# A Cell Burst Scheduling for ATM Networking Part II: Implementation

C. Tang, A. T. Chronopoulos, Senior Member, IEEE
Computer Science Department
Wayne State University
email:ctang, chronos@cs.wayne.edu

E. Yaprak, Member, IEEE
College of Engineering
Wayne State University
yaprak@et1.eng.wayne.edu

## Abstract

*The scheduling scheme of a switch affects the delay, throughput and fairness of a network and thus has a great impact on the quality of service (QoS). In Part I, we present a theoretical analysis of a burst scheduling for ATM switches and proved QoS guarantees on throughput and fairness of the applications. Here, we use simulation to demonstrate the superiority of the burst based weighted fair queueing over the non-burst version. Our simulation study is based on backbone and access subnetworks, which are common in the real world.*

**Key Words** *: Burst based weighted fair queueing, Implementation, Simulation.*

## 1 Introduction

As the infrastructure for future digital communication, ATM networks can accommodate various kinds of applications, and applications may have different traffic characteristics. The order in which application sessions will be offered service is very important. An ATM switch needs a well-designed scheduling algorithm to achieve this instead of First-Come-First-Served (FIFO) queuing. The scheduling algorithm has a great impact on the delay (especially on the queuing delay), the delay variation and the throughput of an application. There are many articles which present the progress on this topic, i.e. [1], [2], [4], [5], [6] and [**?**]. In the next sections, we present an efficient implementation for the algorithms of our article in Part I.

## 2 An efficient implementation of BBWFQ

In this implementation, we build five procedures for the main loop: *cell_enqueue*, *compute_start_time*, *compute_finish_time*, *virtual_time_function*, *burst_departure*. *Cell_enqueue* determines the burst boundary upon the cell arrival and cell interval arrival time. The actual data unit in the queue is a variable length burst. The *compute_start_time* computes the virtual start time of a burst instead of a cell. The *compute_finish_time* obtains the virtual finish time of the enqueued bursts. The *virtual_time_function* keeps track of the virtual time of the server upon the cell arrival and gives a server-wide consistent virtual time to all other procedures. The *burst_departure* is in charge of the scheduling of the burst.

The scheduling pseudocode is as follows:

```
init(); /* Initialize the global constants and global variables*/
loop
   if( cell arrival)
      cell_enqueue();
   if( system backlog isn't empty )
      burst_scheduling();
```

Following are the procedures used in the main loop:

```
cell_enqueue()
{
the_vci=VCI in the cell header arrived;
the_vpi=VPI in the cell header arrived;
queue_index=get_queue_index(the_vci, the_vpi);
arrival_time=get_current_physical_time();
if(queue is empty)
   previous_cell_arrival_time[queue_index]
   =arrival_time;
if(system is idle)
   burst_finish_time[queue_index]
      =virtual_time_function(queue_index)+
      1.0/bandwidth_portion[queue_index];
/* The bandwidth_portion is a global array which stores
the preassigned portion of each session and it is initialized
in the procedure init() */
if(arrival_time-previous_cell_arrival_time[queue_index]>
   BURST_BOUNDARY_THRESHOLD or
   burst_size[queue_index]≥
   burst_max_size[queue_index] or
   system backlog is empty)
/* The BURST_BOUNDARY_THRESHOLD is a global
```

```
constant */
    {
    the_start_time=compute_start_time(queue_index,
        burst_arrival_time[queue_index],
        burst_finish_time[queue_index]);
    the_finish_time=compute_finish_time(queue_index,
        burst_size[queue_index], the_start_time);
    burst_finish_time[queue_index]=the_finish_time;
    #ifdef PREEMPTIVE
    if(the_finish_time<
        get_burst_finish_time(current_served_session))
        burst_departure();
    #endif
    set_burst_finish_time(queue_index, the_finish_time);
    burst_arrival_time[queue_index]=arrival_time;
    } else {
        burst_size[queue_index]=burst_size[queue_index]+1;
    previous_cell_arrival_time[queue_index]=arrival_time;
    enqueue_burst(queue_index);
    }

    compute_start_time(queue_index,
        arrival_time, previous_finish_time)
    {
    the_virtual_time=virtual_time_function(queue_index,
arrival_time);
    if ( previous_finish_time > the_virtual_time)
        start_time=previous_finish_time;
    else
        start_time=virtual_time;
    return start_time;
    }

    compute_finish_time(queue_index, burst_size,
start_time)
    {
    return start_time+burst_size *CELL_SLOT/
        bandwidth_portion[queue_index];
    }
    /* CELL_SLOT is a global constant which equals to
the time spent for processing a cell and it depends on the
out-link capacity and the processor capacity*/

    virtual_time_function(queue_index, burst_arrival_time)
    {
    total_active_portion=get_total_bandwidth_portion();
    the_physical_time=get_current_physical_time();
    the_virtual_time=virtual_time+
        (the_physical_time-burst_arrival_time)/
        total_active_portion;
    virtual_time=the_virtual_time; /* virtual_time is a global
variable*/
    return the_virtual_time;
    }

    burst_departure()
    {
    for(index=0 to active_session_number)
    {
        queue_index=get_queue_index(index);
        burst_at_header[i]=get_burst(queue_index);
    }
    burst_index=get_least_finish_time(burst_at_header,
        active_session_number);
    current_served_session
=get_queue_index_from_burst(burst_index);
    send_burst(burst_index, current_served_session);
    }
```

