# An Efficient Implementation of Burst Fair Queuing for ATM Networking

Anthony T. Chronopoulos[*], Caimu Tang[†]

## Abstract

*Fair queueing, which was developed in last decade and was aimed at general packet switching systems with varying packet length, is not suitable for use in the ATM networking. The reason is that the ATM cell length is very small and fixed, and so the scheduling scheme on per cell basis isn't practical. The scheduling scheme of a switch affects the delay, throughput and fairness of a network, and thus it has a great impact on the quality of service (QoS).*

*In this paper, we give an efficient implementation of burst fair queueing for ATM networking. We also give a maximum burst delay bound under this algorithm implementation. One kind of flow regulator is employed for the algorithm. Furthermore, a numerical method is used for the virtual time function computation which achieves higher computation efficiency than previous methods. we use simulation to study the preemptive and nonpreemptive versions of the algorithms. The flow regulator performance under this algorithm is also examined.*

**Key Words** *: Fair Queueing, Cell Burst, Quality Measurement Unit, Quality of Service(QoS), Computation Efficiency*

[1]Dr. Anthony T. Chronopoulos is an associate professor at Computer Science Department, Wayne State University. His e-mail address is chronos@cs.wayne.edu

[2]Mr. Caimu Tang is a Ph.D student at Computer Science Department, Wayne State University, His e-mail address is ctang@cs.wayne.edu

## 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. [3], [4], [8], [10], [12] and [13].

We studied the Burst Based Weighted Fair Queuing (BBWFQ) in [11]. In section 2, we study an efficient implementation for this algorithm when a flow regulator is present in the network. In section 3, We present a new numerical method to approximate the virtual time function. In section 4, we present the simulation results.

## 2 An Implementation of BBFQ

In this section, we study the implementations of both the preemptive version and nonpreemptive version of BBFQ and the delay guarantees of BBFQ. The flow regulator which works with the leaky bucket algorithm is also studied.

### 2.1 The Algorithms

Let's first give the description of the algorithm, and postpone describing the pseudocode of the algorithm till the simulation section (where a full implementation is also given and the results are shown through the OPNET simulator).

Let $V(t)$ be a virtual time function. Here we don't limit the function to a specific one, any effective function could be used. For the simulation we will choose the modified version of the function used in [18]. Let i, k be the session-index and burst-number respectively, let $c(i,k)$ be the burst size and let $S(i,k)$, $F(i,k)$ be the virtual start-time and finish-time of the k-th burst from session i, respectively. Also, let $a(i,k)$ be the burst arrival time. The algorithm steps are as follows:

- Starting system busy period( at physical time $t_{start}$):
  $S(i,0) = F(i,0) = 0$; for arbitrary session index i. and $V(t_{start}) = 0$;
- Burst arrival:
  1. Burst-start:
  $$c(i,k) = 0, S(i,k) = max\{F(i,k-1), V(a(i,k))\}.$$
  2. Cell arrival:
  $$c(i,k) = c(i,k) + 1.$$

3. End of a burst

$$F(i,k) = S(i,k) + c(i,k) * \frac{\tau_0}{\phi_i} \ ,$$

where $\tau_0$ is the cell slot and $\phi_i$ is the bandwidth portion available for session i.

- Burst departure(Non preemptive)
  The scheduler serves the bursts in sessions according to the ascending order of the $F(i,k)$ which are backlogged in the server so far.

- Burst departure(Preemptive)
  The scheduler chooses the burst with smallest $F(i,k)$ among the backlogged sessions in the server. In case a burst with smaller virtual finish-time arrives, then the scheduler will preempt the currently served burst at the closest boundary of a cell and start to schedule this burst.

### 2.2 Delay Property

Let $A_i(\tau,t)$ be the amount of cells from session i that leaves the leaky bucket and enters the network in the time interval $[\tau,t]$. We say session i is leaky bucket constrained if

$$A_i(\tau,t) \leq \min\{(t-\tau)Q_i, \sigma_i + \rho_i(t-\tau)\}, \ \ for\ arbitrary\ t \geq \tau \geq 0\ , \tag{1}$$

where $\sigma_i$ is the depth of the bucket, $\rho_i$ is the token generation speed, $Q_i$ is the maximum rate allowed at the exit of the leaky bucket. We denote the leaky-bucket-constraint on $A_i$ by the triplet $(\sigma_i, \rho_i, Q_i)$.

**Theorem 1** *Consider session i is leaky-bucket constrained by $(\sigma_i, \rho_i, Q_i)$. If the scheduling scheme is non-preemptive, then the session i has delay guarantee equal to $\frac{\rho_i + C_{max}}{\rho_i}$.*

Proof: First note that session i is a FIFO queue, $\widehat{NS}_i(t_0,t_1)$ is the number of served cells under PGPS during interval $[t_0,t_1]$ from session i, $NS_i(t_0,t_1)$ is the number of served cells under GPS during interval $[t_0,t_1]$ from session i, and $NA_i(t_1,t_2)$ is the number of cells arrived during interval $[t_1,t_2]$ . Let $a(i,j)$ be the arrival time of burst $(i,j)$, and let $d(i,j)$ be the departure time of burst $(i,j)$. Let $t_0$ be the latest time before $d(i,j)$ when the session i is backlogged. Then we have,

$$\widehat{NS}_i(t_0, d(i,j)) = NA_i(t_0, a(i,j)) \ \ ,$$

and then inequality ( 1) also holds for $NA_i$:

$$NA_i(t_0, a(i,j)) \leq \min((a(i,j)-t_0)Q_i, \ , \ \ \sigma_i + r_i(a(i,j)-t_0)) \ \ .$$

So we obtain $\hat{NS}_i(t_0, d(i,j)) \leq \sigma_i + r_i(a(i,j)-t_0)$. By Theorem 2 [11], we have

$$NS_i(t_0, d(i,j)) - C_{max} \leq \widehat{NS}_i(t_0, d(i,j)) \ \ .$$

and since GPS server is backlogged for session i during $[t_0, d(i,j)]$, we have

$$r_i(d(i,j) - t_0) \leq NS_i(t_0, d(i,j)) \ ,$$

where $r_i$ is the available rate to session i. So, we get

$$r_i(d(i,j) - t_0) - C_{max} \leq \sigma_i + r_i(a(i,j) - t_0) \ ,$$

Therefore, $d(i,j) - a(i,j) \leq \frac{\sigma_i + C_{max}}{r_i}$. $\Box$

This theorem is useful because at the traffic entry, as long as the traffic characteristics are known, one can determine the maximum delay the traffic will experience in the network. Also, in order to have guaranteed delay quality, the user can compute the demanded bandwidth and subscribe for this bandwidth from the carriers or service providers. This is critical for delay-sensitive application.

### 2.3 Burst Boundary Determination

BBFQ has two outstanding advantages. One is the relaxed deadline in terms of end-to-end delay. The other is the computation efficiency. Since the high level packet is segmented according to ATM cell size, usually one high level packet will cause a cell burst in the underlying network, and sometimes the high level packet is the unit of quality of performance measurement. For example, a TELNET packet will generate a TCP packet and the TCP packet is further encapsulated in an IP packet and is sent out. Also, many currently multimedia applications use UDP as the transport layer protocol. One application level packet will need several UDP packets to be sent. These UDP packets will generate a cell burst in the underlying ATM network. Since the high level information is totally encapsulated in these packets, from the lower level network point of view, it's almost impossible to get this burst information (except for other mechanisms to pass this information down to the network node such as through setup signaling.

