Quantitative Study of Differentiated Service Model Using UltraSAN *

Jun Wang, Ying Wang, Klara Nahrstedt †
Department of Computer Science
University of Illinois at Urbana-Champaign
junwang3@cs.uiuc.edu, ywang10@uiuc.edu, klara@cs.uiuc.edu

Abstract

In today's Internet, only best-effort service is provided. With up-coming Quality of Service (QoS) requirements raised by a wide range of communication-intensive, real-time multimedia applications, the best-effort service is no longer sufficient. As a result, Differentiated Service Model (DiffServ) has been proposed as a cost-effective way to provision QoS in the Internet.

In this paper, we first briefly introduce the basic queueing and scheduling schemes in DiffServ. Then we focus on quantitative analysis on the performance of these schemes, in terms of the queueing delay and loss rate. UltraSAN is used as the analyzing tool and simulations are conducted. Our results show that the *Premium* class traffic, which has the highest priority, has significant influence on other classes of traffic.

1 Introduction

In the early 90's, in order to provide Quality of Service (QoS) guarantees in the Internet, the Integrated Service Model (IntServ) was proposed. However, it suffers the scalability problem and is too expensive to deploy. As a result, DiffServ is proposed as a cost-effective and scalable way to achieve QoS instead.

The DiffServ model provisions end-to-end QoS guarantees by using service differentiations and works as follows. Incoming packets are classified and marked into different classes, using Differentiated Services CodePoint (DSCP) [2] (e.g., IPv4 TOS bits or IPv6 Traffic Class bits in a IP header). Complex traffic conditioning such as classification, marking, shaping, and policing are pushed to network edge routers or hosts. Therefore, the functionalities of the core routers are relatively simple - they classify packets according to the DSCP and forward them using corresponding Per-Hop Behaviors (PHBs) [1, 3]. Three service classes have been proposed in the DiffServ model: the premium class, the assured class and the best-effort class [3]. They have different priorities. The premium class has the highest priority while the best-effort class is the lowest one. Figure 1 shows the structure of a DiffServ router.

In order to differentiate these three service classes and enforce the priorities between them, a certain queueing and scheduling scheme is used. Basically, two queues are used. The Premium Service queue (PS-queue) for the premium class and the RIO-queue (Random Early Detection with distinction of In-profile and Out-profile packets) for both assured and best-effort classes. The PS-queue is a simple FIFO queue, while the RIO-queue is more complicated. Figure 2 shows both queues. For the RIO-queue, if the queue length exceeds the threshold B_b, then the incoming best-

*This work was supported by NSF Grant under contract number NSF ANI 00-73802 and NSF CISE Grant under contract number NSF EIA 99-72884. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

†Please address all correspondences to Jun Wang and Klara Nahrstedt at Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, phone: (217) 244-5841, fax: (217) 244-6869.

