Burst Scheduling Networks*

Simon S. Lam and Geoffrey G. Xie
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

August 31, 1996

Abstract

Most application-level data units are too large to be carried in a single packet and must be segmented for network delivery. To an application, the end-to-end delays and loss rate of its data units are more relevant measures of performance than ones specified for individual packets. From this observation, we introduced the concept of a *burst* (which subsumes the concept of a *block* in the ATM literature). A flow is modeled as a sequence of bursts, each of which models a sequence of packets that encapsulate an application data unit. We describe an approach towards designing integrated services packet-switching networks that provide QoS guarantees to bursts. We present a burst-based flow specification, an architecture and algorithms for packet scheduling, and tight bounds on end-to-end burst delays. In particular, we illustrate how to exploit the burst-based flow specification to improve implementation efficiency. We describe how burst scheduling networks can be designed to provide a real-time VBR service with no loss and, with burst-based admission control, a real-time VBR service at a specified loss rate.

Keywords: packet scheduling, delay guarantee, firewall property, flow specification, admission control, variable bit rate service, packet switching, ATM block transfer, block transfer delay, block loss rate

*Research supported in part by National Science Foundation under grant no. NCR-9506048, the Texas Advanced Research Program grant no. 003658-220, and by an Intel Graduate Fellowship awarded to Geoffrey Xie for the 1995-96 academic year. Xie's current address: Computer Science Department, Naval Postgraduate School, Monterey, CA 93943. An earlier version of this paper appeared in *Proceedings INFOCOM '95*, April 1995 [7].

1 Introduction

In all packet switching networks, packets have a maximum size (in number of bits). Most application-level data units are too large to be carried in a single packet and must be segmented for network delivery. To an application, the end-to-end delays and loss rate of its data units are more relevant performance measures than ones specified for individual packets. For example, a video picture being sent over an IP network may be segmented into a sequence of IP datagrams. To an end user, the delay incurred to deliver the entire video picture is more important than the delays of individual IP datagrams. As another example, an email message may be segmented into a sequence of cells for delivery over an ATM network. The delay incurred to deliver the email message is more important than the delays of individual cells.

Motivated by the above observation, the concept of a *burst* was introduced [7]. Consider a traffic source that generates a sequence of application data units for network delivery to some destination. An application data unit may be segmented and is encapsulated in one or more packets. The sequence of packets encapsulating an application data unit is called a burst. In this model, a traffic flow is still a sequence of packets, but with the addition of a many-to-one mapping from packets to bursts.

We believe that integrated services packet-switching networks should be designed to provide QoS guarantees to bursts rather than packets. In particular, we advocate that the QoS parameters in the ATM Forum Traffic Management specification, cell transfer delay and cell loss rate, be generalized to block transfer delay and block loss rate, respectively.[1] Note that such a generalization is backward-compatible since a block is by definition a sequence of cells, with a single cell as a special case. The concept of a block (burst) already exists in the ATM literature. Specifically, the ATM block transfer (ABT) capability being standardized by ITU-T [4] allows a traffic source to dynamically negotiate its bandwidth reservation on the basis of a block of cells. In this paper, we refer to such a reservation method as *burst-based rate allocation*.

The objective of this paper is to describe an approach towards designing integrated services packet-switching networks to provide QoS guarantees to bursts rather than individual packets. Our approach is illustrated with the design of a particular class of networks, called *burst scheduling networks*, first presented in [7].

The balance of this paper is organized as follows. In Section 2, our design approach is described. In Section 3, a burst-based flow specification is introduced. In Section 4, we describe how to modify an existing packet scheduling discipline to provide burst delay guarantees, and exploit the burst-based flow specification to significantly improve implementation efficiency. In Section 5, we present tight bounds on end-to-end burst delays. Depending upon the admission control policy, each flow that satisfies the flow specification at its network entrance will be provided a burst delay guarantee with zero loss or at a specified loss rate. In Section 6, we discuss recent extensions to this line of research. In Section 7, we present experimental results from a discrete-event simulation driven by MPEG video traces.

[1] The terms, *block* and *cell*, in ATM terminology are special cases of *burst* and *packet*, respectively, with a cell being a fixed-size packet.

2 Burst QoS Approach to Network Design

To introduce the performance measures of interest, consider a flow that traverses a fixed path of $K+2$ nodes through a network, where node 0 is the source, node $K+1$ is the destination, and nodes 1 to K are switches. We assume that packets in the *same* flow are served in FIFO order at each switch. The *delay of a burst* in the flow is defined to be from the arrival time of a burst's first packet at its network entrance (node 1) to the arrival time of the burst's last packet at its destination (node $K+1$). The *loss rate* of the flow is defined to the fraction of bursts in the flow not delivered to the destination.

At a switch (node k, for $k = 1, \ldots, K$), each output channel (the channel from node k to node $k+1$) is statistically shared among many flows according to some scheduling algorithm (to be designed). Some of the flows may require QoS guarantees from the network, and others may not.

A flow that requires QoS guarantees from the network is assumed to satisfy a flow specification at its network entrance. The guarantees are *conditional* in that the network is not obligated to provide any guarantee to a flow that does not conform to its flow specification.

An important objective in network design is the *firewall property* [8, 11]. That is, a flow may misbehave and not conform to its flow specification. In this case, the misbehaving flow may not be provided QoS guarantees by the network; however, the network must be designed such that its QoS guarantees to other flows are unaffected by the presence of some misbehaving flow.

Note that with the firewall property, when a policing mechanism at the network entrance of a flow fails, the failure's impact on the network is limited. In fact, a network-enforced policing mechanism may not be necessary; instead, the network may rely upon the source of each flow (e.g., a workstation) to police itself because of self-interest.

To provide burst QoS, a burst-based flow specification is needed for network design. In our model, a flow is a sequence of bursts, each of which is a sequence of packets that encapsulate an application data unit. We make two observations: (i) bursts vary greatly in size[2] and (ii) for any network to provide an upper bound on burst delay, the packets of a burst must arrive to the flow's network entrance within a bounded duration of time.

The first observation above suggests that networks should be designed to exploit information on the size of a burst or, alternatively, the bandwidth required over a time duration specified for the burst; this information can be supplied by the source of the burst. The second observation above suggests that a jitter constraint over the packet arrival times of each burst should be included in the burst-based flow specification (see Section 3 for a more detailed design.)

The next step in network design is the choice of a packet scheduling discipline that provides a delay guarantee to packets with the firewall property. Several rate-based packet scheduling disciplines meet this requirement [2, 10, 11, 14, 15], among others.[3] In Section 4, we choose one and modify it to provide a delay guarantee to bursts with the firewall property.

[2] For example, the pictures in a compressed video sequence vary greatly in the number of encoded bits [5].

[3] The FIFO discipline cannot be used because it does not have the firewall property.

In our flow model, the *rate of a burst* is defined to be the burst's size (bits or packets) divided by a source-specified duration for the burst (seconds). The peak rate of a flow is the largest rate over all bursts in the flow. In integrated-services networks, each channel carries traffic belonging to different service classes. A channel is said to be *overbooked for a service class*, if the sum of the peak rates of all flows in this service class carried by the channel exceeds the channel bandwidth allocated to the service class.

For a network to provide end-to-end delay guarantees to flows, an admission control mechanism is needed. Two different admission control policies can be used with the same packet scheduling discipline leading to two service classes with different end-to-end delay guarantees (in addition to best-effort service):

- *Real-time VBR service with no loss.* Each flow in this class is admitted at connection setup on the basis of its peak rate (no overbooking for this service class). Subsequently, each burst in the flow will be admitted by every switch in its path.[4]
- *Real-time VBR service at a specified loss rate.* For applications that can tolerate a small burst loss rate, their flows are admitted at connection setup on the basis of their peak and sustained rates, with overbooking allowed at each channel for this service class. The burst loss rate of flows in this class can be calculated and negotiated at connection setup [13].