In order to make the BBFQ practical, a proper method needs to be employed to distinguish between the bursts. In the literature, there are usually four methods: Two are deterministic and the other two are not ([5], [7]). The first method uses a constant burst size for each session to approximate the real burst size, this method is the same as the time-slicing. The second method passes the burst size information explicitly at the header cell of the burst. This method will lead to some difficulties in the protocol design and also needs extra bandwidth for this burst size information. The third method is introducing a concept called flow regulation which makes the burst boundary distinguishable by the underlying network switches. This method is very attractive because there is no modification of currently widely used protocols and saves bandwidth. The disadvantages are that this method needs hardware and software supports and further complicates the switch design and implementation. The fourth method uses a statistical model to predict the next burst size. It makes use of the burst correlation of applications, which is based on the assumption that a well-behaved application traffic model always has some pattern. Once the scheduler catches the pattern, the burst size almost are the same through the session. In order to catch the traffic pattern, the switch needs some historical data and some computation. These are the obvious shortcomings of this method.

Our work will further study the flow regulation method and enforce the regulation at the entrance of the traffic. More specifically, we will put the flow regulation function into the leaky bucket constraint, and make the design work much easier.

The model of a leaky bucket constraint without flow regulator is as follows:

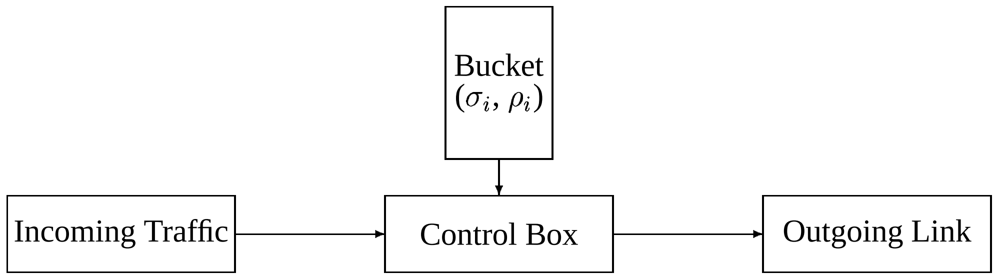


Figure 1:A leaky bucket

In this model, the tolerable burst size is

$$\frac{\sigma_i}{r_i - \rho_i} * \rho_i + \sigma_i \quad ,$$

for session i. The characteristics of the leaky bucket are extensively studied in [1] and [2].

In the leaky bucket-flow regulator combined system, the token generation rate is not the constant $\rho_i$ (for each session i), but it's a function denoted by $T_i(t)$, which depends on the remaining tokens in the bucket. Let $K_i(t)$ be the number of tokens in the leaky bucket of session i at time t. Let $\rho_i$ be the maximum token generation speed allowed for session i. We define

$$T_i(t) = \begin{cases} \frac{\rho_i}{\tau_0 - t_0}(t - t_0) & t_0 \le t \le \tau_0 \\ \rho_i & \tau_0 \le t < \tau_1 \\ 0 & t = \tau_1 \quad , \end{cases}$$

where $K_i(t_0^+) = 0$ and $K_i(\tau_1) = \sigma_i$.

In the definition of $T_i(t)$, we use a linear function in the interval $[t_0, \tau_0]$ instead of a constant $\rho_i$. We call this the slow-start stage of the bucket. Usually, the slow-start stage begins just after a burst or during the near-end period of a burst. One big advantage of using $T_i(t)$ instead of $\rho_i$ is that it's easy to handle many greedy sessions[10] and still allow the same burstiness as a normal bucket. Another way to measure the burstiness is by using the traffic on-off time interval. Here, the on-time interval is constrained by $\tau_1 - t_0$, and the off-time interval starts at $\tau_1$, and the length depends on the inter-arrival time of the burst.

What we need to do next is to give a proper time interval for the slow start stage, i.e. the value for $(\tau_0 - t_0)$. Several cell slots are needed for the network to locate the burst boundary. We use simulation to determine an approximation of this value. From now on, we denote this interval by $T_0 = \tau_0 - t_0$. At the incoming link of a switch, we maintain two counters for each session, one is the cell number counter $C_i$ and the other is the cell slot counter $S_i$. Once we detect an idle slot on one session, we reset the two counters to zero and start to record the idle slots since then. If we find that the value of $C_i$ is greater than a threshold $S_{idle}$, then we find the burst boundary, we compute the virtual finish-time of the last burst and we reset the two counters. If $S_i < S_{idle}$, and if after $T_0$ slots, the value of $C_i$ is less than $\frac{1}{2}\rho_i T_0$ (which is the integral of $T_i(t)$ over $[t_0, \tau_0]$), then we also detect the burst boundary and we compute the virtual finish-time of this burst.

It's easy to see that the method above will cause the scheduling algorithm to become nonwork-conserving if we apply the above procedure at the beginning of a system busy period and the system burst backlog is zero. But each session is waiting the arrival of the burst boundary, even though the output link is idle and there are cell backlogs in the system. We introduce an initialization stage. Whenever the output link is idle, we will interrupt the active session at this moment and send it out when a burst backlog forms in the system.

## 3 Numerical Method

As stated before, the computation of virtual time function is a critical point for the whole efficiency of the PGPS server and its variants. In ([6], [10], [12]) the virtual time function is expressed as an ordinary differential equation as follows:

$$\begin{aligned} v(0) &= 0, \\ \dot{v}(t) &= \frac{1}{\sum_{i \in B(t)} \phi_i} \quad , \end{aligned} \tag{2}$$

where $B(t)$ is the index set of sessions which are backlogged at time t. This equation can't be solved explicitly because the index set $B(t)$ is changing dynamically, and so the summation changes too. Numerical solution offers a simple way to get the value of function $v(t)$ at some time t.

Let's use $f(t)$ to denote the right hand side of equation ( 2):

$$f(t) = \frac{1}{\sum_{i \in B(t)} \phi_i} \ .$$

From the mathematical point of view, $f(t)$ is a non-linear function. We rewrite the equation with the initial condition:

$$\begin{aligned} v(0) &= 0, \\ \dot{v}(t) &= f(t) \ . \end{aligned} \tag{3}$$

Equation ( 3) is a standard ordinary differential equation. The variable step size Euler method is used for the numerical computation of the virtual time function ( 3). The iteration is given as following:

$$v(0) = 0 \ ,$$

$$v(t_{j-1} + \tau_{j-i}) = v(t_{j-1}) + \tau_{j-i} * f(t_j) \ , \quad (4)$$

where $t_j = t_{j-1} + \tau_{j-i}$.

For the PGPS with variable packet length, this method offers good accuracy. The iteration time step is usually chosen to be the inter-arrival time of two contiguous packets. However, in the BBFQ for ATM networking, a better result can be obtained. For, in the packetized networking where the variable size packet is an indivisible switching data unit, and the correlation among cells in the burst is loose. But in BBFQ, it's possible to use a larger iteration step to compute the virtual time (at some physical time) which will reflect accurately the queue status in the server. Our method uses the iteration step based on the queue backlogs in the server and on the bandwidth portions of the backlogged queues. The smaller the total summation of the bandwidth portions of all the backlogged sessions is, the smaller the size of the iteration step. The smallest step is one cell slot. And also, if the set of active sessions doesn't change since the last virtual time computation, the invocation is not necessary. Through the simulation, we find many invocations of the virtual-time routine is this case.

Using $v(t)$ obtained by the Euler method with $\tau_{j-1} = 1$ cell-slot (for all j) as the exact solution, we use the same number of the invocations of the virtual function, we can get more accurate approximation for $v(t)$ in BBFQ than the non-burst version method. Or we can get the same accuracy as PGPS with much fewer invocation of the function $f(t)$.

We next give the simulation comparison between these two methods which demonstrates that our method almost always has a better overall performance even through we use fewer iterations steps (almost half the number), because we can eliminate lots of unnecessary invocations. The unnecessary invocations are the invocation where the number of backlogged sessions don't change since last invocation.