## 3 Simulation Study

In this section, we simulate extensively the algorithms and test the related parameters. Our simulation model is based on a two-level ATM network environment consisting of backbone switches and access switches. Our simulation package is OPNET modeler 3.0.B from MIL 3, Inc. It has a prominent advantage over other simulation packages, for it supplies adequate in-built C functions to support accurate algorithm implementation using discrete-event mechanism and it also supplies many flexible analysis tools [3]. Towards building a model representing a real-world network, OPNET allows a model specification, complete with network, node, process and parameter definitions to capture the characteristics of a modeled system's behavior at different modeling hierarchies. Geographically distributed sites are referred to as nodes or subnetworks. These nodes are connected through point-to-point links, bus links, or radio links, in the OPNET network editor.

A simulation is executed by taking the simulation program and a set of data files representing the model's parameters to dynamically model the behavior of the actual system. Pre-defined statistics of interest can be collected on sample simulation runs by using the Probe Editor to specify which built-in statistics should be recorded by the Simulation Kernel. Each simulation run can be viewed as an experiment on a certain group of parameters of the system. It exhibits a repeatable behavior each time without changing the execution environment. This property is useful for isolating problems and analyzing or demonstrating interesting behaviour. However, for simulating stochastic elements (for example, a packet generator module) different seed (one of the parameters) values should be used to produce distinct sequences so that each particular simulation run can be thought of as representing one possible scenario of events for the modeled system. Statistics of interest can be collected as output vector files during the sim-

ulation runs. Our simulation plots are generated this way. The ATM switch model design is shown in Figure 1, where: **AAL** = ATM Adaptation Layer, **MGMT** = traffic management, **ATM** = ATM Layer, **FRM BKB** = From Backbone Switch, **To BKB** = To Backbone Switch, **To LOC** = To Local Switch, **Switch** = ATM Switch Fabric, **TRANS** = ATM Cell Transmission. ATM layers are **Sessions**(Application Layer), **AAL**, **MGMT/ATM**, **TRANS/SWITCH**(Physical Layer). Our scheduling server is part of **AAL**.

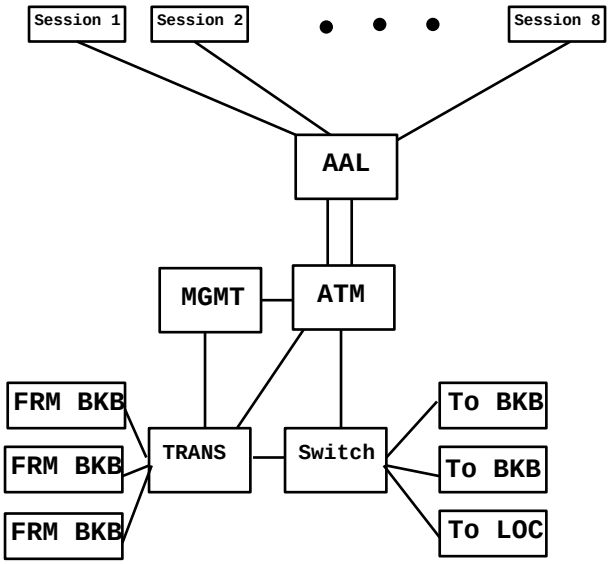


**Figure 1: Switch Model**

The network topology for this simulation is shown in Figure 2. Table 1 gives the source traffic parameters in our experiment with eight sessions, Note: $p0$ = *session number,* $p1$ = *bandwidth portion,* $p2$ = *packet size,* $p3$ = *leaky-bucket burst size,* $p4$ = *leaky-bucket maximum rate,* $p5$ = *leaky-bucket token generation rate,* $p6$ = *source interarrival time,* $p7$ = *source start time and* $p8$ = *source end time.*

In the figures (generated by OPNET), the ordinate axis unit is the *number of cells* for throughput, or for bucket-depth and *number of cell slots* for delay. The abscissa axis unit is the *simulation time* measured in *cell slots* (not in seconds), the number after the legend in each figure patch is the session number.

