Parallel switching in connection-oriented networks[*]

James Anderson **Sanjoy Baruah** **Kevin Jeffay**

The University of North Carolina at Chapel Hill

Abstract

*Packet switching in connection-oriented networks that may have multiple parallel links between pairs of switches is considered. An efficient packet-scheduling algorithm that guarantees a deterministic quality of service to connections with real-time constraints is proposed – this algorithm is a generalization of some recent multi*processor *scheduling algorithms, and offers real-time performance guarantees similar to those offered by earlier fair-scheduling strategies such as Weighted Fair Queueing and proportional-share schemes.*

1. Introduction

Asynchronous Transfer Mode (ATM) networking technology is centered around the concept of switching 53-byte *cells*, where the cell is the basic unit of data transfer. One of the major reasons behind this design choice is that it is possible to build *parallel* switches, in which several of these small, fixed-size, cells are simultaneously processed by different switching elements and each switching element takes the same amount of time to complete its job. While much research has been performed on the issue of providing deterministic quality of service (QoS) guarantees to connections in packet-switched networks, none of this research, to our knowledge, exploits the parallelism capabilities of such networks. In part, this is due to the inherent intractability of most problems in deterministic multi-resource (as opposed to *uni*-resource) scheduling.

Particularly with respect to multimedia applications, the data-transfer requirements of an application may often be modelled as a data "stream" that is generated at a fairly regular rate, subject to occasional bursts. This model has been formalized into the concept of a *flow* [13, 12, 5, 6], which considers the traffic of a particular connection to be a sequence of packets or cells generated by the source of the connection: each packet belonging to a flow passes through the same sequence of switches along a path, established at connection-admission time, from the source to the destination in the network. A connection request specifies the rate at which it intends to generate data (i.e., it specifies its flow parameters), and the request is admitted by the network if and only if this flow would not overload the network and cause a consequent unacceptable degradation of service to previously admitted connections.

1.1. Fairness

Consider a link between two switches that is of bandwidth B. Suppose that n connections $c_1, c_2, \ldots, c_n$ share this link, and that each connection c_i is characterized by a flow rate of $f(c_i)$. The ratio $w(c_i) \overset{\text{def}}{=} f(c_i)/B$ denotes the fraction of the total bandwidth on the link that connection c_i requires (clearly, it is necessary that $\sum_{i=1}^{n} w(c_i) \leq 1$).

We would ideally like to be able to make the following guarantee to each connection c_i: over any time interval $[t_x, t_y]$, c_i is able to send $(t_y - t_x) \cdot f(c_i)$ units of data through this switch (provided, of course, that there is always some traffic waiting to be transmitted — i.e., that the connection is always *backlogged* during the interval $[t_x, t_y]$). However,it can be shown that achieving this would require that cells be infinitesimally small: since this is not the case, the "granularity" of our guarantee can be no smaller than the cell-size. More specifically, let a *time slot* denote the amount of time required to transmit one cell over the link — each time slot is equal to p/B, where p denotes the cell size. Assuming that time at each switch is measured in time slots numbered beginning with zero, the "fairest" bandwidth allocation scheme would be one that allocates at least $\lfloor t \cdot w(c_i) \rfloor$ slots out of any consecutive t slots to each connection c_i (provided, of course, that

[*]Supported in part by the National Science Foundation (Grant Nos. CCR-9510156, CDA-9624662, CCR-9704206, CCR-9732916, and CCR-9972211). The research of the first author was also supported in part by an Alfred P. Sloan Research Fellowship.

the connection is backlogged over this entire duration). Our attempt here is to achieve a slightly weaker form of fairness, one that is similar to the fairness guarantees made by previous fair-queueing algorithms such as WFQ [8, 4]: Suppose that there are no cells of connection c_i queued at the switch at the start of time slot $t_o - 1$, and that a cell arrives at the beginning of time slot t_o. In our version of fairness, *a bandwidth allocation scheme is fair if it allocates at least* $\lfloor (t_1 - t_o) \cdot w(c_i) \rfloor$ *slots out of the* $(t_1 - t_o)$ *slots numbered* $t_o, t_o + 1, \ldots, t_1 - 1$ *to connection* c_i (provided, once again, that the connection is backlogged during these slots).

1.2. High-bandwidth connectivity

Modern networks, such as ATM-based ones, are typically constructed using optical fibers. One method of obtaining a high-bandwidth connection between a pair of neighbouring switches is to connect them with a high-bandwidth fiber. Another method is to have several fibers in parallel connect the two switches — it costs significantly less to connect two locations with m fibers (typically, within the same cable) than it does to connect them with one fiber that has a bandwidth m times as high; more important, fast serial switching elements (i.e., switching elements that can handle a large number of cells per unit time) are very difficult to construct, and currently constitute the technological bottleneck to the implementation of higher-bandwidth networks. In this second method of achieving a high-bandwidth connection between two switches S1 and S2, we therefore have $m > 1$ lower-bandwidth links (rather than a single higher-bandwidth link) between them. At the beginning of each time slot, switch S1 would select up to m waiting cells in parallel and transmit them out on the m links to S2. These cells would arrive simultaneously at S2, and have to be processed in parallel; i.e., based upon the *Virtual Channel Indicator (VCI)* on each cell, the cell would be routed to the appropriate output queue in S2.

