

# PWP: An Architecture for Guaranteed Network Services

Mark R. Lindsey

May 7, 2002

## Abstract

Some distributed applications function well only when guarantees are made on the maximum latency which the applications' packets will experience. Common packet-switched networks, such as the current public Internet, do not provide straightforward mechanisms for deriving such guarantees. Two approaches to the problem of bounding packet-switched network latency are presented: PWP-Sporadic, and PWP-Rate, each of which corresponds to a well-known model of real-time tasks. Feasibility relations are presented, and the relative benefits of each are discussed. Finally, critical issues in the deployment of PWP-Rate are explored.

## 1 Introduction

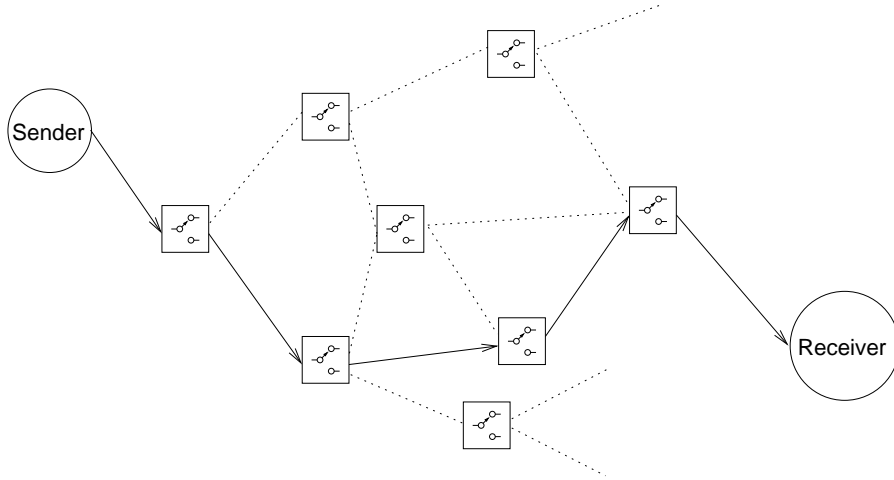
Networks are provided to support distributed applications, such as email, file delivery, and teleconferencing. Some distributed applications, such as teleconferencing, function only when the communication latency between members of the distributed system is less than some constant time. In teleconferencing, each participating party has a real-time process which samples data from cameras and microphones and transmits this data into a network. When the data is received at another site, a corresponding real-time process decodes and displays the sampled data. If this entire, end-to-end process is consistently performed with a sufficiently-small delay between sampling and display, then the conference is useful to its participants.

When direct physical connections are provided between members of the distributed system, then the task of determining communication latencies is a straightforward analysis of propagation and coding delays along each of the links involved.

Packet-switched networks present greater difficulty for purposes of assessing end-to-end latency fundamentally because the path that a packet takes from the sender to the receiver is determined by independent routing decisions at each switch along the path.

With each routing decision, a link is selected along with its service queue, hence end-to-end delays are determined by

Figure 1: A packet-switched network, showing a path from the sender to the receiver in the black line, and other links in dotted lines. Note that each switch along the path may have inuts from many other sources, which may include real-time (guaranteed-service) and non-real-time flows, both of which may be scheduled on the outbound link server.

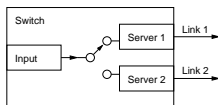


- intrinsic delays in the routing process,
- queueing delays of each link server along path, and
- the propagation delays of each link server along the path

Basic packet-switched networks, such as the Internet, typically provide only ‘best-effort’ service: i.e., packets are queued for service by a link if queueing capacity is available. More sophisticated Active Queue Management (AQM) schemes, such as Random Early Detection [6], may drop packets even when there is queueing capacity available, but do so in order to signal the senders of impending congestion. Several other schemes have been proposed and standardized for providing guaranteed latency services across a packet-switched network, including the Berkeley Tenet project [1] and Delay-EDD [5], but neither of these exploit recent developments in scheduling analysis in order to determine feasibility.

The fundamental goal of this research area is to make a guarantee about the performance that a single sender will experience when transmitting to a single receiver. When a maximum end-to-end latency can be guaranteed, then a minimum amount of throughput is thus assured. For example, if the network promises that it can deliver a 1,000 bytes every 20ms, then the network has explicitly made a latency guarantee and has implicitly made a throughput guarantee of  $\frac{1,000}{20}$  bits-per-millisecond.

Figure 2: A minimal switch having one input and two outbound links. Queuing occurs only at outbound links. As a flow’s path will use at most one outbound link on any switch, we can disregard other links that are not used by the flow.



The term *Precisely-Warmed Potato* refers to the general approach suggested here: each switch along the path chooses a local deadline which is precisely right to ensure that the packet is delivered just in time.<sup>1</sup>

PWP forwarding has two fundamental goals:

1. Consistency: Provide latency guarantees on network services
2. Efficiency: Make use of all available network resources

Two techniques for providing guaranteed services are described here: PWP-Sporadic which models the input data flow as a Sporadic task [12], and PWP-Rate which models the input data flow as task in the Rate-Based Execution model [8].

This paper is organized as follows; section 2 formulates the problem and details related work; section 3 details the PWP approach to guaranteed network services; sections 4 and 4 detail the PWP-Sporadic and PWP-Rate approaches, respectively, and section 6 discusses some practical issues surrounding deployment of this system.

## 2 Background

### 2.1 Problem Formulation

A *packet* is an ordered set of  $\ell$  bits which together form some useful, decodable message. A *store-and-forward packet-switched network* performs delivery of packets using a network of switches connected by.

Conceptually, a *link* is a unidirectional conduit for bits connecting a bit sender to a bit receiver. Each link has a capacity,  $C$ , which is the number of bits which may enter and exit the link during a unit of time. Also, each link has a propagation delay,  $\pi$ , which is the time between a bit’s injection into the link by the sender until its receipt.

A *switch* receives packets on one or more links, and transmits each packet to another link; i.e., it is the receiver for some links, and the sender for some links.

<sup>1</sup>By contrast, ‘Hot-Potato’ or ‘Deflection’ Routing delivers the packet immediately, possibly deflecting it further away from the destination. PWP does not address routing issues, and assumes that nodes in the network do have buffering.

Figure 3: A path from a sender to a receiver, showing the parameters at each node. For example, at node 3, the maximum queueing time is 2 units time; the outgoing link rate is 1000 bits per units time, and the propagation delay along that link to node 4 is 24 units of time. The sender has infinite capacity and no latency to the first queueing node.