The session bandwidth portions and related arguments are given in the following table. Note that the unit of burst size argument is an octet and the inter-arrival argument (Inter-arg item in Table 3) is for the Poisson traffic generator.

| Session No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Burst Size | 8988 | 1581 | 1581 | 8988 | 12400 | 12400 | 2400 | 24000 |
| Bandwidth Portion | 0.2 | 0.1 | 0.3 | 0.1 | 0.6 | 0.5 | 0.5 | 1.0 |
| Start-Time | 1000 | 1500 | 200 | 500 | 1000 | 100 | 1300 | 0 |
| Stop-Time | 3000 | 2800 | 2700 | 2000 | 3000 | 2000 | 2800 | 3000 |
| Inter-arg | 200 | 200 | 200 | 200 | 500 | 300 | 200 | 100 |

**Table 1. The bandwidth portion table for virtual function numerical analysis**

The virtual time function and the curve of total number of invocations obtained by using the standard Euler method is given in Figure 2.

The virtual time function obtained by using the BBFQ Euler method and the total invocation number curve is presented in Figure 3.

In the BBFQ Euler method, we changed the step size based on the total active bandwidth portion. If the total active bandwidth portion is greater than a threshold, we set $\tau_{j-1}$ = 'the number of cells in the current burst'. The total sum of the bandwidth portions of all the sessions whether or not they are backlogged, in a bandwidth portion normalized server, equals 1.0. In our experiment, the total sum of the bandwidth portions equals 3.3, so the slope isn't necessarily greater than 1.0. Therefore, the QoS's are still guaranteed. In this experiment, we use the bandwidth portion threshold as one fifth, the result shows that our method offers almost the same approximation as does the PGPS but it uses much fewer invocations of the virtual time function.

## 4 Simulation Study

In a real network, the traffic always has some pattern, therefore it's very difficult to establish different traffic models to test an algorithm under various situations. Simulation is an effective way to do this.

In this section, we will simulate 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. Usually, the ATM backbone switch doesn't directly support applications, only the access switch can connect to data terminal devices (like computer, video codec, LAN and etc) via network interface card (NIC). The bandwidth of the links between backbone switches is higher than that of links between access switches and backbone switches. From the cell scheduling point of view, the main functionality of a backbone switch is scheduling the cells from one backbone switch to another backbone switch. Sessions are represented by virtual paths. In an access switch, the scheduler inside the switch will schedule the cells from an application such as a LAN, a video-on-demand server etc. to the backbone switches. The link capacity between access and backbone switches is lower than that on the backbone network. This is a typical setup for a ATM network if the B-ISDN is employed.

### 4.1 The simulation environment and the simulation model

Our simulation package is the OPNET modeler 3.0.B from MIL 3 Inc. [9]. OPNET modeler supplies adequate in-built C functions to support accurate algorithm implementation with the discrete-event mechanism besides its flexible analysis tools.

In this simulation, we have five simplified ATM backbone switches, see Figure 4. The switches on the four corners on the topology map are called peripheral switches and the one at the center is called the center switch. The link bandwidth between these switches is $155 Mbits/sec$, and the topology is a *partial mesh* (i.e. a mesh where some nodes do not have direct links between them). Each backbone switch is connected to two access switches. Each access switch serves eight applications. Four of the sources follow a Poisson traffic model for a general application and the other four sources follow a clocked traffic model for a multimedia application. The link bandwidth between a backbone switch and an access switch is $155 Mbits/sec$. Although the network is simple, it is sufficient to simulate the scheduling part of the ATM networking. The OPNET Modeler 3.0.B version runs on the Sparcstation20 with SUN OS 4.1.3.

The following table gives the source traffic parameters:

| Session No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Bandwidth Portion | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.5 | 0.5 | 1.0 |
| Pk Size Args | 8988 | 1581 | 1581 | 8988 | 1240 | 1240 | 2400 | 2400 |
| Leaky-Bucket Burst Size | 50 | 80 | 100 | 100 | 100 | 200 | 100 | 200 |
| Leaky-Bucket Max Rate | 2000 | 2000 | 2000 | 2000 | 4000 | 4000 | 5000 | 5000 |
| Leaky-Bucket Token rate | 50 | 10 | 30 | 20 | 50 | 10 | 10 | 10 |
| Source Interarrival | 200 | 200 | 200 | 200 | 100 | 100 | 100 | 100 |
| Source Start Time | 100.0 | 0.0 | 200.0 | 0.0 | 100.0 | 100.0 | 20.0 | 0.0 |
| Source End Time | 2000 | 1800 | 1700 | 1500 | 2000 | 2000 | 1800 | 1700 |

**Table 2. The Source traffic parameter**

In the figures (generated by OPNET), the ordinate unit is usually the number of cells for throughput, bucket-depth and cell slot for delay. The abscissa unit is usually the simulation time measured in cell slots rather than seconds, the number after the title in each patch of these figures are the session numbers.

### 4.2 Performance analysis of flow regulator

Flow regulator is more important for the backbone switches than for the access switches. Since the traffics are multiplexed into one virtual path between two backbone switches, and at the access switches, the real-time traffic source always has some temporal relation. So the scheduler can detect the boundary of bursts and schedule the whole burst at one time. In this experiment, we will give the flow regulator performance analysis under different boundary detection thresholds at both the access switch and backbone switch. We give the comparison at the access switches using two different thresholds. This experiment uses the traffic arrival presented in Figure 5 measured in cells.

We use the threshold 20 cell slots to get the source bucket depth, the bucket token generation rate, the delay and the backlog of each session in Figures 6-9. Performances on bucket depth, delay, backlog without flow regulator at the access switches are presented in Figures 10-12.

In this experiment, we find the scheduler can get the exact burst size almost always as long as the boundary threshold is greater than 20 cell slots for these traffic arrival models. We believe that under different traffic arrivals, the needed value of the threshold is also very small. The overall gain using flow regulator can be seen by comparing Figures 6-9 to Figures 10-12. From this comparison, we can also see that the Quality of Presentation (QoP) of the multimedia applications and the services' qualities of the best effort traffic are also guaranteed. Figure 7 shows the token generation rate of the leaky-bucket of a traffic source session which reflects the effects of the flow constraint. We also use 5 cell slots threshold to perform the experiment and the token generation rate is displayed in Figure 13.

Next, we present the performance index figures of delay and throughput at the backbone switches in Figure 14 and Figure 15. Note that, because of the connection oriented feature of ATM, it's possible that we can distinguish the source session at the backbone switch for both switched VP/VC and permanent VP/VC.

## References

[1] Rene L. Cruz, "A calculus for network delay, Part I: network elements in isolation", *IEEE Transactions on Information Theory, vol.37 No.1 Jan. 1991.*

[2] Rene L. Cruz, "A calculus for network delay, Part II: network analysis", *IEEE Transactions on Information Theory, vol.37 No.1 Jan. 1991.*

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

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

[5] ITU-T Rec. I.371. Traffic Control and Congestion Control in B-ISDN. *ITU Telecommunication Standardization Sector, March, 1993.*

[6] Albert G. Greenberg, Neal Madras, "How fair is fair queueing?", *The Journal of ACM, vol.37, No.3, July 1992.*

[7] Simon S. Lam and Geoffrey G. Xie, "Burst scheduling networks", *Technical Report TR-95-28, University of Texas at Austin, July 1995.*

[8] E.Hahne, "Round robin scheduling for fair flow control", *Ph.D thesis, Dept. Elect. Eng. and Comput. Sci., M.I.T., Dec. 1986.*

[9] OPNET Modeler: Modeling , Simulation Kernel, Mil 3, Inc. 1996.