Assuming that each fiber has bandwidth B we could now, in principle, have n connections $c_1, c_2, \ldots, c_n$ share the network edge between switches S1 and S2, provided that $\sum_{i=1}^{n} f(c_i) \leq m \cdot B$.

One of the major requirements of such an approach to increasing bandwidth is that, since the network offers a connection-oriented service, the relative ordering of the cells belonging to any particular connection be preserved. Specifically, if more than one cell of connection c_i is transmitted over the edge from S1 to S2 during any time slot, it is desirable that S2 be able to order these cells correctly before passing them on. This could be achieved, e.g., by adding a "cell-number" to each cell; however, such an approach would (i) slow down the switching process at S1 and S2 by increasing the amount of work that needs to be done at these switches, and (ii) be incompatible with the ATM standard as currently defined. Even if this were to be nevertheless done, sending several cells of the same connection during the same slot would require that these cells be compared and reordered at the receiving switch; these inherently sequential operations would reduce the amount of parallelism achieved in switch S2. All told, a better design decision would be to permit at most one cell of each connection to traverse the link during any time slot[1]. The problem of bandwidth allocation on the network edge S1$\rightarrow$S2 thus reduces to the following problem:

The parallel switching problem: *Given n connections $C = \{c_1, c_2, \ldots, c_n\}$ and m parallel links, with connection c_i needing to switch cells for* at most *a fraction $w(c_i)$ of the time slots, choose, for each time slot, a subset of C of size at most m of the connections that will be permitted to transmit cells on the links during this time slot.*

Of course, we would like to be able to do this in as fair a manner as possible, where the idea of "fairness" should be closely related to the ideas discussed above. This is a *multi*-resource version of the problem of sharing a single resource in a fair manner; however, as far as we can tell, none of the single-resource fair queueing schemes suggested in the literature (such as Weighted Fair Queueing [4, 8], Worst-case Fair WFQ [3], Start-time Fair Queueing [6], etc.), nor the proportional-share schemes [11, 10, 9] generalize to this multi-resource problem — detailing the exact reasons why this is so is beyond the scope of this report.

1.3. Proportionate Progress and Pfairness

The notion of *proportionate progress*, and the associated concept of pfairness [1], deals with the following scheduling problem:

The multiprocessor periodic scheduling problem: *Given n tasks $\tau = \{x_1, x_2, \ldots, x_n\}$ and m identical processors such that each processor must be allocated for each fixed (indivisible) quantum of time to a single task, and no task may use more than one processor during any time quantum, with task x_i needing*

[1]This design decision has two significant consequences: First, no individual connection with a bandwidth requirement greater than B can be admitted over this link, despite the fact that the total link capacity is $m \cdot B$. Second, we are no longer using a *work conserving* scheduling descipline — if less than m connections have cells waiting to use the network edge, then the edge will not be used at full capacity despite the fact that there are cells that need to use it.

to execute on a processor for exactly *a fraction $w(x_i)$ of the time quanta, choose, for each time quantum, a subset of τ of size at most m of the tasks that will be permitted to use the m processors during this time slot.*

It has been shown [1] that the m processors can be allocated in a *pfair* manner — i.e., in such a manner that, over the quanta numbered $0, 1, \ldots, t-1$, each task x_i will have executed on a processor for exactly $\lfloor w(x_i) \cdot t \rfloor$ or $\lceil w(x_i) \cdot t \rceil$ quanta, for all $t \in \mathbf{N}$; Algorithm PF [1] is a scheduling algorithm that determines the subset of tasks that obtain the m processors during each time quantum.

It should be evident that the multiprocessor periodic scheduling problem is closely related to the parallel switching problem. However, the two problems have some major differences. The main difference — and the reason why Algorithm PF cannot be directly used to solve the parallel switching problem — lies in the fact that, while tasks are always available to use a processor, there may simply not be any cells of a connection queued up at the time that Algorithm PF would want to service that particular connection, but a cell arrives on this connection immediately after its "turn" has gone by. This is a consequence of the fact that, while everything about an instance of the periodic multiprocessor scheduling problem — the number of tasks, their weights, the number of processors, etc. — is known beforehand, the parallel switching problem is inherently *on-line*, in that the exact times at which cells of a particular connection will arrive at a switch is not *a priori* known. Adapting multiprocessor periodic fair scheduling algorithms such as Algorithm PF to a dynamic, on-line environment while continuing to obtain high utilization of the available bandwidth, is the major algorithmic challenge in being able to design fair bandwidth allocation strategies for networks that may have multiple parallel links between pairs of switches. This report describes our attempts as designing such strategies.

The remainder of this report is organized as follows. In Section 2, we briefly summarize previous research in pfair scheduling – this research will form the basis of the scheduling schemes introduced in this paper. In Section 3, we present (and prove correct) Algorithm NF, an algorithm for scheduling cells on parallel links in a fair manner. In Section 4, we address some concerns regarding the run-time computational complexity of Algorithm NF, and propose certain techniques for increasing its efficiency. We conclude in Section 5 with a discussion on alternative means of exploiting parallelism in network links.

2. A brief introduction to pfairness

In this section, we briefly describe previous research on the multiprocessor periodic scheduling problem. We review the concept of pfairness (first introduced in [1]), and present some notation, terminology, and important results that will be used in later sections.

