with simulation using real traces of video traffic. The scheduler presented here may be readily adapted for use in scheduling flows with multi-level priorities such as in some real-time video streams, as well as in other emerging service models of the Internet that mark packets to identify drop precedences.**

Index Terms: **Controlled load service, scheduling, real-time traffic, Integrated Services architecture.**

I. INTRODUCTION

With the rapid evolution of the Internet as a commercial infrastructure, there has been an increasing demand for differentiated services based on user and/or application requirements. Real-time traffic, in particular, has stringent delay and bandwidth requirements which are not guaranteed by the best-effort service provided by much of today's Internet. The guaranteed service model in the *Integrated Services* architecture defined by the IETF seeks to achieve this by making per-flow reservations using the Resource Reservation Protocol (RSVP), and then expecting schedulers in the routers to abide by the reservations [1]. One of the challenges in providing these guaranteed services is in the management of reservations and scheduling states corresponding to thousands of traffic flows that may all be active at the same time. Therefore, the Integrated Services framework also specifies a more scalable option called the *controlled load* service [2]. This paper is concerned with designing a scheduler for routers which serve packets belonging to this service category. The algorithm presented here may also be used in other contexts where one has to schedule flows consisting of packets at different priority levels, such as in some real-time video streams [3] as well as in other emerging service models of the Internet that mark packets to identify drop precedences [4].

Controlled load service is distinguished by the fact that it seeks to provide users with a quality of service similar to that in a lightly loaded or unloaded network, and without requiring or specifying a target upper bound on the delay or loss probabilities. The idea behind this service model is that many real-time applications do receive adequate performance and quality of service in a lightly loaded network, eliminating the need for very strict performance guarantees. The desired quality of service is intended to be assured through capacity planning and admission control rather than through per-flow management during packet scheduling and forwarding. When a user exceeds traffic specifications approved by the admission control policy, the service obtained by the excess packets degenerates to the best-effort service.

Controlled load service allows a scalable means to achieve the required quality of service since it does not require the network to distinguish between flows beyond the admission control stage at the edges of the network. Each user/application provides an estimate of its traffic specifications, *Tspec*, and the service provider admits the traffic based on one of several possible admission control strategies that determine whether or not supporting the new user will still keep the network "lightly loaded" [5]. Packets sent by an application in excess of the *Tspec* agreed upon by the user and the service provider are marked by a traffic policer at the entry point into the network. As required by the definition of the controlled load service, the unmarked packets receive service similar to that in a lightly loaded network but the marked packets receive only a best-effort service. To preserve the generality of our solutions, we refer to these excess packets as simply *marked* packets and the rest as *unmarked* packets. Such marking of packets to indicate their level of importance for dropping policies within the network is also used in the Differentiated Services model defined by the IETF [4] and in related buffer management and congestion control strategies as in the RED with In/Out (RIO) [6]. In addition, certain applications such as the multicasting of video streams employ schemes where the signal is encoded in layers so that progres-

Manuscript received February 26, 2002; approved for publication by Weiguo Wang, Division III Editor, October 27, 2002.

The authors are with Department of Electrical and Computer Engineering, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104-2875, USA, e-mail: {shi, sethu}@ece.drexel.edu.

This work was supported in part by U.S. Air Force Contract F30602-00-2-0501.

sively higher quality in the received stream can be achieved by receiving packets from more layers [7], [3]. Packets from layers corresponding to higher quality may be selectively dropped by the routers at congested points in the network.

Several admission control [5], packet forwarding and scheduling strategies [8], [9] have been suggested for use in routers to support the controlled load service. The authors in [9] propose a strategy that dynamically alters the priority of the packets (for e.g., marked or unmarked) to appropriately achieve the expected arrival time of each packet. The complexity of the algorithm is of $O(n \log n)$ with respect to n, the number of flows. However, the controlled load service was designed as a scalable alternative to providing guaranteed service for applications that do have certain quality of service requirements but which can, to some extent, adapt to changes in network conditions. One of the goals in the design of our scheduler for controlled load service is that it should be efficient with no per-flow management of flows, and with an $O(1)$ dequeuing complexity with respect to the number of flows and also the number of packets awaiting service. We consider it important to preserve this original intent in the implementation of mechanisms that support the controlled load service. To this end, as intended by the designers of this class of service and as also suggested in [8], we believe that a First-Come-First-Serve scheduling strategy is adequate for controlled load service, in combination with an effective admission control policy and a simple threshold-based buffer management strategy.

In addition to the simplicity and the scalability desired in the mechanism that supports the controlled load service, it is also desirable that we provide some guidelines on how to treat marked packets in relation to unmarked packets. For example, if capacity planning and admission control are reliably and correctly executed, the scheduler will have enough bandwidth for the unmarked packets. However, it is desirable that as many marked packets be transmitted as possible while the delay caused to unmarked packets because of marked packet transmissions is bounded. Even though scheduling algorithms have been studied extensively for best effort traffic as well as for guaranteed services, scheduling strategies for merged packet streams with marked and unmarked packets requiring different levels of service have not been studied within a theoretical framework. Scheduling of packets seeking the controlled load service provides such a context with packets of the same flow belonging to different priority levels. In the design of the scheduler for controlled load service, our goal is to provide a framework and a mechanism that recognizes the trade-offs in the conflicting requirements of sending as many marked packets as possible, while ensuring that the unmarked packets of the same or other traffic streams continue to experience delays consistent with that in a lightly loaded network.

