CPU Scheduling for Active Processing using Feedback Deficit Round Robin

Tilman Wolf and Dan Decasper

July 19, 1999

{wolf, dan}@arl.wustl.edu
Applied Research Laboratory
Washington University
St. Louis, Missouri 63130

Abstract

In active networks, much processing power is required for the execution of packet processing code that is carried in or referenced by active packets. In high-bandwidth environments the processor performing this active processing poses a bottleneck. We present a processor scheduling algorithm called Feedback Deficit Round Robin (FDRR) that reduces the overhead for fair scheduling and context switching by estimating the processing time for each active packet. Simulation results show that FDRR performs significantly better when compared to Round Robin scheduling.

1 Introduction

The increasing demand for flexibility in communications has introduced general purpose processing capabilities into the network layer. These active networks are capable of processing packets with the instruction code that is included in or referenced by each active packet. Active applications can be very demanding with respect to computational power, especially when the packet payload is modified (e.g. encryption [2]). In high bandwidth environments it becomes visible that there is a gap between transmission speed and available computational power [6]. While designs with multiple processors per port increase the total amount of processing power, the processing power is still the bottleneck for active processing. Efficient process scheduling is therefore an important issue to avoid the waste of precious cycles. The overhead introduced by context switching in particular is a dominant factor.

Context switching times of several thousand cycles are typical for switching between different processes [4]. Even light-weight threads require hundreds of cycles for a context switch. In relation to packet processing times in the range of hundrets to tens of thousands cycles, context switching is a considerable overhead.

Especially in active networks, where every packet can potentially carry or reference different instruction code for processing, the context for each packet can be different. This causes additional context switches, because the caches instructions from the previous packet cannot be reused. In this context, scheduling for active networks differs from traditional CPU scheduling.

Additionally, fair sharing of the processing resources and isolation between different flows is required to avoid that malicious or erroneous programs use the processor excessively.

In Section 2, related fair queuing and scheduling algorithms are presented on which Feedback Deficit Round Robin (FDRR) is based. Section 3 describes the FDRR algorithm. Section 4 is focused on different ways to estimate the processing time of an active packet, and Section 5 shows simulation results onn how FDRR performs.

2 Related Work

The scheduling that is required for the active network environment described above can be viewed from two sides. On one hand, packets are being processed, which points toward a fair queuing or packet scheduling scheme. On the other hand, each flow of packets can be viewed as a process that can be scheduled in the fashion in which CPU scheduling is performed.

2.1 Fair Queuing

A widely used fair queuing algorithm is Deficit Round Robin (DRR) [5]. DRR maintains a queue and a deficit counter for each flow of packets. In each round a quantum is added to the deficit of each non-empty queue. While the deficit exceeds the size of the next packet to be scheduled, the packet is processed/sent. This scheme is of $O(1)$ computational complexity for each packet, and the difference in fairness between flows is bounded by the maximum packet size. The simplicity of DRR allows an easy implementation in hard or software in high-performance routers. DRR and other fair queueing schemes, though, cannot easily be used for CPU scheduling. To achieve the fairness properties, it is necessary to know the amount of processing required for each packet beforehand. Except for very simple programs this cannot be computed reliably.

2.2 CPU Scheduling

A simple scheduling scheme that guarantees fair sharing of the processor is Round Robin. A synchronous timer interrupts the active process and makes it possible to give control to a different process. The decision which process gets to use the processor is made by the operating system, which maintains and adjusts the priorities for each process. In the simplest case each process uses the processor for one timeslice after which the next process is activated (Round Robin). This timeslicing is implemented in the 4.4BSD operating system [3]. In the context of active networks, this scheme of scheduling ignores the packet-by-packet nature of the data flows. The interruption of the process is not synchronized with termination of the packet processing for each packet, which incurs additional context switching overheads. This is significant when the scheduling is done on a very fine-grained basis.

3 Feedback Deficit Round Robin

We propose a scheduling scheme, Feedback Deficit Round Robin (FDRR), that gets rid of the shortcomings of fair queueing and timeslicing when used for active processing scheduling. The outline of FDRR is shown in Figure 1.

