Burst Scheduling: Architecture and Algorithm for Switching Packet Video*

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

Abstract

We observed that variable bit rate (VBR) video, which is a sequence of encoded pictures, has very large rate fluctuations from picture to picture. In designing a new traffic model, we retain the basic notion of a flow but allow the flow rate to fluctuate. In particular, we introduce the concept of a burst which, in a video flow, is a sequence of packets that carry the bits of an encoded picture. We present the architecture of a class of packet switching networks, called Burst Scheduling networks, for carrying video, audio, and data traffic. The class is characterized by (i) use of virtual clock value as priority in scheduling, (ii) end-to-end delay and delay jitter guarantees provided to flows conforming to the new traffic model, and (iii) traffic flows (in particular, video flows) scheduled efficiently in bursts. Some experimental results are presented from a discrete-event simulation in which traces from several MPEG video sequences were used as video sources.

1 Introduction

Real-time video and audio require stringent network performance guarantees. Even though circuit switching can be used to provide the required performance guarantees, the current technology trend is towards the use of packet switching for all types of traffic. In ATM networks, for example, video, audio, and data are all to be carried in 53-byte cells.

In a packet switching network, each communication channel is statistically shared among many traffic flows. Typically, packets are queued and scheduled for transmission on a first-come-first-served (FCFS) basis. The service received by a particular flow is necessarily impacted by the behavior of other traffic flows that share the same queue. For this reason, it is very difficult for packet switching networks that employ FCFS scheduling to offer the following kinds of guarantees: (i) a flow gets a specified throughput rate, (ii) the end-to-end delays of packets in a flow are bounded, and (iii) the delay jitter over a set of packets is bounded.[1]

Such throughput, delay, and delay jitter guarantees, however, are precisely the ones needed to support many multimedia applications. To provide some or all of these guarantees in packet switching networks, a variety of rate-based service disciplines have been proposed [1, 2, 3, 4, 9, 11, 12]. For a network employing one of the rate-based disciplines to offer performance guarantees, network users are assumed to generate traffic according to a traffic specification. The traffic specifications assumed in [1, 3, 4, 12] are essentially the same and very simple: an average rate, and an interval over which the average rate is calculated. The traffic specification in [2, 9, 11] has one additional requirement, namely: a minimum interarrival time, which imposes an upper bound on the instantaneous flow rate.

This paper is concerned with the design of packet switching networks to support multimedia applications. In particular, we present the architecture of a class of packet switching networks, called Burst Scheduling networks, for carrying video, audio, and data traffic. An efficient scheduling algorithm is specified. The architecture and algorithm are motivated by two recent findings in our research project: (i) From studying the characteristics of MPEG video [5], we found that traffic specifications assumed by existing rate-based disciplines are inadequate for representing video; in particular, the packet generation rate of a VBR video source changes substantially and

*Research supported in part by National Science Foundation under grant no. NCR-9004464, and in part by NSA INFOSEC University Research Program. This paper, an abbreviated version of [7], will appear in *Proceedings INFOCOM '95*, April 1995.

[1] The delay jitter over a set of items is the maximum difference between the delays of any two items in the set.

frequently. (ii) We discovered and proved a delay guarantee for the Virtual Clock service discipline [10].

The balance of this report is organized as follows. In Section 2, we present the delay guarantee of a Virtual Clock server and its properties. In particular, the concept of a conditional delay guarantee is introduced. In Section 3, we discuss why a new traffic model is needed for packet video. We then introduce the concept of bursts and specify the traffic model. In Section 4, we show how to convert the delay guarantee to a delay bound for flows that conform to the traffic model. In Section 5, we discuss the need to restructure and retime bursts as they travel through a network. The network architecture and scheduling algorithm are presented in Section 6. In Section 7, we present end-to-end delay and delay jitter bounds. In Section 8, we present some experimental results from a discrete-event simulation.

The network architecture and scheduling algorithm in this paper have been designed with the following objectives: (i) End-to-end delay and delay jitter guarantees provided by a network to a traffic flow are independent of the behavior of other traffic flows sharing the network. (ii) The scheduling algorithm is fast enough for high-speed packet switches. Substantial improvements in algorithm efficiency are obtained by exploiting the traffic model and delay guarantee. In particular, a switch stores only one virtual clock value per flow.[2]

Due to length limitation, many important network design issues are beyond the scope of this paper, such as: demand assignment, connection admission control, overbooking and statistical guarantees, modeling and scheduling of audio and delay-sensitive data, and fault tolerance. They are being investigated and our findings will be presented in a forthcoming technical report.

2 Delay Guarantee

Consider a service facility with several sources of traffic. Each source generates a sequence of packets, called a *flow*. Prior to generating packets, the source of flow f negotiates with the facility a reserved flow rate $r(f)$ bits/second. Generally, different sources negotiate for different rates depending upon their needs and how much they are willing to pay. For all flows, the length of a packet varies from a minimum of l_{min} to a maximum of l_{max} bits. For an arbitrary packet p, we use $l(p)$ to denote its length in bits, $A(p)$ its arrival time to the facility ($A(p) \geq 0$), and $L(p)$ its departure time from the facility upon service completion. The concept of virtual clock is introduced next.