This paper presents a scheduler of $O(1)$ per-packet complexity with respect to the number of flows and the number of packets awaiting service, and which ensures that the impact of marked packets on the delay experienced by an unmarked packet is bounded. In addition, it transmits very close to the maximum possible number of marked packets that may be transmitted while meeting the above goals. The paper is organized as follows: In Section II of this paper, we elaborate on the above and related requirements of a scheduler in an Internet router providing controlled load service to traffic from real-time sources with marked and unmarked packets. This section also formally outlines these requirements upon which we base our scheduling strategy. Section III presents the CL(α) scheduler designed to satisfy these requirements. Section IV analytically proves that the algorithm meets the requirements specified in Section II. Section V presents a brief set of simulation results using real video traffic traces that verify the analysis in Section IV and also show that the CL(α) scheduler sends close to the maximum possible number of marked packets under the stipulated restrictions. Section VI concludes the paper.

II. REQUIREMENTS OF THE SCHEDULER

Our primary goal in the design of a scheduler for controlled load service is to preserve the original intent in the design and specification of the controlled load service. Certainly, it would be inappropriate to add implementation complexity to the service by adding per-flow management in the routers. Therefore, it is desirable that the scheduler use some simple discipline such as first-come-first-served (FCFS), while aggregating packets from all flows awaiting service by the scheduler into the same queue. Note that, in a lightly loaded network with regulated traffic, the FCFS scheduling discipline is expected to be more than adequate. This facilitates the design of an efficient scheduler with a per-packet dequeuing complexity that is independent of the number of flows and also the number of marked or unmarked packets awaiting service in the queue. Also, by such a strategy which places all the packets in the same queue, the packets within the same flow, marked or unmarked, are delivered in order.

Secondly, the controlled load service packets do not have a delay or bandwidth specification. Therefore, a scheduler cannot make decisions based on delay requirements as in traditional guaranteed-service schedulers such as virtual clock or weighted fair queueing [10]. Instead, it is the capacity planning phase and admission control mechanism, based on the *Tspec* provided by the applications, that are responsible for ensuring that the packets can receive a delay approximating that in a lightly loaded network. Therefore, given effective capacity planning and admission control, it is sensible for the scheduler to assume that the unmarked packets of one flow will not affect the unmarked packets of another flow to the point that the network appears congested to any flow. However, the marked packet arrival characteristics are not part of the *Tspec* and therefore, unregulated. The scheduler does have to ensure that the impact of too many marked packets on the quality-of-service received by the traffic flows is kept under control within a certain acceptable bound. Since delay is the primary QoS parameter for real-time traffic, we can define the scheduler requirement as follows: the scheduler should guarantee that, *for any unmarked packet, the additional delay caused by marked packet transmissions is no more than a certain constant, α.* In other words, if an unmarked packet, in the absence of marked packets, could be forwarded with a delay of Δ, then the delay of the same packet in the presence of marked packets should be no more than $\Delta + \alpha$. The quantity α may be defined by the router or may be a negotiated

quantity between service providers.

Finally, we do wish to send as many marked packets (unregulated, best-effort packets) as possible without violating the above requirement on the impact of marked packets on the delays experienced by unmarked packets, and also without violating the requirements on the efficiency and complexity of the scheduler. This set of requirements is non-trivial to meet, especially in the absence of per-flow management. Note that in the absence of per-flow tracking and management of packet arrivals, the scheduler cannot predict with sufficient precision the new packet arrival characteristics, and therefore, cannot know whether sending a marked packet at a certain instant of time can be a cause for additional delay for unmarked packets at some later time.

In summary, the following are the goals in the design of our scheduler for controlled load service:

1. The scheduler should be efficient with no per-flow management of flows, and with an $O(1)$ dequeuing complexity with respect to the number of flows and with respect to the number of packets awaiting service.
2. The scheduler should be able to ensure that the impact of marked packets on the delay experienced by an unmarked packet is bounded.
3. Given the above two goals, the scheduler should be able to transmit as many marked packets as possible. For example, the scheduler should not trivially achieve the above goals by dropping all marked packets.

We assume that, within any given router, the quantity α is the same for all flows in the network. This critical parameter cannot be used in a cumulative fashion across multiple routers in the path of a flow since this would imply that schedulers would have to manage per-flow states. We assume that the capacity planning and the admission control phases will determine and set this parameter for each of the routers in the network prior to the beginning of transmissions that require such service.

III. THE CL(α) SCHEDULER

The CL(α) scheduler for Controlled Load service presented in this section meets the requirements specified in the previous section. For any given α, the CL(α) scheduler ensures that the increase in the delay experienced by an unmarked packet due to the presence of marked packets is bounded by α.

The CL(α) scheduler maintains a single FCFS queue for all arriving packets. Marked as well as unmarked packets are all added to the tail of the same queue in the order of their arrival times. The CL(α) scheduler removes packets from the head of the queue for transmission, dropping marked packets if necessary. In our presentation of the scheduler, we assume that a marked packet transmission will not be pre-empted for the transmission of an unmarked packet. Consequently, we also assume that α is no smaller than the maximum possible length of time it may take to transmit a marked packet. Without this assumption, however, pre-emption will be necessary to ensure that marked packet transmissions do not increase the delay of an unmarked packet by more than α. While our presentation assumes no pre-emption for purposes of improved clarity, the CL(α) scheduler can be trivially changed to allow pre-emption if so desired.

In this paper, we use the following definitions of the delay of a packet and the extra delay of an unmarked packet at time instant t.

Definition 1: The *arrival time* of a packet is the instant of time that the last bit of the packet arrives into the queue of packets awaiting transmission by a scheduler. The *departure time* of a packet is the instant of time that the last bit of the packet is transmitted by the scheduler. The *delay* of a packet at a scheduler is the length of the time interval between the arrival time and the departure time of the packet.

