End-to-End Scheduling in Real-Time Packet-Switched Networks

Ian R. Philp and Jane W.S. Liu
Real-Time Systems Laboratory
Department of Computer Science
University of Illinois at Urbana, Urbana, IL 61801
{philp, janeliu}@cs.uiuc.edu

Abstract

This paper presents the budgeted-weighted-round-robin (BWRR) algorithm for scheduling the transmission of hard-real-time messages in a packet-switched network. The BWRR algorithm provides a bound on the worst case delay of a message through the network. In addition, with the BWRR algorithm, the necessary buffer space at each switch is predictable and can be fixed at system configuration time. The per switch delay and required buffer space provided by the BWRR algorithm are significantly less than those provided by previous methods. The BWRR algorithm has a simple hardware implementation and admission test, and it does not require a global clock or any explicit flow control messages. It can achieve high utilization of the network under most practical scenarios and reacts predictably under transient overloads.

1 Introduction

Proper functioning of real-time systems requires not only that the correct results be produced but also that the results be produced by specified times called the deadlines. Applications in a distributed real-time system consist of communicating processes. One process may do some computation and then send a message over the network to another process which does some more computation and so on. In order to meet the end-to-end timing requirements of such processes, strict bounds must be placed not only on the completion times of the processes but also on the times for message transmissions.

The problem of scheduling real-time messages in packet-switched networks has been studied extensively in recent years. Some of the algorithms that have been proposed are: Delay Earliest-Due-Date (D-EDD) [3], Jitter Earliest-Due-Date (J-EDD) [9], Virtual Clock (VC) [12], Fair Queuing (FQ) [2], Rate-Controlled Static-Priority Queuing (RCSP) [10], Stop-and-Go (S&G) [4], and Hierarchical-Round-Robin (HRR) [5, 8]. Aras *et al.* [1] and Zhang and Keshav [11] provide overviews on real-time communication in multihop packet-switched networks. Among these algorithms, D-EDD, J-EDD, VC, FQ, and RCSP are priority-based, i.e., they explicitly assign priorities to packets, and the scheduler transmits the packet with the highest priority. In S&G and HRR, time is explicitly fragmented into fixed size frames. Each message stream is allotted some number of slots within each frame; thus, a minimum (and maximum) bandwidth is guaranteed for each stream. The advantage of the priority-based algorithms is that they are more flexible in the way they assign delay and bandwidth to each stream. Their disadvantage is that their implementation is more complicated.

In this paper, we present a modification of the weighted-round-robin (WRR) [2, 7, 6] algorithm, called the budgeted-weighted-round-robin (BWRR) algorithm, for scheduling real-time messages in a multihop packet-switched network. The WRR algorithm cycles through the message streams transmitting a limited number of packets from each stream, thus guaranteeing each stream a minimum fraction of the transmission bandwidth. The WRR algorithm allows the actual transmission rate to exceed the minimum rate which, as we shall see, increases jitter and the required buffer space. Like the WRR algorithm, the BWRR algorithm also ensures a minimum transmission rate, but it also ensures that the maximum transmission rate of each message stream does not exceed its allocated rate. In this way, the BWRR algorithm significantly reduces the jitter in message delivery and the buffer space requirement per message stream. This algorithm has the advantages of the WRR algorithm: it has a simple hardware implementation; the admission test for determining whether a new message stream is schedulable is simple; and it does not require a global clock or any explicit flow control messages. It can achieve high utilization of the network under most practical scenarios and reacts predictably under transient overloads. In addition, with the BWRR algorithm, the necessary buffer space is predictable and can be fixed at system configuration time.

The BWRR algorithm is more closely related to the

0-8186-7453-9/96 $05.00 © 1996 IEEE

frame-based algorithms, S&G and HRR, than the priority-based algorithms because of the way in which it explicitly assigns a bandwidth to the message streams. The BWRR algorithm has the same advantages and disadvantages as S&G and HRR compared with the priority-based algorithms: it has simpler implementation but is not as flexible in assigning delay and bandwidth to streams. Therefore, in this paper we compare the BWRR algorithm to the S&G and HRR algorithms. We shall show that the BWRR algorithm provides a significant improvement in per switch delay and required buffer space over the two frame-based algorithms.