We start with some conventions:

- Let τ denote a set of n periodic tasks, that are to be scheduled on m identical processors.
- We adopt the standard notation of having $[a, b)$ denote the contiguous natural numbers $a, a+1, \ldots, b-1$.
- Scheduling decisions are made at integral values of time, numbered from 0. The real interval between time t and time $t+1$ (including t, excluding $t+1$) will be referred to as **slot** t, $t \in \mathbf{N}$.
- The quantity $(\sum_{x \in \tau} w(x))$ is referred to as the *density* of the set τ of periodic tasks.

Now some definitions:

- A *schedule* S for periodic task system τ is a function from $\tau \times \mathbf{N}$ to $\{0, 1\}$, where $\sum_{x \in \tau} S(x, t) \leq m$, $t \in \mathbf{N}$. Informally, $S(x, t) = 1$ if and only if task x is scheduled in slot t.
- The quantity $\mathsf{allocated}(S, x, t)$ of a task x at time t with respect to schedule S is defined as follows:

  $$\mathsf{allocated}(S, x, t) \stackrel{\text{def}}{=} \textstyle\sum_{t' \in [0,t)} S(x, t').$$

- The *lag* of a task x at time t with respect to schedule S, denoted $\mathsf{lag}(S, x, t)$, is defined by:

  $$\mathsf{lag}(S, x, t) = w(x) \cdot t - \mathsf{allocated}(S, x, t).$$

  The quantity $w(x) \cdot t$ represents the amount of time for which task x *should* have been allocated the processor over $[0, t)$, and $\mathsf{allocated}(S, x, t)$ is equal to the number of slots for which task x was actually scheduled. Therefore, a positive lag indicates that a task has been scheduled for less than its "fair" share, while a negative lag indicates that it has been overscheduled. A lag of exactly zero indicates that the task has received exactly its fair share thus far.

- A schedule S is *pfair* if and only if

  $$\forall x, t : x \in X, t \in \mathbf{N} : -1 < \mathsf{lag}(S, x, t) < 1.$$

  That is, a schedule is pfair if and only *if it is never the case that any task x is overallocated or underallocated by an entire slot.*

With respect to a given task x, let $\mathsf{earliest}(x, j)$ (resp., $\mathsf{latest}(x, j)$) denote the earliest (resp., latest) slot during which x may be scheduled for the jth time in any pfair schedule, where $j = 1, 2, \ldots$. We can easily derive closed-form expressions for $\mathsf{earliest}(x, j)$ and $\mathsf{latest}(x, j)$:

$$\begin{aligned}
&\mathsf{earliest}(x, j) \\
&\quad = \min t : t \in \mathbf{N} : w(x) \cdot (t+1) - j > -1 \\
&\quad = \left\lfloor \frac{j-1}{w(x)} \right\rfloor \qquad (1)
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\mathsf{latest}(x, j) \\
&\quad = \max t : t \in \mathbf{N} : w(x) \cdot t - (j-1) < 1 \\
&\quad = \left\lceil \frac{j}{w(x)} \right\rceil - 1 \qquad (2)
\end{aligned}$$

Let S denote a partial schedule in which scheduling decisions have been made only for the time slots $0, 1, , \ldots, (t-1)$. Task x is **eligible** in schedule S at time t if it may receive the processor during slot t without becoming overallocated; i.e., if

$$\mathsf{allocated}(S, x, t) = k \wedge (\mathsf{earliest}(x, k+1) \leq t) \ .$$

Suppose that a task x is eligible in schedule S at time t. By definition, x can be scheduled during slot t without having its lag fall below -1; i.e.,

$$w(x) \cdot (t+1) - [\mathsf{allocated}(S, x, t) + 1] > -1 \ .$$

The above is logically equivalent to

$$w(x) \cdot (t+1) - \mathsf{allocated}(S, x, t) > 0 \ . \qquad (3)$$

The LHS of Inequality 3 represents $\mathsf{lag}(S, x, t+1)$, provided S does not schedule x during slot t (i.e., provided that $\mathsf{allocated}(S, x, t+1) = \mathsf{allocated}(S, x, t)$). But a lag > 0 indicates underallocation, from which it follows that

Lemma 1 *If a task is eligible during a slot, then not scheduling it during this slot guarantees that it will be* under*allocated at the start of the next slot.* □

The concept of pfairness was initially introduced in [1], in the context of constructing periodic schedules for a system of periodic tasks on several identical processors — the multiprocessor periodic scheduling problem (described in Section 1). The following theorem was proved there:

Theorem 1 *A system of periodic tasks can be scheduled in a pfair manner on m processors provided the weights of all the tasks sum to at most m.* □

In addition, an on-line scheduling algorithm – Algorithm PF – was presented that generates a pfair schedule for any such system of periodic tasks. In order to describe this algorithm, we need to introduce some more terminology:

Let S denote a partial schedule in which scheduling decisions have been made only for the time slots $0, 1, , \ldots, (t-1)$. For any positive integer i, the ***i*'th pseudo-deadline** of task x at time t in schedule S is defined to be equal to $\mathsf{latest}(x, \mathsf{allocated}(S, x, t) + i)$. Intuitively, the i'th pseudo-deadline of task x denotes the latest slot by which task x must be scheduled i more times in S, if it is to not violate its lag bound.

