Cell Scheduling to support CBR traffic

Naeem Ahmed Khan

31 December 1994

1 Introduction

The purpose of this document is to describe an attempt made to give guarantee to the Constant Bit Rate (CBR) traffic cells passing through a Fairisle Switch. Different types of traffic cells (e.g audio, video, file, etc) in an ATM network have different time constrains associated with them. For example, for a digital audio one does not want the delaying of cells, though some loss of cells is acceptable, whereas in case of a file transfer one does not want to loose any cell, but a reasonable delay is acceptable. Therefore when the cells of different traffic types arrive at a switch-port, there should be some kind of mechanism which keeps in consideration the timing requirements of the different types of traffic cells and forward cells accordingly. Therefore cell scheduling is required at an ATM switch ports. In a switch, another source of delay may the contention in a switch-fabric, so cells should be scheduled in such a way that no contention in the switch fabric should be experienced by the CBR traffic cells. DEC's AN2 ATM switch uses a scheduling mechanism which is used to reserve bandwidth for CBR connections. Bandwidth allocations are made on the basis of *frames* that consist of a fixed number of *slots*, where a slot is the time required to transmit one cell. The cells arriving at the input are buffered and CBR traffic is handled by building an explicit schedule of input-output pairings for each time slot in a frame. The frame schedule is constructed to accommodate the guaranteed traffic through the switch.

2 Handling of cells in a Fairisle Switch

The input ports of a Fairisle switch work independently (in the sense that they forward the cells to the fabric on their own without considering which output port other input ports are sending to in a particular time slot) [1]. This may cause contention and so cells are delayed, particularly having severe effect on CBR traffic.

The cells arriving at the input port of the switch are buffered. When a cell arrives, an interrupt is signalled to the ARM processor by the hardware and a fast interrupt routine (fiq5) is invoked. This interrupt routine places the cell which has just arrived in an appropriate queue. There are three queues in which cells are placed; (i) the *forwarding-queue*, which contains the cells to be forwarded to some other switch or end system (ii) the *arriving-queue*, containing the cells for the Wanda kernel and (iii) the *trash-queue*, which contains corrupted cells (which are discarded). There is another queue, the *leaving-queue*, which contains the cells generated by the Wanda kernel which are to be forwarded. The cells which are to be forwarded (from the *forwarding-queue* and *leaving-queue*) are placed in a *transmit-queue* by the fiq5 interrupt routine. The hardware picks the cells from the *transmit-queue* and sends them into the fabric. A cell sent into fabric from the *transmit-queue* by the hardware is either transmitted successfully or gets nacked because a cell, in the same time slot, from some other port was able to make its way to the same output port to which it was going and the transmission module on this port stops sending cells. The fiq5 routine gets invoked in case of a nack condition and deals with the nacked cell appropriately and restarts the transmission module. Before raising a nacked condition, the hardware tries to send this cell in seven consecutive time slots (7 retries). If hardware is unable to transmit the cell successfully in 7 retries, then software tries to send once again (with 7 hardware retries). If still not successful, the cell is thrown away by the software.

Note that the *forwarding-queue* contains cells in the order in which they arrived at the port and may have a mixture of constant bit rate (CBR) and variable bit rate (VBR) traffic cells. Consider a situation when VBR cells are queued up in front of CBR cells and furthermore what would happen if contention occurs for these VBR cells. The obvious result will be the delaying of CBR cells which we want to avoid.

3 Proposed scheme for scheduling of cells

To avoid contention, one needs to force the port of a switch to behave in such a way that it should not send a cell to an output port when some other input port is about to send a cell to the same port in the same time slot. This can be achieved by keeping a *schedule* of time slots at each input port. Each entry in the *schedule* should indicate the port number to which to send a cell in this time slot; no other port's *schedule* entry for this time slot should indicate to send a cell to that output port. Now more queues are required for the cells to be forwarded, if one wants to keep-up with the *schedule*. Therefore for scheduling the cells at a port one needs to do the following:

1. Maintain a *schedule* of the time slots.
2. Maintain the queues.
3. Implement a mechanism for negotiation between the ports for time slots.
4. Synchronise all the ports of a switch.

Obviously this requires modifications to the fiq5 routine so that it keeps in consideration the *schedule* and queues.

4 Maintaining the *schedule*

We conceptually consider a *schedule* consisting of n time slots. Reserved time slots contain the port number of the port to which cell should be sent in the slot.The *schedule* should be repeated on a periodic basis. All the CBR traffic requests for connection through the switch (or from the switch) must be made in terms of x time slots per *schedule* (i.e. this tells how much bandwidth is required by the connection). The input ports then negotiate with each other and if there are x time slots available out of total of n at this input port, they would be reserved for forwarding the cells for this connection. In case of a disconnect request, the slots reserved for that connection would be freed. No time-slots are marked reserved for VBR connection requests and any free slot can be used for forwarding the VBR cells. The reservation of time slots is done in such a way that there is no contention. As an example, reservation of the time slots at the input ports is shown below. This is a snap-shot of a 4x4 Fairisle switch schedules at input ports. The time slots are from left to right in the *schedule*. Each line represents the input slots in the *schedule* at the mentioned port (commas are just to show separation of input slots). One can see that there is no conflict. The *schedule* size is a configurable feature. A larger frame size allows for finer granularity in bandwidth allocation and a smaller frame yields lower latency. The most likely use of CBR guarantees will be to bound latency for multi-media flows.

```
               <----------------- Input time slots --------------->
Input-port-00  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  , 3, 2, 3, 2,  ,  ,
Input-port-01  ,  , 1, 3,  ,  , 1,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,
Input-port-02  , 1, 0,  , 1, 1, 0, 0,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,
Input-port-03  , 2,  , 2,  ,  ,  ,  , 0, 0,  ,  ,  ,  ,  ,  ,  ,  ,
```

In AN2, on a new connection request or a disconnect request, the pre-schedule is recalculated, whereas in our case the input ports start negotiating with each other and time slots may have to shuffle.

5 Maintaining the queues

Keeping one *forwarding-queue* would not be much help in scheduling the cells, because if, for example, the cell at the head of the queue is not to go to the output port to which the scheduling entry is indicating, that time slot (or many other) would be wasted and the CBR cells in the queue would be delayed.

To avoid this problem, at each port one queue for CBR cells for each output port and a queue for variable bit rate (VBR) cells is maintained. The VBR cells to be forwarded are placed in the *VBR-queue* and the CBR cells are placed in the corresponding *CBR-queues*. Now by reading the schedule entry one can pick the cell from the corresponding queue. Now cells for all the CBR connections going to one particular output port will queue in one *CBR-queue*. If more cells for one connection are in front of the cells of another connection, then those cells would be transmitted first. Better approach would be to create a queue for each new connection and in the schedule, instead of port numbers, store the pointers to the queues corresponding to the connection. This would make the things more complicated. But just to test the idea of scheduling first approach was chosen.