[10] Abhay K. Parekh and Robert 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.*

[11] Caimu Tang, Anthony T. Chronopoulos, Ece Yaprak, "A Cell Burst Scheduling for ATM Networking Part I: Theory and Part II: Implementation", *To Appear in Proceeding of ISCC'98.*

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

[13] Lixia Zhang, "VirtualClock: A new traffic control algorithm for packet-switched networks", *ACM Transactions on Computer Systems, Vol.9, No.2 May 1991.*

**Figure 2. Virtual time function obtained**
using standard Euler method

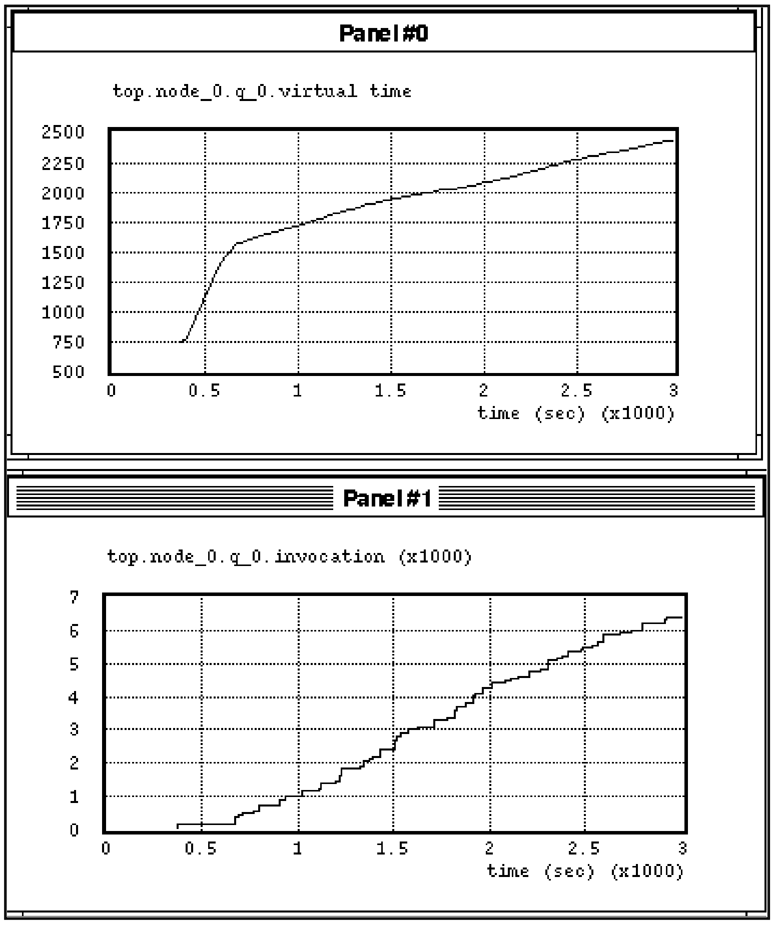


**Figure 3. Virtual time function obtained using revised Euler** method(with fewer iterations)

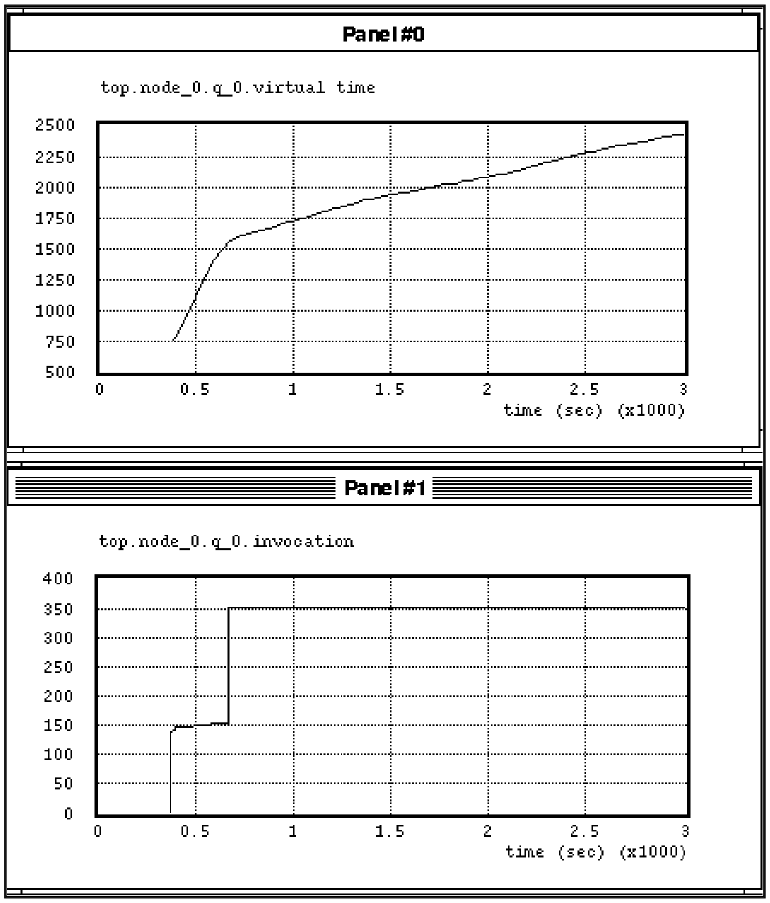


**Figure 4. The network topology**

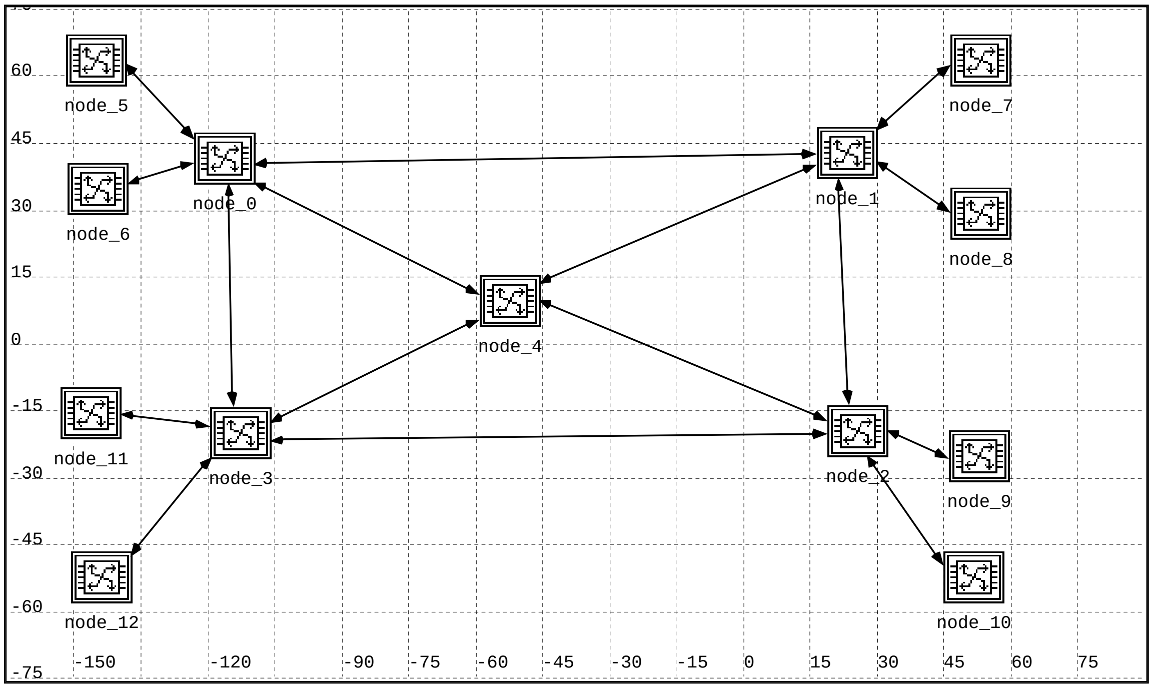


**Figure 5. The traffic arrival patterns for flow regulator**

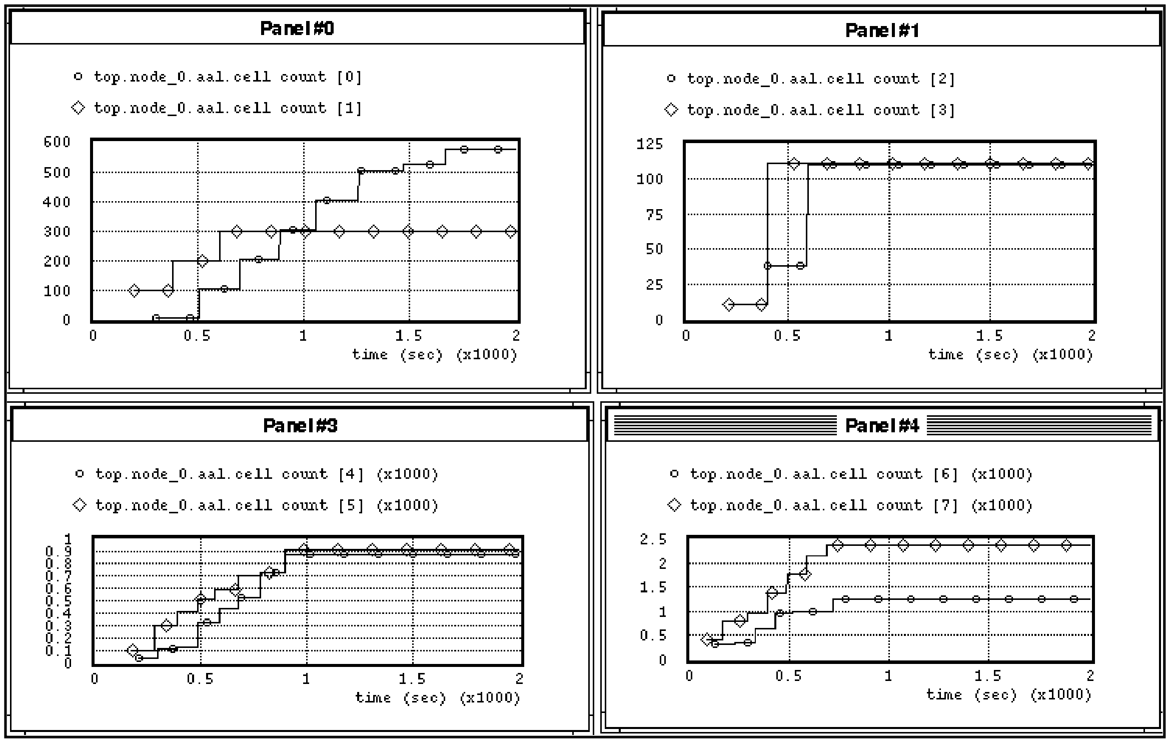