Furthermore, we say that this i'th pseudo-deadline is a **solid pseudo-deadline** if

$$\mathsf{latest}(x, \mathsf{allocated}(S, x, t) + i) \equiv \frac{\mathsf{allocated}(S, x, t) + i}{w(x)} - 1.$$

Intuitively, a solid pseudo-deadline occurs when the $\lceil \ \rceil$ function used in Equation 2 introduces no rounding error since the argument to the $\lceil \ \rceil$ is itself integral.

ALgorithm PF schedules tasks according to their **pf-priorities**. A formal definition of pf-priorities is provided in Figure 1; informally, the relative pf-priorities of two tasks at a particular time in a given schedule are determined by comparing their i'th pseudo-deadlines, with i initially set to one:

L1: If the i'th pseudo-deadlines are unequal, then the task with the earlier i'th pseudo-deadline has greater priority.

Else if they are equal but one of the pseudo-deadlines is a solid pseudo-deadline, then the other task has greater priority (if they are equal and both are solid pseudo-deadlines, then either task can be arbitrarily assigned the greater priority)

Else the pseudo-deadlines are equal but neither is a solid pseudo-deadline: in this case, the next pair of pseudo-deadlines of the tasks must be compared in the same manner. I.e., set i to $(i+1)$, and go to L1.

Algorithm PF is now easily described: *at each time slot, schedule the (at most) m eligible tasks with the highest pf-priorities.*

3. Fairness and the parallel switching problem

While scheduling periodic tasks on multiple processors, Algorithm PF can decide to schedule any task at any time-slot (subject, of course, to the fairness constraints we have chosen to impose). The situation is

Task x is defined to have greater **pf-priority** than task y in schedule S at time t if, in schedule S at time t, there in an integer $i \geq 1$ such that

for all j, $1 \leq j < i$,
  the j'th pseudo-deadline of x is not a solid pseudo-deadline, **and**
  the j'th pseudo-deadline of y is not a solid pseudo-deadline, **and**
  the j'th pseudo-deadline of x equals the j'th pseudo-deadline of y
and
  the i'th pseudo-deadline of x is smaller than the i'th pseudo-deadline of y, **or**
  the i'th pseudo-deadline of y is a solid pseudo-deadline

Figure 1. Definition of pf-priorities

rather different in the case of network traffic: if there are no cells of a particular connection queued at the start of slot t, then this connection *cannot* be serviced during this slot, regardless of the scheduling decision made by Algorithm PF. We illustrate by means of an example.

Example 1 Consider a situation in which three connections c_1, c_2, and c_3, with $w(c_1) = w(c_2) = w(c_3) = 2/3$, share a parallel link composed of two fibers (i.e., with $m = 2$). Assume that all three connections have their first cells arrive at the switch at time 0, and subsequent cells arrive exactly 1.5 time units apart (i.e., at exactly the rate implied by their weights). At the start of the time-slot zero, all three connections are eligible; without loss of generality, assume that Algorithm PF selects c_1 and c_2 to send their cells during this slot. Algorithm PF would then let c_3 send its first cell during time-slot one, and would also select one of c_1 or c_2 to send its second cell during this time slot. However, this second cell has not yet arrived at the start of the time slot. $\square$

The problem illustrated by Example 1 is, in effect, insoluble — there is simply nothing that any algorithm could do about the fact that there are only three cells available for transmission during the first two time-slots. Our approach — formalized below as **Algorithm NF** (for "Network Fair") — handles this by *weakening* the performance guarantee: in Example 1, we will guarantee that connection c_i gets its first cell transmitted during slots 0 or 1; its second cell during slots 2 or 3; its third cell during slots 3 or 4; and so on. That is Algorithm NF may transmit a cell one time-slot later than is mandated by the pfairness requirement — this is a consequence (as Example 1 illustrates) of the fact that cells that arrive *during* a slot are only available for transmission during the *next* slot. However, Algorithm NF does guarantee that *no cell is delayed by more than one slot from where it should have been transmitted in accordance with pfairness.*

We now describe the operation of Algorithm NF. This is how we will proceed. First, we will explain how Algorithm NF performs *admission control* — determining whether a new connection request is accepted by a switch. Next, we will discuss how incoming cells are queued at a switch while awaiting transmission. And finally, we will describe how Algorithm NF uses the notion of pf-priorities (described in Section 2) to determine which cells are to be transmitted during each time slot.

§1. Connection establishment. Suppose that there are m parallel links, each of capacity B, leading from switch S_1 to switch S_2 — we will refer to these links cumulatively as the $\underline{\textit{network edge } (S_1, S_2)}$. With each real-time connection x_i that has already been established passing through network edge (S_1, S_2), we associate a *weight* $w(x_i)$, $0 \leq w(x_i) \leq 1$, denoting that connection x_i has reserved a bandwidth equal to $w(x_i) \cdot B$ on this edge.

Suppose that a new real-time connection x now desires to use this edge. Among the information that x must make available during connection establishment is its bandwidth requirement, from which its weight $w(x)$ is computed (by dividing this bandwidth requirement by B). Admission control now consists of switch S_1 validating that $w(x)$ plus the weights of all already-admitted connections on network edge (S_1, S_2) does not exceed m. This step is repeated by switch S_ℓ on each network edge (S_ℓ, S_k) that the connection wishes to traverse, and the connection is admitted if and only if doing so would not cause any network edge to exceed its capacity.