6 Mechanism for negotiation for time slots among the ports

Each port must have a *schedule* so that it can forward the cells according to the *schedule*. One approach could be to assume one port as the master port responsible for keeping time slots information for all the ports, calculating new schedules and then delivering these schedules to the other ports. So a connect/disconnect request at any port would be diverted towards the master port, which will then recalculate the schedule and pass it over to other ports as well. This has the disadvantage of burdening one port and also there is the problem of how all the ports can switch to the new schedule atomically. Instead, the approach chosen has an evenly distributed burden on all the ports.

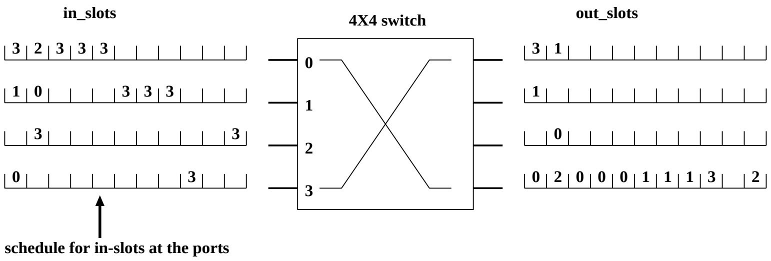

Figure 1: In_slots and Out_slots reservation at a 4x4 switch

In this approach, each port keeps information about its in-slots (slots at the input of the port) and out-slots (slots at the output of the port). The in-slots are used to reserve input slots in the *schedule* for a connection request and also one needs to reserve corresponding out-slots to avoid contention. So only those in-slots are reserved for which corresponding out-slots are available. In in-slots, the port number to which to send the cell is written and in out-slot the port number from which the cell is coming. Port i will be the sole authority for granting the out-slots for port i. For example, if a connection request at port 2 needs to send its cells to port 4, then port 4 will be requested to grant out-slots to port 2. Figure 1 shows slot reservations at the ports of a switch.

A control algorithm has been developed, which is used to exchange information between the ports of a switch when a connect/disconnect request arrives at a port. This algorithm was simulated to test for its convergence and was found working satisfactorily.

To make the algorithm simple, though more costly, the slots reservation for a connection request are made one by one. That is, first one slot is reserved, then second, and so on. Similarly the slots are released one by one for a disconnect message. A diagram highlighting the working of the algorithm for a 4x4 switch is given in Figure 2. The dotted arrows in the Figure indicate that the messages can go to any port and may follow an arbitrary number of loops. A typical slot reservation activity is shown in Figure 3 (which shows how this algorithm works).

The above mentioned algorithm cannot be used for a 16x16 delta Fairisle switch made up of 4x4 cross-bars, because at the first-stage output there could be contention which has not been catered for in the above algorithm. Now we have to consider the time-slots at the output of the first stage as well (let's call them stage-slots). Therefore to reserve an in-slot one has to reserve a corresponding stage-slot and out-slot. The stage-slot would contain the port number from which the cell is coming (as for out-slots). The Figure 4 shows that a connection request from port 3 to port 0 (for two time slots) cannot be honoured because stage-slots are not available even though out-slots are available.

It has been assumed that when ports exchange information, they know which port number they are and which port number they want to send the cell to. When a port reserves an in-slot, it knows the port number to which to send the request for the out-slot, and port number to which to send the request for a stage-slot. The port number to which a request for a stage-slot is sent is calculated by the formula as given in Equation (1). Where "portno" is the current port number, "to" is the output port number to which the data cells of this connection will be sent and "stageport" is the

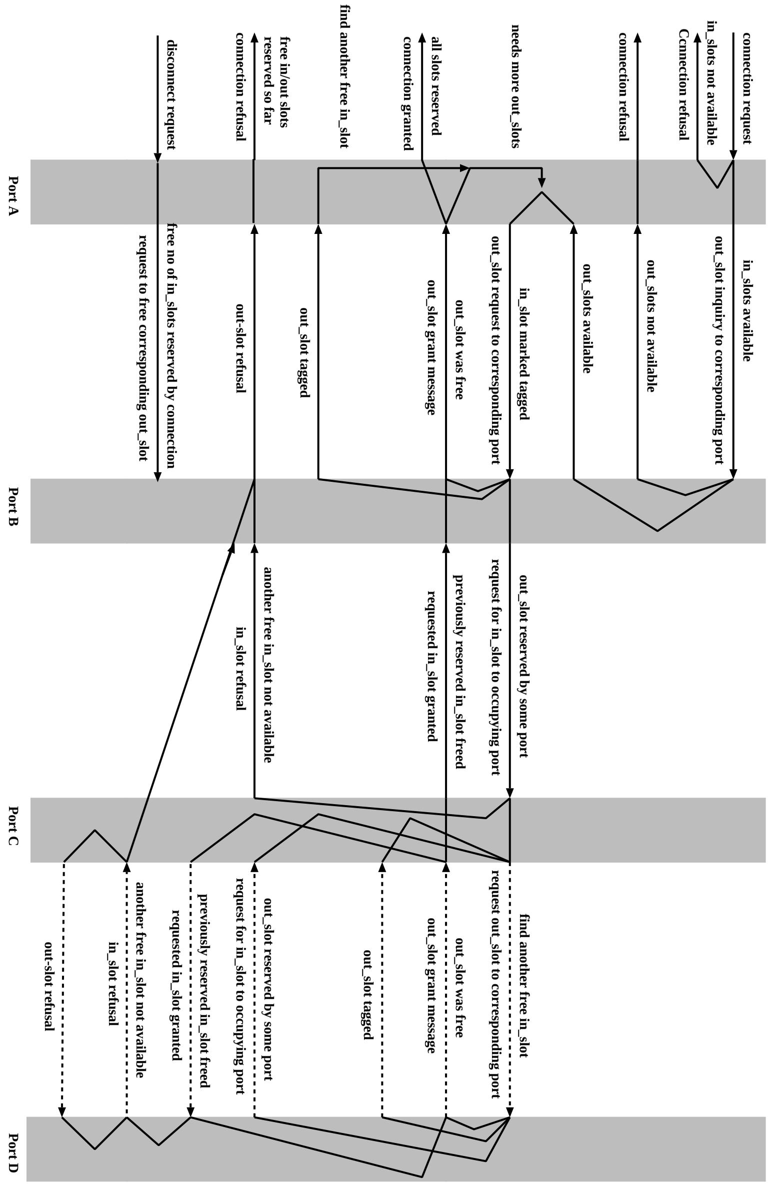

Figure 2: Working of algorithm for 4x4 switch

port number whom the stage-slot will be requested for the reservation of the corresponding in-slot. Integer arithmetic only is used in the formula, making computation efficient and simple.

$$stageport = (portno/4) * 4 + (to/4) \tag{1}$$

The actions taken by the port when different message types arrives at the port are shown in Figures 5 to 8. The "+" sign (white on black circle) indicates that more than one message is to be sent in response to the incoming reservation request.