Definition 2: The *extra delay*, denoted by $ED_P(t)$, experienced at a scheduler by an unmarked packet, P, at time instant t is the cumulative additional delay that is experienced by P caused by transmissions of marked packets before time t.

In a scheduling policy in which all marked packets are always dropped, the extra delay of an unmarked packet is always zero. The extra delay of a packet at a scheduler represents the difference between the delay experienced by the packet at the scheduler and the delay it would experience with a reference scheduler that drops all marked packets.

The extra delay, as defined above, is a function of time and changes whenever the scheduler chooses to send marked packets. If a scheduler chooses to drop all marked packets after time t, the extra delay of unmarked packets in the queue will not increase after time t. Obviously, when a marked packet is scheduled for transmission, all the unmarked packets in the queue will suffer an extra delay. In addition, some of the extra delay is "passed on" further to the unmarked packets which arrive after the transmission of the marked packet. This is because, in a first-come-first-served queue, a packet's delay depends on the time at which its predecessor is served. Thus, the extra delay caused to one unmarked packet can cause an extra delay to unmarked packets that arrive later as well.

The extra delay of a newly arrived packet, however, is not merely equal to the extra delay suffered by its predecessor until this time, and can actually be less than that of its predecessor in the queue. This is best illustrated by considering an unmarked packet that arrives during a period of low congestion with only a small number of packets in the queue. However, the unmarked packet ahead of it in the queue, i.e., the predecessor packet, may have arrived in the queue during a period of heavy congestion when the queue length was large and thus may have a large extra delay associated with it. It is possible that the queue length later reduces and the newly arrived packet will only inherit a portion of the delay suffered previously by the predecessor packet. We will analyze these aspects of the extra delay in greater detail in Section III-A.

The goal of the CL(α) scheduler is to ensure that $ED_P(t) \leq \alpha$ for all unmarked packets at all time instants t. In achieving this goal, the system has to (i) keep track of the changes in the extra delay of each packet in the queue and also (ii) determine the extra delay inherited by each new arriving packet from its predecessor packet. In the following, we describe an efficient algorithm that manages these two important functions in the CL(α) scheduler.

```
Initialize: (Invoked when the scheduler is initialized)
    HeadED ← 0;
    TailED ← 0;

Enqueue: (Invoked when a packet P arrives)
    if (P is unmarked) then
        if (EDDQueueIsNotEmpty AND U < TailED) then
            AddToEDDQueue(TailED − U);
            TailED ← U;
        else
            AddToEDDQueue(0);
        end if;
    end if;
    AddPacketToQueue(P);
```

Fig. 1. Intialization and enqueueing routines of the CL(α) scheduler; U, at any given time instant t, and stands for $U(t)$.

A. *Tracking Changes in* $\mathrm{ED}_P(t)$

To track the extra delay of packets in the queue as it changes with time, a naive method is to maintain an extra delay counter for each unmarked packet in the queue. The scheduler in such a case would have to check each of the extra delay counters before sending a marked packet. Upon sending a marked packet, it would have to update each counter by the transmission time of the marked packet. Obviously, this scheme has a processing delay proportional to the number of unmarked packets in the queue, with the potential of severely limiting scheduling efficiency when the queue length is large. The CL(α) scheduler, however, achieves significantly better scalability by inferring the extra delay of each unmarked packet from that of its predecessor packets in the queue. In fact, the CL(α) scheduler succeeds in achieving an $O(1)$ per-packet work complexity. Figs. 1, 2, and 3 present a pseudo-code description of the CL(α) scheduler.

Recall that the FCFS queue consists of both marked and unmarked packets. We denote the first unmarked packet in the queue that has not yet completed transmission as the *unmarked head*. Note that an unmarked packet that is being transmitted at a certain instant of time is the unmarked head at that instant. Similarly, we define the *unmarked tail* as the last unmarked packet in the queue that has not yet completed transmission. The CL(α) scheduler maintains a record of the extra delays for the unmarked head and tail packets, denoted by *HeadED* and *TailED*, respectively. Both of these values have to be updated whenever a marked packet is transmitted while there are unmarked packets in the queue.

Suppose the arriving *unmarked* packets are labeled as 1, 2, $\cdots$, in the order of their arrival times. Let a_i be the arrival time of packet i. Let d_i be the departure time of packet i. If packet i arrives before packet $i-1$ completes its transmission (i.e., $a_i < d_{i-1}$), note that neither of the packets would have completed transmission during the interval between a_i and d_{i-1}. The additional accumulated extra delay due to marked packet transmissions during this time interval is the same for both packets. That is, for $a_i \le t \le d_{i-1}$, we have,

$$ED_i(t) - ED_i(a_i) = ED_{i-1}(t) - ED_{i-1}(a_i).$$

```
Dequeue:
    while (QueueIsNotEmpty) do
        P ← PacketAtHeadOfQueue;
        if (P is unmarked) then
            TransmitUnmarkedPacket(P);
        else                    /* P is marked. */
            if (EDDQueueIsNotEmpty) then
                if (HeadED + TxTime(P) ≤ α) then
                    HeadED ← HeadED + TxTime (P);
                    TailED ← TailED + TxTime(P);
                    TransmitPacket(P);
                else
                    V ← UnmarkedHeadOfQueue;
                    Drop all packets ahead of V;
                    TransmitUnmarkedPacket(V);
                end if;
            else
                TransmitPacket(P);
            end if;
        end if;
    end while;
```

Fig. 2. Dequeueing routine of the CL(α) scheduler.

```
TransmitUnmarkedPacket(P)
    if (EDDQueue.length ≥ 2) then
        Remove EDDHead from EDDQueue;
        E ← CurrentHeadOfEDDQueue;
        if (E.EDD < HeadED) then
            HeadED ← HeadED − E.EDD;
        else
            HeadED ← 0; TailED ← 0;
        end if;
        TransmitPacket(P);
    else
        TransmitPacket(P);
        Remove EDDHead from EDDQueue;
        /* New unmarked packets may arrive during
           the transmission. */
        E ← CurrentHeadOfEDDQueue;
        if (E ≠ NULL AND E.EDD < HeadED ) then
            HeadED ← HeadED − E.EDD;
        else
            HeadED ← 0; TailED ← 0;
        end if;
    end if;
```

Fig. 3. *Transmit Unmarked Packet* routine used by the dequeueing routine of the CL(α) scheduler.