We also allow burst-based rate allocation such that the reserved rate of a VBR flow is variable, i.e., it changes from one burst to the next. Specifically, at each channel, a reserved rate is not allocated to an admitted flow until the first packet of a burst arrives, and the rate is subsequently deallocated when the last packet of the burst departs. The reserved rate allocated to a burst is equal to the burst's rate, if it is not less than a minimum rate that depends upon the flow's negotiated QoS parameters (see Section 5 and [9]).

Burst-based rate allocation has the advantage of allocating a reserved rate that is exactly what a VBR flow needs at all time. In particular, for each real-time flow admitted at connection setup on the basis of its peak rate, any difference between the peak rate and current reserved rate is unallocated and available to flows in other service classes.

We next consider a real-time flow admitted at connection setup into a service class with overbooking. Each burst in such a flow is subject to admission control when its first packet arrives at a switch. Because of overbooking, the flow's outgoing channel may not have enough residual capacity to accommodate the burst. Specifically, the burst is admitted only if the burst's reserved rate does not exceed the channel's unallocated capacity.

Thus admission control decisions are made by a switch at two levels of time granularity: once for a flow at connection setup and once for each burst in the flow when its first packet arrives. Overbooking is allowed for flow admission. However, when an individual burst is admitted, channel capacity must not be exceeded by the aggregate rate allocated. Also, all packets of a burst are admitted or discarded as a whole. Such burst-based admission control is similar to the ATM block transfer (ABT) Immediate Transmission capability [4].

To put network design on a sound foundation, tight bounds on the end-to-end delays of bursts must be proved. Furthermore, a formula that relates the extent of overbooking

[4]Note that loss due to buffer overflow is possible.

for a service class to its burst loss rate must be derived. For the class of burst scheduling networks presented in Section 4 below, the end-to-end burst delay bounds can be found in Section 5. A burst admission control algorithm, derived from a generalized central limit theorem, is presented in [13]. Lastly, to achieve a high-speed implementation, we make priority computation for flows highly efficient by exploiting information in the burst-based flow specification (see Section 4).

3 Burst-based Flow Specification

For clarity of exposition, we assume in the balance of this paper that packets are of fixed size (such as ATM cells). The results and specifications to be presented can be modified in a straightforward manner for networks where the packet size is variable, but bounded. (In particular, the end-to-end delay guarantee theorem in [9] is applicable to variable-size packets.)

Recall that a flow is modeled as a sequence of bursts, each of which is a sequence of packets. The burst concept is needed for two purposes: (i) partitioning a flow into intervals that have substantially different rates, and (ii) specifying delay jitter bounds.[5]

For video, in particular, a burst is a sequence of packets that carry the encoded bits of a picture. Even with live video capture, note that as soon as the video source has encoded a picture, the size of the burst (in number of encoded bits or packets) is known and its rate (in bits/second or packets/second) can be determined. For example, if a picture has N encoded bits, the video display rate is 30 pictures per second, and the packet payload is 48 bytes, then the burst's rate is $30N$ bits/second or $(30N)/(48 \times 8)$ packets/second.

In what follows, we consider a particular flow f specified by the following notation:

(m, l)	the lth packet in the mth burst of flow f
$A(m, l)$	arrival time of packet (m, l)
b_m	size of burst m (packets)
δ_m	maximum duration of burst m (seconds)
λ_m	$\geq b_m/\delta_m$, reserved rate of burst m (packets/second)

The values of b_m and δ_m for burst m are to be supplied by the flow's source.

Flow Specification.

- The first and last packets of a burst can be uniquely identified. The first packet of burst m carries information on its rate, λ_m , in packets/second.[6]
- Packets in burst m satisfy a *jitter* timing constraint, namely: for $l = 1, 2, \ldots, b_m$,

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

[5] Two types of delay jitters can be defined: delay jitter over packets in a burst, and delay jitter over bursts in a flow.

[6] This part of the specification is implementation-dependent. The information in the first packet may be the burst's size in lieu of the burst's rate, or both may be included. Some other parameter value for the burst may also be included [9]. In ATM block transfer, the first and last packets are RM cells.

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

$$A(m+1,1) - A(m,1) \geq \frac{b_m}{\lambda_m} \tag{2}$$

Timing constraint (1) specifies a jitter bound over the packets of each burst; e.g., this constraint is satisfied if all packets of a burst arrive at the same time. Such a jitter bound is necessary if a network is to provide a burst delay bound.

Timing constraint (2) specifies a minimum separation between two consecutive burst arrivals in a flow. This is a form of source control. Furthermore, the constraint ensures that within a switch each active flow contains at most one active burst at any time; with this property, it is simple for a switch to check that the capacity of an output channel is not exceeded by the aggregate reserved rate of active flows.[7]

Though motivated by video traffic, the Flow Specification can be used for audio and data traffic that require a burst delay guarantee. In what follows, a flow that requires burst QoS guarantees from a network and conforms to the Flow Specification when entering the network is called a *guaranteed* flow.

4 Network Design

The next step in network design is to choose a packet scheduling discipline for each channel, and modify it to provide a burst delay guarantee.

Consider a channel shared by a set of flows, each of which is a sequence of packets. A deadline is associated with each packet. Packets in the same flow are given increasing deadlines, so that they are served in arrival order. (We follow the convention that a packet with a smaller deadline has a higher priority for service.) A queue is maintained for each flow. At the end of each packet transmission, a scheduler searches the head-of-line packets over all flows, and selects a packet with the smallest deadline to transmit next, if one (or more) exists. Service is nonpreemptive, i.e., each packet transmission, once begun, will not be preempted by the arrival of a higher-priority packet.

There are several packet scheduling disciplines that fit the above description, and provide a delay guarantee with the firewall property [2, 10, 11, 14, 15]. The disciplines differ mainly in how packet deadlines are defined. We have chosen the Virtual Clock (VC) algorithm for priority computation [15].[8] Specifically, a virtual clock value is associated with each packet, and packets are scheduled in increasing order of their virtual clock values. The virtual clock value of packet $i+1$ in a flow f is computed from[9]

$$P(i+1) = \max\{P(i), A(i+1)\} + \frac{1}{\lambda(f)} \tag{3}$$

where $P(i)$ is the virtual clock value of packet i, $A(i+1)$ is the arrival time of packet $i+1$, and $\lambda(f)$ denotes the reserved rate of flow f in packets/second.

[7] See [11] for a formal definition of *active* flow.

[8] We use the terms, *priority* and *deadline* interchangeably. In what follows, we have chosen to use the virtual clock value of a packet as its priority (deadline).

[9] Here, a flow is simply a sequence of packets.

In what follows, we shall refer to a channel that transmits packets in order of virtual clock value as a *VC server*. It has been shown that a VC server provides the following delay guarantee [11] to packets in flow f,

$$L(i+1) \leq P(i+1) + \frac{1}{\gamma} \tag{4}$$

where $L(i+1)$ is the departure time of packet $i+1$, and γ is the transmission rate in packets/second.

We have chosen the VC algorithm because its priority computation is highly efficient. It can be made even more efficient by exploiting information in Flow Specification (see Section 4.8).

The VC algorithm is often classified in the literature as being unfair, i.e., if a VC server has some capacity that has not been allocated to flows being served, such residual capacity is not shared among the flows in proportion to their reserved rates [14]. We note that this fairness property is not important to flows that are *not greedy*, i.e., flows that do not need any more bandwidth than their reserved rates. In designing a network for flows that conform to Flow Specification, it is easy to see that efficiency is an important concern, while fairness is not.