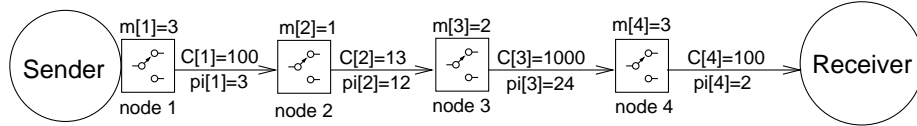


Figure 2 provides a brief visualization for this model. We assume that queueing does not occur on the link's inputs (links for which it is the receiver), but may occur on its outputs (links for which it is the sender). This corresponds to the realistic observation of actual equipment, for which sufficient capacity is often provided on the network interface which receives data from a link. However, unless each output link has a capacity at least as large as the sum of the capacity of all of the input links (i.e.,  $C_{out} \geq \sum_{j=1}^n C_{in,j}$ ,  $n$  is the number of input links), then queueing may occur on the output links. This is because all of the input links may, simultaneously, receive a packet which must be transmitted on a given outbound link.

Thus, a queue must be provided for each outbound link, the server for which removes a bit from the queue and transmits it on the outbound link. The queue for a link having capacity  $C$  has a service rate of  $C$ . We assume that the server can dequeue a bit and transmit it on the link simultaneously.

**Lemma 1 (Packet Transmission Time)** *A queue server for a link having capacity  $C$  requires  $\frac{\ell}{C}$  units of time to transmit a packet of  $\ell$  bits from the head of the queue.*

**Proof** By definition, a server for a link having capacity  $C$  services the queue at a rate of  $C$  bits per unit of time. That is equivalent to saying that each bit requires  $\frac{1}{C}$  units of time to dequeue it and to transmit it on the link. Thus,  $\ell$  bits can be dequeued and transmitted in  $\frac{\ell}{C}$  units of time.  $\square$

We assume further that each switch has a maximum enqueueing time of  $m$ ; formally,  $m$  is the maximum of amount time which will elapse between the time of receipt of the last bit of a packet on an input link and the time at which the packet is in the appropriate output queue. This delay encapsulates the delay intrinsic to the routing decision and the queue management time.

A *path* from  $\alpha$  to  $\beta$  is the set of links and switches that a packet from the sender  $\alpha$  will be sent along to reach the receiver  $\beta$ . It is assumed that the selection of an output link for a packet in a given switch is dependent only on the destination of the packet; formally, that  $\forall$  switch  $j$ ,  $\forall$  destination  $\beta$ ,  $\exists$  unique link  $l$  of which  $j$  is the sender and which  $j$  would select to transmit a packet

destined for  $\beta$ . This restriction is reasonable, and reflects a typical property in deployed packet switched networks. Figure 3 shows a path from a sender to a receiver along a four-node path.

We assume also that the sender transmits to the first queueing node with infinite capacity and no propagation delay. This describes the function of actual systems, in which the producer of packets is a part of the same machine as the first queueing node connected to the network.

**Lemma 2 (Node Transit Time)** *The arrival time  $a_j^k$  of a packet  $p_f^k$  at node  $j$  is related to the arrival time  $a_{j+1}^k$  of the packet at the next node along the path  $j + 1$  such that*

$$a_{f,j+1}^k - a_{f,j}^k \leq m_j + \frac{\ell_f^k}{C_j} + \pi_j$$

where  $m_j$  is the maximum queue eligibility delay in node  $j$ ,  $C_j$  is the capacity of the link from node  $j$  to node  $j + 1$  in bits per unit of time,  $\pi_j$  is the propagation delay along the link from node  $j$  to node  $j + 1$ ,  $\ell_f^k$  is the number of bits in the packet  $p_f^k$ , and where no queueing delay occurs in node  $j$ .

**Proof** By definition, the packet  $p_f^k$  is received at node  $j$  at the time  $a_{f,j}^k$ . No more than  $m_j$  time units later, the packet is known to be enqueued, by the definition of  $m_j$ . Because no queueing delay occurs, transmission begins immediately, and is completed  $\frac{\ell_f^k}{C_j}$  time units later, by lemma 1.  $\pi_j$  time units after transmission completes, the last bit reaches node  $j + 1$ . Thus we have that the packet is received at node  $j + 1$  no later than  $a_{f,j}^k + m_j + \frac{\ell_f^k}{C_j} + \pi_j$  which equivalent to saying

$$a_{f,j+1}^k \leq a_{f,j}^k + m_j + \frac{\ell_f^k}{C_j} + \pi_j \quad (1)$$

$$a_{f,j+1}^k - a_{f,j}^k \leq m_j + \frac{\ell_f^k}{C_j} + \pi_j \quad (2)$$

as was to be shown.  $\square$

**Lemma 3 (Minimum No-Queueing Path Transit Time)** *The time for a packet of length  $\ell$  to transit a packet-switched network having  $n$  nodes when the packet experiences no queueing delay is upper-bounded by*

$$\sum_{j=1}^n (m_j + \frac{\ell}{C_j} + \pi_j)$$

**Proof** Each node  $j$  requires no more than  $m_j + \frac{\ell_f^k}{C_j} + \pi_j$  units of time after it has received a packet to transmit the packet to node  $j + 1$ , assuming that it experiences no queueing delay, by lemma 2. Because there is no latency between

the time that a packet is produced until it is available for queueing in the first node. By definition of our model, there are no other delays present in the path; thus the path transit time in the absence of queueing is simply the sum of the node transit times.  $\square$

A *flow* is a sequence of packets  $(p_f^k, p_f^{k+1}, \dots)$  sent from a single sender  $\alpha$  to a single receiver  $\beta$  [3]. By our assumptions above, it is assumed that every packet in this flow transits the same path. In the absence of queueing on outbound links, the minimum no-queueing transit time is straightforward to compute. However, in general, each packet will experience nonnegative queueing delay at each node.

To guarantee a maximum end-to-end latency for the flow in the presence of queueing, mechanisms must be provided to ensure that the amount of end-to-end delay that a flow  $f$  experiences is bounded to be less than the maximum tolerable end-to-end latency of the flow; this value is called  $L_f$ . Each flow is also specified with a maximum packet size,  $M_f$ .

The *slack* in the path of a flow, denoted  $S_f$ , is a flow's tolerance for latency beyond the minimum no-queueing path transit time of largest packet of the flow; formally,

$$S_f = L_f - \sum_{j=1}^n (m_j + \frac{M_f}{C_j} + \pi_j)$$

$S_f$  is an upper bound on the amount of delay that each packet of flow  $f$  can experience in queues along its end-to-end path.

**Observation 1** *For a flow  $f$ , if  $S_f < 0$ , then the flow cannot be delivered within tolerable delay.*

Intuitively, this is because a negative slack means that the minimum end-to-end transit time is greater than the maximum tolerable latency. <sup>2</sup>

The flows having nonnegative slack may be *feasible*. All mechanisms within this field of research fundamentally operate to manage the use of slack as each packet traverses the network.