For each queue, a deficit counter and an estimate is maintained. The deficit represents the amount of processing that this queue can use. The estimate represents the amount of processing that is expected fot the next packet of this queue. The DRR scheduler forwards packets of a queue to the processor as long as the deficit is larger than the estimate of the next packet. With each packet, a timer is started that interrupts the processor in case a packet uses excessive processing. When the processing is finished or terminated, the actual processing time is used to adjust the deficit, as well as the estimate that is used for the next packet.

3.1 Algorithm

The pseudo code for the FDRR algorithm is shown in Figure 2. There are n queues. Each queue has initially no deficit and the estimated processing time is set to a default. How to choose the estimated time is explained in Section 4. For simplicity, we assume that all queues are always backlogged. The handling of empty queues is described in [5]. To enqueue a packet, it is simply appended to the queue that is determined by the flow classification.

The main loop of the dequeueing process adds the quantum to the deficit of the current queue. While the deficit is positive (i.e. the flow can use more processor time), the deficit and the estimated processing time for the next packet is compared. If the estimate is less than the deficit, the packet is processed by `process()`. In case the current packet was previously interrupted, the old state is restored. At the same time a timer is set to the deficit and started. There are two possibilities how the processing ends:

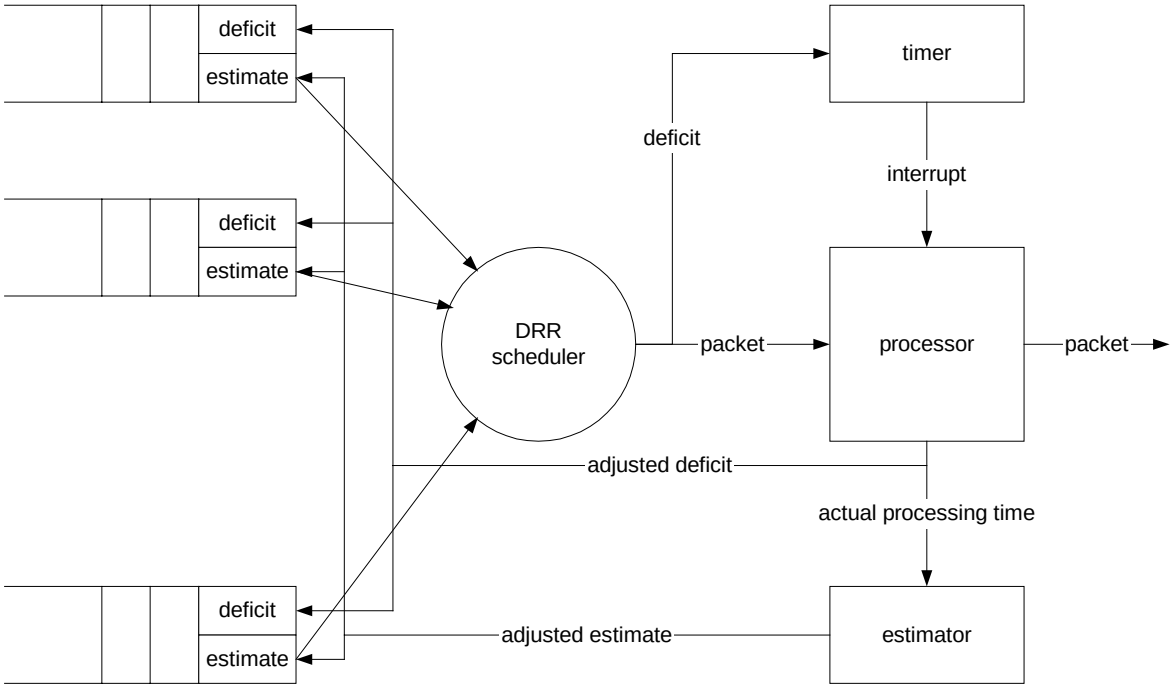

Figure 1: Feedback Deficit Round Robin.