Thus,

$$ED_i(t) = ED_{i-1}(t) - \left[ED_{i-1}(a_i) - ED_i(a_i)\right]. \quad (1)$$

Note that the quantity $ED_{i-1}(a_i) - ED_i(a_i)$ does not change with time, and is a constant for a given i. To further simplify our presentation and analysis, we define this quantity below.

Definition 3: Consider an unmarked packet i, and its predecessor unmarked packet $i-1$. At the instant a_i when packet i arrives, if the predecessor packet has not yet completed its transmission, define *ExtraDelayDifference*$_i$ or EDD_i of an unmarked packet i as the difference between the extra delay of packets $i-1$ and i at time instant a_i. If the predecessor packet has already completed its transmission, define EDD_i as 0.

In other words, $EDD_i = ED_{i-1}(a_i) - ED_i(a_i)$ if unmarked

packet i arrives while unmarked packet $i-1$ is still in the system, and $EDD_i = 0$ otherwise.

Now, from (1),

$$ED_i(t) = ED_{i-1}(t) - EDD_i. \tag{2}$$

Therefore, $ED_i(t)$ can be obtained from EDD_i, and the extra delay of the predecessor packet in the queue, $ED_{i-1}(t)$.

For each unmarked packet i in the queue awaiting transmission, one may associate a constant value, EDD_i. The scheduler maintains these *EDD* values in a separate queue, which we shall denote by *EDDQueue*. The head of this queue, denoted by *EDDHead*, contains the *EDD* value corresponding to the unmarked head. Similarly, the tail of this queue, denoted by *EDDTail*, contains the *EDD* value corresponding to the unmarked tail. Let packet h be the unmarked head at time t. If there is another unmarked packet in the queue, then the next unmarked packet in the queue, packet $h+1$, should have a corresponding entry in the EDD queue equal to EDD_{h+1}. Note from (2) that $ED_{h+1}(t)$ can be computed from EDD_{h+1} and the *HeadED* (extra delay of packet h) at time t. After packet h is transmitted, the value computed for ED_{h+1} becomes the new *HeadED*, since packet $h+1$ is now the new unmarked head. Thus, *HeadED*, always contains the extra delay corresponding to the current unmarked head which has not yet completed transmission.

B. *Computing* EDD

The above mechanism of tracking the extra delay of each unmarked packet relies upon knowledge of the correct *EDD* value corresponding to the packet. The CL(α) scheduler sets this value for each unmarked packet at the instant that the packet arrives into the queue. We will need the following definition and lemma to explain the algorithm used to determine the *EDD* value of an unmarked packet.

Definition 4: Consider the system at time instant t. Define $U(t)$ as the minimum possible additional time it will take for the unmarked tail in the system at time t to complete its transmission.

$U(t)$ may also be thought of as the additional time it will take a packet that arrives at time t to begin its transmission if all marked packets in the system that have not yet begun transmission at time t are dropped. In this paper, we assume that unmarked packets will not pre-empt the transmission of a marked packet. Therefore, $U(t)$ includes the residual transmission time of the packet being transmitted at time t even if it is a marked packet. Thus, $U(t)$ is the sum of this residual transmission time and the transmission times of all the unmarked packets in the queue at time t awaiting the beginning of transmission.

We now proceed to obtain an expression for $ED_i(a_i)$ that facilitates the computation of the *EDD* values.

Lemma 1: During an execution of the CL(α) scheduler, when unmarked packet i arrives into the queue at time instant a_i,

$$ED_i(a_i) = \min\{ED_{i-1}(a_i), U(a_i)\},$$

if its predecessor, packet $i-1$, has not yet completed transmission ($a_i \leq d_{i-1}$), and,

$$ED_i(a_i) = U(a_i),$$

otherwise.

Proof: Since $ED_i(a_i)$ records the cumulative additional delay of packet i caused by marked packet transmissions before time a_i, we prove the statement of the theorem by comparing the departure time of the packet in the CL(α) scheduler assuming no marked packet is transmitted after a_i and its departure time in a reference system which drops all marked packets.

In this proof, we now separately consider each of the two cases in the statement of the lemma.

Case (1): *($a_i \leq d_{i-1}$)*: In this case, the predecessor packet has not yet completed transmission at time a_i. If an unmarked packet i arrives at time a_i, the earliest possible time at which it can begin transmission is $a_i + U(a_i)$, which can occur only if no additional marked packets are transmitted after time a_i. Its earliest departure time is $a_i + U(a_i) + L_i/R$, where L_i is the length of the packet and R is the peak rate of the link. Let $\hat{d}_i$ be the departure time of the packet using the reference scheduler that drops all marked packets. The extra delay of packet i at time a_i, $ED_i(a_i)$, is the component of the delay caused to the packet due to transmissions of marked packets before time instant a_i. Therefore, this is nothing but the difference between $a_i + U(a_i) + L_i/R$ and $\hat{d}_i$. In other words,

$$ED_i(a_i) = a_i + U(a_i) + L_i/R - \hat{d}_i. \tag{3}$$

We will now consider two sub-cases and use (3) above to prove the lemma.