If a connection x is successfully established, then certain resources – buffers, registers, etc. — are allocated to it on each switch through which it passes[2].

[2]Some precomputation is also done, which facilitates the run-time execution of Algorithm NF during cell-switching time —

Some of the variables that are created and maintained in each switch are the following (the use to which these variables are put will be explained during the remainder of this section):

baseTime(x) **:** A positive integer, denoting the time from which we begin considering connection x for the purposes of transmitting its cells across the network edge. This is determined by when cells begin arriving at the switch. (A value of ∞ denotes that the connection is not currently "active".)

count(x) **:** A non-negative integer, denoting the number of cells of connection x that have been transmitted since time baseTime(x).

$Q(x)$ **:** A FIFO queue of the cells of connection x that have arrived at the switch but have not yet been transmitted.

§**2. Queueing newly-arrived cells.** At the start of each time-slot t, all the cells that had arrived during the interval $(t-1, t]$ are processed. Suppose that a cell of (already admitted) real-time connection x has arrived; this is processed as follows:

```
if (baseTime($x$) == $\infty$){
    (This indicates that the connection was
    previously not active)
    baseTime($x$) = $t$
    count($x$) = 0
    }
add the cell to Q($x$).
```

§**3. Transmitting cells.** Once all cells that arrived during $(t-1, t]$ have been queued as described above, the m cells that are to transmitted during time-slot t are selected. Only eligible connections may contend to send cells during this slot: connection x is deemed to be eligible if the following condition, which checks whether $(t-\text{baseTime}(x))$ — the interval during which this connection has been "active" — is at least $\mathsf{earliest}(x, \text{count}(x)+1)$ (see Equation 1), holds:

$$\text{baseTime}(x) + \left\lfloor \frac{\text{count}(x)}{w(x)} \right\rfloor \leq t \ .$$

However, if an eligible connection x has no cells queued for transmission (as would have been the case in Example 1 at the start of time-slot one), then it cannot contend for bandwidth. In this case, Algorithm NF "deactivates" this particular connection:

```
if x is deemed eligible and $Q(x)$ is empty
    baseTime($x$) = $\infty$
```

this is described in more detail in Section 4.

Process cells that arrive at the start of slot t**.**
A cell of connection x has arrived:

```
    if (baseTime($x$) == $\infty$){
            baseTime($x$) = $t$
            count($x$) = 0
    }
    add the cell to Q($x$).
```

Choose cells for transmitting during slot t**.**

```
if (baseTime($x$) + $\left\lfloor \frac{\text{count}(x)}{w(x)} \right\rfloor \leq t$)  {
//Connection $x$ is eligible
    if $Q(x)$ is empty
        baseTime($x$) = $\infty$
    else {    //$Q(x)$ is not empty
        if $x$ is one of the $m$ highest-priority connections
            {
            transmit a cell from $Q(x)$
            count($x$) = count($x$) + 1
            }
    }
}
```

Figure 2. Pseudocode

(When the *next* cell of this connection arrives at a later time t' –in Example 1, at the start of time-slot two; i.e., $t' = 2$– the connection is reactivated, when this next cell is processed, with baseTime(x) set to t' and count(x) to zero.)

Of the remaining eligible connections (i.e, those with non-empty queues), the m connections with the highest pf-priorities[3] get to each transmit a cell:

```
if $x$ is one of the $m$ highest-priority connections {
        transmit a cell from $Q(x)$
        count($x$) = count($x$) + 1
}
```

(If there are fewer than m eligible connections, then all transmit a cell as above, and any unused bandwidth can be used to transmit non-real-time cells.)

The pseudocode presented above is collected together in Figure 2.

3.1 Correctness

For the most part, the correctness of Algorithm NF follows directly from the correctness of Al-

[3]We postpone discussion on how this determination of pf-priorities is actually implemented to Section 4.

gorithm PF [1]. The major difference between the behaviours of the two algorithms arises when Algorithm NF "deactivates" an eligible connection whose queue is empty, and subsequently reactivates it when its queue becomes populated. To ensure that a connection cannot obtain *more* than its reserved share of the shared bandwidth by strategically timing the arrivals of its cells, we must ascertain that such deactivation and subsequent activation does not cause a connection to ever become overallocated. But this is easily seen to hold: observe that in order to become deactivated, a connection must first be eligible; once deactivated, it cannot be reactivated until the beginning of the *next* slot. But by Lemma 1, not scheduling an eligible connection will guarantee that this connection is underallocated at the start of the next slot. Hence a previously deactivated connection is guaranteed to have consumed less than its reserved share of bandwidth by the time it is next activated (although it could have been overallocated – i.e., have had a negative lag – at the instant it was deactivated).

Performance guarantees:

A connection x is said to be *backlogged* at a switch throughout an interval $[t_1, t_2]$ if $Q(x)$ at this switch is non-empty at each instant $t_1, t_1+1, \ldots, t_2$. Notice that

- Whether a connection is backlogged or not over an interval depends upon both the time at which cells arrive at the switch, and the scheduling decisions made over the interval. For instance, connection c_3 is backlogged over the interval $[0, 1]$, while connections c_1 and c_2 are not, in Example 1.