Let $priority(f)$ denote the virtual clock of flow f. It can be implemented as a variable, which is zero initially and updated as follows [12] whenever a flow f packet, say p, arrives to the facility.

$$priority(f) := \max\{priority(f), A(p)\} + \frac{l(p)}{r(f)} \quad (1)$$

The new value of $priority(f)$ above is assigned to packet p as its virtual clock value, denoted by $P(p)$. Thus, the virtual clock of flow f is a variable that holds the virtual clock value of the *most recent arrival* of f. Note that the virtual clock values of flow f are determined by the sequence of packet arrival times of f, and are independent of the service discipline.[3]

Now consider a server and a set F of flows. The Virtual Clock (VC) service discipline specifies that whenever the server is ready to serve a new packet, the packet in queue with the smallest virtual clock value is selected for service. Furthermore, the service discipline is work-conserving and nonpreemptive. We will refer to such a server as a VC server. In what follows, we use the term *system* to refer to both queue and server.

Definition 1. A flow f is active at a VC server at time t if and only if the following condition holds: The system is not empty, and the value of $priority(f)$ at time t satisfies

$$t < priority(f) \quad (2)$$

Simply stated, *a flow is active as long as its virtual clock is running faster than real time* and the system is nonempty. Whenever the system is empty, all flows are inactive by definition.

Definition 2. Let C denote the capacity, in bits/second, of a VC server. The server's capacity is exceeded at time t if the following condition holds:

$$\sum_{f \in a(t)} r(f) > C \quad (3)$$

where $a(t)$ is the subset of flows that are active at time t.

[2] It was observed [11] that priority-based service disciplines may be infeasible for high-speed implementation, because a sorted priority list requires $O(\log N)$ insertion operations, where N is the number of packets queued for a channel. In the Burst Scheduling algorithm, the parameter N is the number of active flows, *not* packets queued.

[3] which is unspecified at this point.

Theorem 1. If the capacity of a VC server has not been exceeded for a nonzero duration since the start of a busy period, then the following holds for every packet p that has been served during the busy period:

$$L(p) \leq P(p) + \frac{l_{max}}{C} \tag{4}$$

A proof of Theorem 1 can be found in [10]. Fixed assignment or demand assignment may be used to ensure that a VC server's capacity is not exceeded. Note that the active flow definition can be exploited to increase the number of flows that statistically share a VC server. Definition 1 has been designed such that a flow's active periods are as short as possible, yet sufficient for the delay guarantee in Theorem 1.

2.1 Properties of delay guarantee

The delay guarantee in Theorem 1 is not a delay bound in the usual sense. Specifically, the deadline provided to a packet is measured from its virtual clock value, rather than its actual arrival time. If a packet arrives early, its deadline is bounded from its expected arrival time, based upon the reserved rate of its flow; see (1) and (4). Thus early arrivals may encounter large delays. On the other hand, packets that arrive late do not get better service. In short, the delay guarantee is a service specification designed to encourage sources to generate on-time arrivals.

We believe that delay guarantees based upon virtual clock values are very appropriate for packets carrying real-time video and audio. In fact, in designing a packet switch, on-time packet arrivals are preferable to early packet arrivals. If a flow's packets can arrive very early, the flow is effectively more bursty, and more buffer space is needed for the flow. Rewarding arrivals that are too early with prompt service is counterproductive.

The delay guarantee is useful because it is *conditional.* As such, it does not require the service facility (actually the designer and implementor of the facility) to be concerned with the behavior of sources, of which the facility has no control [6]. There is no requirement that sources be flow controlled or well-behaved. Even though the reserved rate of flow f is $r(f)$, its source can misbehave, i.e., its traffic generation rate can be arbitrary. However, we do assume that each flow is allocated its own buffers, so that if a source generates traffic at a rate higher than $r(f)$, it will fill up its own buffers but not those of other flows.

The delay guarantee in (4) has a desirable firewall property, namely: The delay guarantee to a flow is independent of the behavior of other flows that share the same service facility. The property is obvious from examining (4) and (1), and recognizing that the virtual clock values of a flow are determined solely by the flow's own arrivals. Thus if a source generates traffic faster than its reserved flow rate, its own packets will encounter large delays, but its behavior will not affect delay guarantees provided to other traffic flows.

2.2 The role of source control

The delay guarantee to a flow becomes an unconditional delay bound if the source of the flow is known to be well behaved, voluntarily or by source control. Consider a flow f with reserved rate $r(f)$. Let i denote the ith packet in flow f. The goal of source control is to upper bound $P(i) - A(i)$, that is, the extent to which the virtual clock of flow f is allowed to run ahead of real time. A simple example of source control is to ensure that the interarrival time between two consecutive packets in the flow is lower bounded, e.g., $A(i+1) - A(i)$ is greater than or equal to $l(i)/r(f)$ for all i. In this case, we can show by induction on i that $P(i) = A(i) + (l(i)/r(f))$. Therefore, a VC server provides the following *delay bound* to every packet p in controlled flows.

$$L(p) \leq A(p) + \frac{l(p)}{r(f)} + \frac{l_{max}}{C} \tag{5}$$

2.3 Network with fixed packet size

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.