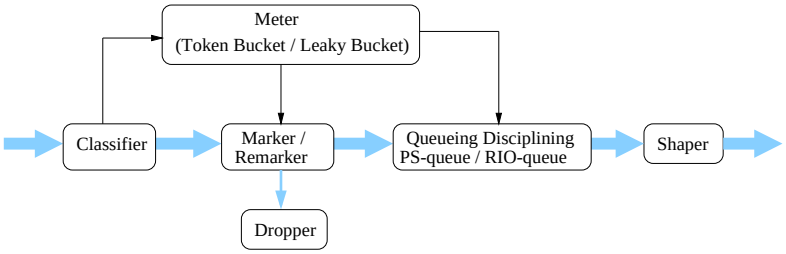


Figure 1: The Structure of a DS Router

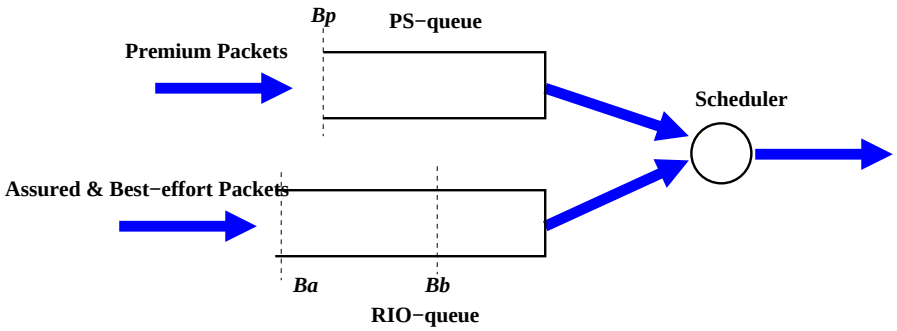


Figure 2: Queueing and scheduling schemes in DiffServ

effort packets begin to drop; if the queue length exceeds the threshold B_a, incoming assured packets begin to drop. By using a smaller B_b, the assured class gets a higher priority over the best-effort one. For the PS-queue, if the queue length exceeds the capacity B_p, the premium packets begin drop. The scheduler uses priority scheduling scheme to fulfill the higher priority of premium class over the other two. Simply speaking, the scheduler will always schedule the premium packets as long as the PS-queue is not empty, i.e., it will schedule the packets from the RIO-queue only when the PS-queue is empty.

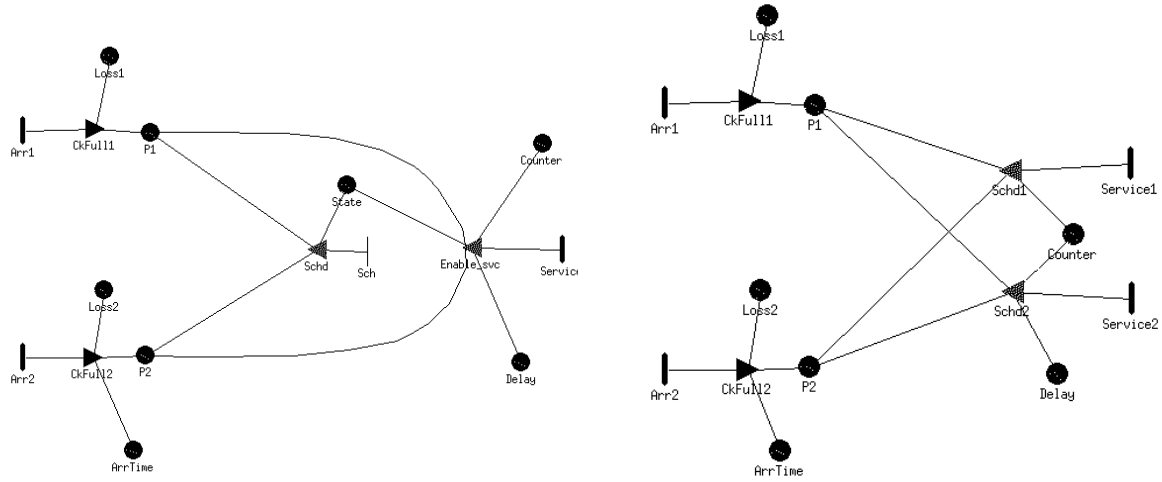
Some papers have already done research on the performance issues for DiffServ model, such as [7, 4]. But in order to get close-form solutions, most of them used simplified assumptions such as Poisson arrivals or exponential distributed service time for packets. And both of them assume a pre-emptive mode for the PS-queue, which is not realistic. Moreover, neither of them combined the three classes together and get overall performance analysis on the whole DiffServ system. So our goal is to enhance the existing studies.

The rest of this paper is organized as follows. Section 2 introduces the design details of SAN models for DiffServ scheme. Several modeling issues are discussed in more details in Section 3. The simulation results are presented and analyzed in Section 4. Finally, Section 5 concludes our work.

2 SAN Models

In this work, we develop five SAN models respectively for 1) Preemptive PS-queue, 2) Non-preemptive PS-queue, 3) RIO-queue with Poisson arrivals and exponential service time, 4) RIO-queue with general assumptions and 5) Combined model of both PS-queue and RIO-queue for all three service classes. During the progress of building these SAN models, we learned some modeling and simulation techniques from [5, 6]. We also encountered some problems. We will introduce them and our solutions to them in Section 3.