## 2.2 Previous Work

Weighted Fair Queueing (WFQ) [4] can successfully partition the use of a link to real-time channels. However, it imposes a high overhead on the system as non-real-time tasks (flows) come and go, and its scheduling does not map easily onto the packet link scheduling of interest here.

The Delay-EDD mechanism [5] applies Liu and Layland's original EDF scheduling mechanism [11] to a priority queue of packets on each link server. Delay-EDD requires that each switch in the network store information about

---

<sup>2</sup>Such cases do occur; e.g., the University of North Carolina *Distributed NanoManipulator* system is constrained by the latency incurred by propagation delays, such that, because of the bound on the speed of light, the two endpoints of the system must be relatively close to one another [7].

Figure 4: The notation used in this paper, much of which was borrowed from Sahni [10].

$p_f^k$	the $k^{th}$ packet of flow $f$
$\ell_f^k$	length of packet $p_f^k$
$m_j$	maximum enqueueing delay in node $j$
$a_{f,j}^k$	arrival time of the last bit of $p_f^k$ at node $j$ on its path
$\pi_j$	maximum possible propagation delay of the link from node $j$ to node $j + 1$
$C_j$	outgoing link capacity and queue service rate at node $j$
$D_f^k$	absolute deadline of $p_f^k$
$t_{f,j}^k$	time at which $p_f^k$ has been received at first queueing node
$L_f$	Maximum end-to-end latency tolerable by flow $f$
$M_f$	Maximum packet size that can be sent by flow $f$
$S_f$	Slack in the path of flow $f$
$s_{f,j}$	Slack assigned to node $j$ for flow $f$

the flows passing through it. However, an error in analysis predicts that Delay-EDD will function for some cases where it will not, because it ignores potential blocking due to the nonpreemptive nature of job queueing.

Further, there are some cases where a guaranteed minimum latency can be provided, even though Delay-EDD would refuse to provide it. This effort provides a mechanism of *assignable slack* which can be used to maximize use of the network.

Later systems, such as the Berkeley Tenet system [1] and its intellectual descendent in the Internet-standardized Integrated Services framework [13] do not require that clocks be synchronized, But do require that state about each flow be stored in each switch. The most recent approaches [14, 10] avoid the use of synchronized clocks and the installation of state in the interior of the network.

### 3 General Problem Approach

The approach of this research is to apply processor-scheduling techniques to the problem of meeting end-to-end latency constraints. To do so, the queue server in each node  $j$  along a path is modeled as a processor operating at rate  $C_j$ .

#### 3.1 Link Scheduling

Non-preemptive scheduling algorithms are necessary for link scheduling; i.e., once the first bit of a packet has been transmitted on a link, the entire remainder of the packet must be transmitted. Otherwise, the receiving node will not be

able to recover any of the packet, because a packet is the smallest useable transmission unit in packet-switched networks.

PWP schedules access to the link using an Earliest-Deadline First (EDF) discipline, discussed by Liu and Layland [11]. EDF has been shown to be an optimal nonpreemptive scheduling algorithm for Sporadic and for Rate-Based task models by Jeffay et al. [9, 8]. Thus it provides a suitable foundation for PWP link scheduling.

**Observation 2** *For a flow  $f$ , if  $S_f = 0$ , and  $f$  is feasible, then  $f$  is the only traffic which can be carried by any link along the path of  $f$ .*

A complete lack of end-to-end slack means that the every packet in the flow must be delivered immediately. In the absence of queueing delays, this is possible. But if other traffic (i.e., packets from other flows) is carried by any link used by  $f$ , then the possibility exists that a packet  $p_f^k$  from flow  $f$  would arrive in a queue the instant after a packet from another flow has commenced transmission along the link. Because the link must be scheduled non-preemptively,  $p_f^k$  will experience some amount of queueing delay. Because  $S_f = 0$ , there is no tolerance for queueing delays, so the packet  $p_f^k$  may miss its deadline.

This effort considers only *work-conserving* scheduling disciplines, as the presence of idle time would violate a secondary goal to make efficient use of our network resources.

### 3.2 Selecting Local Deadlines

To guarantee end-to-end delivery within a bounded latency, each node on the path perform scheduling. The use of the deadline-driven scheduling algorithm EDF implies that a local deadline be chosen at each node.

Each packet  $p_f^k$  is ‘released’ at time  $t_f^k$ , and must be delivered to the destination no later than the time  $t_f^k + L_f$ . To ensure this, both PWP techniques partition the available slack  $S_f$  among the nodes along the path; the slack allocated to each node is denoted as  $s_{f,j}$ .

### 3.3 Admission Control

Each flow which needs to be added to the network is subject to an *admission control* test to determine whether it can be carried. Conceptually, the admission control test requires that some specification of the flow’s requirements be sent to every node along the path. Feasibility decisions are made at each node, and if local guarantees can be made which together do not violate the end-to-end requirements, then the flow is admitted to the network.

These specifications include the maximum end-to-end latency  $L_f$  the maximum packet size  $M_f$ , and an indication of the frequency of the packets. More detail about the flow specifications is given in sections 4 and 5 which detail the PWP-Sporadic and PWP-Rate mechanisms, respectively.

### 3.4 Producer Task Models

To make guarantees to the packets of a flow, the packets must be injected into the network according to some fixed discipline. This is because each packet has a deadline which relates to the time at which it is injected into the network; if the producer is allowed to inject an arbitrarily-large number of packets in an arbitrarily-small amount of time, then an arbitrarily-large amount of capacity is required along the path to ensure that every packet is delivered within the latency tolerance of the flow,  $L_f$ . Because we assume that link capacities are finite, we must thus constrain the behavior of the producer.

In this effort, we consider two models of the tasks: the Sporadic task model [12], and the Rate-Based Execution model [8]. In this context, the producer task model describes the releases of packets into the first queuing node of the network.

## 4 Flow Production as a Sporadic Task

The sporadic task model described by Mok [12] describes a real-time task  $T$  by the tuple  $(c, p)$  where  $c$  is the maximum execution cost of each instance of the task, and  $p$  is the minimum time which elapses between the release of each task instance. Each release of a task instance of  $T$  must be completed no later than  $p$  time units after it is release; i.e., if  $t_k$  is the time of the release of the  $k^{th}$  job of task  $T$ , then the execution of this job must complete no later than  $t_k + p$ .

Sporadic producers are those for which the minimum packet-generation time is known, so that a minimum spacing can be ensured between the release of each packet into the network. A *sporadic flow* is the flow of packets generated with a producer which is producing according to a sporadic task specification.