For networks with a fixed packet size, we use the following notation:

$\lambda(f)$	rate of flow f in packets/second
γ	server rate in packets/second

The virtual clock update in (1) becomes

$$priority(f) := \max\{priority(f), A(p)\} + \frac{1}{\lambda(f)} \tag{6}$$

The delay guarantee in (4) becomes

$$L(p) \leq P(p) + \frac{1}{\gamma} \tag{7}$$

3 New Traffic Model

Full-motion video is a set of pictures displayed sequentially. In uncompressed form, each picture is a two dimensional array of pixels, each of which is represented by three values (24 bits) specifying both luminance and color information. From such uncompressed video data, a video encoder produces a coded bit stream representing a sequence of encoded pictures (as well as some control information for the decoder). There are three types of encoded pictures in MPEG video: I (intracoded), P (predicted), and B (bidirectionally predicted). An I picture is one that is encoded, and decoded, without using information from another picture. In general, an I picture is much larger than a P picture (in number of bits), which is much larger than a B picture. Typically, the size of an I picture is larger than the size of a B picture by an order of magnitude.

Consider an MPEG encoder as a traffic source that produces encoded pictures at a rate of 30 pictures per second. The size of each picture is known as soon as it is encoded. It ranges from a low of about 10,000 bits for a B picture to a high of about 300,000 bits for an I picture (for some MPEG sequences encoded at a spatial resolution of 640×480 pixels [5]). For delivery by a packet switching network, each picture is segmented to be the payload of a large number of packets (e.g., ATM cells). Thus, the rate of such a source changes every 1/30 second, varying from a low of 0.3 Mbps to a high of 9 Mbps.

For traffic flows generated by multimedia applications, we generalize the flow model to a *sequence of bursts*. The concept of bursts is needed for two purposes: (i) for specifying delay jitter guarantees,[4] and (ii) for partitioning a flow into intervals that have substantially different flow rates.

For a video flow, a burst is a sequence of packets that carry the bits of an encoded picture. With live video capture, note that the average rate of the entire video flow is unknown at the start of the video sequence. However, as soon as the video source has encoded a picture, the size of the burst (number of packets) is known and its rate, to be called the picture's *packet rate*, can be determined. Specifically, if lossless smoothing is used, the rate is available from the smoothing algorithm [5]. If not, the rate can be computed from the picture size, the picture rate (e.g., 30 pictures per second), and the packet payload size. Information on the rate and size of each burst can be exploited in scheduling packets.

We use n_i to denote the number of packets in burst i. The jth packet in burst i is denoted by (i,j). The arrival time, virtual clock value, and departure time of packet (i,j) at a service facility are denoted by $A(i,j)$, $P(i,j)$, and $L(i,j)$, respectively.

[4] With the concept of bursts, two types of delay jitters can be defined: delay jitter over packets in a burst, and delay jitter over bursts in a flow.

Definition 3. Flow Specification:

- A flow is a sequence of bursts, each of which is a sequence of packets. The first and last packets in each burst are marked. The first packet of burst i carries information on its rate, λ_i , in packets/second, and its size n_i .[5]
- Packets in burst i satisfy a *jitter* timing constraint, namely: for $j = 1, 2, \ldots, n_i$,
$$0 \leq A(i,j) - A(i,1) \leq \frac{j-1}{\lambda_i} \tag{8}$$
- Bursts in the flow satisfy a *separation* timing constraint, namely: for $i \geq 1$,
$$A(i+1,1) - A(i,1) \geq \frac{n_i}{\lambda_i} \tag{9}$$

Timing constraint (8) specifies a delay jitter bound. Burst Scheduling networks are designed to preserve this jitter bound. Timing constraint (9) specifies a minimum time separation between two consecutive bursts in a flow. This is a form of source control which is used to convert conditional delay guarantees into delay bounds (see Lemma 2 below).

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

4 VC Server for Guaranteed Flows

Consider flows which conform to the Flow Specification arriving at a VC server. Each flow is a sequence of bursts that do not overlap in time. Since the first packet of a burst carries information on the rate of the burst, the burst rate can be used for virtual clock update; specifically, the flow rate $\lambda(f)$ in (6) is updated each time a new burst arrives. We 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 jth packet in burst i, $1 < j \leq n_i$, is
$$P(i,j) = P(i,1) + \frac{j-1}{\lambda_i} \tag{10}$$

[5] This part of the specification is implementation-dependent. Some information, such as n_i, may not be required.