**Figure 6. The bucket depth with flow regulator**

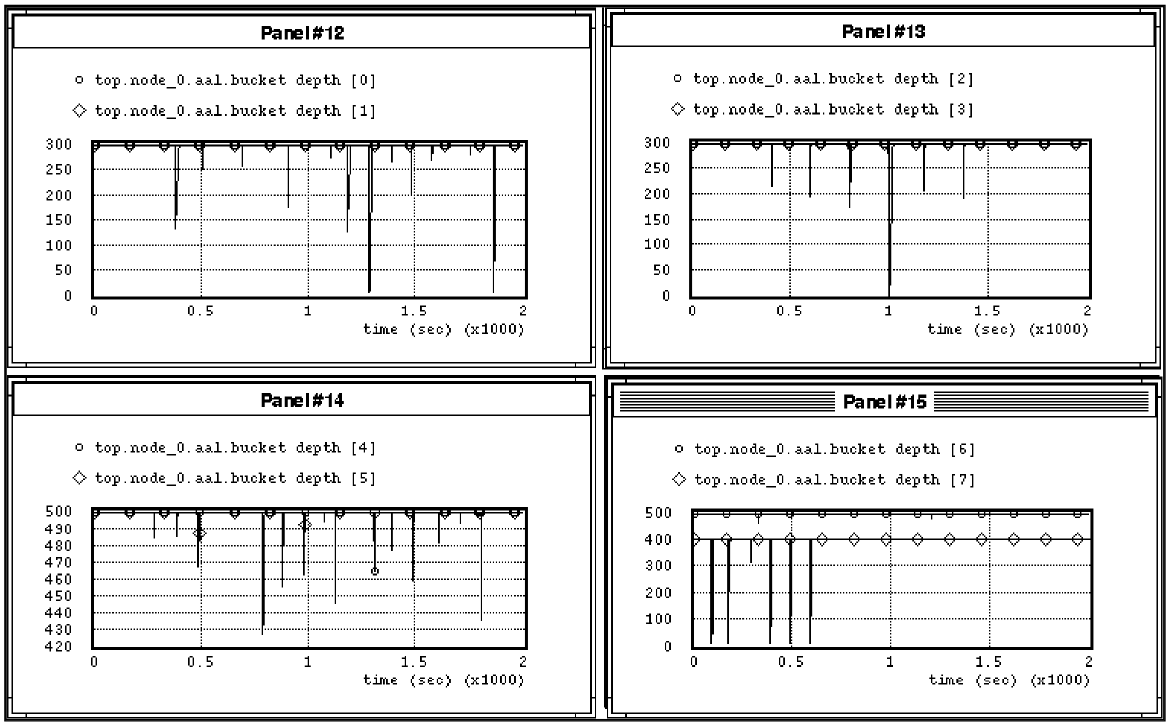


**Figure 7. The leaky-bucket token generation rate (threshold 20) with flow regulator**

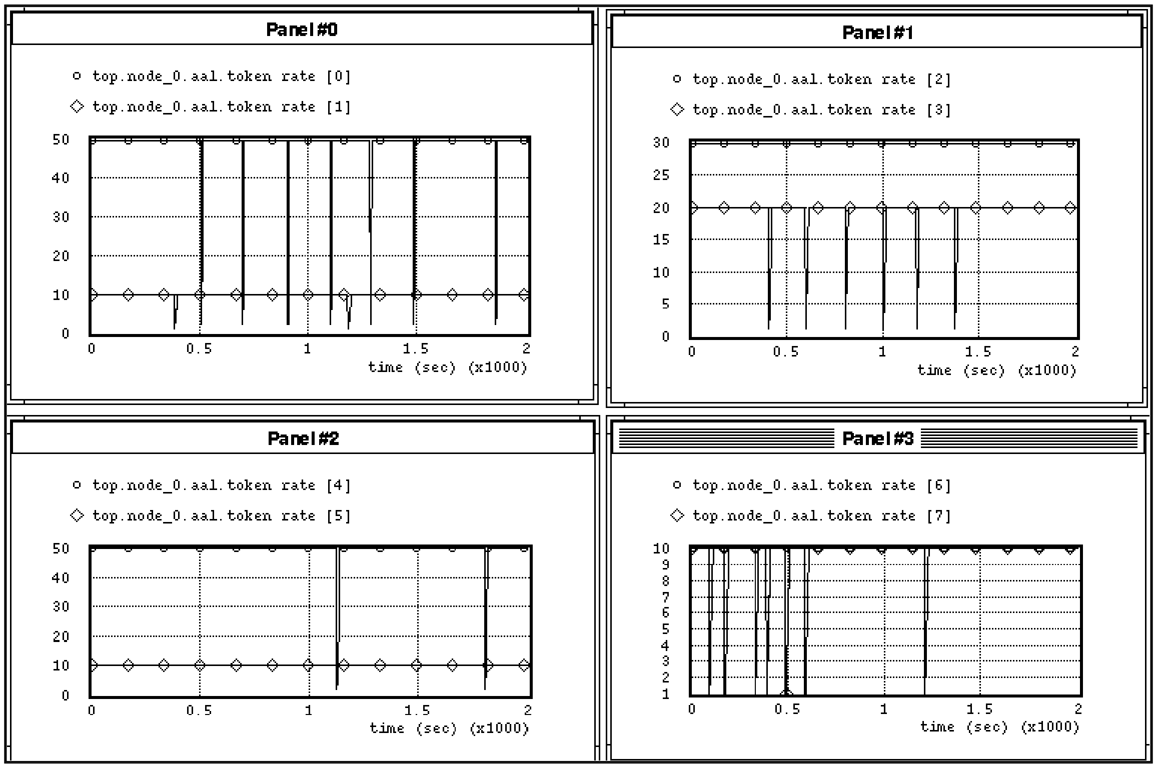


**Figure 8. The session delay with flow regulator**

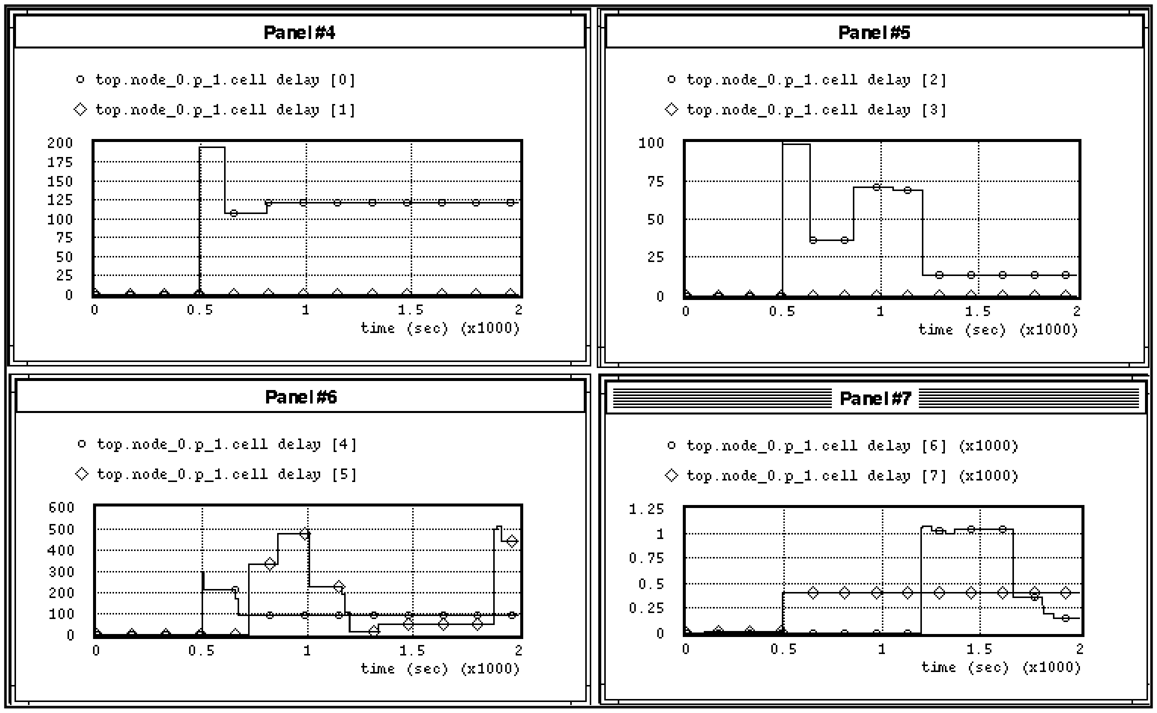


**Figure 9. The queue backlog with flow regulator**

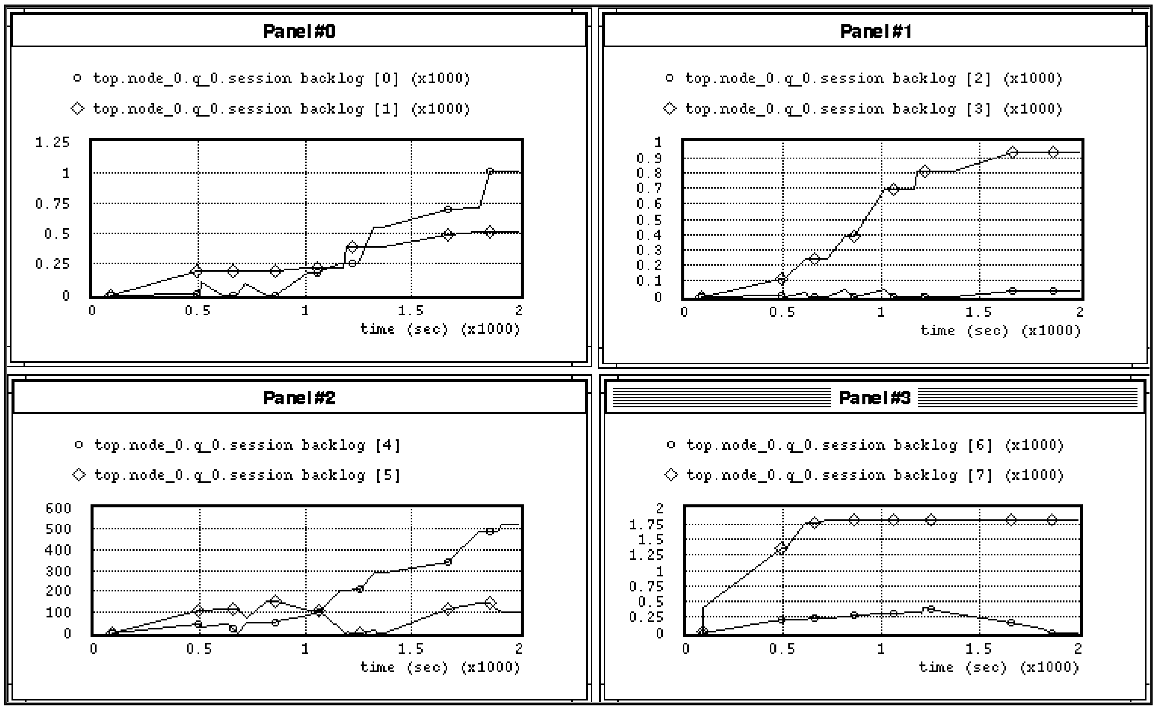