Following this introduction, Section 2 presents the problem formulation. Section 3 introduces the WRR scheduling algorithm and discusses its timing properties and buffer requirements when applied to a multihop packet-switched network. In this section, we also discuss the S&G and HRR algorithms. Section 4 presents the BWRR algorithm and discusses its timing properties and buffer requirements. Section 5 discusses the network utilization achieved by BWRR. We conclude in Section 6 with a summary of our results.

2 Problem Formulation

We assume here that real-time messages are generated by periodic processes. A periodic process is characterized by three parameters: its period, its worst case execution time, and its deadline. Since real-time messages are generated by periodic processes, they too fit the periodic model. Thus, we characterize a periodic message stream M_i by the triple (c_i, p_i, d_i) where c_i is the length of the message (measured by the number of packets), p_i is its period, and d_i is its relative end-to-end deadline, i.e., the maximum allowable end-to-end delay. We refer to d_i simply as its deadline. Each message stream M_i consists of a periodic sequence of message instances, i.e., real messages. We call the j-th instance of message stream M_i $M_{i,j}$. When it is not necessary to distinguish between individual instances, we refer to both the message stream and its instances as M_i.

We take our basic time unit to be the time required to transmit one packet and call this time a *slot*. Each packet is transmitted in one slot. We are primarily interested in finding the achievable value of the end-to-end delay of each message stream when given the lengths and periods of all the message streams in the network. Therefore, we usually leave out the deadline in the tuple and simply refer to M_i by (c_i, p_i).

A message M_i originates at some source switch S_{src}. The message is broken up into c_i fixed size packets and is sent across a path of length L, measured in the number of hops, to its destination S_{dst} where the message is finally reassembled. Each hop is a one way point-to-point connection between two switches. Since each hop in the path of M_i is shared by many other message streams, the problem is how to schedule the message streams traversing each hop such that every message instance $M_{i,j}$ meets its end-to-end deadline d_i. The total end-to-end delay of a message is the sum of the queuing delays, transmission delays, and propagation delays along all L hops. Here, we only consider the queuing and transmission delays at each hop. A message stream M_i is admitted to the network if and only if it can be scheduled on each hop so that its end-to-end delay is no more than its deadline and its admission does not cause messages from other streams to miss their deadlines.

3 WRR Scheduling

According to the WRR scheduling algorithm [2, 7, 6], a weight w_i is assigned to each message stream M_i such that the sum of the w_i is less than or equal to some integer C which we call the scheduler *cycle time*. The scheduler transmits packets from the message streams in cyclic order. If message stream M_i has w_i or more packets ready, the scheduler transmits w_i packets from M_i before transmitting packets from M_{i+1}. If M_i has only k packets ready, where $k < w_i$, the scheduler transmits all k packets from M_i and then immediately transmits packets from M_{i+1}. Because $\sum_i w_i \leq C$, the scheduler offers at least w_i slots to stream M_i in *every* time interval of length C. Hence, according to the WRR algorithm, the actual length of each cycle is at most equal to C but can be less than C. A description of a WRR hardware scheduler can be found in [6].

By an appropriate assignment of the weights, the WRR algorithm can guarantee that each M_i is given at least c_i slots within every period p_i. When $C \leq p_i$, for all i, the following assignment of w_i satisfies this requirement:

$$w_i = \left\lceil c_i / \lfloor p_i / C \rfloor \right\rceil \tag{1}$$

Here, the number of complete cycles within a period p_i of M_i is $\lfloor p_i/C \rfloor$. Since w_i is the ceiling of the total number of packets generated per period divided by the number of complete cycles per period, M_i receives at least c_i slots within every time interval of length p_i.

When a new message stream which requires w_i slots per cycle attempts to enter the system, it is admitted if the sum of the weights of the existing message streams and w_i is no greater than C. If less than w_i slots per cycle are free, admission is denied.

We now state a theorem and a corollary that state upper bounds on the end-to-end delay incurred by a periodic message scheduled according to the WRR algorithm. Their proofs can be found in the appendix. Similar results can be found in [11] in the analysis of the FQ algorithm. However, our results are stated slightly differently, i.e., in terms of the periodic message model. We include them to make the

paper complete and also to provide a basis for comparison of the WRR and BWRR algorithm.