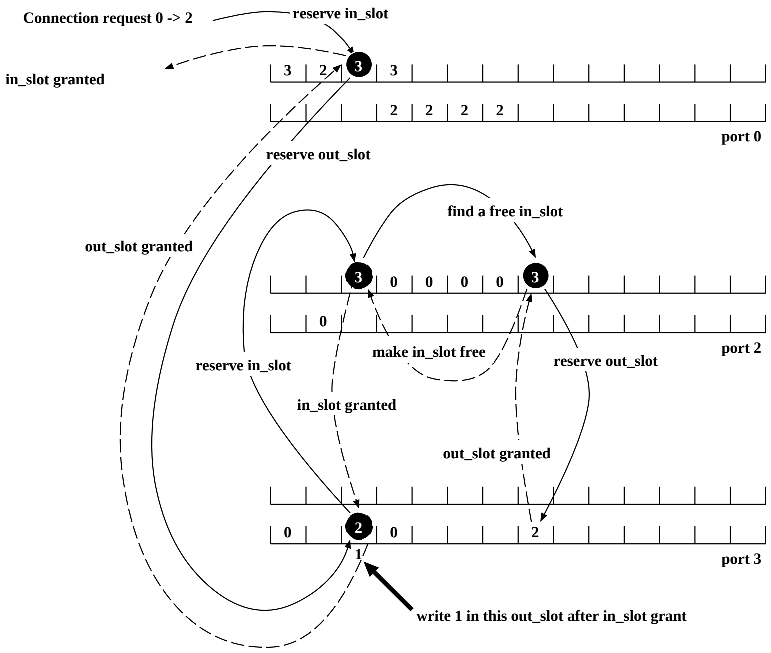

Figure 3: A typical slot reservation activity

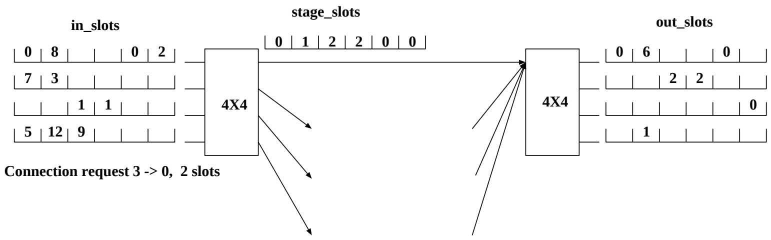

Figure 4: in_slots, stage_slots and out_slots at 16x16 switch

7 Synchronisation of ports

If a port is to send cells according to the *schedule*, it must know the current (global) slot number so that it can access the corresponding entry of the schedule. Luckily, the FPC3 port controller boards have a 16 bit free running clock (FRC) on them. The current slot number can be determined by reading the FRC and using a certain number of low order bits (depending on the frame size; e. g. lower 10 bits for a frame of 1024 slots) as an index into the *schedule*. But the ports do not have the same FRC value at the same time, because when a switch is booted, its ports start with some difference in time. This simply means that by only reading the FRC value, the port's *schedules* would not be synchronised. To synchronise the ports one needs to add the FRC difference with respect to some port (the master-port).

To obtain the FRC delta, a program has been written which runs in user-space on Wanda. The master-port makes connections using a particular port number with the other ports and sends cells at a regular interval to each other port. Information that it is a master port is given through the environment variable (SYNC-MASTER) defined in the hostinfo file. If it is a master port then a list of port addresses to which it is to connect is also given through the environment variable

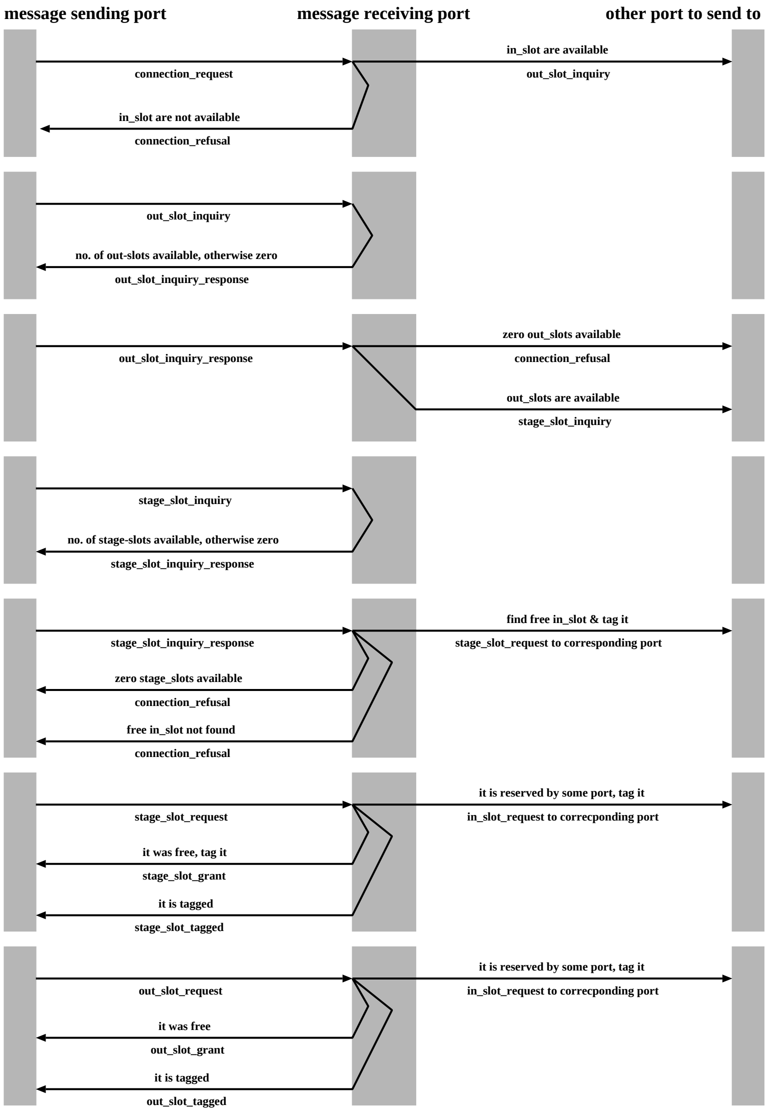

Figure 5: Working of algorithm for 16x16 switch

(SYNC-PORTS). If a port is not the master, it only listens on a specific port. The master-port sends cells in such a way that in the pay-load they have the FRC value of the master-port when they leave and FRC value of the receiving port when they arrive. The receiver now can calculate the FRC delta and store it for future use.