4.1 Burst-based rate allocation

Consider a flow, say f, which conforms to Flow Specification, arriving at a VC server. The flow is a sequence of bursts, each of which is a sequence of packets. Recall that (m, l) denotes the lth packet in the mth burst of the flow. For the VC server, define the following notation:

$P(m,l)$	virtual clock value of packet (m,l) at server
$L(m,l)$	departure time of packet (m,l) from server

The first packet of a burst, say m, carries information on the reserved rate of the burst, λ_m. Each switch uses this information to decide whether the burst can be admitted.[10] The burst is admitted only if its outgoing channel has enough residual capacity for flow f to be allocated a reserved rate equal to λ_m; consequently, the flow rate $\lambda(f)$ in (3) is set to λ_m for computing virtual clock values for packets in burst m. A decision not to admit the burst means that *all* packets in the burst will be discarded.

We next present two lemmas: Lemma 1 is a consequence of the jitter timing constraint, and Lemma 2 is a consequence of the separation timing constraint.

Lemma 1. The virtual clock value of the lth packet in burst m, $1 < l \leq b_m$, is

$$P(m,l) = P(m,1) + \frac{l-1}{\lambda_m} \tag{5}$$

[10] The reserved rate is $\max\{b_m/\delta_m\ ,\ \lambda_{m,min}\}$ where $\lambda_{m,min}$ is specified in Section 5.1. The decision would be unnecessary if the burst belongs to a flow that was admitted on the basis of peak rate at connection setup.

Lemma 2. If the VC server's capacity has not been exceeded for a nonzero duration since the start of a busy period, then the following bound holds for the lth packet of burst m served during the busy period, for $m \geq 1$, $l = 1, 2, \ldots, b_m$,

$$L(m,l) \leq A(m,1) + \frac{l}{\lambda_m} + \frac{1}{\gamma} \tag{6}$$

Proofs of the two lemmas are presented in the Appendix. Note that Lemma 2 provides a burst delay bound which depends on the length of burst m.

From Lemma 1 and the jitter timing constraint, observe that once the first packet of a burst has arrived, its flow will remain continuously *active* (as defined in [11]) until some time after the last packet of the burst arrives. Note that $P(m,l)$ can be computed more efficiently from (5) than from (3).

Lastly, note that the separation timing constraint ensures that within a switch each flow has at most one active burst at any time; thus the rate allocated to a flow is equal to the rate allocated to its active burst. (Otherwise, in determining whether the server capacity is exceeded, the rate allocated to a flow would be equal to the sum of the rates allocated to simultaneously active bursts in the flow.)

4.2 Restructuring and retiming of bursts

Consider the packets of a guaranteed flow, which traverse a sequence of nodes indexed by $0, 1, 2, \ldots, K+1$, where node 0 denotes the source, and node $K+1$ the destination. The other nodes are switches, where each outgoing channel is a VC server. At the network entrance, a *source regulator* ensures that the flow's packets conform to Flow Specification when they arrive at node 1.

Note that the sequence of packets leaving node 1 may or may not satisfy the jitter and separation timing constraints. The same can be said about packets leaving node 2, etc. Since the timing constraints are assumed by every switch along the path, packets are delayed by *flow regulators* to ensure that both timing constraints are satisfied when packets *become eligible* at their next VC server. Specifically, the times when packets become eligible at nodes $2, 3, \ldots, K$ are taken to be their arrival times for the purpose of checking satisfaction of timing constraints (1) and (2), and applying Lemma 2.

To see how to design flow regulators in switches, consider the packets of a guaranteed flow as they leave a VC server. From (4) and (5), we have for $m \geq 1$, $l = 1, 2, \ldots, b_m$,

$$L(m,l) - P(m,1) \leq \frac{l-1}{\lambda_m} + \frac{1}{\gamma} \tag{7}$$

For the departure times of packets to satisfy the jitter timing constraint, the following is needed, for $m \geq 1$, $l = 1, 2, \ldots, b_m$,

$$0 \leq L(m,l) - L(m,1) \leq \frac{l-1}{\lambda_m} \tag{8}$$

Comparing (7) and (8), we see that upon leaving a VC server the packets in burst m need to be restructured as shown in (9) before they are eligible at the next VC server, for

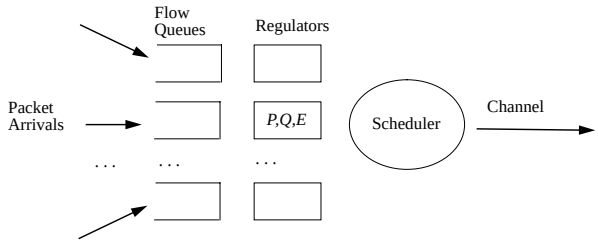

Figure 1: Architecture of a channel.

$m \geq 1$, $l = 1, 2, \ldots, b_m$,

$$L(m,l) := \max\{L(m,l),\ P(m,1) + \frac{1}{\gamma}\} \tag{9}$$

In addition to restructuring the packets within a burst to preserve the jitter timing constraint, burst m may have to be retimed in relation to burst $m-1$ to satisfy the separation timing constraint. Burst retiming is carried out by delaying the times when packets become eligible at the next VC server as follows, for $m > 1$, $l = 1, 2, \ldots, b_m$,

$$A(m,l) := \max\{A(m,l),\ A(m-1,1) + \frac{b_{m-1}}{\lambda_{m-1}}\} \tag{10}$$

In the algorithm to be presented in Section 4.4, both the restructuring in (9) and the retiming in (10) are needed only for the first packet of each burst, and are not performed until this packet gets to the head of its queue in the next switch along the path. Both operations are achieved by delaying the packet slightly.

Note that if the regulator, queue and VC server within a switch are considered together as a system, such system is not work-conserving, because the server may be idle when there are packets being delayed by regulators. However, it is important to note that neither burst restructuring nor retiming affect the worst-case end-to-end delay in burst scheduling networks, i.e., the end-to-end delay upper bound presented in Section 5 is the same in the absence of burst restructuring and retiming. (The end-to-end delay lower bound would be different.)

4.3 Channel architecture

Packets arrive to a switch from sources and other switches. Each packet, depending upon its destination, is routed to one of the outgoing channels of the switch. The architecture of a channel is illustrated in Figure 1. For each channel, separate queues are maintained for packets belonging to different flows; each guaranteed flow is allocated its own buffers. There is a regulator for the queue of each flow. There is a scheduler for each channel.

4.4 Source regulator

A source regulator is used at the network entrance of a guaranteed flow to ensure that its packets conform to Flow Specification. Specifically, the first packet of burst m has two fields,[11]

λ_m	reserved rate of burst m (in packets/second)
u_m	time ahead (in seconds); initialized to zero at source

The source regulator performs the following tasks:

- Store values in the λ_m and u_m fields of the first packet of burst m (using zero for u_m). For each burst, mark the first and last packets.
- Ensure that the jitter and separation timing constraints are satisfied.

4.5 Flow regulator

Consider the bursts in a guaranteed flow. The separation timing constraint requires that burst $m+1$ follows burst m, for all m. The jitter timing constraint requires that packet l in a burst, $l > 1$, cannot overtake the first packet in the burst. It is easy to see that both of these properties are preserved by point-to-point transmission channels. We require that both properties be preserved by the queue of each flow. (It is sufficient to use a FIFO queue.) As a result, the timing constraints in Section 4.2, specified by (9) and (10), are satisfied for all l if they are satisfied for $l = 1$. This observation is used to simplify the regulator specification below.