Theorem 1 *When a periodic message M_i with period p_i is scheduled according to the WRR algorithm on a path of L hops in a packet-switched network in which the cycle lengths are C at all hops and $C < p_i$, the worst case end-to-end delay of M_i is no longer than $\lceil c_i/w_i \rceil C + (L-1)C$.*

Corollary 1 *The worst case end-to-end delay of M_i when scheduled according to the WRR algorithm along L hops in a packet-switched network with equal values of C at all hops and $C < p_i$ is no longer than $p_i + (L-1)C$.*

Intuitively, Theorem 1 says that the first hop completes the transmission of all c_i packets by time $\lceil c_i/w_i \rceil C \leq p_i$ and each of the $L-1$ successive hops completes its transmission within C more slots.

The WRR algorithm provides a high degree of isolation for each message stream. Even if every other stream floods the system with packets, M_i still gets its allotted slots. Under a transient overload, the message or messages that behave badly will suffer longer response times while the messages that conform to their declared parameters will meet their contracted deadlines.

However, when messages are transmitted according to the WRR algorithm, the jitter in message delivery and the required buffer space may be large. We say that two message instances have zero jitter if their interdelivery time to the destination is equal to the period of the message stream. Jitter, then, is the absolute value of the difference of the interdelivery times and the period. From the analysis of work-conserving schedulers in [11], we can conclude that

Theorem 2 *The worst case jitter for a message transmitted on an L-hop packet-switched network according to the WRR algorithm is $p_i - c_i + (L-1)(C-1)$.*

In other words, the jitter introduced by the WRR algorithm grows with the number of hops and may cause problems for jitter sensitive applications. A closely related problem with the WRR algorithm is the excessive buffer usage which also grows with the number of hops as is stated by the following theorem [11].

Theorem 3 *The maximum buffer space required for a message stream M_i at hop L when scheduled according to the WRR algorithm is $(1 + \lceil C(L-1)/p_i \rceil)c_i$ packets.*

The HRR [5] and S&G [4] algorithms are frame-based schedulers which eliminate some of the problems of WRR. According to both these methods, time is explicitly fragmented into fixed size frames of length $F = C$. Within a frame, each message stream is allocated a fixed number of slots which cannot be exceeded. Thus, HRR and S&G are non-work-conserving algorithms. Even when there are no packets at a hop from a particular message stream, a newly arriving packet from that stream might not be transmitted until the end of next outgoing frame. Therefore, the worst case delay at each hop for packets scheduled according to the HRR and S&G algorithms is $2F$. The worst case buffer space requirement depends on whether the input and output frames are synchronized. If the input and output frames are not synchronized, $3w_i$ space per stream is required; if they are synchronized, $2w_i$ space is required. These results are summarized in Table 1 along with the results for the BWRR algorithm which we describe in the next section.

	packet delay	buffer space
HRR and S&G	$2C$	$3w_i$
Synchronized HRR and S&G	C	$2w_i$
BWRR	C	$2w_i$

Table 1. Packet delay and buffer requirements.

4 Budgeted-Weighted-Round-Robin Scheduling

We now describe an algorithm, called the budgeted-weighted-round-robin (BWRR) algorithm, that not only has most of the same advantages as the WRR algorithm but also reduces the jitter and required buffer space at the intermediate hops. BWRR achieves the same end-to-end delay as WRR, requires no clock synchronization, and requires only $2w_i$ buffer space at each hop. The BWRR algorithm is non-work-conserving and is based on a similar idea as the frame-based algorithms, i.e., a budget of slots is allocated to each message stream and the budget is replenished at minimum (and maximum) intervals. The maximum average output rate of a message stream from a hop scheduled according to the BWRR algorithm is the same as the output rate when scheduled according to the frame-based algorithms. There are, however, two important differences. First, BWRR does not replenish the budget at fixed times defined by a frame; hence, no synchronization of frames is required. Second, the BWRR scheduler cycles through the streams continuously, transmitting packets from a stream whenever that stream's budget is greater than zero. In the frame-based algorithms, if a stream misses its turn in a frame, its packets do not become eligible for transmission until the beginning of the next frame and may not be transmitted until the end of that next frame, i.e., $2F = 2C$ slots later. In the BWRR algorithm, as in the WRR algorithm, a group of w_i packets is always

transmitted within C slots. Because the BWRR algorithm transmits packets earlier than the frame-based algorithms and the input rate from the previous hop is the same as in the frame-based methods, the BWRR algorithm has lower buffer space requirements.