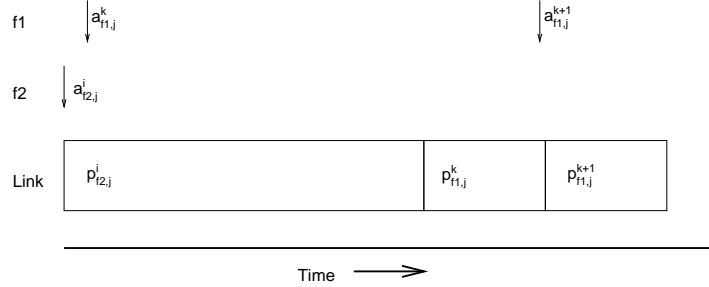
A generalization of the Sporadic model allows deadlines to be unrelated to periods; i.e.,  $T = (c, p, D)$ , where each instance of  $T$  must be completed before at-most  $D$  time units after the instance's release. Jeffay et al. gave feasibility conditions for a set of sporadic tasks scheduled non-preemptively with EDF on a single processor in [9].

Current networking applications usually adhere to such a model implicitly by employing *window-based* rate control. In such a scheme, many of an applications packets are 'in-flight' through a network concurrently, so that the inter-release time  $p$  of the application's packets is less than the relative deadline  $D$  of each packet. Naturally, we would like to provide service guarantees to such applications.

**Observation 3 (Loss of Interarrival Spacing)** *Due to the nonpreemptive nature of link scheduling, we cannot guarantee minimum inter-arrival spacing of packets into every node along a flow's path, without inserting idle time.*

To see this, consider the example depicted in figure 5. The flows of two sporadic tasks,  $f_1$  and  $f_2$ , are shown with respect to node  $j$ . Packet  $p_{f_2,j}^i$  arrives when the link is idle, and transmission commences immediately. Shortly

Figure 5: Packets may arrive according to a guaranteed minimum spacing, but do not exit with any guaranteed spacing.



thereafter, packet  $p_{f1,j}^k$  arrives and is queued behind  $p_{f2,j}^i$ . A full period of the task  $f1$  elapses before the transmission of  $p_{f1,j}^k$  elapses, and  $p_{f1,j}^{k+1}$  arrives. In the absence of competing traffic with other deadlines, and without inserting idle time,  $p_{f1,j}^{k+1}$  is transmitted immediately after the transmission of  $p_{f1,j}^k$  completes. Thus the next switch along the path will receive  $p_{f1,j}^k$  and  $p_{f1,j}^{k+1}$  with no spacing between them.<sup>3</sup>

Note, however, that this ‘compression’ occurs when two packets of a sporadic flow are present in the path concurrently. We can eliminate this effect by ensuring that each packet of a flow exits the network before the next packet of that flow enters the network; this is exactly the case we have where relative deadlines of the packets at least its period, which is certainly the case in the basic Sporadic model shown above.

#### 4.1 Sporadic Flow Admission Control

We now show the conditions under which a Sporadic Flow can be guaranteed service through a path. A flow is here called *feasible* if the network can guarantee that each of the flow’s packets will be delivered within the flow’s maximum tolerable latency  $L_f$  time units after the packet was released into the first queueing node along the path.

The total end-to-end slack available to a flow,  $S_f$ , is the total amount of time which all nodes may allow the flow to reside in queues. Each node  $j$  is given a part of the slack for a flow  $f$ ; this local slack is denoted  $s_{f,j}$ , such that

<sup>3</sup>Many actual link media have an inter-packet delay necessary for synchronization of the sender with the receiver, so that the packets would be spaced by that delay. Other media, specifically synchronous systems, do not have an interpacket delay. Also left unconsidered in this work is the effect of link encapsulation on inter-packet delay. However, in all of these cases, the amount of delay inserted is intentionally minimized.

for  $n$  queueing nodes,

$$\sum_{i=1}^n s_{f,i} \leq S_f$$

The allocation of the slack is discussed further in section 4.2.

Having selected local slack values for all nodes along the path, we can determine whether the flow is feasible by performing a set of individual admission-control checks for each of the queueing nodes along a flow's path. Rather than craft a set of feasibility conditions specific to link scheduling, we can re-use processor scheduling feasibility checks. Clearly, a flow places a demand on each node through which it passes, and we can model that demand as a conventional processor demand; thus each node is modeled as a processor. When the 'processor' of each node  $j$  along path can feasibly schedule a task  $T_{f,j}$  corresponding to the flow  $f$ , then we can make corresponding guarantees to the flow.

**Theorem 1** *A flow  $f = (\ell_f, p_f, L_f)$ ,  $p_f \leq L_f$  is feasible through a path of  $n$  nodes where  $\ell_f$  is the maximum number of bits in a packet of  $f$ ,  $p_f$  is the minimum spacing between releases packets of  $f$  into the network, and  $L_f$  is the maximum latency after a release of a packet before it must be received, if  $\forall$  node  $j$  the sporadic task  $(c_{f,j}, p_{f,j}, d_{f,j})$  is feasible on a single processor which operates at a rate of  $C_j$  bits per unit of time, using nonpreemptive scheduling, together with other flows which  $j$  has already admitted where*

$$c_{f,j} = \frac{\ell_f}{C_j} \quad (3)$$

$$p_{f,j} = p_f - \sum_{i=1}^{j-1} s_i \quad (4)$$

$$d_{f,j} = m_j + \frac{\ell_f}{C_j} + s_j \quad (5)$$

and  $\forall 1 \leq j \leq n, s_j \geq 0$  and  $\sum_{i=1}^n s_i \leq S_f$  where  $S_f$  is defined as above.

Sufficient relations for determining feasibility of sporadic tasks with deadlines unrelated to periods are given by Zheng and Shin in [15].

**Proof** Under this model, the queue server in node  $j$  is modeled as a processor. The 'execution cost' of each packet  $p_f^k$  on this processor is exactly  $\frac{\ell_f}{C_j}$ , by lemma 1, because this is precisely the amount of time which the queue server requires to inject the packet into the link.

Each packet  $p_f^k$  has an absolute deadline  $a_{f,1}^k + L_f$ . We see that, with the deadlines specified, each packet arrives at or before its deadline by the following:  $d_{f,j}$  specifies the local deadline; i.e., if the flow is determined to be feasible in a node, then each of its packets  $p_f^k$  will exit the node no later than  $a_{f,j}^k + d_{f,j}$ .  $\pi_j$  after exiting node  $j$ ,  $p_f^k$  arrives at node  $j + 1$ , by the definition of propagation

delay. Thus the packet reaches its destination no later than

$$a_{f,1}^k + \sum_{j=1}^n d_{f,j} + \pi_j \tag{6}$$

$$= a_{f,1}^k + \sum_{j=1}^n m_j + \frac{\ell_f}{C_j} + s_j + \pi_j \tag{7}$$

$$\leq a_{f,1}^k + S_f + \sum_{j=1}^n m_j + \frac{\ell_f}{C_j} + \pi_j \tag{8}$$

End-to-end slack  $S_f$  is defined as  $L_f - \sum_{j=1}^n m_j + \frac{\ell_f}{C_j} + \pi_j$  thus we have that each packet arrives no later than  $a_{f,1}^k + L_f$ .