- If connection x is backlogged over $[t_1, t_2]$ when scheduled by Algorithm NF, then Algorithm NF would not deactivate x at any time instant during this interval.

The performance guarantee made by Algorithm NF can now be stated: *If the queue $Q(x)$ associated with a connection x is empty at time-instant $t_1 - 1$ but a cell arrives at t_1 and the connection is backlogged over the interval $[t_1, t_1 + t)$, then Algorithm NF will have transmitted at least $\lfloor t \cdot w(x_i) \rfloor$ cells of connection x during the interval $[t_1, t_1 + t)$.*

To see that this guarantee does in fact hold, notice that, in the absence of deactivating and reactivating, Algorithm NF behaves exactly like Algorithm PF. Now, if connection x is backlogged during $[t_1, t_1 + t)$, then Algorithm NF never deactivates and reactivates x. Since (as we have seen above), no other connection could obtain more that *its* fair share of the resource by such deactivating/ reactivating either, Algorithm NF will transmit at least as many cells of x as would have been transmitted in a pfair schedule starting at time t_1.

4. Efficient determination of pf-priorities

A major drawback of the pf-priority determination procedure described in Section 2 is that it does not *a priori* limit the number of pairs of pseudo-deadlines that need to be compared in order to determine the relative pf-priorities of two tasks. Indeed, so long as two tasks have corresponding pairs of their pseudo-deadlines equal –and none of these pseudo-deadlines are solid– we must continue comparing further pairs of pseudo-deadlines. Some research has been done [1] on estimating the exact worst-case computational complexity of such comparison, and attempts have been made [2] – and continue to be made – at obtaining more efficient comparison routines.

In this section, we describe a transformation on sets of real-time connections which *increases* the weight of each connection by an *a priori* bounded amount, such that the pf-priorities of the resulting connections can be more efficiently determined. While the schedule obtained by using these pf-priorities will not in general be pfair with respect to the *original* weights of the connections (since the weights will have been increased, the connections will in general be overallocated as compared to a pfair schedule), this need not be a concern to us here in the context of our network parallel switching application since our desire is to avoid *under*allocation (rather than worry about overallocation).

Our transformation is designed to take advantage of the following fact:

Lemma 2 *A connection x with weight $w(x) = \frac{k}{\ell}$, where k and ℓ are both positive integers, can have at most $(k-1)$ consecutive pseudo-deadlines that are not solid.*

Proof Sketch: The k pseudo-deadlines of task x at time t in schedule S are $\mathsf{latest}(x, \mathsf{allocated}(S, x, t) + 1)$, $\mathsf{latest}(x, \mathsf{allocated}(S, x, t) + 2)$, $\ldots$, $\mathsf{latest}(x, \mathsf{allocated}(S, x, t) + k)$. By the definition of *solid* pseudo-deadlines, one of these pseudo-deadlines is solid if

$$\frac{\mathsf{allocated}(S, x, t) + j}{w(x)} - 1$$

is an integer for some j, $1 \leq j \leq k$. Since $w(x) = \frac{k}{\ell}$, this is equivalent to requiring that

$$\frac{\mathsf{allocated}(S, x, t) + j}{k} \cdot \ell - 1$$

be an integer. But at least one of the k consecutive integers $(\mathsf{allocated}(S, x, t) + 1)$, $(\mathsf{allocated}(S, x, t) + 2)$, $\ldots$,

$(\mathsf{allocated}(S, x, t) + k)$ is divisible by k; consequently, at least one of these k pseudo-deadlines is a solid pseudo-deadline. $\square$

We now describe our transformation on connections. Let k be some pre-specified constant integer, $k \geq 1$. (We will describe later how the value of k is detemined.) Given a connection x, with weight $w(x)$, assign x a weight $w'(x)$ as follows:

$$w'(x) \overset{\text{def}}{=} \frac{k}{\lfloor k/w(x) \rfloor}$$

Notice that the transformed connection will have at most $(k-1)$ consecutive pseudo-deadlines that are not solid — this follows from Lemma 2. Hence determining the pf-priorities of a pair of transformed connections will involve comparing at most k pairs of pseudo-deadlines: by choosing k to be reasonably small, determining the relative pf-priorities of tasks can thus be done efficiently.

But what does this increase in the efficiency of priority determination cost us? Lemma 3 addresses this question:

Lemma 3 *The above transformation increases the weight of a connection by less than a factor of* $(1 + \frac{1}{k})$*; i.e.*

$$w'(x) \quad < \quad w(x) \cdot \left(1 + \frac{1}{k}\right)$$

Proof Sketch: Let $\ell \overset{\text{def}}{=} \lfloor k/w(x) \rfloor$; i.e., $w'(x) = k/\ell$ for some integer ℓ. Since $w(x) \leq 1$, it follows that $k/w(x) \geq k$; i.e., $\lfloor k/w(x) \rfloor \geq k$; i.e., $\ell \geq k$.