**Figure 10. The bucket depth without flow regulator**

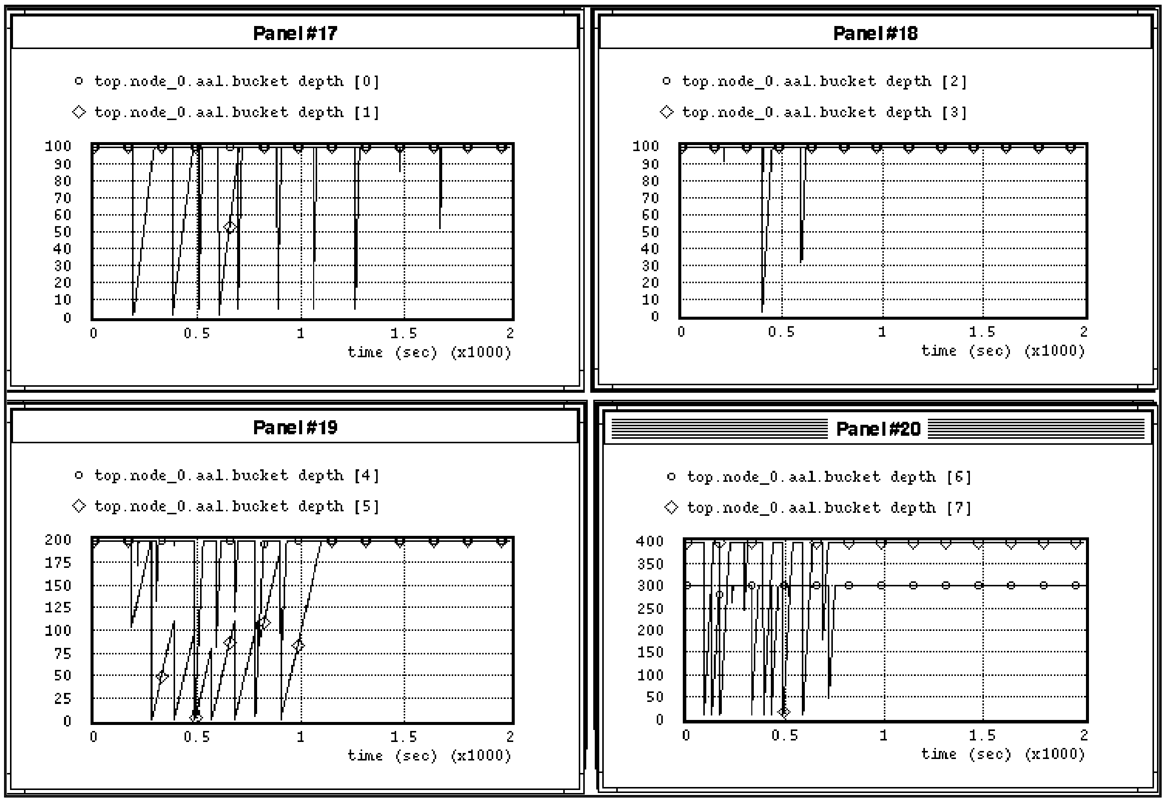


**Figure 11. The session delay without flow regulator**

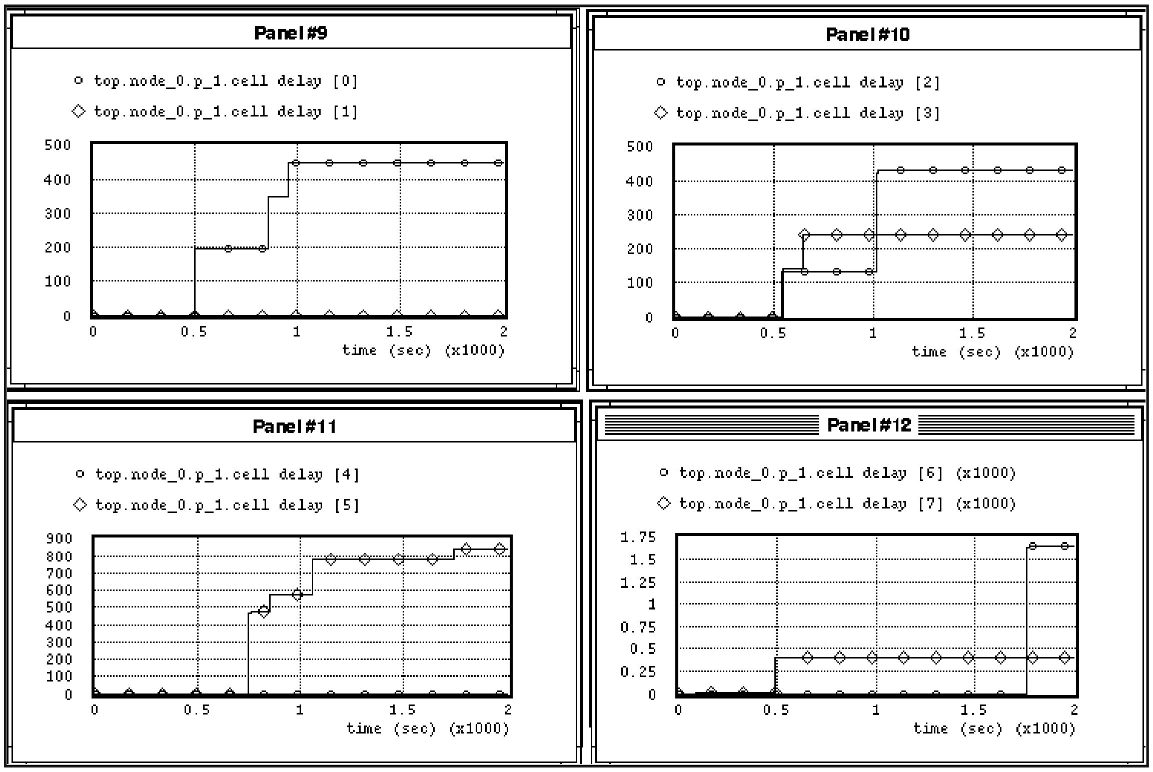


**Figure 12. The queue backlog without flow regulator**

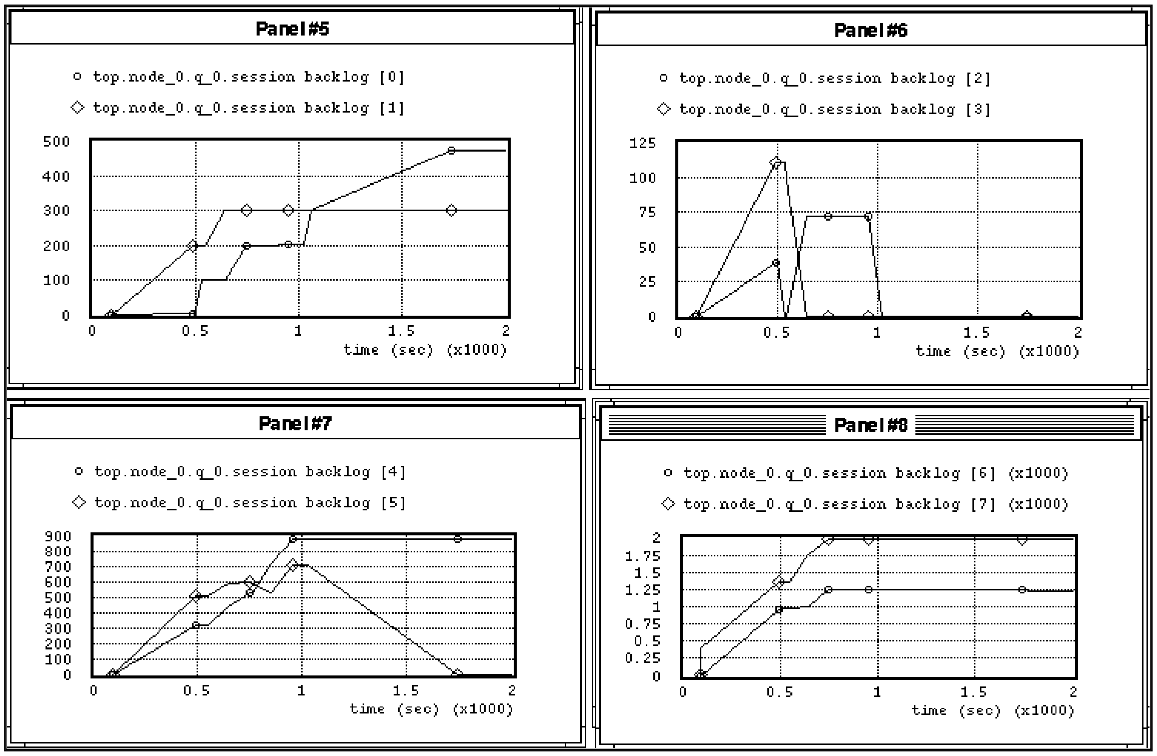


**Figure 13. The leaky-bucket token generation rate (threshold 5) with flow regulator**

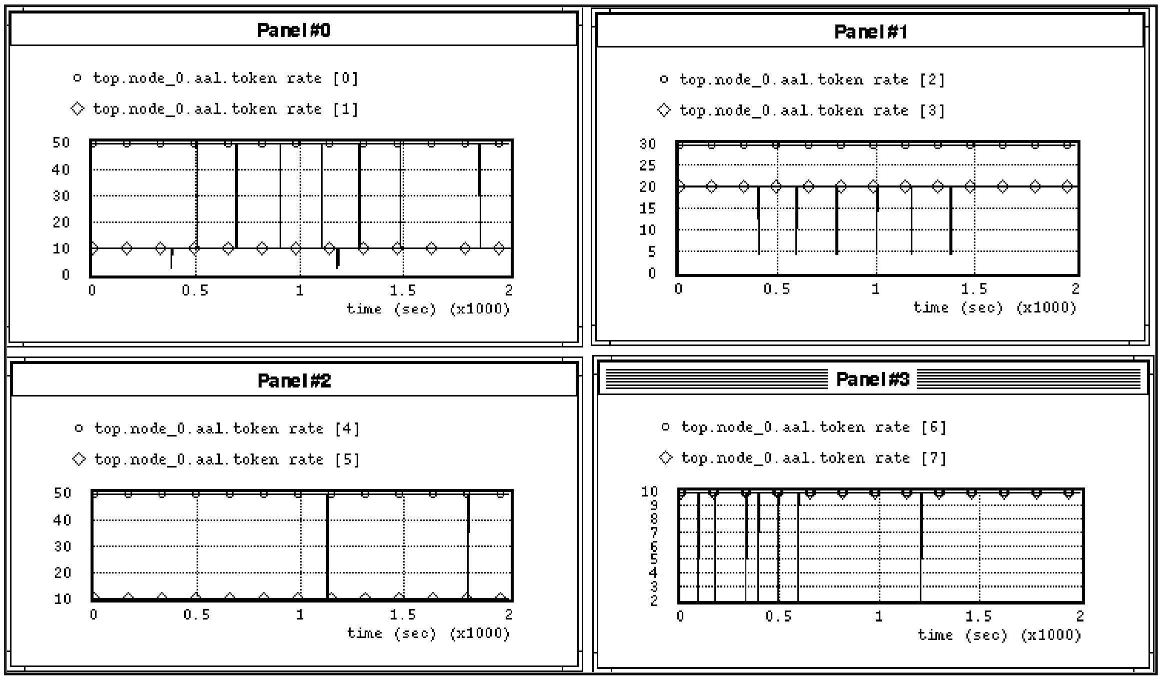


**Figure 14. The session delay at a backbone switch with flow regulator**

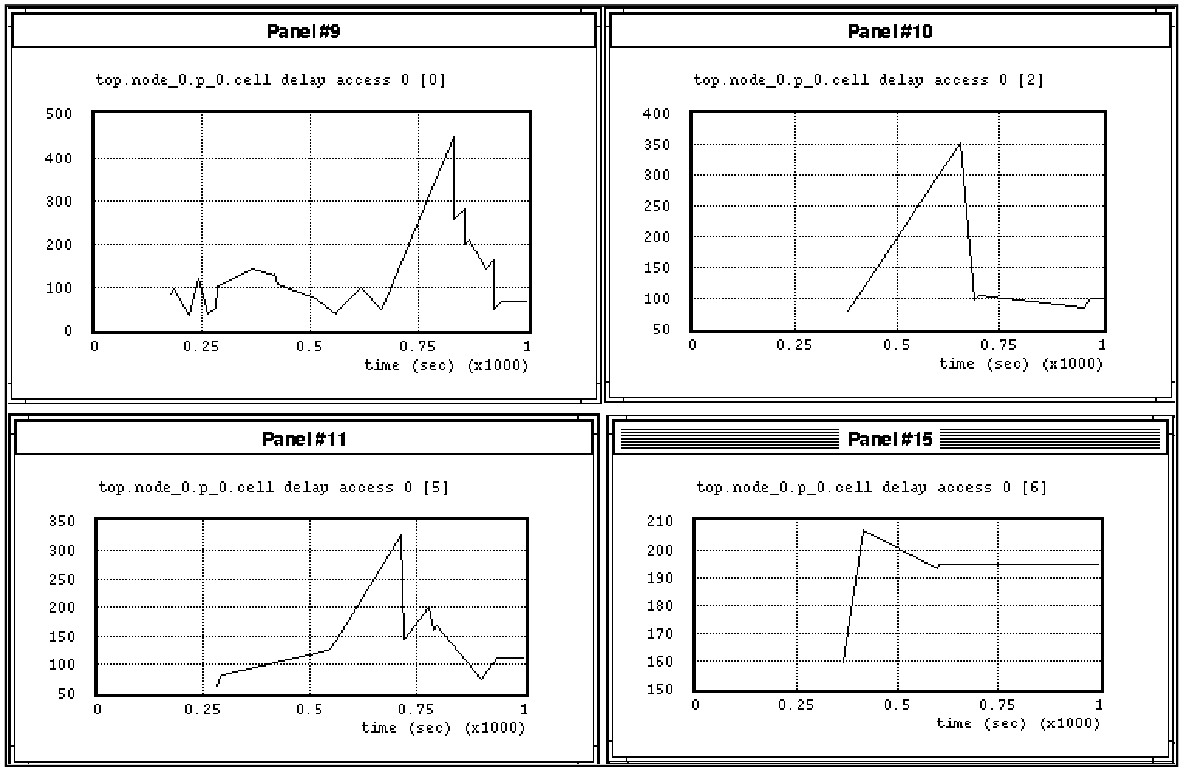