The inter-arrival spacing at node  $j$  is lower-bounded by  $p_f - \sum_{i=1}^{j-1} s_i$ , as can be seen thus: consider a packet  $p_f^k$  which experiences the maximum-tolerable queueing delay as it transits nodes 1 to  $j - 1$ ; i.e., it arrives *as late as possible* to node  $j$ ; then the very next packet,  $p_f^{k+1}$ , is released into node 1 at precisely  $p_f$  time units after the release of  $p_f^k$ , and it transits nodes 1 to  $j - 1$  as quickly as possible, experiencing no queueing delay. The extreme variance in queueing delays will cause the earlier packet  $p_f^k$  to be ‘compressed’ toward the later packet  $p_f^{k+1}$  by  $\sum_{i=1}^{j-1} s_i$  units of time. And because the packets were released precisely  $p_f$  time units apart, their separation at node  $j$  will be  $p_f - \sum_{i=1}^{j-1} s_i$ , as was to be shown.  $\square$

## 4.2 Selection of Local Slack

Each node along the path is allocated a portion of a flow’s slack. The simplest way to allocate the slack among the nodes is to divide it evenly, such that  $\forall 1 \leq j \leq n, s_{f,j} = \frac{S_f}{n}$ . However, because of the compression effect shown in theorem 1, the flow’s virtual task  $T_{f,j}$  near the end of the path must defend against an artificially-low period. So we would expect that giving more slack to the nodes near the end of the path may increase the number of flows which those nodes can carry.

The computation of  $n$  local slacks is an expensive procedure. However, because of the deadline constraints which are conferred by an allocation of slack, an appropriate slack allocation can allow a flow to be feasible in a case when another slack allocation may not allow the flow to be feasible. The cost of computing the appropriate slack can be delegated to either the sender or the receiver, which can collect the appropriate parameters from each node along the path, perform the slack allocations, and return the structure to the nodes along the path. The nodes can then quickly verify that the slack allocations are appropriate, and admit the new flow.

## 4.3 Weakness of Sporadic Model for Network Flows

As mentioned above, we can only support a sporadic flow which keeps no more than one packet in the network. The actual use of guaranteed-latency applications is often for interactive applications having a maximum tolerable delay of 100ms or less; for current networks which carry 1500-byte packets, this means

that such a sporadic flow could carry 120,000 bits per second. Further, under such a scenario each 1500-byte packet must satisfy the display requirements of the receiver for a full 100ms.

These are significant constraints on the usefulness of such a system. However, for applications which do produce data according to a sporadic discipline and require a corresponding delivery latency, then this is all we have. Such applications may include industrial control and measurement applications, where the intrinsic propagation delays are not very large, but we must defend against occasional bulk transfers of data which can cause queueing through the network.

The fundamental problem which causes constraining difficulties for sporadic flows is that the local deadlines in each node is coupled to the arrival process in each node, rather than its end-to-end deadline. A burst of packets in the middle of the network therefore requires a burst of deliveries in the middle of the network, even when the producer process did not originate the burst.

Ultimately, though, we would like to have multiple packets in flight; i.e., have an inter-origination period which is significantly less than the relative deadline. But if we do that, then we cannot ensure that the input to the second node on the path will follow the same sporadic minimum-interarrival guarantees. Jeffay et al. observed this, also [8].

## 5 Flow Production as an RBE Task

The Rate-Based Execution (RBE) model described by Jeffay et al. [8] provides a more suitable framework for describing such network flows, because it decouples arrival from deadlines. Thus packets may arrive at a node in a burst without violating the scheduling conditions.

An RBE task  $T = (x, y, d, c)$  where  $y$  is an interval of time,  $x$  is the maximum number of task releases which *are expected* to occur in any window of  $y$  time units,  $d$  is the desired relative deadline of each task, and  $c$  is the maximum processor demand which will be incurred by each release of a job of  $T$ .

A key relevant difference between an RBE task and a Sporadic task is that the absolute deadline of each job release is chosen to ‘smooth out’ bursts which occur. Specifically, the  $j^{th}$  job of the  $i^{th}$  task  $T = (x_i, y_i, d_i, c_i)$  has the absolute deadline

$$D_i(j) = \begin{cases} t_{i,j} + d_i & \text{if } 1 \leq j \leq x_i \\ \max(t_{i,j} + d_i, D_i(j - x_i) + y_i) & \text{if } j > x_i \end{cases}$$

where  $t_{i,j}$  is the time of release of the  $j^{th}$  job of the  $i^{th}$  task.

As described in [8], this deadline allocation functions to bound the demand over an interval. Thus, while a burst of the sort shown in figure 5 are still likely to occur, the burst does not cause a corresponding increase in the demand in the next switch along the path, because demand is partially decoupled from the arrival process by the deadline-assignment function  $D_i(j)$ .

## 5.1 Flow Admission Control

As in the case of PWP-Sporadic, an admission control test is necessary to determine whether a path is capable of making guarantees to flow having an RBE producer task.

**Theorem 2** *A flow  $f = (x_f, y_f, L_f, \ell_f)$  is feasible through a path of  $n$  nodes where  $\ell_f$  is the maximum number of bits in a packet of  $f$ ,  $x_f$  is the number of packet releases expected each  $y_f$  time units, and  $L_f$  is the maximum latency after a release of a packet before it should be received, if  $\forall$  node  $j$  the sporadic task  $(x_{f,j}, y_{f,j}, d_{f,j}, c_{f,j})$  is feasible on a single processor which operates at a rate of  $C_j$  bits per unit of time, using nonpreemptive scheduling, together with other flows which  $j$  has already admitted where*

$$c_{f,j} = \frac{\ell_f}{C_j} \tag{9}$$

$$x_{f,j} = x_f \tag{10}$$

$$y_{f,j} = y_f \tag{11}$$

$$d_{f,j} = m_j + \frac{\ell_f}{C_j} + s_j \tag{12}$$

and  $\forall 1 \leq j \leq n, s_j \geq 0$  and  $\sum_{i=1}^n s_j \leq S_f$  where  $S_f$  is defined as above.