- If the timer expires, then the packet used more time than it was permitted, and the processing is interrupted. In this case the state of the processing is saved and the packet is pushed back into the head of the queue. In the next round, the processing can then continue with that packet.
- If the processing terminates before the timer expires, the packet is sent on and the next packet in the queue is considered.

In both cases the processing function returns the actual time that was used by the packet. This amount is subtracted from the deficit. The actual processing time is also used by the estimator, `adjust_estimate()`, to adjust the estimation for the queue. If there is more deficit for computation time, then the next packet is considered for processing, otherwise the next queue is considered.

3.2 Correctness

It can be shown that each queue receives only its fair share of processing time. The correctness argument for FDRR is based on correctness of DRR, which has been proven to be a fair queueing scheme [5]. Again only backlogged queues are considered. The fairness of DRR is based on the following observations:

- The deficit counter is increased only once per round by the alloted quantum.
- No queue can receive more of the ressource than the deficit counter indicates.
- If a queue does not make use of all its share in a round, the amount is carried over to the next round in the deficit counter.

In FDRR these properties still hold true:

- The deficit counter is increased the same way as in DRR.

Initialization:

```
for (i = 0; i < n; i = i + 1)
  deficit_i = 0;
  estimate_i = default_estimate;
end for;
```

Enqueuing of packet p:

```
i = flow_classification(p);
if (not_full(queue_i)) then
  enqueue_at_tail(p);
else
  drop(p);
end if;
```

Dequeuing:

```
while (true) do
  i = next_queue();
  deficit_i = deficit_i + quantum_i;
  while (deficit_i > 0) do
    if (deficit_i ≥estimate_i) then
      start_timer(deficit_i);
      p = head(queue_i);
      if (is_interrupted_packet(p)) then
        restore_state(i);
      end if;
      start_timer(deficit_i);
      actual_time = process(p);
      if (process_interrupted) then
        save_state(i);
        enqueue_at_head(p, queue_i);
      end if;
      deficit_i = deficit_i - actual_time
      estimate_i = adjust_estimate(estimate_i, actual_time);
    else
      break;
    end if;
  end while;
  if (empty(queue_i)) then
    deficit_i = 0;
  end if;
end while;
```

Figure 2: FDRR pseudo code. There are n queues numbered 0 to $n-1$. For each queue i, a deficit counter, deficit_i, and an estimate of the expected processing time, estimate_i, is maintained. All queues are assumed to be backlogged.

- No queue receives more processor time than the deficit counter indicates. If the processing is interrupted by the timer, then the queue used its whole deficit and has to wait for the next round to receive more. If processing terminates earlier, the deficit was not exceeded either.
- The deficit is charged only for the actual time that the processor was used. In particular, the charging is independent from the possibly incorrect estimate.

These points assure that the fairness properies of DRR also hold true for FDRR.

4 Estimation of Processing Time

Estimating the correct amount of processing that is used by each packet is an important part of the FDRR scheme. The results in Section 5 show that significant performance gains can only be achieved if the estimate is within 50% of the actual processing time. It should be noted, though, that a wrong estimate only decreases the performance of FDRR, but does not cause the algorithm to produce incorrect results.

The simplest estimation scheme is to measure the actual computation time offline, and include this value in all packets. The FDRR scheduler can use this value then for the estimation. This scheme has some drawbacks, though. The execution time of a program is dependent of the data and also dependent on the particular machine where it is executed. Different cache sizes, for example, can cause a program to take different amount of times, although the same sequence of instructions is executed. Additionally, a protocol is required to include the estimates in the packets, which is a considerable overhead.