To describe the BWRR algorithm, we note that the WRR algorithm can be implemented by using a variable to keep track of the number of slots used up by the current message stream, i.e., the message stream being transmitted at the time. The scheduler sets the variable to w_i when it begins to transmit packets from M_i and continues to transmit from M_i as long as the variable is greater than zero [6]. To implement the BWRR algorithm, we also use such a variable, called bgt_i, for each message stream M_i which gives the budget of packets that M_i may transmit. As packets from M_i are transmitted, bgt_i is decremented. Packets from M_i are transmitted until either M_i has no packets to transmit or $bgt_i = 0$. When the scheduler finishes transmitting packets from M_i, it immediately begins transmitting packets from M_{i+1}. As we shall see, the budget is never greater than w_i, and therefore, the BWRR scheduler cycle time is never greater than C.

The BWRR algorithm is defined by the way it replenishes each of the bgt_i which are initially set to w_i. The pseudocode for the initialization, the packet arrival processing, and the scheduler processing is shown in Figure 1.

We associate two replenishment-time variables M_i.rplsh1 and M_i.rplsh2 with each message stream. M_i.rplsh1 always holds the closest replenishment time, i.e., if both variables are set, M_i.rplsh1.time $<$ M_i.rplsh2.time. Whenever a packet arrives, the packet count for M_i is incremented, and a new replenishment time is set if the arrived packet is the w_i-th packet in a packet group. The new replenishment time is set to either C slots after the arrival of the w_i-th packet or C slots after the pending replenishment time, whichever is later. This rule ensures that the budget replenishments are at least C slots apart. When the scheduler begins to transmit packets from stream M_i, it checks if the budget needs to be replenished and if so sets bgt_i to w_i.

The arrival and transmission pattern shown in Figure 2 illustrates the replenishment rule. In the figure, a downward arrow indicates the arrival of the w_i-th packet in a packet group, and an upward arrow indicates a budget replenishment. The arrival of the first group of w_i packets at a hop is labeled $a1$, the arrival of the second group $a2$, and so on. Similarly, the budget replenishments are labeled $r1$, $r2$, etc. The slots in which each packet group in a message instance of M_i is transmitted at each hop are labeled with the number of the packet group. We assume that the first group of w_i packets arrives at the first hop H_1 at $t = 0$, the second group arrives at $t = C$, the third at $t = 2C$, and so on. According to the BWRR scheduler, the first group of w_i packets is transmitted by time C at the first hop. The

```
1. Initialize

   For all i
      bgt_i = w_i;
      M_i.rplsh1.set = false;
      M_i.rplsh2.set = false;

2. Packet Reception

   M_i.pkt-cnt++;
   if (M_i.pkt-cnt = w_i)
      M_i.pkt-cnt = 0;
      if (M_i.rplsh1.set = false)
         /* No pending replenishment;
            set rplsh1 C slots later */
         M_i.rplsh1.set = true;
         M_i.rplsh1.time = clock + C;
      else /* rplsh1 is set;
            set rplsh2 C slots after rplsh1 */
         M_i.rplsh2.set = true;
         M_i.rplsh2.time = M_i.rplsh1.time + C;

3. When beginning to transmit packets from M_i

   if (M_i.rplsh1.set = true) and (M_i.rplsh1.time ≤ clock)
      bgt_i = w_i;
      /* rplsh1 is set again only if rplsh2 is set */
      M_i.rplsh1 = M_i.rplsh2;
      M_i.rplsh2.set = false;
```

Figure 1. Pseudocode for the BWRR scheduler.

budget is replenished C slots later at time C, so the second packet group is transmitted by time $2C$, and so on. At the second hop, after the arrival of the w_i-th packet in the first group at time $a1 < C$, M_i.rplsh1 is set to replenish the budget at time $r1 = a1 + C$, by which time the first group of packets is transmitted at the second hop. The second group of packets arrives at time $a2 < r1$, so M_i.rplsh2 is set to time $r2 = r1 + C$. Because $r2$ is C slots after $r1$ and the second group of packets is ready by time $r1$, the second group of packets is transmitted by time $r2$. When the third group of packets arrives, there is no pending replenishment time set, so M_i.rplsh1 is set to replenish the budget at time $r3 = a3 + C$, by which time the third group of packets is transmitted.