Sub-Case A: *($\mathrm{ED}_{i-1}(a_i) \leq U(a_i)$)*: This sub-case is illustrated in Fig. 4(a). In this case, packet $i-1$ departs at time $\hat{d}_{i-1} = a_i + [U(a_i) - ED_{i-1}(a_i)]$ in the reference system. In the reference system, packet i can begins its transmission at time $\hat{d}_{i-1}$ and completes the transmission at time $\hat{d}_i = \hat{d}_{i-1} + L_i/R$. Using (3),

$$\begin{aligned} ED_i(a_i) &= \left(a_i + U(a_i) + \frac{L_i}{R}\right) \\ &\quad - \left(a_i + [U(a_i) - ED_{i-1}(a_i)] + \frac{L_i}{R}\right) \\ &= ED_{i-1}(a_i). \end{aligned}$$

Sub-Case B *($\mathrm{ED}_{i-1}(a_i) > U(a_i)$)*: This sub-case is illustrated in Fig. 4(b). In this sub-case, packet $i-1$ has already completed its transmission in the reference system at time instant a_i. Therefore, in the reference system, packet i will find the queue empty upon arrival at time a_i and will begin transmission immediately. Thus, the departure time for packet i in the reference system is $\hat{d}_i = a_i + L_i/R$. Using (3) again, we have,

$$ED_i(a_i) = \left(a_i + U(a_i) + \frac{L_i}{R}\right) - \left(a_i + \frac{L_i}{R}\right) = U(a_i).$$

Case (2) *($a_i > d_{i-1}$)*: For the case that packet $i-1$ has completed its transmission at time a_i in the CL(α) scheduler, the extra delay of packet i is not simply zero even though it is the only unmarked packet in the queue. When packet i arrives, the scheduler may be transmitting a marked packet. Since the scheduler does not pre-empt this transmission, the unmarked packet will

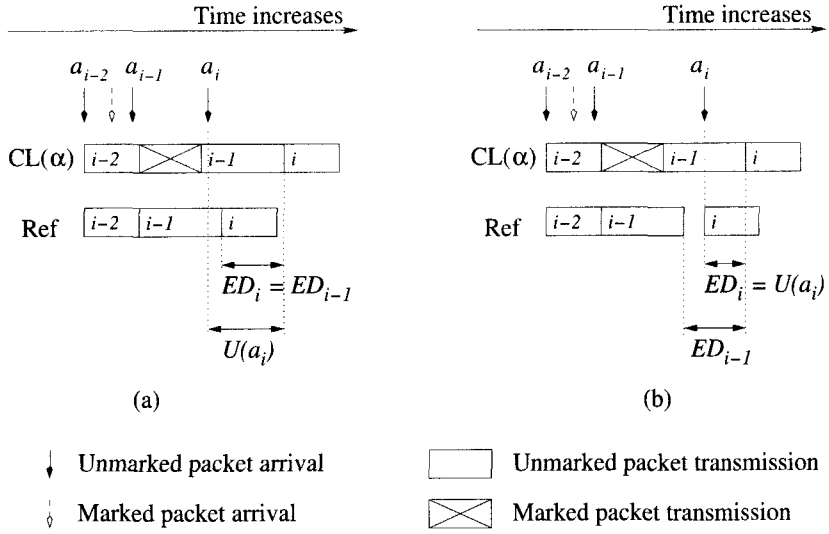


Fig. 4. Illustration of the sub-cases 1A and 1B in the proof of Lemma 1.

acquire an extra delay equal to $U(a_i)$, the residual transmission time of the marked packet currently being transmitted. □

Lemma 1 relates the extra delay of a newly arrived packet at time a_i to that of its predecessor packet and to $U(a_i)$. The extra delay of the predecessor packet is nothing but the *TailED* maintained by the scheduler. $U(t)$ is also easily maintained by the scheduler with updates upon the arrival and departure of packets. This allows an easy computation of the *EDD* value corresponding to each packet, representing the difference between the extra delay values at time a_i between that of packet i and its predecessor.

C. Limiting ED to α

In Lemma 1, it is proved that $ED_k(t)$ is no more than $ED_{k-1}(t)$. Thus, the unmarked head has the largest $ED(t)$ for any given t among all the unmarked packets in the queue. When the scheduler tries to send a marked packet, it only needs to make sure that the *HeadED* will not exceed α. Thus, checking the *ED*s of all the packets in the queue is rendered unnecessary, and thus reduces the per-packet work complexity to $O(1)$.

If *HeadED* will exceed α upon transmission of a marked packet, the scheduler will drop that marked packet and all marked packets ahead of the unmarked head before beginning the transmission of the unmarked head. Searching the queue for the next unmarked packet is obviously not a scalable option, and will not preserve the $O(1)$ complexity of this algorithm. Therefore, the CL(α) scheduler associates with each element of *EDDQueue* a pointer that indicates the position or address of the corresponding unmarked packet in the queue. Recall that each element of *EDDQueue* corresponds to a unique unmarked packet in the queue. When the scheduler determines that the marked packet at the head of the queue cannot be transmitted, it can simply look up the pointer associated with the *EDDHead* and send the unmarked packet corresponding to it.

IV. ANALYSIS

In this section, we present an analysis of the performance and the efficiency of the CL(α) scheduler and prove that the scheduler satisfies the requirements listed in Section II.

Theorem 1: The CL(α) scheduler has a per-packet work complexity of $O(1)$.

Proof: All of the operations in the enqueueing and dequeueing routines are shown in Figs. 1, 2, and 3. The theorem is proved by showing that the number of these operations is $O(1)$.