When a new flow arrives at a switch (after an end-to-end session has been established), a queue is created for the flow, as well as a flow regulator for the queue. There are three variables associated with each queue: P, Q, and E, defined below.[12] There are four variables associated with burst m: a_m and m, defined below, and λ_m and u_m, defined in Section 4.4. The flow regulator can read and write all of these variables.

P	virtual clock value of head-of-line packet in queue; initially 0
Q	time when burst is eligible for selection by scheduler; initially 0
E	boolean flag, indicating that the flow has an eligible burst; initially **false**
m	burst number; initially 1
a_m	arrival time of first packet of burst m

The regulator specification uses a function, $now()$, and a procedure $update(P,E)$. When called, $now()$ returns the current time from a local clock in the switch. The *update* procedure is specified as follows:

[11] These fields are implementation-dependent. Other parameter values may be specified instead.

[12] The specifications can be rewritten without using the variable Q. It is included here for clarity of exposition.

```
procedure update(P, E)        // execute once per burst
Q := max{a_m + u_m, P} ;      // compute time when burst m is eligible
delay(Q − now()) ;
P := Q + 1/λ_m ;              // from (34) in Appendix
E := true;
```

where the procedure $delay(x)$ introduces a delay equal to x if $x > 0$; else, it is a null operation with no delay. A flow regulator is specified by the following two actions:

- Enabling condition: arrival of a burst m packet to queue

```
if (packet is first in burst m)
        then  record arrival time in a_m and values of u_m and λ_m ;
              if (queue was empty before arrival)
                    then update(P, E) ;
```

- Enabling condition: departure of a burst m packet from queue (selected for service by scheduler)

```
if (departed packet is not last in burst m)
        then  P := P + 1/λ_m    // from (29) and (30) in Appendix
        else  E := false ;
              m := m + 1 ;
              if (queue is not empty)
                    then update(P, E) ;
```

Note that procedure $update(P,E)$ is executed only for the first packet of each burst. Specifically, u_m contains information on how much the first packet is ahead of schedule and can be delayed to achieve burst restructuring specified by (9). When the procedure is called, P contains the earliest time when the retiming constraint in (10) is satisfied. Thus both restructuring and retiming of burst m are achieved by executing the second statement in $update(P,E)$; the updated value of Q is the time when the first packet of burst m is *eligible* for selection by the channel scheduler. Note that after the first packet in a burst becomes eligible, all other packets in the burst are eligible.

The updated value of Q should be interpreted as the arrival time of packet $(m,1)$ at a VC server, as used in Lemma 2. Also the virtual clock value of packet $(m,1)$ is given by the third statement of $update(P,E)$, which is based upon (34) proved in the Appendix.

4.6 ABR traffic

Burst scheduling networks are designed for integrated services. Some packet flows do not require burst delay guarantees. In this paper, we refer to them as available bit rate (ABR) traffic. An ABR flow does not conform to Flow Specification. Some of the flows in Figure 1 may be ABR flows. For an ABR flow, its flow regulator has only one task, which is to

compute virtual clock values. To illustrate, we provide a regulator specification below for the case of a single ABR flow sharing a channel with a set of guaranteed flows. (Note that multiple flows can be handled as a single ABR flow when no minimum rate is allocated to any of the flows. If some flows are allocated minimum rates, they should be handled as separate ABR flows; modifying the following regulator specification is straightforward.)

Let α denote the fraction of channel capacity that has been allocated to guaranteed flows. Thus at least $(1-\alpha)$ of the channel capacity is available to the ABR flow. Whenever there is nothing to transmit by any of the guaranteed flows, the entire channel capacity is available to the ABR flow.

The regulator for the ABR flow is specified differently from the flow regulator in Section 4.5. In particular, the E flag is true if and only if the queue is nonempty; it is omitted from the regulator specification below. Also, the variable Q is not needed. The variable P is initially zero. The ABR flow regulator is specified by the following two actions:

- Enabling condition: packet arrival to ABR queue

```
1    if (queue was empty before arrival)
2        then  P := max{P, now()} + 1/((1 - α)γ)
                                   // now() returns packet arrival time
```

- Enabling condition: packet departure from ABR queue (selected for service by scheduler)

```
1    if (queue is not empty after departure)
2        then  P := P + 1/((1 - α)γ)
                                   // from applying (4)
```

Note that if the ABR flow regulator can detect the boolean condition,

$$G_empty = \text{(for all guaranteed flow :: queue is empty or its } E \text{ flag is false)}$$

it may reset the the virtual clock value of ABR to $now()$ whenever G_empty is true. Such virtual clock resets allow the ABR flow to be greedy without being punished later; the resets would not affect the delay bounds of guaranteed flows, provided that the aggregate reserved rate of guaranteed flows does not exceed $\alpha\gamma$.

4.7 Channel scheduler

The channel scheduler can read variables P and E of every queue.[13] We use S to denote the set of nonempty queues where E is true, i.e., the set of flows with eligible packets waiting. The channel scheduler is specified by the following action:

[13] The flow regulator can read and write both variables. If the channel scheduler and flow regulators are to execute concurrently, some access constraints may be required for their actions to be *atomic*.

- Enabling condition: end of a packet transmission or wakeup

```
if (S is not empty)
    then select from S the flow with smallest P ;
         remove head-of-line packet from flow ;
         if (packet is first in burst m)
             then write value of ( P − now() ) into u_m field of packet ;
         transmit packet;
```

In the above action, the current time returned by $now()$ is the time when the packet transmission begins (assuming no intervening delay). The value of $P - now()$ written into u_i is guaranteed by (4) to be nonnegative.

Note that if the set S is empty, the channel scheduler may go to sleep. When the scheduler is sleeping, it would be necessary for a flow regulator to send a wakeup signal when a packet arrives. Such implementation details are beyond the scope of this paper.

4.8 Algorithm efficiency

The algorithms presented in Section 4.7 have been designed to be highly efficient. For a guaranteed flow, its regulator executes the procedure $update(P,E)$ only once per burst, i.e., for the first packet in each burst. For any other packet in burst m, the flow regulator simply increments P by the value of $1/\lambda_m$, instead of executing the algorithm in (3). This is made possible by Lemma 1, which follows from the jitter timing constraint.

In computing virtual clock values for the ABR flow, only the first packet of each busy period of the ABR queue requires the algorithm in (3). For any other packet in a busy period, the regulator simply increments P by the value of $1/((1-\alpha)\gamma)$. This is a consequence of the delay guarantee in (4) and the assumption of a fixed packet size.

With the exception of the first packet of each burst in guaranteed flows, there is no need to store packet arrival times, as suggested in the original proposal [15]. Furthermore, at any time, only one virtual clock value is stored per flow, rather than one per packet.

5 Burst Delay Bounds

Consider a guaranteed flow traversing a path of $K+2$ nodes, where node 0 denotes the source, node $K+1$ the destination, and nodes $1, 2, \ldots, K$ switches.

Notation.

γ_k	capacity of channel from node k to node $k+1$ (packets/second)
$\tau_{k,k+1}$	channel propagation time from node k to node $k+1$ (seconds)
α_k	$= (1/\gamma_k) + \tau_{k,k+1}$ (seconds)
$D(m,l)$	end-to-end delay of lth packet in mth burst, measured from time of arrival at node 1 to time of arrival at node $K+1$

In the following analysis, it is assumed that each channel delivers the flow's packets reliably and in order. Also, for every channel in the path, the aggregate reserved rate of active

guaranteed flows does not exceed the capacity allocated to guaranteed flows. Furthermore, processing times of the router, regulator and scheduler functions in switches do not increase the delay of any packet.