### 3.1 Performance comparision of the cell PGPS and BBWFQ

In this experiment, we compare our burst version BBWFQ with the cell version. Through the simulation, we find that the performances on the queue delay, session throughput and session backlog are almost the same. This means that under the new algorithm, the buffer and source traffic Usage Parameter Control (UPC) parameters need not change and we can still obtain the required performance. Further work on the detail correlation between these algorithms (such as the cell loss) is needed. Here we only focus on the performance indices mentioned above.

| p0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| p1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.5 | 0.5 | 1.0 |
| p2 | 8988 | 1581 | 1581 | 8988 | 1240 | 1240 | 2400 | 2400 |
| p3 | 50 | 80 | 100 | 100 | 100 | 200 | 100 | 200 |
| p4 | 2000 | 2000 | 2000 | 2000 | 4000 | 4000 | 5000 | 5000 |
| p5 | 50 | 10 | 30 | 20 | 50 | 10 | 10 | 10 |
| p6 | 200 | 200 | 200 | 200 | 100 | 100 | 100 | 100 |
| p7 | 100 | 0 | 200 | 0 | 100 | 100 | 20 | 0 |
| p8 | 2000 | 1800 | 1700 | 1500 | 2000 | 2000 | 1800 | 1700 |

**Table 1: The source traffic parameter**

Now, we test the performance of the two algorithms under the same source traffic pattern. The source traffic arrivals measured in cells are presented in Figure 3. The curves presenting performance indices using BBWFQ under the prementioned traffic arrival on session backlog, session throughput and delay are shown in Figure 4-6. Then, we present the experiment results for cell PGPS under the same traffic pattern. The Figures 7-9 show the session backlog, the session throughput and the cell delay.

Through these comparisons, we find the backlogs variation in the BBWFQ servers is relatively larger than that in the cell-PGPS server. It's reasonable because we schedule the cells in burst (group of cells in one time from one session and not one cell one time). As for the delay, the BBWFQ server has an overall smaller maximum delay than that of the cell-PGPS server. The throughputs of the real-time sessions are very consistent in a BBWFQ server as we showed above. In contrast, the throughputs in a cell PGPS server vary greatly, e.g. this can't offer a smooth playback of the multimedia applications.

### 3.2 Performance comparision between the non-preemptive and preemptive schemes

There are several advantages of the preemptive scheme. For example, (a) the average queue delay is shorter; (b) there is no queue indefinitely waiting for service, i.e. there is no starvation under the scheme.

We next present the backlog and delay of the preemptive scheme. The traffic arrivals, measured in cells, are in

Figure 10. The session backlogs in the server is given in Figure 11. In this figure, we find the backlogs in the server with preemptive scheme have a large variation but the absolute backlog is small. The reason for this is that we set the packet length of the real-time session less than that of the nonreal-time session and make the real-time bursts preempt the currently served nonreal-time session once their packets arrive. The bandwidth for those sessions is greater than that of nonreal-time sessions, so the computed virtual finish-time is smaller than that of the packets from nonreal-time sessions. This is why the preemption occurs.

We present cell delay in Figure 12. We note that in the figure, the delays on real-time sessions are under control compared to the delay of nonreal-time sessions varies widely.

## 4 Conclusion

In this paper, we implement the BBWFQ, which is highly efficient in terms of the computation complexity and the major QoS indices. The simulation study demonstrates the design advantages of the BBWFQ over the standard schemes.

**Acknowledgement:** Some comments of anonymous referees, which helped improve the presentation of some parts of the paper are gratefully acknowledged. This work was supported in part by NSF Grant ASC-9634775.

## References

[1] S.J., Golestani, "Congestion-free transmission of real-time traffic in packet networks", *proc. IEEE Infocom 90, June 1990*.

[2] P. Goyal, H. M. Vin, Haichen Cheng, "Start-time Fair Queueing: A scheduling algorithm for integrated services packet switching networks", *SIGCOMM'96 8/96.*

[3] OPNET Modeler: Modeling and Simulation kernel, Mil 3, Inc. 1996.

[4] 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, vol. 1, No. 3, June 1993.*

[5] D. Saha, S. Makherjec, S.H Tripahi "Carry-over round robin: A simple cell scheduling mechanism for ATM networks", *In Proc. IEEE Infocom'96, 1996.*

[6] Hui Zhang et al. "$WF^2Q$: Worst-case fair weighed fair queuing", *In Proc. IEEE Infocom'96, March 1996.*

**Figure 2. Network topology**

**Figure 3. Traffic arrival pattern**

**Figure 4. Session backlog of BBWFQ**

**Figure 5. Session throughput of BBWFQ**

**Figure 6. Session delay of BBWFQ**

**Figure 7. Session backlog of cell PGPS**

**Figure 8. Session throughput of cell PGPS**

**Figure 9. Session delay of cell PGPS**

**Figure 10. Traffic arrival patterns**

**Figure 11. Session backlog of preemptive scheme**

**Figure 12. Session delay of preemptive scheme**