Necessary and sufficient relations for determining feasibility are given in [8].

**Proof** The reasons for this are similar to those for theorem 1. Specifically, that feasibility on a single processor is suitable for establishing feasibility of traffic through a node.

The deadlines guarantee appropriate delivery by the same argument given for theorem 1.

Also as in the sporadic case, compression can occur by extreme differences in the queuing delay of consecutive packets. However, because the underlying local deadline-assignment scheme ensures that bursts of packets do not cause corresponding bursts in demand, *we need not defend against it.*

We can see that  $x_{f,j} = x_f$  and  $y_{f,j} = y_f$  because no packets are inserted or removed in the path after the sender, and therefore over long periods of time, the expected number of packets  $x_f$  per averaging interval  $y_f$  is the same at all points through the network.  $\square$

## 6 Deployment of a PWP Scheme

The previous sections of this paper developed a theoretical model of packet-switched networks, and described a mapping from real-time network flows to processor scheduling for purposes of determining feasibility. The remainder of this paper deals with the issues of deploying the PWP-Rate scheme on a contemporary, operating internet.

## 6.1 Choosing a PWP Scheme

Two schemes for modelling real-time traffic flows have been presented in this paper: PWP-Sporadic (§4), and PWP-Rate (§5). PWP-Sporadic is imminently unsuitable for deployment for a number of related reasons, as discussed above: first and fundamentally, the sporadic model is unsuitable for link-scheduling applications because it cannot be guaranteed minimum inter-arrival delays when multiple packets from a single flow are in the network simultaneously; this limits the overall throughput rate of a PWP-Sporadic flow.

Second, the minimum inter-arrival times against which PWP-Sporadic must defend are extremely pessimistic. This is revealed in theorem the specification of the period parameter in theorem 1 – conceptually, because two consecutive packets *can* arrive very quickly behind each other due to compression, PWP-Sporadic must pessimistically schedule as if *every* packet arrives very quickly. That is, the arrival rate of packets from a given flow  $f$  at a node  $j$  is considered to be significantly greater than the arrival rate at the first node in the path. Clearly, such a case cannot persist over an extended period of time, and therefore PWP-Sporadic is far too pessimistic.

Finally, to generate the sporadic task for a given node  $j$  corresponding to a PWP-Sporadic flow, we must have information about the inter-node latency of all nodes  $k < j$ , i.e., switches closer in the path to the sender.

The RBE model provides a suitable framework for modelling real-time network flows, however. It deals with the packet compression issue by assigning local deadlines which correspond to the expected arrival rate; PWP-Rate inherits this benefit directly.

Further, conventional flow specifications can be easily transformed to PWP-Rate flow specifications. For example, an application which intends to transfer 100 kilobits per second using 10,000-bit packets and which has a maximum latency tolerance of 40 milliseconds could represent its PWP-Rate flow specification as  $(\frac{100000}{10000} = 10\text{packets}, 1\text{second}, 40\text{ms}, 10000\text{bits})$  – i.e., ten packets every second, each of which may be as large as 1,250 octets, and which should experience a latency of no more than 40 milliseconds after entering the network.

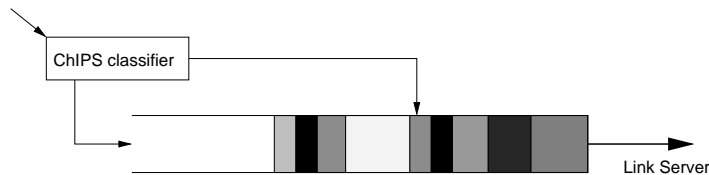
For these reasons, only deployment of the PWP-Rate scheme is discussed, and will be abbreviated as simply PWP in this section.

## 6.2 Deployment Platform

It is only useful to discuss deployment in the context of an operating environment. This effort assumes that we are deploying PWP-Rate on a contemporary IPv4 network which carries only best-effort traffic. Further, we would be interested in the usefulness of PWP when it is deployed only along part of the path, such that hard guarantees cannot be made, but overall improvements for soft-real-time traffic may be realized.

The degree of administrative access which is available will directly affect the degree to which PWP can make hard guarantees.

Figure 6: ChIPS allows some packets to ‘cut ahead’ in the queue in order to regulate jitter.



### 6.3 Queueing

All analysis for PWP assumes that access to the link can be scheduled using a nonpreemptive Earliest-Deadline-First (EDF) service discipline. However, most existing router queueing techniques use first-in-first-out (FIFO) queueing, or possibly a collection of queues with differing relative priorities. Theoretically, it would be possible to replace the FIFO queues with a general priority queue; however, the complexity cost would be significant.

Instead of general priority queueing, we can extend an idea – Cut-In Packet Scheduling (ChIPS) – given by Chung and Claypool in [2]. ChIPS is a multimedia-biased adaptation of FIFO scheduling. Under certain conditions, ChIPS will insert UDP packets into a point in the queue corresponding to the average queue length. ChIPS has an attractive  $O(1)$  complexity, making it a candidate for usage on the data path through routers.

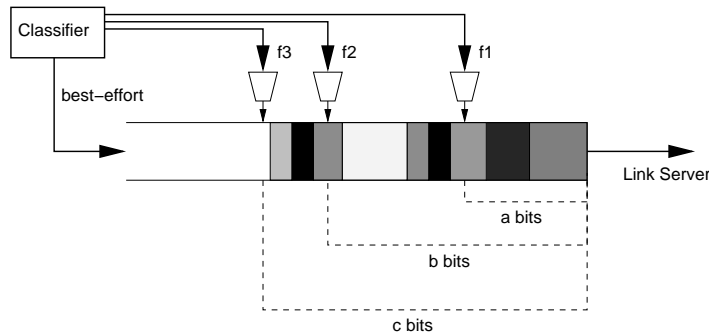
A key insight in the design of ChIPS is the relationship between queue position and a packet’s waiting time: by inserting each of a class of packets at the position the packet would have in an average queue, the packets are held in the router just as if the queue actually did have that average length. In PWP, the acceptable waiting time for a flow’s packet is the local slack for that flow.

By using the same technique in a PWP router, we can control the maximum amount of time that the router will hold a given packet. Specifically, when a packet is inserted at a position  $b$  bits from the service-end of the queue, we know that the packet will begin transmission at  $\frac{b}{C}$  units of time later, where  $C$  is the service capacity of the link.

However, we do not yet have a mechanism for assigning the local deadlines which are specified by the RBE deadline function,  $D_i(j)$  (§5). We can use the *token bucket* traffic-shaping abstraction to enforce the RBE parameters. Specifically, an RBE flow which specifies  $x$  packets every  $y$  units of time can be regulated by a token bucket which releases  $x$  tokens every  $y$  units of time.