Lemma 2. If the capacity of a VC server has not been exceeded for a nonzero duration since the start of a busy period, then the following bound holds for the jth packet of burst i served during the busy period, for $i \geq 1$, $j = 1, 2, \ldots, n_i$,

$$L(i,j) \leq A(i,1) + \frac{j}{\lambda_i} + \frac{1}{\gamma} \tag{11}$$

Proofs of the two lemmas are presented in [7]. Note that if we consider each burst as an integral unit, Lemma 2 provides a burst delay bound which depends on the length of burst i.

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 until some time after the last packet of the burst arrives. Also if a flow does not satisfy the separation timing constraint, the flow may have two or more bursts being active at the same time with the following consequence: In determining whether or not the capacity of a server is exceeded, the rate of the flow is the aggregate rate of simultaneously active bursts in the flow. The separation timing constraint is needed to ensure that each flow has at most one active burst at any time.

5 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 packet switches, where each outgoing channel is a VC server. At the network entrance, a *source regulator* ensures that the flow's packets satisfy the 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 packet switch along the path,[6] 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 purposes of checking satisfaction of timing constraints (8) and (9), and applying Lemma 2.

[6] The jitter timing constraint is assumed to compute virtual clock values efficiently. The separation timing constraint is assumed to determine the aggregate rate of active flows.

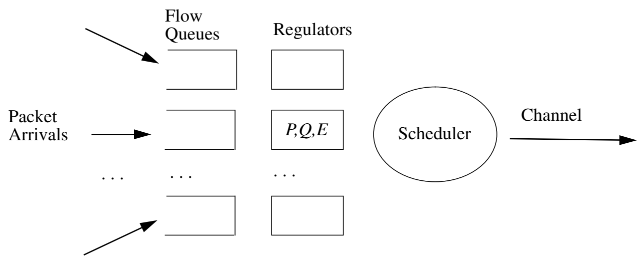

Figure 1: Architecture of a channel.

Delaying packets to satisfy the jitter and separation timing constraints is called burst *restructuring* and *retiming*, respectively [7]. In the algorithm to be presented in Section 6, both the restructuring and retiming of a burst are achieved by delaying just the first packet of the burst, and are not performed until the packet gets to the head of its flow queue in the next switch along the path.

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. It is noteworthy 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 7 is the same in the absence of burst restructuring and retiming. (The end-to-end delay lower bound and the burst delay jitter bound would be different.)

6 Architecture and Algorithm

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 flow queues are maintained for packets belonging to different flows; each flow queue is allocated its own buffers. All of the flow queues, except one, are for guaranteed flows. There is a regulator for each flow queue. There is a scheduler for each channel. Additionally, there is a regulator at the network entry point of every guaranteed flow (not shown in Figure 1).

6.1 Source regulator

A source regulator is used to ensure that the packets of a flow enter a network in accordance with the Flow

Specification. Specifically, the first packet of burst i has two fields,[7]

λ_i	rate of burst i (in packets/second)
u_i	time ahead (in seconds); initialized to zero at source

The source regulator performs the following tasks:

- Store values in the λ_i and u_i fields of the first packet of burst i (using value of λ_i supplied by the encoder or lossless smoothing algorithm, and zero for u_i). For each burst, mark the first packet as *first* and last packet as *last*.
- Ensure that the jitter and separation timing constraints are satisfied.

6.2 Flow regulator

Consider the bursts in a guaranteed flow. From timing constraint (9), burst i+1 follows burst i, for all i. From timing constraint (8), packet j in a burst, for $j > 1$, 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 are preserved by flow queues. (This is easy to do. It is sufficient to implement each flow queue as a FIFO queue.) As a result, the restructuring and retiming of packets in a burst are accomplished for all j if they are performed for the first packet in the burst [7].

When a new flow arrives at a switch (presumably after an end-to-end session has been established), a flow queue is created for the flow, as well as a flow regulator for the queue. There are three variables associated with each flow queue: P, Q, and E, defined below.[8] There are four variables associated with burst i: a_i and i, defined below, and λ_i and u_i, defined as in Section 6.1. The flow regulator can read and write all of these variables.

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

[7] The burst size n_i is not used in the algorithm to be presented.

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

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 packet switch. The *update* procedure is specified as follows:

procedure $update(P, E)$
    /* execute once per burst */
{ $Q := \max\{a_i + u_i, P\}$;
    /* compute time when burst i is eligible */
  $delay(Q - now())$;
  $P := Q + 1/\lambda_i$;
  $E :=$ **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 i packet to flow queue

  if (packet is *first* in burst i) **then**
  { record arrival time in a_i and
      values of u_i and λ_i;
    if (flow queue was empty before arrival)
      then $update(P, E)$; }

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

  if (departed packet is *not last* in burst i) **then**
    $P := P + 1/\lambda_i$;
  else
  { $E :=$ **false**;
    $i := i + 1$;
    if (flow queue is not empty) **then**
      $update(P, E)$; }

Note that the procedure $update(P, E)$ is executed only for the first packet of each burst. Specifically, u_i contains information on how much the first packet should be delayed to achieve burst restructuring [7]. When the procedure is called, P contains the earliest time when the burst retiming constraint is satisfied. Thus both restructuring and retiming of burst i 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 i 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 (i,1) at a VC server, as used in Lemma 2.