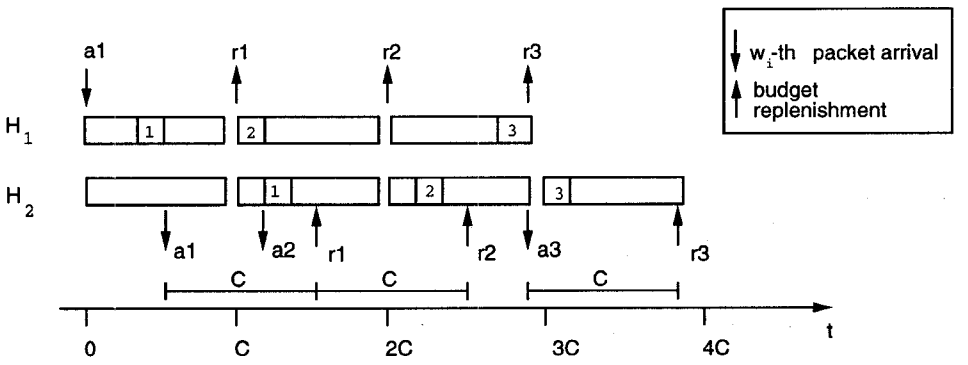

Figure 2. Replenishing the budget in BWRR.

The following theorem shows that the BWRR algorithm provides the same delay bound shown for the WRR algorithm in Theorem 1,

Theorem 4 *The BWRR algorithm provides the same worst case end-to-end delay as the WRR algorithm.*

Proof: In order to show that the delay bound proved in Theorem 1 for the WRR algorithm holds in general for the BWRR algorithm, we need to bound the times at which the budget is replenished. At H_1, the budget for the k-th packet group ($k > 1$) is replenished by time $(k-1)C$, so the k-th packet group is transmitted by time kC. At H_2, the first packet group arrives by time $a1 \leq C$ and is transmitted by time $a1 + C \leq 2C$. The budget is replenished by time $r1 = a1 + C \leq 2C$. The second packet group arrives at H_2 at time $a2 \leq 2C$. Since the budget is replenished by time $2C$, the second packet group is transmitted by time $3C$. The second budget replenishment occurs at time $r2 = a2 + C$ or $r2 = r1 + C$, whichever is later. Since both of these are less than $3C$ and the third packet group arrives at H_2 by time $3C$, the third packet group is transmitted at H_2 by time $4C$. Continuing this argument, we find that the replenishment time for the k-th group of packets at the l-th hop is always at or before $(l-1)C+(k-1)C$, so the BWRR scheduler always transmits the k-th group of packets by time $(l-1)C+kC$, and the end-to-end delay proved in Theorem 1 holds. □

Because the replenishment times are at least C slots apart, the number of packets that can be transmitted in any time interval is limited. Based on this property, the following theorem bounds the buffer space required by the BWRR algorithm.

Theorem 5 *The maximum buffer space required per message at each hop when scheduled according to the BWRR algorithm is c_i at the first hop and $2w_i$ at each successive hop.*

Proof. Since the application may send all c_i packets in a message to the first hop together, the first hop requires c_i buffer space. We view the c_i packets of message instance $M_{i,j}$ as $\lceil c_i/w_i \rceil$ groups of packets where each group has w_i packets. If packet group k is transmitted at hop l by time t and there are no packets at hop $l+1$, then at hop $l+1$ the latest time group k is transmitted is by time $t + C$. Group $k+1$ could be transmitted at hop l as early as time $t + w_i$, so $2w_i$ space is necessary at hop $l+1$. The earliest time at which group $k+2$ could be transmitted at hop l is $t+w_i+C$. Thus, group $k+2$ (and all successive groups) cannot arrive at hop $l+1$ before the transmission of group k completes, so $2w_i$ space suffices. □

Corollary 2 *If the packets that enter the first hop are controlled by the BWRR algorithm, only $2w_i$ buffer space is needed per message at the first hop.*

The fact that only $2w_i$ buffer space is required at each hop justifies the use of two replenishment variables in the pseudocode of the BWRR algorithm.

Corollary 3 *The total buffer space required at each hop is no more than $2C$ packets.*

Proof. Each message reserves w_i slots at a hop and requires $2w_i$ buffer space. Since the sum of the w_i is less than or equal to C, the maximum buffer space required at each hop is at most $2C$ packets. □

This last result implies that the amount of buffer space required for each hop can be determined at system design time based on the transmission rate and the scheduler cycle length C. When a message stream requests service, it only has to request a certain number of slots within a cycle; no additional checking is necessary to determine if there is enough buffer space.