Now since $\lfloor k/w(x) \rfloor = \ell$, it must be the case that $k/w(x) < (\ell + 1)$, from which it follows that $w(x) > \frac{k}{\ell+1}$. Therefore,

$$\frac{w'(x)}{w(x)} < \frac{k/\ell}{k/(\ell+1)} = 1 + \frac{1}{\ell} \ .$$

Since $(1 + \frac{1}{\ell})$ decreases with increasing ℓ, this ratio is maximized for the smallest value of ℓ, i.e., for $\ell = k$. Hence $\frac{w'(x)}{w(x)} < 1 + \frac{1}{k}$, from which it follows that

$$w'(x) \quad < \quad w(x) \cdot \left(1 + \frac{1}{k}\right) \ .$$

$\square$

Lemma 3 tells us that transforming a connection x in the manner described above causes no more than a $(1 + \frac{1}{k})$-fold increase in the connection's weight. Recall that Algorithm NF's admission control test required only that the sum of the weights of all admitted connections not exceed m (where m is the number of parallel links comprising the network edge). For a set of connections γ to all be admitted after each connection in γ has been so transformed, it is therefore sufficient that

- Each individual connection's weight, after the transformation, should not exceed 1; i.e., $w(x) \leq 1/(1 + \frac{1}{k})$ (which is equivalent to requiring that $w(x)$ be no larger than $\frac{k}{k+1}$), and
- $\sum_{x \in \gamma} w'(x)$ be no larger than m; i.e., $\sum_{x \in \gamma} w(x) \cdot (1 + \frac{1}{k}) \leq m$, which is equivalent to requiring that
$$\left(\sum_{x \in \gamma} w(x) \quad \leq m \cdot \frac{k}{k+1} \right) .$$

Notice that the bound above — $\left(m \cdot \frac{k}{k+1}\right)$ — increases with increasing k (as $k \to \infty$, this approaches m). On the other hand, the procedure for determining the relative pf-priorities of a pair of connections takes time $\mathcal{O}(k)$, and hence the efficiency of this procedure decreases with increasing k. We thus see that (for a given value of m) there is a tradeoff between the total weight of sets of connections which are guaranteed admitted after such transformation, and the efficiency of actually determining the schedule. In particular, suppose that we wish to guarantee that all sets of connections with a total weight no larger than $U \cdot m$ are always admitted: algebraic manipulation of the above inequality yields the result that the desired value of k is $\left\lceil \frac{U}{1-U} \right\rceil$.

In the context of our network parallel switching application, this information can be used as follows: Suppose we know beforehand that the real-time traffic across a network edge will not exceed U times the capacity of the edge, for some $U < 1$ — the rest of the bandwidth is to be used for non-real-time traffic. (I.e., the real-time capacity of the link is calibrated at U times its actual capacity.) This allows us to decide that we can transform each connection such that determining pf-priorities will require the comparison of no more than $\left\lceil \frac{U}{1-U} \right\rceil$ pairs of pseudo-deadlines[4]. For example, If we desire that U be 80%, the necessary value of k is $\left\lceil \frac{0.8}{1-0.8} \right\rceil$ which equals 4: this means that we can transform all admitted connections with k set to

[4]Although we will not discuss this further in this manuscript, if an upper bound on the weights of all possible connections is known beforehand then we may be able to obtain even lower values of k for a desired value of U. Specifically, suppose that it is known that no connection will have a weight greater than some constant W, then

$$k = \left\lceil W \cdot \left\lceil \frac{U}{1-U} \right\rceil \right\rceil$$

is sufficient. For example, if a U of 0.90 is desired but it is *a priori* known that no connection's weight will exceed 0.20, then $k = \left\lceil 0.20 \cdot \left\lceil \frac{0.90}{1-0.90} \right\rceil \right\rceil = 2$; i.e., at most two pseudo-deadlines must be compared to resolve the pf-priorities of any pair of connections

4, and will then never need to compare more than four pseudo-deadlines to determine the relative pf-priorities of a pair of connections.

Some additional observations:

- Notice that such a transformation permits a connection to "cheat" by actually generating enough cells to consume a fraction $w'(x)$ of the bandwidth of a single link (rather than a fraction $w(x)$ as originally promised: Algorithm NF guarantees timely delivery of all these extra cells as well). If this is undesirable – in the sense that it would have the effect of reducing the bandwidth available to non-real-time data – traffic policing can be done to prevent a connection from injecting more than the permitted amount of cells into the network.

- It has been observed that traffic flows tend to become more bursty as they traverse a network, consequently requiring greater buffering at intermediate switches and resulting in jittery delivery at the destination. The ability of Algorithm NF to actually service a connection x at a rate $w'(x)$ which may be greater than the rate $w(x)$ can help reduce this burstiness — if a buildup of cells of connection x occurs at a switch, the excess available bandwidth $(w'(x)-w(x))$ will be used by Algorithm NF to get rid of this buildup by transmitting cells of x at a faster rate.

Implementation. In Section 3, we had postponed discussion on how pf-priorities of connections are actually computed by Algorithm NF. We take this discussion up now.