6.3 Flow queue for other traffic

Burst Scheduling networks are designed to carry video, audio, and data traffic. Some packet flows do not require delay and delay jitter bounds. In this paper, we refer to them as *other* traffic.[9] For clarity of exposition, and without any loss of generality, we assume that one flow queue is used for all other traffic. Modifying the following specification to the case of multiple flow queues for other traffic is straightforward.

Let α denote a fraction of the channel capacity that has been reserved for guaranteed flows. Thus $(1-\alpha)$ of the channel capacity is reserved for other traffic, and is allocated using virtual clock values. Whenever there is nothing to transmit from the guaranteed flow queues, the entire channel capacity is available to other traffic.

The flow regulator for other traffic is specified differently from the flow regulator in Section 6.2. The E flag of the flow queue is true if and only if the queue is nonempty; it is omitted from the regulator specification below. The variable Q is not needed. The variable P is initially zero. The flow regulator for other traffic is specified by the following actions:

- Enabling condition: packet arrival to flow queue

if (flow queue was empty before arrival) **then**
  $P := \max\{P, now()\} + 1/((1-\alpha)\gamma)$;

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

if (flow queue is not empty after departure)
  then $P := P + 1/((1-\alpha)\gamma)$;
      /* from applying (7) */

Note that if the flow regulator for other traffic can detect the boolean condition,

G_empty = (for all guaranteed flow queue ::
      queue is empty or its E flag is false)

it may reset the virtual clock of its flow queue to $now()$ whenever G_empty is true. Such aggressive behavior would not affect the delay and delay jitter bounds provided to guaranteed flows, given that the reserved rate $\alpha\gamma$ is not exceeded by the aggregate rate of guaranteed flows at every VC server in the network.

[9] In the literature, they are called available bit rate (ABR) traffic.

6.4 Channel scheduler

The channel scheduler can read variables P and E of every flow queue.[10] We use S to denote the set of nonempty flow queues where E is true, i.e., the set of flow queues with eligible packets waiting. The channel scheduler is specified by the following action:

- Enabling condition: end of a packet transmission or wakeup

if (S is not empty) **then**
{ select from S the flow queue with smallest P;
  remove packet at head of selected flow queue;
  if (packet is *first* in burst i) **then**
    write value of ($P - now()$)
    into u_i field of packet;
  transmit packet; }

In the above specification, 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 (7) to be nonnegative.

6.5 Algorithm efficiency

Among service disciplines based upon the use of priority [1, 2, 8, 9], we believe that the Virtual Clock service discipline will have the most efficient implementation. By using the virtual clock value of a packet directly as its priority, it is easy to see that Virtual Clock is much more efficient than PGPS [1, 8]. Compared to the EDD disciplines [2, 9], Virtual Clock requires no schedulability test.

Furthermore, the Burst Scheduling algorithm has been designed to exploit the jitter timing constraint and the delay guarantee in (7) to compute virtual clock values efficiently.

For guaranteed flows, a flow regulator executes the procedure *update*(P, E) only once per burst, specifically, for the first packet in each burst. For any other packet in a burst, the flow regulator simply increments P by the value of $1/\lambda_i$, instead of executing the algorithm in (6). This is made possible by Lemma 1, which follows from the jitter timing constraint.

In computing virtual clock values for other traffic, only the first packet of each busy period of the flow queue requires the algorithm in (6). For any other

[10] 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*.

packet in a busy period, the flow regulator simply increments P by the value of $1/((1-\alpha)\gamma)$. This is a consequence of the delay guarantee in (7) 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 stamp packets with their arrival times, or store the arrival times of queued packets, as suggested in the original proposal [12]. Furthermore, at any time, a switch stores only one virtual clock value per flow queue (rather than one per packet).

7 Delay and Delay Jitter Bounds

Consider a guaranteed flow traversing a path of $K+2$ nodes in a Burst Scheduling network, where node 0 denotes the source, node $K+1$ the destination, and nodes $1, 2, \ldots, K$ packet switches. In the following analysis, it is assumed that channels in the path deliver the flow's packets reliably and in order. Also, for every channel in the path, the channel capacity reserved for guaranteed flows is not exceeded by the aggregate rate of active guaranteed flows. Furthermore, processing times of the router, regulator, and scheduler functions in packet 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 work on a packet that is waiting. We can think of two exceptions: (1) when a packet arrives to a channel where all flow 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 $1/\gamma_s$ slightly).

Consider an arbitrary packet (i, j) of the flow. Let $D(i, j)$ denote its end-to-end delay, which is measured from the time the packet leaves node 0 to the time it arrives at node $K+1$. Define

- τ_s propagation time of channel from node s to $s+1$, in seconds, $s = 0, 1, \ldots, K$
- γ_s capacity of channel from node s to $s+1$, in packets/second, $s = 1, 2, \ldots, K$

Since the flow satisfies the jitter timing constraint when its packets become eligible at a node (except possibly at node $K+1$, the destination), the delay jitter over packets within the same burst, say burst i, is bounded by the duration of the burst, namely: n_i/λ_i.