This last assumption is reasonable because the functions can be carried out in parallel with an ongoing transmission, i.e., the router, regulator, and scheduler process packets that are queued. (We can think of two exceptions: (1) when a packet arrives to a channel where all queues are empty, and (2) when the channel scheduler writes the value of $P - now()$ into the u_i field of the first packet of a burst; this delay can be accounted for by increasing $1/\gamma_k$ slightly.)

Theorem 1. The end-to-end delay of the first packet of burst m, for $m = 1, 2, \ldots$, has the following upper and lower bounds:

$$D(m,1) \leq \frac{1}{\lambda_m} + (K-1) \max_{1 \leq n \leq m} \{\frac{1}{\lambda_n}\} + \sum_{k=1}^{K} \alpha_k \tag{11}$$

$$D(m,1) \geq (K-1)\frac{1}{\lambda_m} + \sum_{k=1}^{K} \alpha_k \tag{12}$$

A proof of Theorem 1 is given in the Appendix. Note that a flow regulator at each node preserves the jitter timing constraint for each burst in a guaranteed flow, except at node $K+1$.[14] The delay of packet (m,l) is bounded as follows:

$$D(m,l) \leq D(m,1) + \frac{l}{\lambda_m} \tag{13}$$

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:

$$D_m \leq D(m,1) + \frac{b_m}{\lambda_m} \leq D(m,1) + \delta_m \tag{14}$$

5.1 QoS parameters

The source of a real-time flow negotiates with a 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 paper, the following QoS parameters are relevant (among others):

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

[14] It is assumed that the first packet of burst m does not undergo burst restructuring when it arrives at node $K+1$. Thus we have l instead of $l-1$ in (13).

For a source to conform to the negotiated maximum rate, λ_{max}, it is sufficient that the source controls its burst sizes, such that, for all m,

$$\frac{b_m}{\delta_m} \leq \lambda_{max} \tag{15}$$

If the flow conforms to Flow Specification at its network entrance, the network will ensure that burst delays do not exceed D_{max}. To do so, the reserved rate allocated to each burst in the flow cannot be too small. From (11) and (14), the reserved rate allocated to burst m must be larger than

$$\lambda_{m,min} = (b_m + K)/(D_{max} - \sum_{k=1}^{K} \alpha_k) \tag{16}$$

Thus the reserved rate of burst m is $\max\{b_m/\delta_m, \lambda_{m,min}\}$. Note that if $\lambda_{m,min}$ turns out to be larger than λ_{max} for some m, there is a conflict between the negotiated values of λ_{max} and D_{max}. A renegotiation between source and network would be required.

5.2 Conditional guarantee and firewall property

A service provider is generally designed to provide service guarantees that are *conditional*.[15] In this paper the network layer being designed offers service guarantees to a higher layer (source of a flow) by making use of service guarantees offered by a lower layer (communication channels). In this environment, the network layer is obligated to provide burst delay bounds to a flow only if

- the flow conforms to Flow Specification at its network entrance, and
- channels deliver the flow's packets reliably (i.e. in-order delivery with no loss).

However, when a flow, say v, misbehaves (as a result of unreliable packet delivery or source regulator malfunctioning), the network's service guarantees to other flows should be unaffected. Burst scheduling networks inherit such a firewall property from the delay guarantee of a VC server [8, 11] under several assumptions. First, switches are reliable. Second, each guaranteed flow is allocated its own buffers in a switch. Third, each switch allocates reserved rates to flows and can ensure that the capacity of each of its output channels is not exceeded by the aggregate rate allocated.

6 Extensions and Related Work

The design of burst scheduling networks, as described in Section 3-5, was first presented in the 1994 Computer Communications Workshop and published in [7]. Subsequently, we have made progress in addressing several implementation issues. We have also generalized the end-to-end delay guarantee from VC servers to a class of guaranteed-deadline servers. A brief overview follows.

[15] For an in-depth treatment of assumptions and guarantees between service providers and consumers, see [6].

The scheduler of a VC server (in fact, any priority server) must repeatedly search for the smallest element in a set of priority values (deadlines). For high-speed networks of the future, it is likely that a channel will be shared by hundreds, and perhaps, thousands of flows. Thus the search algorithm should be highly efficient. Furthermore, each search must be carried out within a time bound, i.e., the search must be finished by the end of the current packet transmission. Otherwise, the channel would be idled, ready packets would incur additional delays, and delay guarantees would not hold. In [12], we present a search algorithm based upon a novel data structure, called *adaptive heap*, which behaves like a heap most of the time, but adaptively changes its strategy when necessary to satisfy the time bound. We have shown that the algorithm has optimal worst-case time performance and good average performance.

To make adaptive heap search even more efficient, particularly when channel utilization is high, we have modified the channel scheduler to implement *group priority*. Specifically, consecutive packet arrivals in a flow are partitioned into groups. The largest deadline among packets in a group is assigned to every packet in the group. (Thus all packets except one in the group have relaxed deadlines.) For each burst, say m, the group size g_m is a parameter whose value can be chosen such that the worst-case end-to-end burst delay of a flow is unaffected by the use of group priority [9].

Group priority has two advantages. First, the channel scheduler's work is much reduced, particularly when the channel is heavily utilized [12]. This is because the priority of each flow changes only once per group rather than once per packet; hence the scheduler updates its priority data structure less often. Second, we discovered empirically that the use of group priority results in much better statistical performance (i.e., delay, queue size, and loss probability) for networks where some channels are heavily utilized. This is because group sizes are chosen such that the ratio g_m/λ_m is approximately the same for all bursts in the same flow. Thus a large group size is used for a large burst with a high reserved rate, resulting in relaxed deadlines for many packets in the burst, and better statistical performance at a heavily utilized channel.

An *end-to-end delay guarantee theorem* for guaranteed-deadline (GD) servers is presented in [9]. The theorem can be instantiated to obtain end-to-end delay bounds for a variety of GD servers and source control mechanisms.[16] In particular, it can be instantiated to obtain the following end-to-end delay upper bound for VC servers, group priority, and Flow Specification:

$$D(m,1) \leq \frac{g_m}{\lambda_m} + (K-1) \max_{1 \leq n \leq m} \{\frac{g_n}{\lambda_n}\} + \sum_{k=1}^{K} \alpha_k \tag{17}$$

$$D_m \leq D(m,1) + \frac{b_m}{\lambda_m} \tag{18}$$

Note that the upper bound in (17) subsumes the upper bound in Theorem 1 as a special case, i.e., $g_m = 1$ for all m.

In Section 4, we have shown how to share a channel among flows belonging to multiple service classes (e.g., real-time service, best-effort service). As discussed in Section 2, a real-time VBR service with no loss can be provided to a flow if the flow is admitted on

[16] Also, packet sizes can be variable as long as they are bounded.

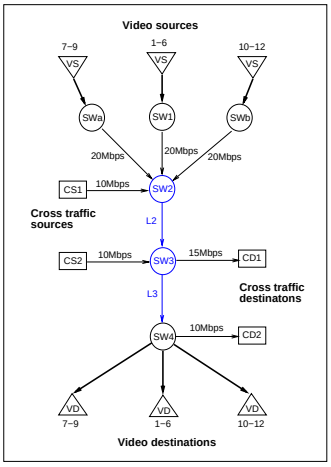

Figure 2: Simulated network.