To enqueue an unmarked packet, one needs to find out the states of the scheduling system, set *EDD* and add it to *EDDQueue*, set *TailED*, and finally append the packet to the end of the queue. To enqueue a marked packet, the server needs only one operation of appending the packet to the end of the queue. In either case, the number of operations of constant time complexity is bounded by a small finite constant. Thus, the *Enqueue* routine in Fig. 1 has a per-packet work complexity of $O(1)$.

In dequeueing an unmarked packet, one needs to update *HeadED* and *TailED*, operations that are readily verified to be of $O(1)$ time complexity. In dequeuing a marked packet, the scheduler needs to first determine if it should be transmitted at all, which is based on a simple comparison operation. If the packet is to be transmitted, the scheduler only needs to update *HeadED* and *TailED*, which again involves only an $O(1)$ complexity in time. If the marked packet is not to be transmitted, the scheduler dequeues the unmarked head packet in $O(1)$ time using the pointer stored in the elements of the *EDDQueue* and without going through a search operation among the packets. Thus, the total time taken to dequeue a marked packet is also of complexity $O(1)$. □

A. Bound on the Extra Delay

In the following, we prove that the CL(α) scheduler correctly computes the extra delay of each unmarked packet, and that the CL(α) scheduler successfully bounds the extra delay of each unmarked packet to α.

Table 1. Settings for MPEG-4 traffic sources and token bucket regulators.

Source	1	2	3	4	5	6	7
Movie Name[1]	J	S	W	B	D	K	F
Video Quality	High	Medium	High	Medium	High	High	High
$L_{\min}$ (bytes)	72	28	26	27	71	307	130
$L_{\max}$ (bytes)	16,745	11,915	9,370	7,565	16,960	15,813	14,431
r_{avg} (Kbps)	770	180	280	180	700	830	840
r_{peak} (Mbps)	3.3	2.4	1.9	1.5	3.4	3.2	2.9
ρ_i (Kbps)	567	135	203	135	473	567	709
σ_i (bytes)	16,755	11,935	9,384	7,570	16,982	15,848	14,458
Link Capacity	2.83 Mbps						
α	50 ms						
Total Time	160 seconds						

Theorem 2: During any execution of the CL(α) scheduling discipline, the additional delay of an unmarked packet caused by the transmission of marked packets is never greater than α.

Proof: The CL(α) scheduler computes the values in the *EDDQueue* based on Lemma 1. When the unmarked head is transmitted, as per Equation (2), a new value of *HeadED* is computed as the difference between the previous value and the EDD value corresponding to the new unmarked head. Thus, *HeadED* represents the extra delay of the new unmarked head. For each marked packet transmission thereafter, the CL(α) scheduler increments the *HeadED* value by the transmission time of the marked packet. Thus, the *HeadED* value always contains the extra delay of the unmarked head packet at all time instants. Recall that the CL(α) scheduler does not transmit a marked packet if its transmission time plus *HeadED* is larger than α. Therefore, if an unmarked packet becomes the unmarked head with an extra delay of less than or equal to α, it will be transmitted early enough to ensure that its extra delay never goes beyond α.

Using induction on the sequence on unmarked packets, we now proceed to show that every unmarked packet has an extra delay less than or equal to α when it becomes the unmarked head.

As the basis step of the induction, consider the very first unmarked packet that arrives at the scheduler. This becomes the unmarked head with a *HeadED* value equal to the residual time of the current marked packet transmission, which is guaranteed to be less than or equal to α.

As part of the inductive step of the proof, assume that each of the unmarked packets until packet $i-1$, i.e., each unmarked packet that arrive at or before a_{i-1}, experiences an extra delay of no more than α when it becomes the unmarked head. We have to now prove that packet i will also experience an extra delay of no more than α when it becomes the unmarked head. We consider two cases:

- If unmarked packet i arrives after packet $i-1$ completes transmission, it immediately becomes the current unmarked head similar to the case of the first unmarked packet that arrives in the system. Therefore, it has an extra delay no more than α when it becomes the unmarked head.
- If unmarked packet i arrives before packet $i-1$ completes transmission, from Lemma 1, the extra delay of packet i is always less than or equal to the extra delay of packet $i-1$ until packet $i-1$ completes transmission. Since packet $i-1$ is transmitted with an extra delay of less than or equal to α, packet i will have an extra delay of no more than α when it becomes the unmarked head.

From the above inductive proof, every unmarked packet has an extra delay less than or equal to α when it becomes the unmarked head. Now, as long as the packet has an extra delay of less than or equal to α when it becomes the unmarked head, it will be transmitted before its extra delay goes higher than α as is ensured by the CL(α) scheduler. This proves the theorem. □

V. SIMULATION RESULTS

The effectiveness of the CL(α) scheduler may be demonstrated using simulation. In our simulation, we use seven sources, each of which generates an MPEG-4 video stream. These video streams of certain popular movies and sports programs are from the traces made available by the Telecommunication Networks Group at the Technical University of Berlin, Germany [11]. In order to remove any effects due to possible correlation between early portions of the video streams, in our simulation, each source begins transmitting at a random point within the movie trace. The generated traffic is policed by token bucket regulators and associated packet markers before it arrives at our CL(α) scheduler. The traffic policer and marker for flow i is configured to allow a long-term average rate of ρ_i and a maximum burst size of σ_i. Packets in the source traffic that do not conform to these token bucket parameters are marked. Fig. 5 illustrates the simulation setup. The details of the video stream sources and the token bucket parameters are listed in Table 1.