Figure 7 depicts an example of this queue-management scheme for three flows,  $f_1$ ,  $f_2$ , and  $f_3$ . The token bucket for each is positioned such that the packet released from the bucket will experience a fixed queueing delay; for example, each packet from flow  $f_1$  will experience no more than  $\frac{a}{C}$  queueing delay

Figure 7: A cut-ahead packet queuing used to guarantee service to three flows,  $f1$ ,  $f2$ , and  $f3$ . Flow  $f1$  has the least local slack.



after it is released from the bucket.<sup>4</sup>

To provide hard guarantees, the buckets must be placed such that the maximum rate of releases ‘down stream’ in the queue will not cause a packet to miss a local deadline. However, the admission control test (specified in theorem 2) already ensures that such an organization will be possible, because the RBE feasibility relations in [8] defend against that worst-case condition.

An alteration should be made to the token bucket which could improve performance: normally, a token bucket would allow traffic only as tokens are available. However, the token bucket for this application specifies when the token bucket *must* allow a packet – i.e., when a packet must be inserted into the queue, so that its deadline is not missed. If the queue below position of a flow’s bucket is empty, then packets from that flow may be allowed to ‘drain’ out, filling the queue to the level of the token bucket.<sup>5</sup> Otherwise, the link may become idle even while data is available in the queues served by the token buckets; this violates the work-conserving nature of our scheduling, and thus our fundamental goal to make efficient use of the network.

Many router manufacturers currently provide token-bucket implementations for use in traffic engineering, so it is to our advantage that re-use the existing, deployed technique. ChIPS is not widely used, however, and would have to be added to routers.

## 6.4 Determining End-to-End Slack

PWP requires that local slack allocations be made to each router along the path from the sender to the receiver, so that each can assign local deadlines which

<sup>4</sup>In this example,  $f1$  produces the hottest potatoes.

<sup>5</sup>We might call it a *Pascal’s Token Bucket*, because it drains to equalize the ‘water level’ of the link-service queue and the individual flow’s queue.

ensure that the end-to-end deadlines are met. The end-to-end slack for a flow  $S_f$  is, conceptually, the latency budget for each packet; time from this budget must be allocated to each of the routers along the path.

PWP-Sporadic required the knowledge of inter-node latencies, but PWP-Rate needs only aggregate information about the end-to-end minimum latency; i.e., the minimal one-way trip time (OTT).

The Minimal OTT can be estimated by tools such as `pathchar` from NLANR, but a fundamental sampling problem is present when doing so – i.e., how long do you have to measure before you have a reasonable minimum? Due to the long-range-dependent nature of traffic bursts on the Internet, it would be difficult to make such measurements over any reasonably-short time scale in order to ascertain an estimate of the minimum OTT. Further, true precision requires that the two endpoints of the path have synchronized clocks, so that OTT can be directly measured by the receiving endpoint.

To get an accurate upper-bound minimum OTT, we must first determine upper-bounds on the minimum queueing delays of each router along the path, and second measure propagation delays along each hop in the path. This first step could be performed by network equipment manufacturers, who would perform timing analysis – for each software release, for each of their equipment models – then encode it as constants with the software. Thus the values could be provided for the admission-control procedure.

Facilities for measuring propagation delay would have to be added to the link-layer protocols, then measurements performed each time a link on a router becomes active (e.g., when the power is applied to the router, or service is restored to the link). Some link-layer protocols, such as PPP, already have a *ping* operation, and for the common case where the link from node  $n$  to node  $n + 1$  has a corresponding, physically-similar link for transmission from node  $n + 1$  to node  $n$ , the round-trip-time measurements could be halved to estimate OTT on the link.

If we improperly estimate the minimal OTT to be larger than it truly is, then we will not be able to use all available slack in the network; i.e., the path could actually provide lower latency than we believe that it can provide. In the worst case, this may cause a refusal to make guarantees to a flow for which sufficient capacity is available in the network. Therefore, greater accuracy in minimal OTT measurement will provide for greater utilization of the network for guaranteed flows.

## 6.5 Per-Node Local Slack Allocation

Once the end-to-end slack has been determined, it must be partitioned among the switches. Conceptually, a switch will be able to schedule more flows with long deadlines than it can schedule flows having short deadlines, because the scheduler has a relatively large amount of time to transmit each packet in a flow with a long deadline. Thus, at first blush, we wish to maximize the local deadlines at each switch; this could be accomplished easily by dividing the available slack among all of the switches.

However, this might prevent PWP from guaranteeing service to some flows which it could have handled. As mentioned above in section 4.2, there may be cases in which a switch  $j$  cannot make a guarantee when its deadlines are determined by this even division of slack  $\frac{S_f}{n}$ , but it would be able to make a guarantee if its local slack was set to  $s_{f,j} = \frac{S_f}{n} + \epsilon$  where  $\epsilon$  is a small number. Further, suppose that in such a case, another switch  $i$  along the path would be capable of making the guarantee even if its local slack were reduced to  $s_{f,i} = \frac{S_f}{n} - \epsilon$ . Such a case is clearly possible with the schemes proposed, because the local slack implies local deadlines, and the feasibility test for nonpreemptive scheduling of RBE tasks considers these local deadlines in its decision.

This observation suggests that the slack-allocation problem for a specific flow may be solved using linear programming, and is, in general, computationally expensive. Traditional admission-control techniques require that the devices which perform the switching also perform the admission-control test, but this pattern may not be necessary. An alternate protocol for admission control is suggested in the next section.

## 6.6 PWP Flow Setup Protocol

This section outlines a protocol for management of a PWP flow. It includes an unconventional admission-control system intended to ameliorate performance of the flow setup, due to the computational complexity of the feasibility decision. Further, this design is shown to have useful application-design properties.

The conceptual structure is as follows: the sender, because it has the flow specification locally, collects the parameters of the path to the receiver. It then computes slack allocations for each node if the flow is feasible, and returns the slack allocations to the nodes.

### 6.6.1 Identify Queueing Path Nodes

When an application on the sender  $\alpha$  needs to transmit a flow with latency guarantees  $F_{\alpha,\beta} = (x_{\alpha,\beta}, y_{\alpha,\beta}, L_{\alpha,\beta}, \ell_{\alpha,\beta})$  to a receiver  $\beta$ , it initiates the PWP Flow Setup Protocol. The first step is to identify the queueing nodes along the path, each of which must provide a service guarantee.

An established technique for determining a unidirectional path is implemented in the tool *traceroute*. It transmits a sequence of packets having artificially-low *time-to-live* (TTL) values. When the first packet reaches the first router on the network, it will ‘expire’. That first router will then return an ICMP *time-exceeded* message to  $\alpha$ , which includes an IP address from the router. The sender would then record that IP address, then send another probe into the network having a TTL of 2. This causes the second router to respond, so that the sender identifies it. This sequence continues until the receiver  $\beta$  finally responds.