MPEG sequences	encoding pattern	λ (Mbits/s)		
		min	max	ave
Terminator	(3, 6)	0.14	3.86	1.15
ParentsSon	(3, 6)	0.37	5.97	1.51
RedsNightmare	(10, 30)	0.089	3.62	0.75
Student	(1, 4)	0.48	2.47	1.27
Driving	(3, 9)	0.17	8.48	1.88
Airwolf 2	(3, 6)	0.14	3.31	0.89
Simpsons I	(3, 6)	0.14	2.60	0.92
Canyon	(3, 6)	0.076	0.70	0.28
FlowerGarden	(3, 6)	1.39	13.25	5.04
UnderSiege	(3, 6)	0.17	2.02	0.59
StarTrek II	(0, 1)	0.28	1.15	0.62
Energizer	(3, 6)	0.17	2.34	0.76

Table 1: MPEG sequences used in experiments

the basis of its peak rate.[17] Alternatively, a flow can be admitted on the basis of its peak and sustained rates, with some overbooking allowed, such that it is provided a real-time VBR service at a specified loss rate. An algorithm for determining how much overbooking is allowed to achieve a specified loss rate at a switch is presented in [13].

Lastly, we envision that future integrated services packet-switching networks will support not only link sharing by multiple service classes but also by multiple administrative classes (e.g., different agencies and organizations) [1, 3]. The end-to-end delay upper bound in (17) above has been further generalized to networks with hierarchical link sharing [13].

[17] Loss due to buffer overflow is still possible. Note also that with burst-based rate allocation, most bursts in the flow are actually allocated a reserved rate less than the flow's peak rate.

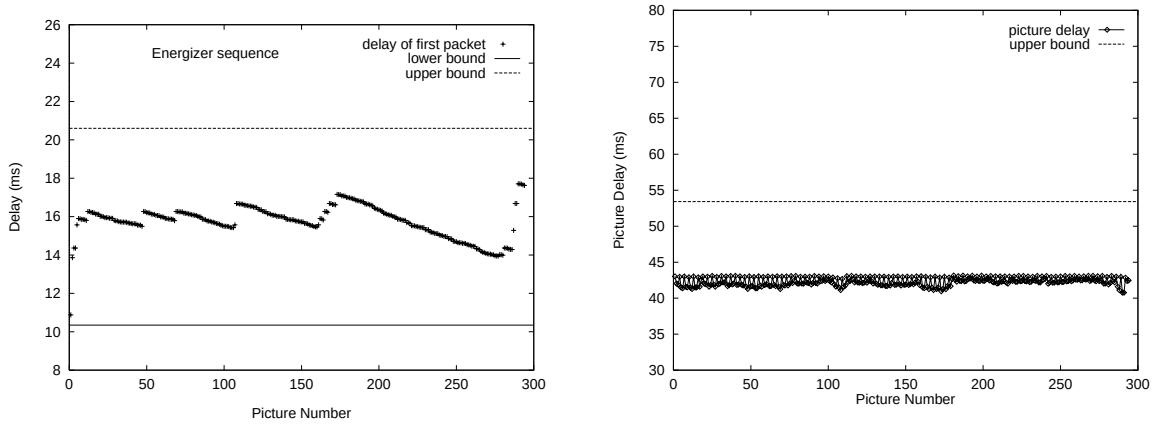

Figure 3: End-to-end delays of Energizer sequence

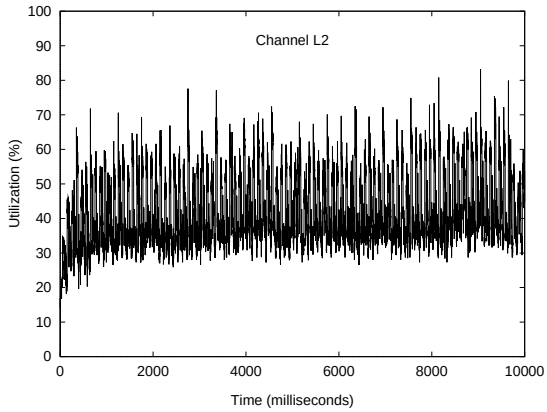

Figure 4: Utilization of channel L2 over time.

7 Experimental Results

We have performed a large number of experiments using a discrete-event simulator driven by MPEG video traces representing real-time VBR flows. The network simulated is illustrated in Figure 2. There are six switches labeled SW. Each switch has a buffer pool for 1200 packets, which is shared by all video flows.[18] For the same reasons discussed at the beginning of Section 5, we have omitted any processing delay in a switch for routing, regulating and scheduling.

Each thin arrow in Figure 2 represents a channel, which (except for L2 and L3) is labeled by its capacity in megabits per second (Mbps). Channel propagation delays vary ranging from 0.3 to 3 milliseconds (ms). Channels L2 and L3 have the same capacity C. The value of C can be changed from one experiment to another to vary the channel utilization. Each

[18]With a large buffer pool, the assumption that each guaranteed flow is allocated its own buffers is unnecessary in the simulation.

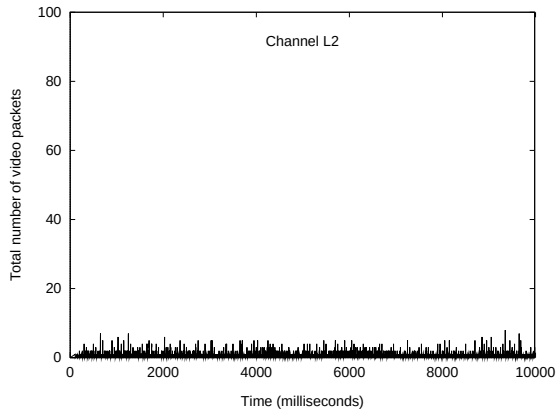

Figure 5: Total number of video packets at channel L2 over time.

thick arrow represents a set of channels, one for each video flow. Each such channel has a capacity larger than the peak rate of the flow it carries; the capacity varies from 10 to 15 Mbps, and the propagation delay also varies. Channels are assumed to be lossless.

The simulated network carries twelve video flows, as well as some ABR traffic. In Figure 2, 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 the MPEG video sequences shown in Table 1.

Each picture in a video sequence is modeled as a burst in Flow Specification. The rate λ_m of picture (burst) m 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.[19] In Table 1, λ_{max} and λ_{min} denote the rates of the largest and smallest pictures (in number of encoded bits), respectively.

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. Whenever there was nothing to send from the video flows, the entire channel capacity was available to ABR traffic.

[19] We conducted several experiments in which the packets of a burst were generated in batches of 40 each, with a fixed interbatch gap. Compared to the results presented herein, we observed that such batch arrivals had no impact on worst-case end-to-end burst delays. The queue sizes were larger. The average end-to-end burst delays were actually smaller because, with batch arrivals, burst durations were smaller on the average.

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.

In Figures 3-5, we present results from an experiment in which the value of C was chosen such that $0.8C$ equals the sum of the peak rates in Table 1, such that each video flow can be allocated a share of the channel capacity equal to its peak rate. In Figure 3, we show (on the left) the end-to-end delay of the first packet in each picture of the Energizer sequence. Note that the upper and lower bounds in Theorem 1 are functions of the rate λ_m of burst m which varies from picture to picture. The upper bound shown is the upper bound of the smallest picture in the sequence, and the lower bound shown is the lower bound of the largest picture in the sequence. Also in Figure 3 (on the right), we show the end-to-end picture delay for every picture in the Energizer sequence. We observed that all of the bounds held for all video flows during the experiment, as predicted by theory (since there was no overbooking).

The channel utilization of L2 is shown as a function of time in Figure 4. The average utilization was 0.42 for the duration of the experiment. The total number of video packets in SW2, summed over all twelve flows, is shown as a function of time in Figure 5. Note that the buffer requirement is very low.

We also carried out experiments in which the channel capacity was overbooked to increase channel utilization. The results of some of these experiments can be found in [9].

8 Conclusion

We have presented an approach towards designing integrated services packet-switching networks that provide QoS guarantees to application data units. In particular, we have presented a burst-based flow specification. A flow is modeled as a sequence of bursts, each of which models a sequence of packets that encapsulate an application data unit.