8 Modification to the fiq5 routine

As mentioned earlier the fiq5 [2] routine is responsible for receiving the cells and then placing them in appropriate queues. Also it picks the cells from the queues for transmission. Because now there are more queues and they are to be serviced differently (according to the schedule), this routine needs to be changed. This routine was changed accordingly. This routine is written in assembly language [3] very carefully so that it can deal with the cells as quickly as possible. Figure 9 shows

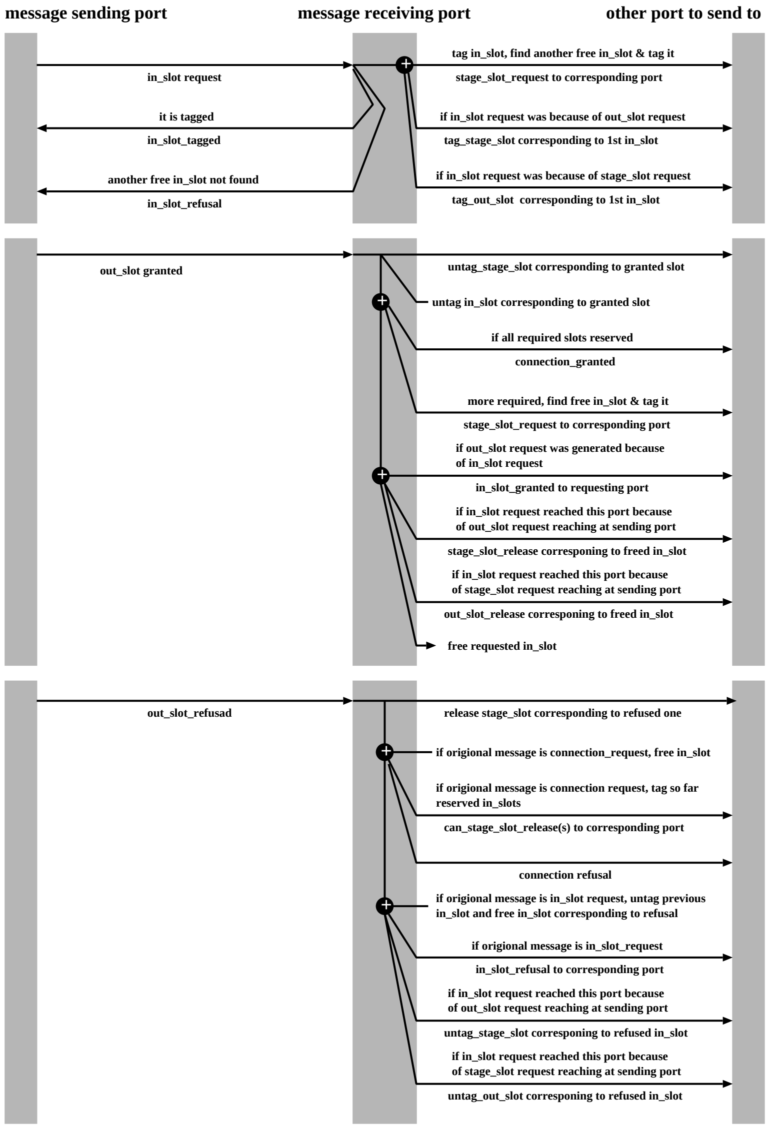

Figure 6: Working of algorithm for 16x16 switch

queues the fiq5 routine deals with.

9 Some modifications

Soon it was realized that in one slot time the processor cannot determine the cell to send in the next time slot because one has to read the FRC value, add the delta (some other things as well) and then get the queue pointer to pick the cell from. Therefore *sub-schedules* (or sub-frames), each of consecutive 8 time-slots, were formed and it was assumed that all the cells in this sub-frame would be sent to the same output port. Therefore now CPU has only to calculate the current sub-frame and get the queue pointer once for this sub-frame. Now there needs to be a mechanism to indicate that a new sub-frame has started. This can be achieved by writing the timer count