Finally, we state the following theorem on the jitter of the BWRR algorithm.

Theorem 6 *The worst case jitter for a message scheduled on an L-hop packet-switched network according to the BWRR algorithm is no greater than* $C - w_i + (L-1)(C-1)$.

Proof. The completion time for a message instance submitted at time t at the first hop is never sooner than $t+(\lceil c_i/w_i \rceil - 1)C + w_i$, i.e., the cycle in which the last group of packets is transmitted begins at time $(\lceil c_i/w_i \rceil - 1)C$ and the packets are transmitted within w_i slots. This last group of w_i packets can be transmitted immediately at each of the remaining $L-1$ hops, giving the shortest end-to-end delay of $t + (\lceil c_i/w_i \rceil - 1)C + w_i + (L-1)$. The latest completion time of a message instance released at time $t + p_i$ is $t + p_i + \lceil c_i/w_i \rceil C + (L-1)C$. Therefore, the worst case jitter is $C - w_i + (L-1)(C-1)$. □

The BWRR algorithm reduces the jitter of a message to approximately one scheduler cycle per hop. This result should be compared with Theorem 2 which states the worst case jitter for the WRR algorithm. Although the jitter does increase with each hop, in practice the jitter can be controlled by appropriately selecting a value for C. For example, in the current telecommunications network a connection traverses at most 6 to 8 hops. If a constant bit rate video stream is being transmitted with a period of 33 ms., then by choosing $C < 4ms$ the jitter is limited to 32 ms which is less than the period of the message.

5 Performance

In this section, we examine, through simulation, the achievable utilization of the BWRR scheduling algorithm on a single link. We use the following notation:

- $U_{act,i} = c_i/p_i$ is the actual utilization of M_i.
- $U_{act} = \sum_i U_{act,i}$ is the actual utilization of all message streams assigned to a link.
- $U_{bwrr,i} = w_i/C$ is the assigned utilization of M_i.
- $U_{bwrr} = \sum_i U_{bwrr,i}$ is the assigned utilization of all message streams assigned to a link.

Since $U_{act,i} \leq U_{bwrr,i}$ for all i, $U_{act} \leq U_{bwrr}$. Hence, even when $U_{bwrr} = 1.0$ (i.e., all C slots are assigned), the actual utilization U_{act} of the link can be less than 1.0. The purpose of the simulation is to measure the actual achievable utilization U_{act}. In our simulation experiment, we varied the cycle length C and the length and period of the message streams as follows:

- C was simulated with values 100, 1,000, and 10,000 slots. With 53 byte ATM cells and a cycle time of 5 ms., these values of C correspond to transmission rates of 8.48, 84.8, and 848 Mbps respectively.
- For each value of C, a random set of messages was generated with $U_{bwrr} = 1.0$. Specifically, we did this by randomly generating values for p_i and c_i and assigning a weight w_i to this stream. This process was repeated until a stream was generated that required more slots than were free. When this happened, we computed the maximum value of c_i for the given value of p_i such that the stream required all the remaining free slots. The stream was then added to the link with this new smaller value of c_i. The value of U_{act} was used to measure the achievable utilization.
- The periods of the messages were chosen from three uniform distributions with the following ranges:
  - small periods: $[C, 10C]$
  - large periods: $[1,000C, 10,000C]$
  - wide range of periods: $[C, 10,000C]$
- The number of packets per message was chosen from a uniform distribution such that the utilization of each message stream $U_{act,i}$ fell into one of the three following ranges. The average value of $U_{act,i}$ determines the number of message streams assigned to the link.
  - small utilization: $[0.001, 0.01]$ ($\approx$ 180 streams)
  - large utilization: $[0.05, 0.10]$ ($\approx$ 13 streams)
  - wide range of utilization: $[0.001, 0.05]$ ($\approx$ 40 streams)

Tables 2, 3, and 4 show the results of the simulation. The first point to notice is that the achievable utilization increases as C increases. This is because of the finer granularity of bandwidth that is allocated with larger values of C. The maximum overallocation per message stream is 1 slot per cycle or $1/C$ overallocation of utilization. The effect of a small value of C shows up in the first column of Table 2 where the utilization per message stream is low; hence, the number of message streams is high. When the number of message streams is high, so is the total overallocation of utilization. Tables 3 and 4 limit the overallocation per stream to 0.1% and 0.01% respectively and thus attain a higher link utilization.