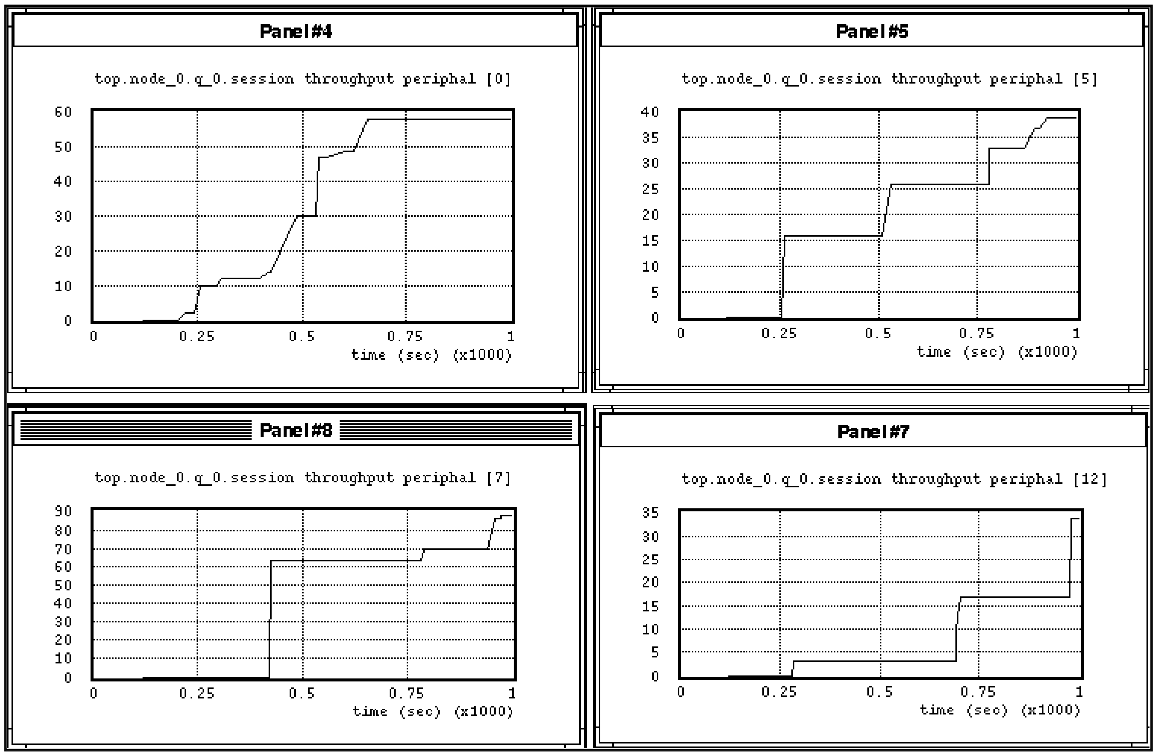


**Figure 15. The session throughput at a backbone switch with flow regulator**


Panel#4
top.node_0.q_0.session throughput periphal [0]
time (sec) (x1000)
Panel#5
top.node_0.q_0.session throughput periphal [5]
time (sec) (x1000)
Panel#8
top.node_0.q_0.session throughput periphal [7]
time (sec) (x1000)
Panel#7
top.node_0.q_0.session throughput periphal [12]
time (sec) (x1000)