We assume that, through the capacity planning phase, traffic policers are configured so that $\sum_i \rho_i \leq R$, where R is the peak link rate at the output of the scheduler. In our experiment, we use values of the token generation rates, ρ_i, such that $\sum_i \rho_i \approx 0.98R$. We also ensure that each source traffic has a higher long-term average rate than allowed by the policer, so that a sufficient number of marked packets are generated to verify the algorithm.

[1]The alphabet letters stand for the following movies and sports programs: J: "Jurassic Park I" S: "Silence Of the Lambs" W: "Star Wars IV" B: "Mr. Bean" D: "Die Hard III" K: "Alpine Ski" F: "Formula 1."

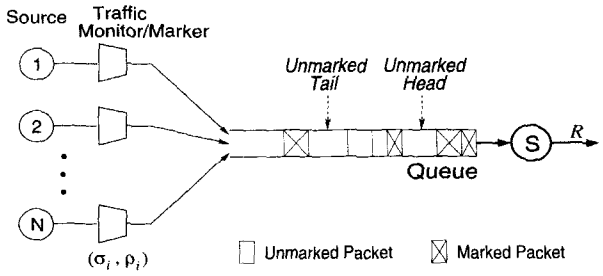


Fig. 5. Simulation set-up of the CL(α) scheduler.

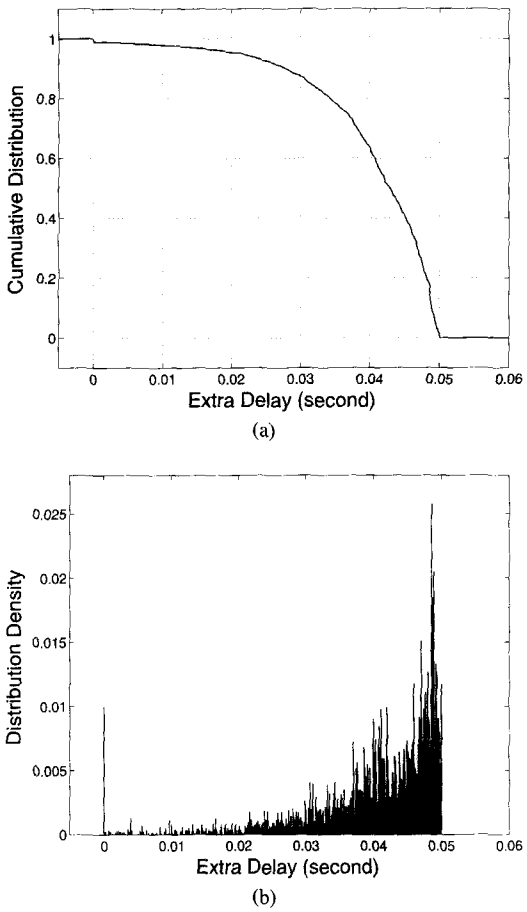


Fig. 6. Distribution of the extra delay from CL(α) scheduler: (a) cumulative distribution (b) distribution density.

In our simulation, the peak link rate R is selected to be smaller than the sum of the average transmission rates (not the same as the corresponding token generation rate) of video streams so that the input queue is backlogged most of time with either marked or unmarked packets, and so that some marked packets would have to be dropped.

Our simulation implements an instance of the CL(α) scheduler where $\alpha = 50$ ms. The duration of the simulation is 160 seconds. Fig. 6(a) shows a cumulative distribution of the extra delay of unmarked packets that go through the CL(α) scheduler over the length of the experiment. Fig. 6(b) shows the distribution density represented as a histogram of the extra delay experienced by these unmarked packets. Fig. 6 verifies that no

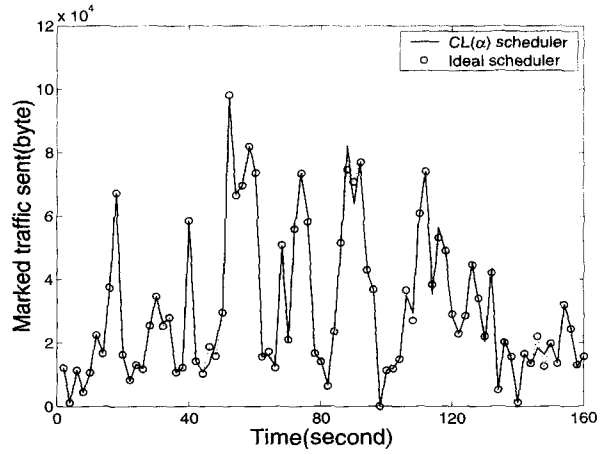


Fig. 7. The amount of data from marked packets sent by the CL(α) scheduler and by the ideal but more complex scheduler.

unmarked packets suffer an extra delay greater than $\alpha = 50$ ms in the CL(α) scheduler.

Once the CL(α) scheduler determines that transmitting the marked packet at the head of the queue will increase the extra delay of an unmarked packet beyond α, it decides to send the unmarked packet and drop all marked packets ahead of it. This ensures the $O(1)$ per-packet complexity of the scheduler. At a greater complexity, one might design a scheduler that tries to send a smaller marked packet that will also not increase the extra delay of the unmarked head beyond α, and thus increase the number of marked packets that are transmitted. In fact, an ideal scheduler that maximizes the data in marked packet transmissions will need to examine each marked packet in the queue ahead of the unmarked head, and send exactly the set of marked packets that together make up the largest amount of data that can be transmitted without increasing the extra delay of the unmarked head beyond α. However, this will require the scheduler to examine each of the marked packets in the queue ahead of the unmarked head, and the number of packets one may have to examine is unbounded except by the size of the queue. Our CL(α) scheduler makes a compromise in favor of achieving simplicity of implementation and a lower per-packet work complexity. Fig. 7 shows the amount of data from marked packets transmitted by the CL(α) scheduler and that transmitted by the ideal scheduler under the simulation setup described earlier. The figure illustrates that the amount of data in marked packets transmitted using the CL(α) scheduler is almost identical to that transmitted by an ideal but more complex scheduler.