To avoind these problems we focus on estimation schemes that use local results to predict the next packet's execution time. We identified three basic posibilities to perform this local estimation:

- *constant*
  The constant estimate is the simplest estimator. The estimated computation time for queue i in round n, $\text{estimate}_{i,\,n}$, is always the same for all packets. If the queues correspond to different traffic classes, this information can be used to select the constant.

  $$\text{estimate}_{i,\,n} = \text{estimate}_{i,\,n-1} = const.$$

- *exponential average*
  The exponential average is a common scheme for an adaptive estimation that combines the most current execution time, actual_n, with the previous results. It is defined as:

  $$\text{estimate}_{i,\,n} = \alpha \cdot \text{actual}_{i,\,n} + (1-\alpha) \cdot \text{estimate}_{i,\,n-1}.$$

  The parameter α specifies how much of the previous history is preserved. This scheme is used in many practical applications, e.g. TCP round-trip delay estimation.

- *packet size dependent estimate*
  While the exponential average works well in practice, it ignores the size of the packet that is going to be processed. This information is used in the third estimation scheme. The packet size dependent estimate is defines as:

  $$\text{estimate}_{i,\,n} = f_n(\text{size}(\text{p}_n)),$$

  where the function f_n maps the packet size p_n to a processing time. The function f_n is adapted by the estimator E in the following way:

  $$f_n = E(f_{n-1},\, \text{actual}_{i,\,n}).$$

  The estimator E maps a packet size dependent estimation function to a new estimation function under consideration of the actual processing time. For the estimation, any function can be used but polynomial functions seem to be most promising, especially, since a polynomial of order o can be represented with $o+1$ variables. Depending on how precisely an estimation is required, higher or lower order polynomials can be used.

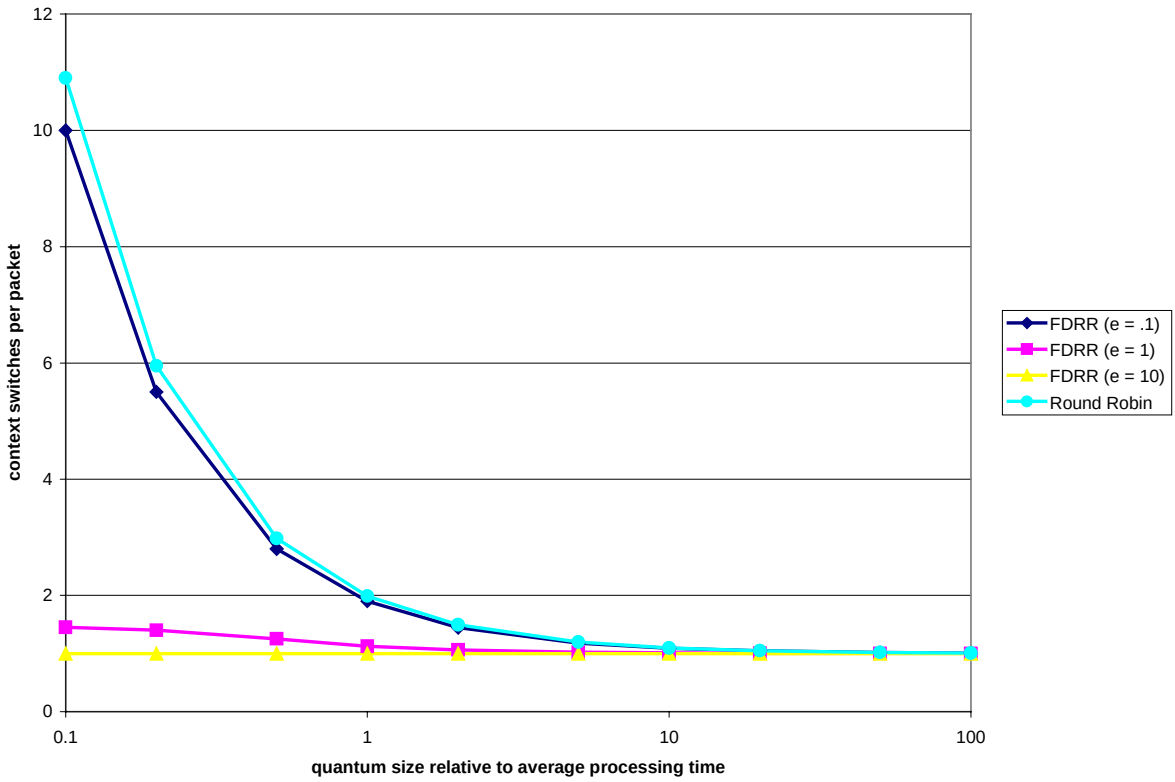

Figure 3: Number of context switches incurred by scheduling algorithm. FDRR is simulated using three different constant estimates, e, where the estimate is .1, 1, or 10 times the actual average processing time.