The other cause of overallocation of utilization arises due to the floor and ceiling in Equation 1. In the evaluation of both the ceiling and the floor functions, the average overallocation of utilization is minimized when the denominator is small relative to the numerator. This fact results in a tradeoff in choosing a value for C for a given set of message periods. In order to minimize the overallocation in the evaluation of the floor in Equation 1 $\lfloor p_i/C \rfloor$, C should be small relative to the message periods. However, when evaluating the ceiling $\lceil c_i/N \rceil$ where $N = \lfloor p_i/C \rfloor$ is the number of complete cycles in M_i's period, N should be small, and hence, C

$p_i \backslash U_{act,i}$	[0.001,0.01]	[0.05,0.10]	[0.001,0.05]
[C,10C]	0.558	0.834	0.777
[1,000C,10,000C]	0.549	0.939	0.838
[C,10,000C]	0.549	0.938	0.840

Table 2. U_{act} **with** $C = 100$.

$p_i \backslash U_{act,i}$	[0.001,0.01]	[0.05,0.10]	[0.001,0.05]
[C,10C]	0.819	0.862	0.856
[1,000C,10,000C]	0.916	0.994	0.981
[C,10,000C]	0.916	0.993	0.981

Table 3. U_{act} **with** $C = 1,000$.

should be large. The effect of a large value of C is shown in the first row of each table where the message periods are in the range $[C, 10C]$. However, even here the average utilization is above 85%. In the last two rows of Tables 3 and 4 where C is small compared with the message periods, the achievable utilization is over 90%.

In this section, we have only discussed the performance of the BWRR scheduler that uses one cycle length. The BWRR algorithm can easily be modified to include multiple cycle lengths just as the frame-based algorithms can include multiple frame lengths [4, 5]. What we have shown in this section is that high average utilization of the network is achieved over message streams with periods that differ by four or five orders of magnitude. Depending on the traffic mix, it appears that high utilization can be achieved with one or at most two different cycle lengths.

6 Conclusion

In this paper, we have addressed the problem of scheduling hard-real-time messages in a multihop packet-switched network. We investigated the use of the budgeted-weighted-round-robin (BWRR) scheduling algorithm. In particular, we showed that the BWRR scheduler can be used to guarantee end-to-end deadlines when used in a multihop packet-switched network. We saw that the BWRR algorithm pipelines the transmission of message packets and achieves an end-to-end deadline that increases by one scheduler cycle per hop. Furthermore, the BWRR scheduler does not require any global clock synchronization or any explicit flow control messages.

We also discussed the stability properties of the BWRR algorithm under transient overload conditions. We saw that the scheduler provides complete isolation for each message stream, i.e., bad behavior of one message will not affect any other messages.

The buffer space required per stream at each hop by the WRR algorithm increases as the scheduler cycle length C and the number of hops L grow. With the BWRR algorithm, the required buffer space does not depend on the number of hops traversed by a message stream and is limited to the maximum number of packets transmitted in two scheduler cycles. Previously proposed frame-based schedulers can achieve this same buffer requirement, but they require global clock synchronization to do so. Finally, we showed that the BWRR scheduler achieves high average utilization of the network over a large range of traffic characteristics.

Appendix

Proof of Theorem 1: The key to the proof lies in viewing the c_i packets produced every period not as one group of c_i packets but rather as $\lceil c_i/w_i \rceil$ groups of w_i packets. We prove by induction that the k-th group of packets is transmitted at the l-th hop by time $kC + (l-1)C$.

The induction is shown graphically in Figure 3. Each entry in the table corresponds to the worst case completion time of the k-th group at the l-th hop. The completion time of the last group at the last hop is the entry in the lower right corner of the table. For the base step of the proof, we first fill out (1) the first row and (2) the first column. (1) Assuming that the beginning of M_i's period is time 0, at the first hop the WRR scheduling algorithm guarantees that w_i packets are transmitted in every interval of time C. Therefore, the first group of w_i packets is transmitted by time C. This first group then arrives at the second hop, and is transmitted within C more slots, for a total of $2C$. In general, the first set of packets is transmitted at the l-th hop by time lC or $C + (l-1)C$. (2) At the first hop, the first group of packets is transmitted by time C, the second group

$p_i \backslash U_{act,i}$	[0.001,0.01]	[0.05,0.10]	[0.001,0.05]
[C,10C]	0.859	0.862	0.863
[1,000C,10,000C]	0.991	0.999	0.997
[C,10,000C]	0.991	0.998	0.997

Table 4. U_{act} **with** $C = 10,000$.