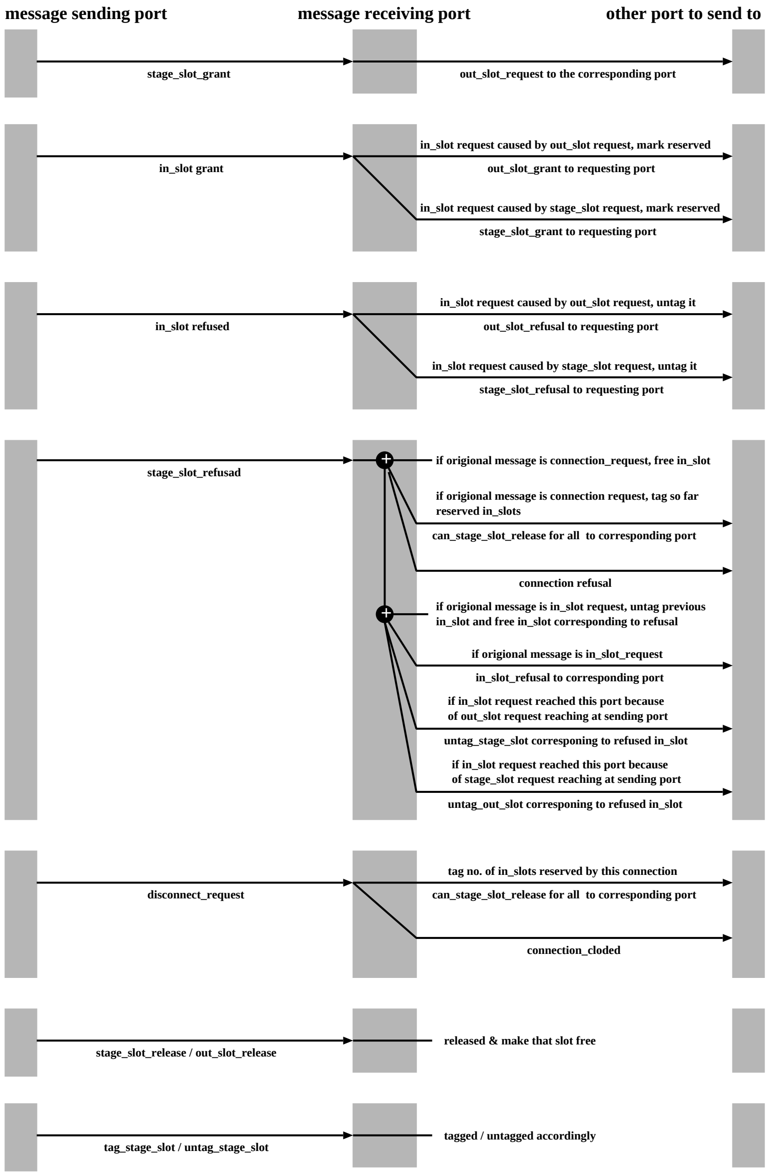

Figure 7: Working of algorithm for 16x16 switch

in the Xilinx and enabling the TCT interrupt. The timer count is decremented by one for each fabric tick until it reaches zero and at that point a TCT interrupt is generated. The fiq5 routine checks for a TCT condition, and if so, it will write a value of 8 immediately into the timer counter so that it can generate TCT interrupt correctly next time. Also cells from the CBR queue and VBR queue can not be mixed, because in case of a nack, we have to deal with them differently. In case of a CBR cell we just throw that cell away and in case of a VBR cell we place the cell back at the head of the VBR queue. In case of CBR cells, fiq5 tries to send as many cells as it can (maximum of 8) in a sub-frame time, but in case of VBR cells, it sends one cell and waits for its successful transmission, then sends the next VBR cell and so on till the sub-frame time finishes. Refer to Experiment 1 for full detail on the algorithm. It was observed in experiment 1 that at the most 3 to 4 cells were being forwarded in a sub-frame time by the fiq5 routine.

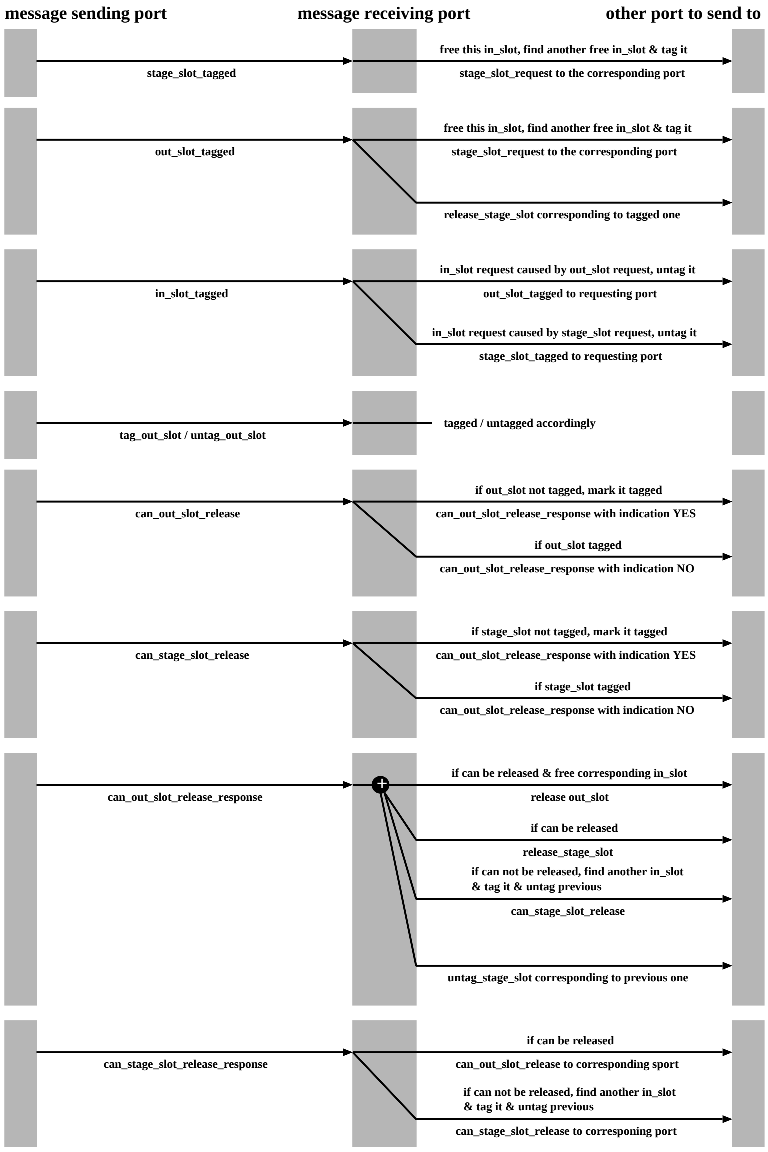

Figure 8: Working of algorithm for 16x16 switch

To make the fiq5 routine fast, it was rewritten with the new strategy. Now the sub-frame size was made 16 time slots. A sub-frame pointing to a VBR queue means, in this sub-frame all the cells forwarded would be from the VBR queue. But if it is pointing to a CBR queue, first it will forward maximum of 8 CBR cells and then as many VBR cells as it can. Now the CBR cells were being placed in the *transmit-queue* very fast by just using 5 assembly language instructions. If a CBR cell is nacked, it is simply thrown away, otherwise it is placed black at the head of the VBR queue. Therefore we can not put more than one VBR cells in the *transmit-queue*, because a nack condition can change the order of the cells. Refer to Experiment 2 for full details.