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

$$\begin{aligned} D(i,1) &\geq \frac{K-1}{\lambda_i} + \sum_{s=1}^{K} \frac{1}{\gamma_s} + \sum_{s=0}^{K} \tau_s \\ D(i,1) &\leq \frac{1}{\lambda_i} + (K-1) \max_{1 \leq h \leq i} \{\frac{1}{\lambda_h}\} + \sum_{s=1}^{K} \frac{1}{\gamma_s} + \sum_{s=0}^{K} \tau_s \end{aligned} \tag{12}$$

A proof of Theorem 2 is given in [7]. The delay of an arbitrary packet (i, j) is bounded by

$$D(i,j) \leq D(i,1) + \frac{j}{\lambda_i} \tag{13}$$

7.1 Assumptions, guarantees, and fault-tolerance

The delay and delay jitter bounds provided to a guaranteed flow are independent of the behavior of other traffic flows in the network. This firewall property is inherited from virtual clock delay guarantees,[11] under the assumption that each flow queue is allocated its own buffers. Furthermore, it is assumed that all network components are reliable.

In what follows, we continue to assume that packet switches are reliable. However, it is possible (albeit infrequently) that channels lose packets and source regulators malfunction (specifically, failing to enforce the jitter and separation timing constraints). Burst Scheduling networks should be designed to be tolerant of such faults. We provide here a brief discussion on how to achieve fault tolerance.

A service provider is generally designed to provide service guarantees that are *conditional*.[12] Note that the network layer offers service guarantees to a higher layer (such as video flows) by making use of service guarantees offered by a lower layer (in this case, reliable packet delivery by channels). In this case, the network layer is obligated to provide its service guarantees to a flow only if

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

[11] A virtual clock delay guarantee, in general, is a deadline measured from the virtual clock value of an arrival.

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

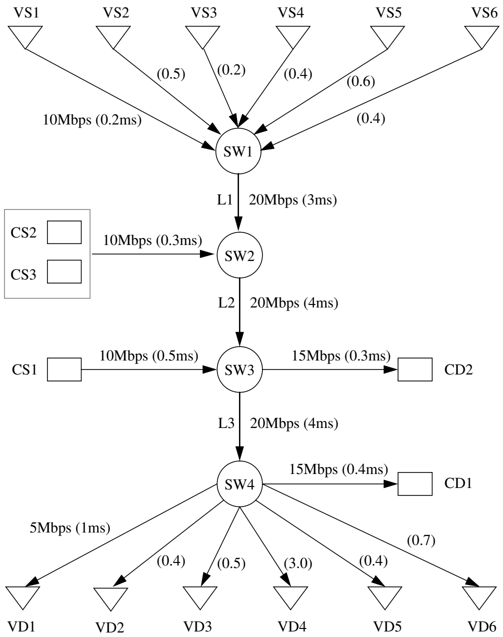

Figure 2: Simulated network.

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. To meet this requirement, it is sufficient that flow regulators compute correct virtual clock values. Recall from Section 2.1 that virtual clock delay guarantees to other flows are independent of the behavior of flow v; it does not matter whether the actual rate of flow v is higher than its reserved rate, whether some packets in flow v have been lost or reordered, or how its packets are partitioned into a sequence of bursts.

8 Experimental Results

Using discrete-event simulation, we studied the performance of a Burst Scheduling network. We are interested in comparing experimental end-to-end delays and delay jitters with the theoretical bounds in Section 7. Additionally, we would like to observe the buffer requirements of video flows in a Burst Scheduling switch.

8.1 Network configuration

The simulated network is illustrated in Figure 2. There are four switches, labeled by SW. (For the same reasons discussed in Section 7, we have omitted any processing delay in a switch for routing, regulating, and scheduling.) Each arrow in Figure 2 represents a channel, labeled by its capacity in megabits per second (Mbps) and its propagation delay in milliseconds (ms); the propagation delay is the number shown in parentheses. Channels are assumed to be lossless. Channels are shared by video flows and other traffic, with 20% of each channel reserved for other traffic.

Six video sessions were active throughout the experiment. The source of each video flow is labeled by VS, the destination by VD. Every video flow is routed through switches SW1, SW2, SW3, and SW4 in that order. Each video source generates bursts of 53-byte packets with the size and packet rate of each burst obtained from a trace file. The trace files are from MPEG sequences that have been smoothed by the lossless algorithm in [5]. When packets were generated within a burst, the interpacket gap was fixed and was computed from the size and rate of each burst in the trace file. We used a fixed interpacket gap to make the simulation run faster. Note that a fixed gap is not required by Burst Scheduling. The gap can be variable as long as bursts satisfy the jitter timing constraint.