We have presented an architecture and algorithms for packet scheduling, and tight bounds on end-to-end burst delays. We have shown how the burst-based flow specification can be exploited to improve implementation efficiency. We have also described how burst scheduling networks can be designed to provide both a real-time VBR service with no loss and, with burst-based admission control, a real-time VBR service at a specified loss rate.

Our concept of a burst subsumes the concept of a block in the ATM literature, where a block represents a sequence of cells. We advocate that the QoS parameters of cell transfer delay and cell loss rate in the ATM Forum Traffic Management specification be generalized to block transfer delay and block loss rate, respectively. We note that such a generalization is backward-compatible since a block, being a sequence of cells, includes a cell as a special case.

Acknowledgement

We thank the anonymous referees for their constructive comments. In response to these comments, our presentation has been improved.

References

[1] David Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM '92*, pages 14–26, August 1992.

[2] 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.

[3] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[4] ITU-T Rec. I.371. *Traffic Control and Congestion Control in B-ISDN*. Perth, U.K., Nov. 6-14, 1995.

[5] 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.

[6] Simon S. Lam and A. Udaya Shankar. A theory of interfaces and modules I: Composition theorem. *IEEE Transactions on Software Engineering*, 20(1):55–71, January 1994.

[7] Simon S. Lam and Geoffrey G. Xie. Burst scheduling networks. Technical Report TR-94-20, University of Texas at Austin, July 1994. An abbreviated version in *Proceedings of IEEE INFOCOM '95*, April 1995.

[8] Simon S. Lam and Geoffrey G. Xie. Burst Scheduling networks: Flow specification and performance guarantees. In *Proceedings of NOSSDAV*, Durham, New Hampshire, April 1995.

[9] Simon S. Lam and Geoffrey G. Xie. Group priority scheduling. Technical Report TR-95-28, University of Texas at Austin, July 1995. Revised, September 1996. An abbreviated version in *Proceedings of IEEE INFOCOM '96*, March 1996.

[10] 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*, 1(3):344–357, June 1993.

[11] Geoffrey G. Xie and Simon S. Lam. Delay guarantee of Virtual Clock server. *IEEE/ACM Trans. on Networking*, 3(6):683–689, December 1995. Presented at *9th IEEE Workshop on Computer Communications*, October 1994.

[12] Geoffrey G. Xie and Simon S. Lam. An efficient adaptive search algorithm for scheduling real-time traffic. In *Proceedings of IEEE ICNP '96*, Columbus, Ohio, October 1996.

[13] Geoffrey G. Xie and Simon S. Lam. Real-time block transfer under a link sharing hierarchy. Technical Report TR-96-19, University of Texas at Austin, June 1996. Available from http://www.cs.utexas.edu/users/lam/NRL/ . To appear in *Proc. of IEEE INFOCOM '97*.

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

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

A Appendix

A.1 Proof of Lemma 1

From (3), for $n = 1, 2, \ldots$ and $1 < l \leq b_m$,

$$\begin{aligned} P(m,l) &= \max\{A(m,l), P(m,l-1)\} + \frac{1}{\lambda_m} && (19) \\ &\geq P(m,l-1) + \frac{1}{\lambda_m} && (20) \\ &\cdots \\ &\geq P(m,1) + \frac{l-1}{\lambda_m} && (21) \end{aligned}$$

Also from (3), we have

$$P(1,1) = A(1,1) + \frac{1}{\lambda_1} \tag{22}$$

and for $m > 1$,

$$P(m,1) = \max\{A(m,1), P(m-1, b_{m-1})\} + \frac{1}{\lambda_m} \tag{23}$$

Therefore,

$$P(m,1) \geq A(m,1) + \frac{1}{\lambda_m} \tag{24}$$

Combining (21) and (24), we have for $1 < l \leq b_m$,

$$\begin{aligned} P(m,l) &\geq [A(m,1) + \frac{1}{\lambda_m}] + \frac{l-1}{\lambda_m} && (25) \\ &= A(m,1) + \frac{l}{\lambda_m} && (26) \end{aligned}$$

Combining (24) and (26), we have for $1 < l \leq b_m$,

$$P(m,l-1) \geq A(m,1) + \frac{l-1}{\lambda_m} \tag{27}$$

By (1) in Flow Specification, we have for $1 < l \leq b_m$,

$$A(m,l) \leq A(m,1) + \frac{l-1}{\lambda_m} \tag{28}$$

Combining (27) and (28), we have for $1 < l \leq b_m$,

$$P(m,l-1) \geq A(m,l) \tag{29}$$

From (19) and (29), we have for $1 < l \leq b_m$,

$$P(m,l) = P(m,l-1) + \frac{1}{\lambda_m} \tag{30}$$

Applying (30) repeatedly, for $1 < l \leq b_m$,

$$\begin{aligned} P(m,l) &= P(m,l-1) + \frac{1}{\lambda_m} && (31) \\ &= P(m,l-2) + \frac{2}{\lambda_m} && (32) \\ &\cdots \\ &= P(m,1) + \frac{l-1}{\lambda_m} \end{aligned}$$

□

A.2 Proof of Lemma 2

From Lemma 1, we have for $1 < l \leq b_m$,

$$P(m,l) = P(m,1) + \frac{(l-1)}{\lambda_m} \tag{33}$$

Next we use induction to show that for all $m = 1, 2, \ldots$,

$$P(m,1) = A(m,1) + \frac{1}{\lambda_m} \tag{34}$$

1. For $m = 1$, (34) holds by (3),

$$P(1,1) = A(1,1) + \frac{1}{\lambda_1} \tag{35}$$

2. Assume that (34) holds for $m = m' \geq 1$.

3. For $m = m' + 1$, by (2) in Flow Specification,

$$\begin{aligned} A(m'+1,1) &\geq A(m',1) + \frac{b_{m'}}{\lambda_{m'}} && (36) \\ &= [A(m',1) + \frac{1}{\lambda_{m'}}] + \frac{b_{m'}-1}{\lambda_{m'}} && (37) \\ \{by\ induction\ hypothesis\} \\ &= P(m',1) + \frac{b_{m'}-1}{\lambda_{m'}} && (38) \\ \{from\ (33)\} \\ &= P(m',b_{m'}) && (39) \end{aligned}$$

From (3),

$$\begin{aligned} P(m'+1,1) &= \max\{A(m'+1,1), P(m',b_{m'})\} + \frac{1}{\lambda_{m'+1}} && (40) \\ \{from\ (39)\} \\ &= A(m'+1,1) + \frac{1}{\lambda_{m'+1}} && (41) \end{aligned}$$

Therefore (34) holds for all $m = 1, 2, \ldots$.

Combining (33) and (34), we have for all $m = 1, 2, \ldots$, and $1 \leq l \leq b_m$,

$$P(m,l) = A(m,1) + \frac{l}{\lambda_m} \tag{42}$$

By (4),

$$\begin{aligned} L(m,l) &\leq P(m,l) + \frac{1}{\gamma} \\ &= A(m,1) + \frac{l}{\lambda_m} + \frac{1}{\gamma} \end{aligned}$$

□

A.3 Proof of Theorem 1

Lower bound

For burst m to travel from node k to node $k+1$, for $k = 1, 2, \ldots, K$, it incurs a transmission delay of $1/\gamma_k$ and a propagation delay of $\tau_{k,k+1}$, which sum to α_k.