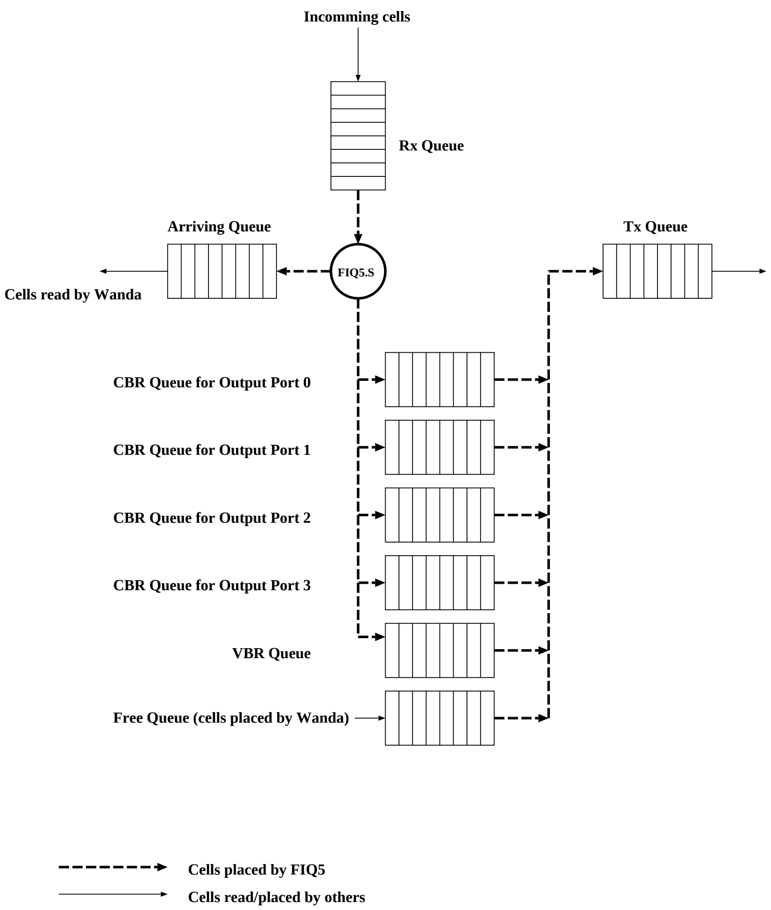

Figure 9: Proposed Working of fiq5.s routine

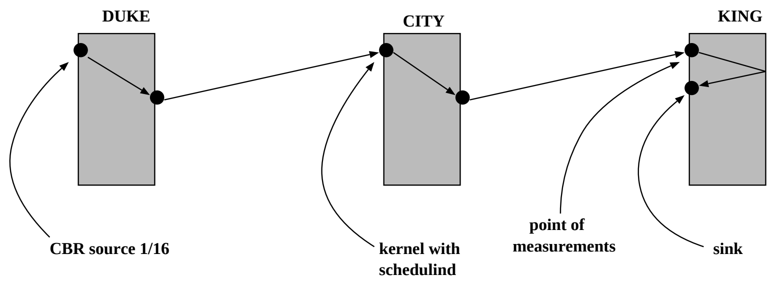

Figure 10: Switch configuration in the experiment for 8-slots sub-frame kernel

10 Experiment 1

To see the behaviour of fiq5 (with 8 time-slot sub-frame) described above, a kernel implementing the above scheduling algorithm was built and traffic was sent through it. The configuration of the switches used in the experiment is shown in Figure 10. The CBR source used was 1/16 (i. e. one cell after every 15 cell time). The *schedule* size was 1024 time slots (or 128 sub-frames, each of 8 time slots). The number of sub-frames theoretically required for this connection is 8. For this experiment the first 32 sub-frames were reserved for this connection.

The cell inter-arrival times observed showed that fiq5 was unable to forward the cells fast enough so that they may be transmitted back to back in a sub-frame. Because fiq5 has to do many things as described above, it takes quite a lot of time and is therefore unable to forward the required number of cells in a sub-frame. The inter-arrival time distribution is shown in the Figure 11. Some cells with inter-arrival time around 775 can be seen in the Figure 11. These are because no sub-frame from 33rd to 128th are allocated for this connection, so no cell for this connection will be transmitted till the *schedule* enters its next period. This means that the kernel is able to forward the cells according to the *schedule* but not fast enough. A customised kernel was developed to run at the "point of measurements" (as shown in the Figure 10) to obtain the inter-arrival times of the cells sent by the kernel implementing the scheduling. The source used in the experiment and the supporting programs to get the statistics from the "point of measurements" had already been developed by another research student (Simon Crosby).

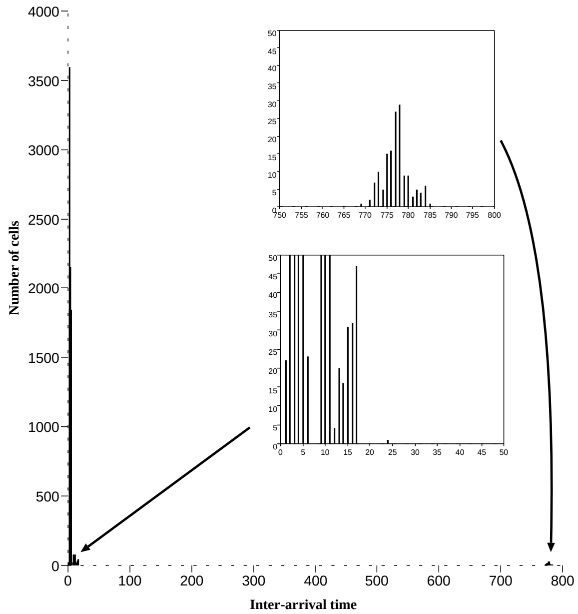

Figure 11: Inter-arrival time for 8 time slot sub-frame

11 Experiment 2

A kernel with with the new approach of 16 slot sub-frame was built and tested. The switch arrangement was the same as shown in Figure 10. Now the total number of sub-frames available is 64. CBR source used was 1/16. The first 12 sub-frames were reserved for this connection. The cell inter-arrival times observed showed that most of the cells were coming back to back, which means that fiq5 was able to forward the cells more rapidly.

No cell from CBR queue (where the incoming cells for the connection are placed) will be transmitted in sub-frames 13th to 64th, and the cells arriving from the source will keep on accumulating in the queue and will be transmitted when the next frame starts. The inter-arrival time histogram for this kernel is shown in Figure 12. The delay of the cells is bounded by the size of the *schedule*. If we reduce the *schedule* size, the delay of cells will be reduced but the granularity of sharing the bandwidth will also be reduced. If we increase the *schedule* size, however it increases the granularity of sharing the bandwidth but increases the delay time.