Since simulations are used, we can model systems with large queue sizes. For example, in Figure 3, SAN models



(a) Non-preemptive PS-Queue

(b) Preemptive PS-Queue

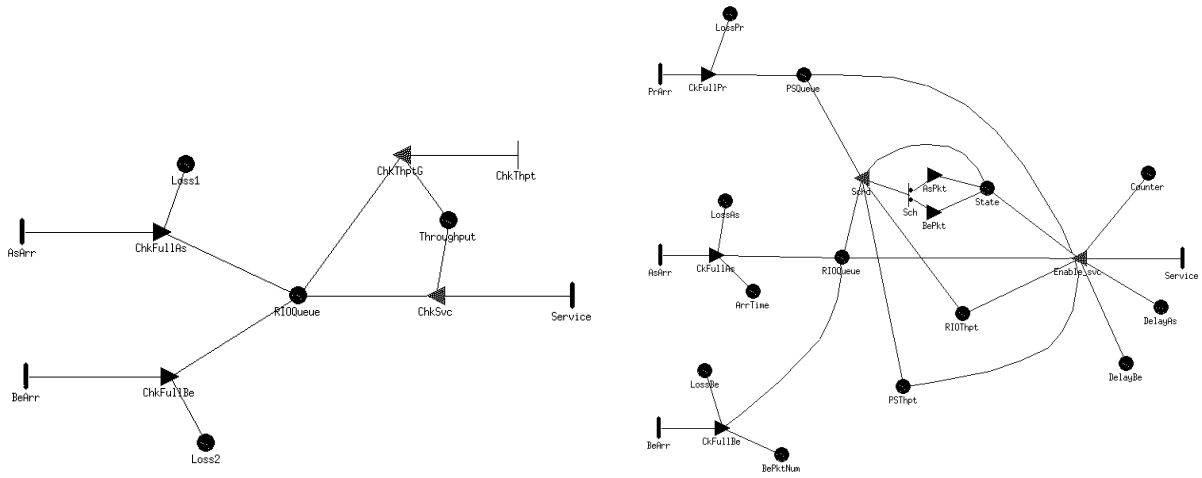
Figure 3: SAN models for non-preemptive PS-queue and preemptive PS-queue

for PS-queue are shown. In both models, the premium queue size is 100 and the best-effort queue size is 1000.

Figure 3(a) gives the SAN model for a non-preemptive PS-queue. As we can see, timed activity “Arr1” and “Arr2” generate premium packets and best-effort packets respectively according to Poisson arrivals. Place “P1” holds the premium (with higher priority) queue and “P2” holds the best-effort (with lower priority) queue. Queue size for “P1” is 100 and queue size for “P2” is 1000. Output gate “CkFull1” is responsible for checking the premium queue. If it finds the premium queue is full, the incoming packet is dropped and the dropping event is recorded in “Loss1”. The similar thing is done on the best-effort queue by “CkFull2” and “Loss2”. In order to enforce the non-preemptive property, packet scheduling is splitted into two parts, implemented by input gates “Schd” and “Enable_svc” as well as instantaneous activity “Sch” and timed activity “Service” respectively. The instantaneous activity “Sch” is enabled only when there is no token in “State” (indicating the server is idle) and there are tokens in “P1” or “P2”. The scheduler always schedules packets in “P1” as long as there are tokens in “P1”, setting “State” to 1 (indicating the server is serving a premium packet). Only when the server is idle, “P1” is empty and “P2” is not empty, does the scheduler schedule a packet from “P2” and set “State” into 2, indicating the server is serving a best-effort packet. When “State” is set to 1 or 2, it disables the scheduler and enables the server through the input gate “Enable_svc”. The timed activity “Service” completes with exponential distributed service time at a normalized rate 1.0. The other three places “Counter”, “ArrTime” and “Delay” are used to record or estimate packet delay for best-effort packets. Notice that while it is easy to get the packet delay for premium packets by applying Little’s Law, it is NOT straightforward to get it for best-effort packets, since the delay of a best-effort packet depends not only on the the condition of the best-effort queue, but also on the condition of the premium queue. So simply applying Little’s Law on the best-effort queue does not work here. This is a tough problem on which we spent a lot of time, since there is no way for us to record the exact arriving time and leaving time for each individual best-effort packet. Finally, we found a tricky way to solve this problem by using these three places. The detailed solution will be introduced in Section 3.