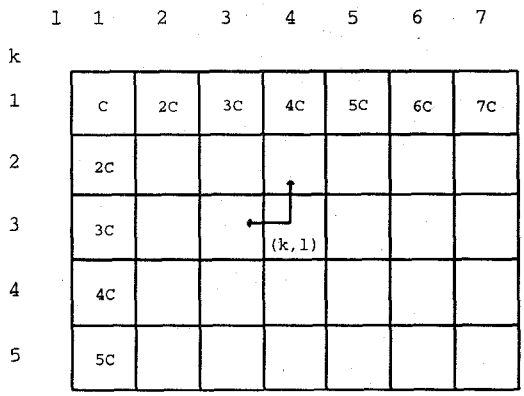

Figure 3. The completion time of the k-th group at the l-th hop depends on the completion time of the k-th group at the $(l\text{-}1)$-th hop and the completion time of the $(k\text{-}1)$-th group at the l-th hop.

by time $2C$, and the k-th group by time kC.

The inductive step proceeds by filling in the rest of the table row by row. The completion time of the (k,l)-th entry depends on the completion time of entry $(k-1,l)$ and the completion time of entry $(k,l-1)$. The inductive hypothesis states that entry $(k,l-1)$ is $kC+(l-2)C$ and that entry $(k-1,l)$ is $(k-1)C+(l-1)C$. Both of these are equal to $(k+l-2)C$. What this says is that the k-th group of packets is transmitted at the $(l-1)$-th hop and therefore is ready at the l-th hop at time $(k+l-2)C$. Also, the previous group of packets is transmitted at the l-th hop by the same time. Now, since the WRR scheduling algorithm guarantees that w_i packets are transmitted from the k-th group within time C, the k-th group is completed at the l-th hop by time $(k+l-2)C+C = kC+(l-1)C$. This completes the inductive part of the proof.

Overall, there are $k = \lceil c_i/w_i \rceil$ groups of packets and $l = L$ hops. Substituting these into $kC+(l-1)C$ we get $\lceil c_i/w_i \rceil C+(L-1)C$ for the worst case completion time at the last hop. □

Proof of Corollary 1: Since $\lceil c_i/w_i \rceil C \leq p_i$, we obtain $p_i+(L-1)C$ as the worst case completion time of the last group at the last hop. □

References

[1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Shulzrinne. Real-time communication in packet switched networks. *Proceedings of the IEEE*, 82(1):122–139, Jan. 1994.

[2] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Proceedings of ACM Sigcomm '89*, pages 1–12, 1989.

[3] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8:368–379, April 1990.

[4] S. Golestani. A framing strategy for congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7):1064–1077, Sept. 1991.

[5] C. R. Kalmanek, H. Kanakia, and S. Keshav. Rate controlled servers for very high speed networks. *Proceedings IEEE Global Telecommunications Conference*, pages 300.3.1–300.3.9, Dec. 1990.

[6] M. G. H. Katevenis. Fast switching and fair control of congested flow in broadband networks. *IEEE Journal on Selected Areas in Communications*, SAC-5(8):1315–1326, Oct. 1987.

[7] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *Proceedings of IEEE Infocom '93*, pages 521–530, March 1993.

[8] H. Saran, S. Keshav, and C. R. Kalmanek. A scheduling discipline and admission control policy for Xunet 2. *Multimedia Systems*, 2:118–128, 1994.

[9] D. C. Verma, H. Zhang, and D. Ferrari. Delay jitter control for real-time communication in a packet switching network. *Proceedings of Tricomm '91*, April 1991.

[10] H. Zhang and D. Ferrari. Rate controlled static-priority queueing. *Proceedings of IEEE Infocom '93*, pages 227–236, March 1993.

[11] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. *Proceedings of ACM Sigcomm '91*, pages 113–121, Sept. 1991.

[12] L. Zhang. Virtual Clock: A new traffic control algorithm for packet switching networks. *Proceedings ACM Sigcomm*, pages 19–29, 1990.