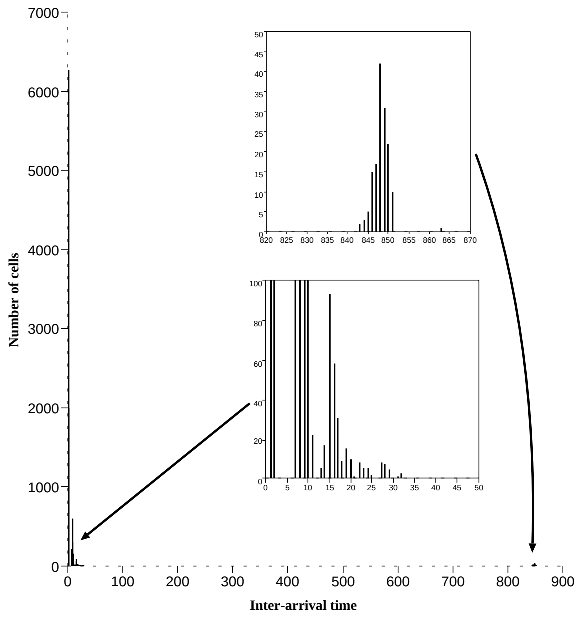

Figure 12: Inter-arrival time for 16 time slot sub-frame

12 Further Experiments

Next, two CBR sources were used. A Few tests with different sources and bandwidth reservations were made. The switch configuration used in the experiments is shown in Figure 13; (a) Two CBR sources of 2/100 each with 3 sub-frames reserved for each source were used and the inter-arrival time histogram is shown in Figure 14 and Figure 15. (b) Two CBR sources of 1/16 each with 16 sub-frames reserved for each source were used and their inter-arrival histogram is shown in Figure 16 and Figure 17. (c) Two CBR sources of 1/10 each with 22 sub-frames reserved for each source were used and the inter-arrival time histogram is shown in Figure 18 and Figure 19. In this case 100,000 cells from each source were sent, but only about 65 percent were received by the intermediate stage. This was because now fiq5 was unable to receive the incoming cells (therefore cells coming-in were being dropped by the port), though forwarding of cells was quite good as per *schedule*. Experiments with bandwidth reservations of 22, 25 and 28 sub-frames for each connection were also run, but the behaviour was the same. But when run with 18 sub-frame reservation, only about 24 percent of cells were received. (d) One CBR source of 1/16 with 16 sub-frames reserved for it and a VBR (Bernoulli) source of roughly 6.85 Mbs were used. The inter-arrival times at the sinks are given in Figures 20 and 21.

13 Concluding remarks

The results of the experiments show that the scheduling of time slots at the Fairisle ports can be achieved as described above but to do cell scheduling in software one needs more processing power. The fiq5 routine may be improved a bit more, but even then it will not be able to keep up

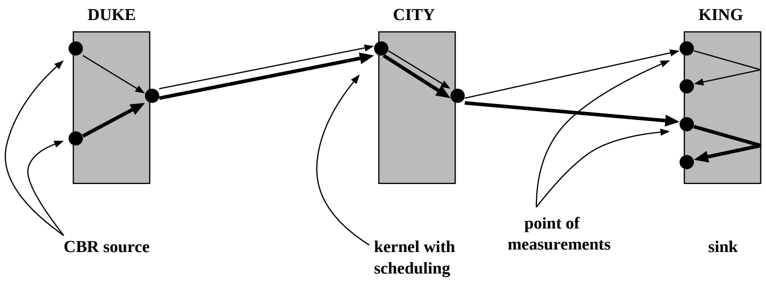

Figure 13: Switch configuration in the experiment for two CBR sources

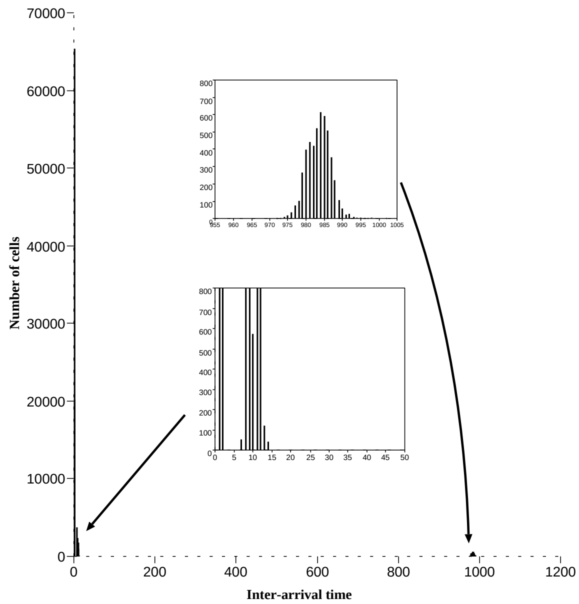

Figure 14: Inter-arrival time (sink 1) for 2/100 source with 3 sub-frames

with faster sources. Also if more hardware support is available, the cell forwarding can be done more rapidly and accurately; For example, at the start of each sub-frame the Xilinx timer count is written again so that it can generate TCT interrupt for the next sub-frame. If reloading of the timer count is done by the hardware, it will save time. Also if FRC can be changed at the port (or on all the ports at the same time), this will ease the problem of port synchronisation and hence adding of FRC delta. Currently the slot reservation is done randomly, but one needs an algorithm which spread the slots evenly throughout the slot-frame. In the experiments, we have reserved first sub-frames (which yields the worst schedule). This causes a long delay and followed by a burst of cells, followed by a long delay followed by a burst of cells, and so on. The exchange of message-cells for reservation of time slots can be sent on the meta-signalling channel used by Fairisle.

References

[1] *ATM Document Collection 3 (The Blue Book)* Computer Lab, University of Cambridge.

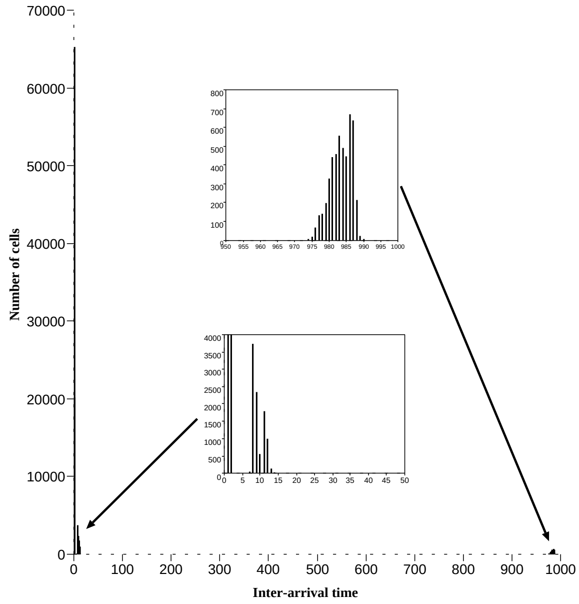

Figure 15: Inter-arrival time (sink 2) for 2/100 source with 3 sub-frames

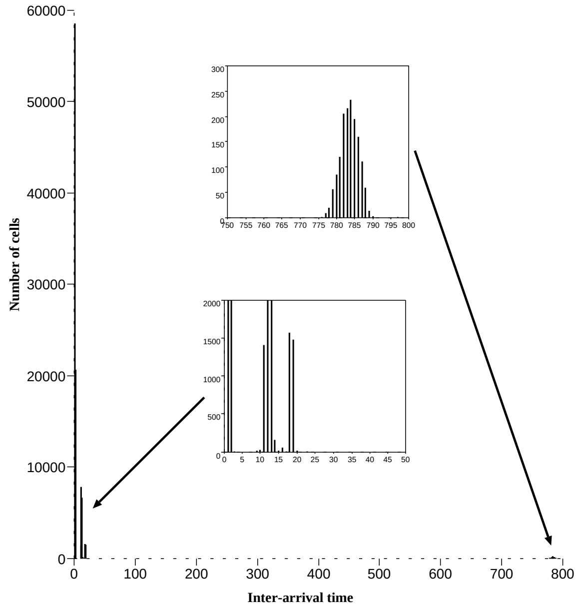

Figure 16: Inter-arrival time (sink 1) for 1/16 source with 16 sub-frames

[2] *Wanda operating system code.*

[3] *Arm6 Macrocell Datasheet* Issue no: 0.5, Nov 8, 1991.