At this point,  $\alpha$  has a list of the queueing nodes.  $\alpha$  together with these nodes form a distributed system which task is to deliver the flow according to temporal constraints imposed by  $\alpha$  and the current workload of the other members.

### 6.6.2 Collect Workload Parameters from Nodes

The sender would then contact each of the nodes to collect its workload – i.e., flows for which service has been guaranteed on the outbound link which  $\alpha$  wishes to employ. Each node  $j$  would then return a task set

$$T_j = ((x_1, y_1, d_1, c_1), (x_2, y_2, d_2, c_2), \dots, (x_n, y_n, d_n, c_n))$$

to  $\alpha$ . The task sets will be used to determine feasibility of the new flow,  $F_{\alpha,\beta}$ . Each node would need to temporarily lock itself against future changes, because it is in the midst of a flow-setup transaction until the sender returns its slack allocation  $s_{(\alpha,\beta),j}$ .

### 6.6.3 Determine Minimum OTT

The minimum one-way trip time must be computed, either by information provided by each node, or by estimation through experiments.

### 6.6.4 Compute Slack Allocation and Inform Nodes

The sender then attempts to compute a slack allocation which will satisfy its flow requirements. Recall that the end-to-end slack  $S_{\alpha,\beta}$  is determined by the application's tolerance of latency,  $L_{\alpha,\beta}$ . Because the feasibility computation takes place on the sender before it has specified its requirements to any of the queueing nodes, it can iteratively test its desired flow specifications until it finds a flow which the path can guarantee. For example, while the latency requirements of a teleconference may remain constant, the path may be able to support a flow of 150 thousand bits per second for a slow frame-rate teleconference, when it could not support a flow of 500 thousand bits per second.

Once a combination has been successfully determined, the local slack allocation  $s_{(\alpha,\beta),j}$  for each node  $j$  is derived, and is sent to the node along with the flow specification of  $F_{\alpha,\beta}$ . The node is then responsible to configure itself for the new flow by enabling a token properly-placed token bucket.

### 6.6.5 Send Traffic

As  $\alpha$  sends data to  $\beta$ , each node along the path can classify the traffic as being part of the flow  $F_{\alpha,\beta}$  only by examining the IP source and destination addresses. Traffic classified thus is submitted to the token bucket which has been configured for the flow.

### 6.6.6 Teardown Flow Guarantee

If completion of the flow ever occurs, then the sender  $\alpha$  should inform each of the routers that the flow is complete, thereby returning resources which had been allocated for the flow.

## 6.7 Incremental Deployment

A public network, such as the current Internet, is not a unified entity under any single administrative authority. Therefore, deployment of protocols such as PWP do occur in any unified way. Thus it's very possible that a flow may include some routers which support PWP, and others which do not support it.

PWP has the attractive property that it is useful even in such situations. By examining the flow-setup protocol described in section 6.6, it is clear that the flow-setup procedure is transformed into a series of protocol interactions between the sender and each of the queueing nodes. If the sender were to somehow 'skip' one of the queueing nodes and therefore have guarantees from only part of the path, then PWP would still operate, but without the full guarantee.

As fewer nodes along the path can make the PWP guarantee, then even less can be said of the maximum latency along the end-to-end path. However, observe that if even one of the routers is capable of guaranteeing a level of service, then the path is therefore at-least a little better than best-effort. As more nodes along the path support the guarantee mechanism, the quality of service committed to a flow will improve. And, of course, in the limit when all nodes along the path support PWP, then hard guarantees can be made.

Given the relative elasticity present in many real-time applications, such partial improvements may be extremely useful. Thus, everywhere PWP is deployed it can provide value.

## 7 Summary

This effort presents the problem of making guarantees to real-time network flows, and describes theoretically two approaches for doing so: PWP-Sporadic models the offered load as a process for which minimum interrelease times are guaranteed, and PWP-Rate supports processes which expect to generate traffic at a fixed rate. The relative advantages of PWP-Rate over PWP-Sporadic are discussed.

The deployment of PWP-Rate is discussed at some length. A queueing structure is proposed which exploits an adaptation of the existing token-bucket technique for providing guaranteed deadlines, and an admission-control scheme is proposed which offloads to the sender the computationally- expensive optimization task inherent to the PWP flow setup. Finally, the advantages to users in even a partial deployment of PWP are explained.

## References

- [1] BANERJEA, A., FERRARI, D., MAH, B. A., MORAN, M., VERMA, D. C., AND ZHANG, H. The Tenet real-time protocol suite: design, implementation, and experiences. *IEEE/ACM Transactions on Networking* 4, 1 (1996), 1–10.

- [2] CHUNG, J., AND CLAYPOOL, M. Dynamic-cbt and chips – router support for improved multimedia performance on the internet. In *Proceedings of the ACM Multimedia Conference* (November 2000).
- [3] CLARK, D. D. The design philosophy of the DARPA internet protocols. In *SIGCOMM* (Stanford, CA, August 1988), ACM, pp. 106–114.
- [4] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair-queueing algorithm. *Journal of Internetworking* 1, 1 (1990).
- [5] FERRARI, D., AND VERMA, D. C. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications* 8, 3 (1990), 368–379.
- [6] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (1993), 397–413.
- [7] JEFFAY, K. Private Communication, February 2002.
- [8] JEFFAY, K., AND GODDARD, S. A theory of rate-based execution. In *20th IEEE Real-Time Systems Symposium* (1999), pp. 304–314.
- [9] JEFFAY, K., STANAT, D. F., AND MARTEL, C. U. On non-preemptive scheduling of periodic and sporadic tasks. In *12th IEEE Symposium on Real-Time Systems* (1991), pp. 129–139.
- [10] KAUR, J., AND VIN, H. M. Core-stateless guaranteed throughput networks. Tech. Rep. TR-01-47, Department of Computer Sciences, University of Texas at Austin, November 2001.
- [11] LIU, C., AND LAYLAND, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20, 1 (Jan 1973), 46–61.
- [12] MOK, A.-L. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, MIT, Department of EE and CS, May 1983.
- [13] SHENKER, S., CLARK, D., AND ZHANG, L. A service model for an integrated services internet, October 1993.
- [14] ZHANG, Z., DUAN, Z., AND HOU, Y. Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services, 2000.
- [15] ZHENG, Q., AND SHIN, K. G. On the ability of establishing real-time channels in point-to-point packet switched networks. *IEEE Transactions on Communications* 42, 2 (1994).