Once a connection x is admitted (during connection establishment), x's weight at a switch $S1$ is transformed to $w'(x) = \frac{k}{\lfloor k/w(x) \rfloor}$, where k is a constant whose value is determined –as explained above– by how much of the switch's bandwidth is reserved for real-time traffic. In addition, an integer variable $\ell(x)$ stores the integer ℓ such that $w'(x) = k/\ell$, and the real numbers

$$0.0, \frac{1}{w'(x)}, \frac{2}{w'(x)}, \ldots, \frac{k-1}{w'(x)}$$

are precomputed and stored in an array $D(x)[[0, \cdots, k-1]$ — this array will be used by Algorithm NF to determine when a connection becomes eligible, and its pseudo-deadlines. For example, determining whether a connection is eligible (the test in the first line of the pseudocode for choosing cells for transmitting during slot t – Figure 2) can be performed by one "floor" operation and additional integer operations, and no floating-point operations:

$$\text{“if baseTime}(x) + \lfloor D(x)[\text{count}(x) \bmod k] \rfloor + \ell(x) \cdot (\text{count}(x) \,\mathsf{div}\, k) \quad \leq t\text{”}$$

More important, the pseudo-deadlines are easily computed, with no floating-point operation (other than a "ceiling" operation): the i'th pseudo-deadline of connection x is at time[5]

$$\text{baseTime}(x) + \lceil D(x)[(\text{count}(x)+1) \bmod k] \rceil + \ell(x) \cdot ((\text{count}(x)+1) \,\mathsf{div}\, k) \ .$$

By precomputing the $D(x)$ array at connection-establishment time, we will have thus considerably reduced the time required by Algorithm NF to determine which connections to service during each time-slot.

5. Conclusions; Other approaches

The networking community has recently been paying considerably attention to the problem of being able to provide determinstic quality of service guarantees to connections that can be characterized as *flows*. Such connections arise primarily in networked multimedia applications such as videoconferencing, telephony, video-on-demand, etc.

From one perspective, a computer network can be modelled as a *graph*, in which nodes correspond to routers and switches, and edges to physical links between pairs of routers. Cell-switching algorithms are concerned with determining the order in which cells are transmitted across these graph edges.

One method of obtaining a high-bandwidth connection between a pair of neighbouring switches is to connect them with a high-bandwidth fiber. Another method is to have several fibers in parallel connect the two switches — it costs significantly less to connect two locations with m fibers (typically, within the same cable) than it does to connect them with one fiber that has a bandwidth m times as high; furthermore, fast serial switching elements (i.e., switching elements that can handle a large number of cells per unit time) are very difficult to construct, and currently constitute the technological bottleneck to the implementation of higher-bandwidth networks.

We have described here an approach for cell-scheduling on such parallel-linked networks. Our approach — formalized as Algorithm NF — offers quality of service guarantees that are comparable to those

[5]While we will not get into the details here, comparision of pseudo-deadlines for the purpose of determining relative pf-priorities can also be conveniently implemented to execute in *constant* time per comparison in hardware, provided there are $\mathcal{O}(k)$ special-purpose registers per connection available at the router.

made by earlier algorithms (such as Weighted Fair Queueing and proportional-share schemes) that were designed for networks without parallel links. Under certain restrictions (in particular, we can guarantee no more than a fraction U of the bandwidth for real-time connections, for some $U < 1$), we have obtained efficient implementations of Algorithm NF.

We conclude with a brief description of other promising approaches towards parallel switching of flow-based traffic that, based upon our preliminary studies, seem to merit further investigation. Of these, potentially the most rewarding is to explore the possibility of extending the single-resource fair-queueing strategies to the parallel-switching domain. While these are not likely to be optimal, the enormous amount of "legacy" research — the algorithms, implementations, and analyses — that has been performed with respect to these strategies may offset a minor loss of efficiency (bandwidth) or fairness. Another possible approach towards using the single-resource fairness results for parallel switching involves *partitioning* the connections among the various fibers at connection establishment time, and then having each connection's cells contend for bandwidth only on its associated fiber; once again, the downside to such an approach is low utilization of available bandwidth.

Recall that we had claimed that the "fairest" bandwidth allocation scheme would be one that allocates at least $\lfloor t \cdot w(c_i) \rfloor$ slots out of any consecutive t slots to each connection c_i. There seems to be a close relationship between this notion of extreme fairness and the concept of pinwheel scheduling [7]; further exploring this relationship may yield interesting results.

References

[1] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[2] S. Baruah, J. Gehrke, and G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 280–288. IEEE Computer Society Press, April 1995. Extended version available via anonymous ftp from `ftp.cs.utexas.edu`, as Tech Report TR–95–02.

[3] J. Bennett and H. Zhang. WF^{2}Q: Worst-case fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, March 1996.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, September 1990.

[5] A. El-Nahas, K. Ahmed, and M. Gouda. Leaky bucket, refreshed bucket, and other flow admission criteria. In *Proceedings of the Fifth International Conference on Computer communications and Networks*, Rockville, MD, October 1996.

[6] P. Goyal, H. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. Technical Report TR-96-02, Department of Computer Sciences, University of Texas at Austin, 1996.

[7] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference on System Science*, pages 693–702, Kailua-Kona, January 1989.

[8] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[9] I. Stoica and H. Abdel-Wahab. Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation. Technical Report TR–95–22, Department of Computer Science, Old Dominion University, 1995.

[10] I. Stoica and H. Abdel-Wahab. A new approach to implement proportional share resource allocation. Technical Report TR–95–05, Department of Computer Science, Old Dominion University, 1995.

[11] I. Stoica, H. Abdel-Wahab, K. Jeffay, J. Gherke, G. Plaxton, and S. Baruah. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the Real-Time Systems Symposium*, pages 288–299, Washington, DC, December 1996.

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

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