There are also three active sessions of other types of traffic in the network, routed as follows:

cross traffic session 1: CS1 $\rightarrow$ SW3 $\rightarrow$ SW4 $\rightarrow$ CD1
cross traffic session 2: CS2 $\rightarrow$ SW2 $\rightarrow$ SW3 $\rightarrow$ CD2
cross traffic session 3: CS3 $\rightarrow$ SW2 $\rightarrow$ SW3 $\rightarrow$ CD2

CS1 and CS2 are Poisson sources which generate 53-byte packets at rates of 15 packets/ms and 6 packets/ms respectively. CS3 is a packet train source [12] with an average train gap of 0.8 ms and end-of-train probability of 0.3.

8.2 Network performance

We ran an experiment for 10 seconds of simulated time. About 300 pictures were transmitted for each video session. (Some of the MPEG sequences were not long enough, and the traces were wrapped around.)

The simulated end-to-end delay of the first packet in each burst is plotted together with the analytical upper and lower bounds in Figure 3 for sessions 1-3. Note that the upper and lower bounds in (12) are functions of the packet rate λ_i of burst i which varies from picture to picture. A uniform upper bound is the upper bound obtained from the smallest λ_i in the video sequence, and a uniform lower bound is the lower bound obtained from the largest λ_i. (These are shown as straight dotted lines in Figure 3.)

Since the channel capacity allocated to video flows was not exceeded for every channel, all of the bounds held for video flows during the experiment as predicted

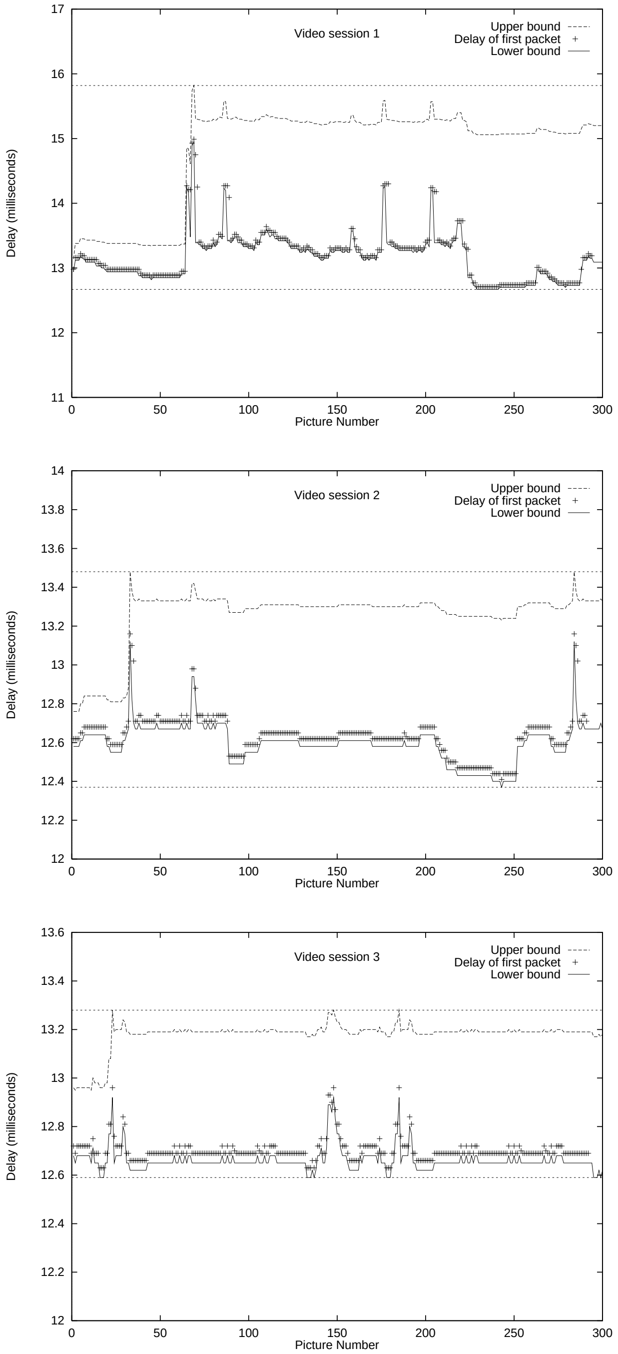

Figure 3: End-to-end delay of first packet in a picture, simulated value and bounds.

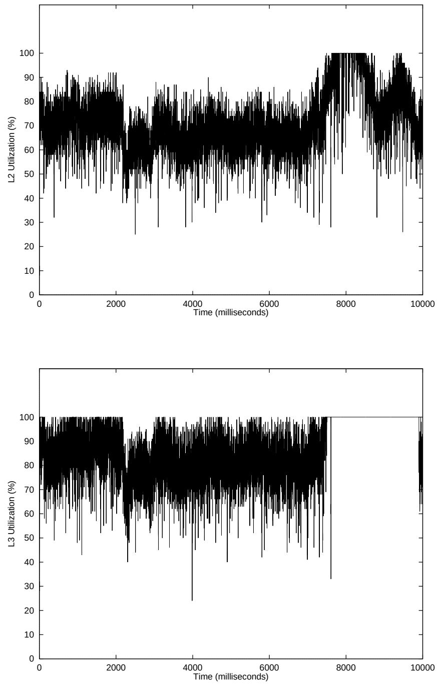

Figure 4: Utilizations of channels L2 and L3.