Additionally, it incurs a combined delay of $1/\lambda_m$ in nodes k and $k+1$, for $k = 1, 2, \ldots, K-1$, as follows: Consider the first packet of burst m in node k. After procedure *update* has been executed, the value of Q is the time when packet $(m,1)$ becomes eligible. The value of P, equal to $Q + 1/\lambda_m$, is its deadline. Suppose the packet is selected for transmission at time $Q + y$, where $0 \leq y \leq 1/\lambda_m$. The scheduler writes the value of $(1/\lambda_m) - y$ into the u_m field of the packet, causing the packet to be delayed this much time in node $k+1$ when procedure *update* executes there. Thus there is a combined delay of $1/\lambda_m$, with a delay of y in node k and $1/\lambda_m - y$ in node $k+1$. The lower bound on $D(m,1)$ in Theorem 1 follows immediately.

Note that the delay of $1/\lambda_m - y$ in node $k+1$, caused by procedure *update*, is precisely the extent to which the packet is ahead of its deadline. Therefore, the worst-case delay of packet $(m,1)$ is unaffected by burst restructuring and retiming.

Upper bound

Since the worst-case delay of packet $(m,1)$ is unaffected by burst restructuring and retiming, the upper bound in Theorem 1 can be obtained as a special case of the end-to-end delay guarantee theorem in [9], which has a simple proof. The following discussion and proof, first presented in [7], provide insights towards understanding the behavior of burst scheduling networks.

For an arbitrary packet i in a flow, let $A^{(k)}(i)$ denote the time when the packet first becomes eligible at node k, i.e., the first time when the packet is at the head of its queue and the queue's E flag is true. Define

$$Gap^{(k)}_{m,m+1} = A^{(k)}(m+1,1) - (A^{(k)}(m,1) + \frac{b_m}{\lambda_m}) \tag{43}$$

where $Gap^{(k)}_{m,m+1}$ is said to be the gap between bursts m and $m+1$ at node k. This gap may increase, decrease, or stay the same as the two bursts travel through the path of nodes,

depending upon the values of λ_m and λ_{m+1}, as follows:

$$Gap_{m,m+1}^{(k+1)} \leq \max\{Gap_{m,m+1}^{(k)} + (\frac{1}{\lambda_{m+1}} - \frac{1}{\lambda_m}),\ 0\} \tag{44}$$

There are two possible cases:

- $\lambda_m > \lambda_{m+1}$

  The gap increases as burst $m+1$ lags further behind burst m after passing through each node, unless burst m is delayed by burst $m-1$ in which case the gap between burst m and burst $m+1$ may actually decrease.

- $\lambda_m \leq \lambda_{m+1}$

  The gap decreases as burst $m+1$ tends to get closer to burst m after passing through each switch, but the interburst gap is bounded below by zero due to burst retiming.

Since the separation timing constraint prevents burst $m+1$ from passing burst m, one can think of the progress of bursts along the path from source to destination as a car race confined to a single lane, where cars travel at different speeds but passing is not allowed.

Proof of upper bound

We will prove the upper bound in Theorem 1 by induction on m. Before doing so, we generalize the gap definition from two consecutive bursts to two arbitrary bursts p and q in the same flow ($q > p$), that is:

$$Gap_{p,q}^{(k)} = A^{(k)}(q,1) - A^{(k)}(p,1) - \sum_{n=p}^{q-1} \frac{b_n}{\lambda_n} \tag{45}$$

$$= \sum_{n=p}^{q-1} Gap_{n,n+1}^{(k)} \tag{46}$$

From the definition in (45), we have

$$A^{(K)}(q,1) - A^{(1)}(q,1) \leq A^{(K)}(p,1) - A^{(1)}(p,1) \Leftrightarrow Gap_{p,q}^{(K)} \leq Gap_{p,q}^{(1)} \tag{47}$$

By definition

$$D(m,1) = L^{(K)}(m,1) + \tau_{K,K+1} - A^{(1)}(m,1) \tag{48}$$

$\{by\ Lemma\ 2\}$

$$\leq (A^{(K)}(m,1) + \frac{1}{\lambda_m} + \frac{1}{\gamma_K}) + \tau_{K,K+1} - A^{(1)}(m,1) \tag{49}$$

where $L^{(K)}(m,1)$ denotes the departure time of packet $(m,1)$ from node K. Let

$$b = \sum_{k=1}^{K-1} (\frac{1}{\gamma_k} + \tau_{k,k+1}) = \sum_{k=1}^{K-1} \alpha_k \tag{50}$$

From (11), (50) and (49), it suffices to show that for $m = 1, 2, \ldots$,

$$A^{(K)}(m,1) - A^{(1)}(m,1) \leq (K-1) \max_{1 \leq n \leq m} \{\frac{1}{\lambda_n}\} + b \tag{51}$$

We will do so by induction on m.

1. For $m = 1$, since packet $(1,1)$ cannot be delayed by any preceding burst in the flow, it achieves the lower bound:

$$A^{(K)}(1,1) - A^{(1)}(1,1) = (K-1) \times \frac{1}{\lambda_1} + b \tag{52}$$

   Therefore (51) holds for $m = 1$.

2. Assume that (51) holds for $1 \leq m \leq m'$.

3. For $m = m' + 1$, we first define

$$p = \max\{x \mid \lambda_x = \min_{1 \leq h \leq m'} \{\lambda_h\}\} \tag{53}$$

   By the induction hypothesis, we have

$$\begin{aligned} A^{(K)}(m',1) - A^{(1)}(m',1) &\leq (K-1) \times \frac{1}{\lambda_p} + b && (54) \\ \{from\ lower\ bound\ of\ Theorem\ 1\} & \\ &\leq A^{(K)}(p,1) - A^{(1)}(p,1) && (55) \end{aligned}$$

   Therefore, by (47),

$$Gap^{(K)}_{p,m'} \leq Gap^{(1)}_{p,m'} \tag{56}$$

   For burst $m' + 1$, there are two possible cases:

   - $\lambda_{m'+1} \geq \lambda_p$ such that

$$\lambda_p = \min_{1 \leq h \leq m'+1} \{\lambda_h\} \tag{57}$$

     There are two possible subcases:

     (a) $Gap^{(K)}_{m',m'+1} \leq Gap^{(1)}_{m',m'+1}$

     In this subcase, from (56) and (46),

$$Gap^{(K)}_{p,m'+1} \leq Gap^{(1)}_{p,m'+1} \tag{58}$$

     From (47),

$$\begin{aligned} A^{(K)}(m'+1,1) - A^{(1)}(m'+1,1) &\leq A^{(K)}(p,1) - A^{(1)}(p,1) && (59) \\ \{by\ induction\ hypothesis\} & \\ &\leq (K-1) \times \frac{1}{\lambda_p} + b && (60) \end{aligned}$$

(b) $Gap^{(K)}_{m',m'+1} > Gap^{(1)}_{m',m'+1} \geq 0$

In this subcase, burst $m'+1$ is not delayed by burst m'; otherwise, the gap between them would become zero and stay at zero. As in the base case, it achieves the lower bound:

$$A^{(K)}(m'+1,1) - A^{(1)}(m'+1,1) = (K-1) \times \frac{1}{\lambda_{m'+1}} + b \quad (61)$$

$$\leq (K-1) \times \frac{1}{\lambda_p} + b \quad (62)$$

Therefore (51) holds for $m = m'+1$.

- $\lambda_{m'+1} < \lambda_p$ such that

$$\lambda_{m'+1} = \min_{1 \leq h \leq m'+1} \{\lambda_h\} \quad (63)$$

In this case, burst $m'+1$ is not delayed by any preceding burst, and thus achieves the lower bound:

$$A^{(K)}(m'+1,1) - A^{(1)}(m'+1,1) = (K-1) \times \frac{1}{\lambda_{m'+1}} + b \quad (64)$$

Therefore (51) holds for $m = m'+1$.

□