5 Simulation Results

As explained in the introduction, the goal of FDRR is to minimize the number of context switches while providing fairness. We assume for our measurements that each active packet can carry different code for its processing. This means that a 'context switch' happens every time the processing of a new packet is started, because the instruction cache will not have the instructions cached, and state information from earlier packets has to be made available.

To compare the performance of FDRR with the Round Robin scheduling, we simulated both algorithms over a range of parameters. The number of context switches per packet are shown in Figure 3. The quantum / timeslice ranges from .1 times to 100 times the average processing time of a packet. FDRR is simulated using three different estimates (.1, 1, or 10 times the average actual processing time).

Round Robin causes most context switches for any quantum size. If FDRR extremely underestimates the processing ($e = .1$), it performs similar to Round Robin. If the estimation is closer to the actual processing time ($e = 1$), FDRR incurs significantly fewer context switches for quantum sizes in the range of the actual processing time. The number of context switches for FDRR are at most 1.5, while for RR they go as high as 11. For larger quantum sizes, FDRR and Round Robin perform similarly, since many packets are being processed in the same round, and differences in scheduling disappear. FDRR causes least context switches for large overestimations ($e = 10$). In this case, it practically never happens that a packet needs more processing time than is estimated. Therefore, the packet is always processed to completion, and no interrupts or context swiches due to scheduling occur. This case comes closest to the ideal number of context switches per packet of 1.

Although FDRR with $e = 10$ seems to perform best, there is the issue of delay that has to be considered. While in Round Robin a packet is processed immediately when it reaches the head of the queue, in FDRR the deficit is accumulated until it reaches the estimated processing time. This accumulation helps to prevent the additional context switched, but it also delays the packet. This is particularly significant when the estimate is

larger than the actual processing. Thus, FDRR with a good estimation ($e = 1$) has the best overall performance.

6 Summary

This paper presents Feedback Deficit Round Robin, a processor scheduling algorithm that is based on a fair queuing scheme. The algorithm is shown to provide fair sharing of the processor, while reducing the number of context switches that are caused by scheduling and processing active packets. Simulation results indicate that FDRR performs significantly better for quantum sizes in the order of the processing time when the actual processing time is predicted well. For larger quantum sizes or incorrect processing time estimates, FDRR performs similar to Round Robin. To achieve correct processing time estimations, three schemes of different complexity are proposed.

To further show the usefulness of FDRR, we plan to implement the different estimation schemes and measure their adaptability to real traffic patterns. We will also implement FDRR in the Active Network Node [1] to test it in a real-world setting.

References

[1] Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B. [1999]. "A Scalable, High Performance Active Network Node," IEEE Network, 13:1.

[2] Franklin, M., Wolf, T. [1999]. "CommBench - A Telecommunications Benchmark for Network Processor Design," submitted to HPCA-6, IEEE, Toulouse, France.

[3] McKusick, M., Bostic, K., Karels, M., Quarterman, J. [1996]. *The Design and Implmanetation of the 4.4 BSD Operating System*, Addison-Wesley, Reading, Mass.

[4] Padmanabhan, V., Roselli, D. [1994]. "The Real Cost of Context Switching," *http://http.cs.berkeley.edu/~padmanab/class_projects/cs252.ps.*

[5] Shreedhar, M., Varghese, G. [1995]. "Efficient Fair Queuing using Deficit Round Robin," *Proc. of SIGCOMM 95*, ACM, Cambridge, Mass.

[6] Smith, J. [1999]. "Selected Challenges in Computer Networking," *IEEE Computer*, 32:1, 40-42.