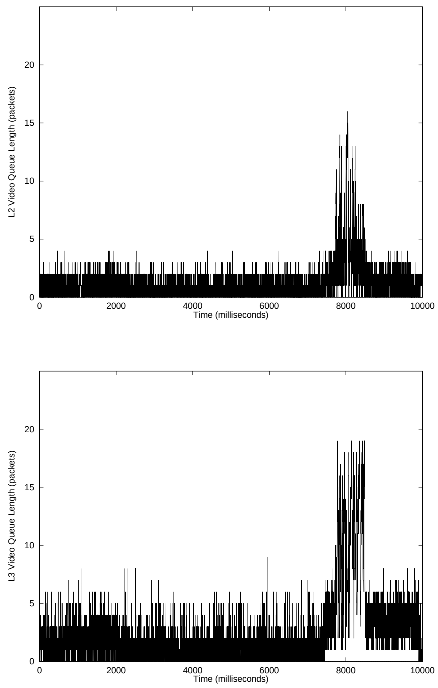

Figure 5: Video queue lengths for channels L2 and L3.

by theory. In fact, the simulated delay is generally much closer to the lower bound than to the upper bound. This is due to the small number of switches. Over a short path, burst k with a large rate λ_k does not get delayed very much by burst i ($i < k$) with a small rate λ_i.

The burst delay jitter bounds were also compared with burst delay jitters obtained from the simulation, and found to hold.

To illustrate the buffer requirements of Burst Scheduling, the utilizations of channels L2 and L3 and their video queue lengths[13] are plotted in Figures 4 and 5, respectively. Note that there was a period of time (from 7500 ms to 8500 ms) when the channel utilizations were 100% and the aggregate video flow rate was very close to the 80% capacity allocated to video traffic. During this period the video queue lengths in both switches increase, then stabilize, and subsequently decrease quickly as the aggregate flow rate comes down. Such good performance is a consequence of the delay guarantee of Virtual Clock, which ensures that on-time packet arrivals to a switch stay in the switch for a short, bounded duration of time.

9 Conclusions

Motivated by VBR video which has very large picture-to-picture rate fluctuations, we propose a new traffic model, called Flow Specification. In designing the model, we retain the basic notion of a flow. Each flow, however, is a sequence of bursts (each of which represents an encoded picture in the case of packet video). A burst is a sequence of packets with the first packet carrying information on the rate and size of the burst (supplied by the video encoder or a lossless smoothing algorithm). Though motivated by packet video, the traffic model is also appropriate for specifying audio and data traffic that require delay jitter bounds.

We have presented an algorithm for scheduling the transmission of both guaranteed flows and available-bit-rate traffic in a packet switch. The algorithm has been designed to compute virtual clock values very efficiently by making use of the new traffic model and the delay guarantee proved in [10]. Such efficiency is needed to enable use of small packets, such as ATM cells, to carry video and audio traffic.

A Burst Scheduling network provides end-to-end delay and delay jitter bounds to flows that satisfy the Flow Specification when entering the network. Bounds provided to a particular flow are independent of the behavior of other traffic flows that share the network, and are thus unaffected by the presence of aggressive or misbehaving traffic sources.

Acknowledgements: We thank the anonymous referees of *INFOCOM '95* for their constructive comments.

[13] The video queue length is sum of all video flow queue sizes for the channel.

References

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

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

[3] S. Jamaloddin Golestani. Congestion-free communication in high-speed packet networks. *IEEE Trans. on Communications*, pages 1802–1812, December 1991.

[4] Charles R. Kalmanek, Hemant Kanakia, and Srinivasan Keshav. Rate controlled servers for very high-speed networks. In *IEEE Global Telecommunications Conference*, San Diego, California, December 1990.

[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*, 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: Architecture and algorithm for switching packet video. Technical Report TR-94-20, Department of Computer Sciences, UT-Austin, July 1994. Revised, January 6, 1995. Available from http://www.cs.utexas.edu/~lam/NRL.

[8] 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.

[9] Dinesh Verma, Hui Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. In *Proceedings of Tricomm '91*, Chapel Hill, North Carolina, April 1991.

[10] Geoffrey G. Xie and Simon S. Lam. Delay guarantee of Virtual Clock server. Technical Report TR-94-24, Department of Computer Sciences, UT-Austin, October 1994. Available from http://www.cs.utexas.edu/~lam/NRL. Presented at 9th IEEE Workshop on Computer Communications, Marathon, Florida, October 1994.

[11] Hui Zhang and Domenico Ferrari. Rate-controlled static-priority queueing. In *Proceedings of INFOCOM '93*, pages 227–236, March 1993.

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