VI. CONCLUDING REMARKS

In this paper, we have defined the requirements of a scheduler serving packets belonging to the controlled load service defined within IETF's Integrated Services architecture. The controlled load service requires source points to regulate the traffic and mark packets that are sent in violation of the traffic contract. One of the requirements we define is that the additional delay of unmarked packets caused due to the transmission of marked packets should be bounded. A $O(1)$ scheduler to achieve this

bound is non-trivial. In this paper, we have proposed the CL(α) scheduler, which bounds this extra delay to α or less.

The principle used in this algorithm may also be used to schedule flows with multi-level priorities, such as in some scalable real-time video streams as well as in other emerging service models of the Internet that mark packets to identify drop precedences [6], [4], and [12]. In such cases with multiple levels of drop precedences, the principle of the CL(α) scheduler would have to be applied in a hierarchical manner to bound the impact of each lower priority layer on the delays experienced by higher priority layers. For example, consider flows of packets with three priority levels labeled as type 1, type 2 or type 3 packets, with type 1 at the highest priority level. In transmissions using such layered coding, one may get tolerable quality from receiving just type 1 packets. The quality of the received video and audio deteriorates if a type 1 packet is delayed or dropped, but not as much if a type 2 packet is delayed or dropped, and even less when a type 3 packet is delayed or dropped. The extra delay of a type 1 packet due to the transmissions of type 2 packets could be required to be less than a certain quantity $\alpha_{1,2}$ and that due to transmissions of type 3 packets could be required to be less than another quantity $\alpha_{1,3}$. Similarly, the extra delay of a type 2 packet due to transmissions of type 3 packets may be bounded by $\alpha_{2,3}$. One may infer *ED* values corresponding to each of these three relationships through maintaining three different *EDD* queues, with each queue managed similarly as in the case of the CL(α) scheduler presented in this paper. Tradeoffs between scheduler complexity, desired quality and bandwidth capacity may be achieved by adjusting the α values and the number of relationships for which an α value is defined.

REFERENCES

[1] J. Wroclawski, "RFC 2210: The use of RSVP with IETF integrated services," Sept. 1997.

[2] J. Wroclawski, "RFC 2211: Specification of the controlled-load network element service," Sept. 1997.

[3] U. Horn and B. Girod, "Scalable video transmission for the internet," *Computer Networks and ISDN Systems*, vol. 29, no. 15, pp. 1833–1842, Nov. 1997.

[4] S. Blake *et al.*, "RFC 2475: An architecture for differentiated services," Dec. 1998.

[5] S. Jamin, S. J. Shenker, and P. B. Danzig, "Comparison of measurement-based admission control algorithms for controlled-load service," in *Proc. IEEE INFOCOM*, 1997, vol. 3, pp. 973–980.

[6] D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, Aug. 1998.

[7] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1996, pp. 117–130.

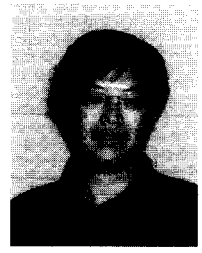
[8] R. Guérin and V. Peris, "Quality-of-service in packet networks: basic mechanisms and directions," *Computer Networks*, vol. 31, no. 3, pp. 169–179, Feb. 1999.

[9] K. Siriwong and R. Ammar, "QoS using delay-synchronized dynamic priority scheduling," in *Proc. Sixth IEEE Symp. Computers and Commun.*, July 2001, pp. 276–281.

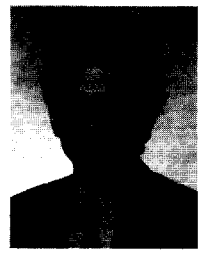
[10] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," in *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.

[11] Telecommunication Networks Group, "MPEG-4 and H.263 video traces for network performance evaluation," available at http://www-tkn.ee.tu-berlin.de/research/trace/trace.html.

[12] W.-C. Cheng *et al.*, "Adaptive packet marking for maintaining end-to-end throughput in a differentiated services internet," *IEEE/ACM Trans. Networking*, vol. 7, no. 5, Oct. 1999.



Hongyuan Shi graduated in 1998 from Tsinghua University in Beijing, China, with a B.S. in electronic engineering. She subsequently joined the Department of ECE at Drexel University as a graduate student and is now a research assistant in the Computer Communications Laboratory. She is currently pursuing a Ph.D. in electrical engineering with a dissertation in the area of packet scheduling strategies for new emerging service models for the Internet. She has also been an intern at Telcordia Technologies, developing component modules for a simulation tool on the transmission performance of optical networks. Her current research interests include quality-of-service in issues in computer networks and the architecture and design of high-speed optical routers.



Harish Sethu obtained his B.Tech in Electronics and Communication Engineering from Indian Institute of Technology (IIT), Chennai, in 1988. He received his Ph.D. in Electrical Engineering from Lehigh University in 1992 with a doctoral dissertation in the area of computer architecture and parallel processing. He spent nearly six years in the RS/6000 SP Division of IBM Corporation where he contributed to the hardware, software and system-level design of more than two generations of the SP family of high-performance parallel computers. He is now an assistant professor in the Department of Electrical and Computer Engineering at Drexel University. He was a recipient of the NSF CAREER award in the year 2000. He has been awarded three U.S. patents. His most recent research efforts have been in the areas of quality of service in computer networks, switch/router architectures, Internet economics and large-scale distributed systems and networks including mobile ad hoc networks and sensor networks.