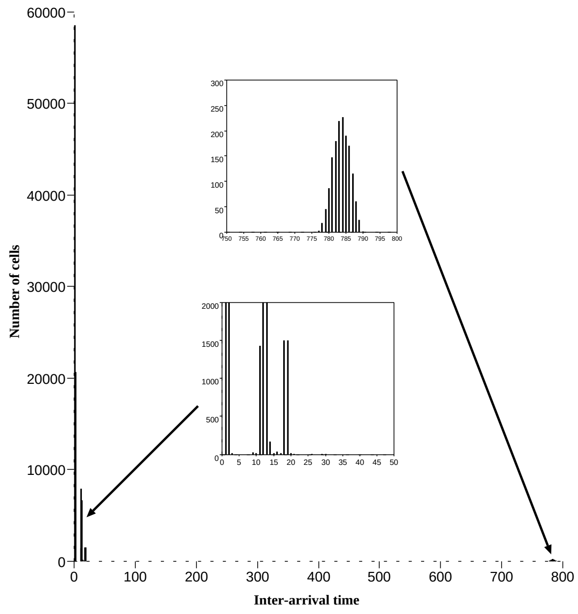

Figure 17: Inter-arrival time (sink 2) for 1/16 source with 16 sub-frames

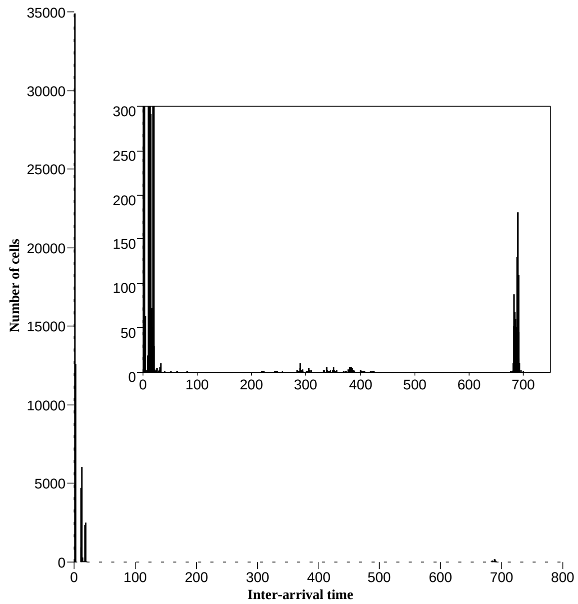

Figure 18: Inter-arrival time (sink 1) for 1/10 source with 22 sub-frames

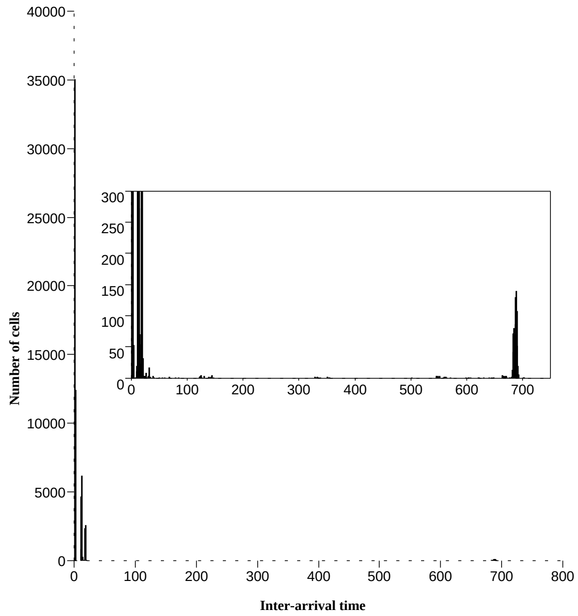

Figure 19: Inter-arrival time (sink 2) for 1/10 source with 22 sub-frames

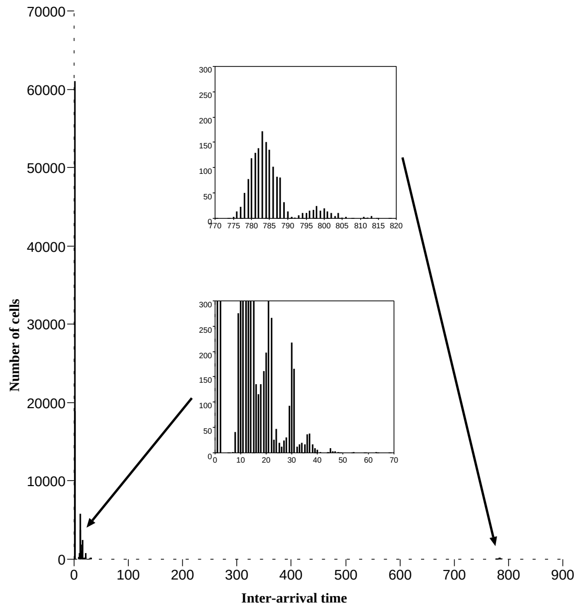

Figure 20: Inter-arrival time (sink 1) for 1/16 source with 16 sub-frames

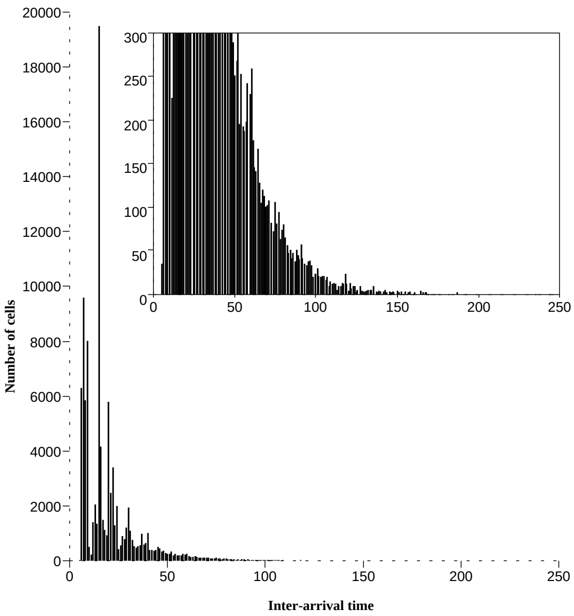

Figure 21: Inter-arrival time (sink 2) for VBR (bernoulli) source of roughly 6.85 Mbs