Figure 3(b) gives the SAN model for a preemptive PS-queue. We do not go through the very detail for this model. But we still want to point out that the preemptive property is enforced by using two input gates “Schd1” and “Schd2”. The enable condition of “Schd1” is that “P1” is NOT empty, while the enable condition of “Schd2” is that “P1” IS

empty AND “P2” is NOT empty. Suppose “Schd2” is passed and “Service2” is enabled but not completed, a premium packet comes into “P1”. Then “Schd2” is disabled immediately and “Service2” aborts, while “Service1” is enabled to serve this premium packet without a delay. In this way, the preemptive property is implemented. Again, the packet delay for best-effort packets is estimated by using three place: “ArrTime”, “Counter” and “Delay”.



(a) RIO-Queue

(b) DiffServ Combined Queues

Figure 4: SAN models for RIO-queue and DiffServ combined queues

Figure 4(a) illustrates the SAN model for an RIO-queue. Again, we skip the design details of this model. The tricky problem here is how to measure the effective throughput for the assured traffic and best-effort traffic. We address this problem in Section 3 too.

In Figure 4(b), we combine the PS-queue scheme and RIO-queue scheme together and build a comprehensive model for a DiffServ system. The design details are skipped due to space limitation.

3 Modeling Problems and Solutions

Problem 1: How to model preemptive and non-preemptive queueing models?

Solution: Basically, the features of UltraSAN provide a way to model a preemptive system in nature. This is because when an activity is enabled but not completed, if its enabling condition changes and becomes disabled, it is aborted. This is exactly what we want when a preemptive system is modeled. As illustrated in Figure 3 and in Section 2, we simply use two input gates “Schd1” and “Schd2” in our PS-queue SAN model to enforce the preemptive feature of the queues. When a packet comes into the premium queue “P1”, the input condition of “Schd2” is disabled. Even if the server is serving a best-effort packet, the service will abort automatically, i.e., the best-effort packet in service will be *preempted* by the premium packet just arrived. But for the non-preemptive case, packet service has to be divided into two steps: scheduling and service, as discussed in Section 2. The scheduling is implemented by using an input gate “Schd”, a place “State” and an instantaneous activity “Sch”. They schedule a packet from either queue according to the priority rule and set “State” to 1 or 2 not only to indicate which queue the packet is from, but also to disable the scheduling simultaneously until the packet is completely served and the “State” is reset to 0 by the server. Therefore,

any packet in service can not be preempted before its service completes. In this way, the system is guaranteed to be working in a non-preemptive mode.

Problem 2: How to evaluate response time (delay) for the low priority service class?

Solution: Actually, there is no straightforward solution to this problem, because there is no way for us to record the arriving time and departure time for each individual packet. We even do not know the time. On the other hand, the packet delay for the low priority service class in a PS-queue system can not be directly calculated from the mean number of packets in queue by applying Little's Law, as we do for the high-priority queue, because the delay for a low-priority packet not only depends on the situation in its queue, but also depends on the situation in the high-priority queues. The solution we use in our models and simulations is an approximative one. However, the simulation results show that our approximated solution works well. In our models, as illustrated in Figure 3, three places "ArrTime", "Counter" and "Delay" are used to solve the problem. "Counter" is a virtual clock which keeps track of time in terms of number of packets served. It increases by 1 each time when a packet gets served. But if there is no packet waiting in the best-effort queue "P2", it is reset to 0 and increase from 0 again (we do this in order to avoid its token from overflow). The output gate "CkFull2" also checks the place "ArrTime". It set the "ArrTime" to the value of "Counter" only when the original value of "ArrTime" is 0. "ArrTime" will be reset to 0 by the server only when a best-effort packet is served AND the best-effort queue is empty. In this way, we can record the exact arriving time of the first packet in a bulk of incoming best-effort packets.

$$v_{i+1} = v_i + \frac{c_i - v_i}{n_i + 1} \tag{1}$$

$$d_i = c_i - v_i \tag{2}$$

Having the departure time of the current packet c_i and the number of packets in the queue n_i, we can estimate the next packet's arriving time v_{i+1} as Equation 1, where v_i is estimated arriving time of the current packet. Then the packet delay d_i can be calculated in Equation 2. For example, as shown in Figure 5, (t_0, t_1, t_2, t_3) are actual arriving times

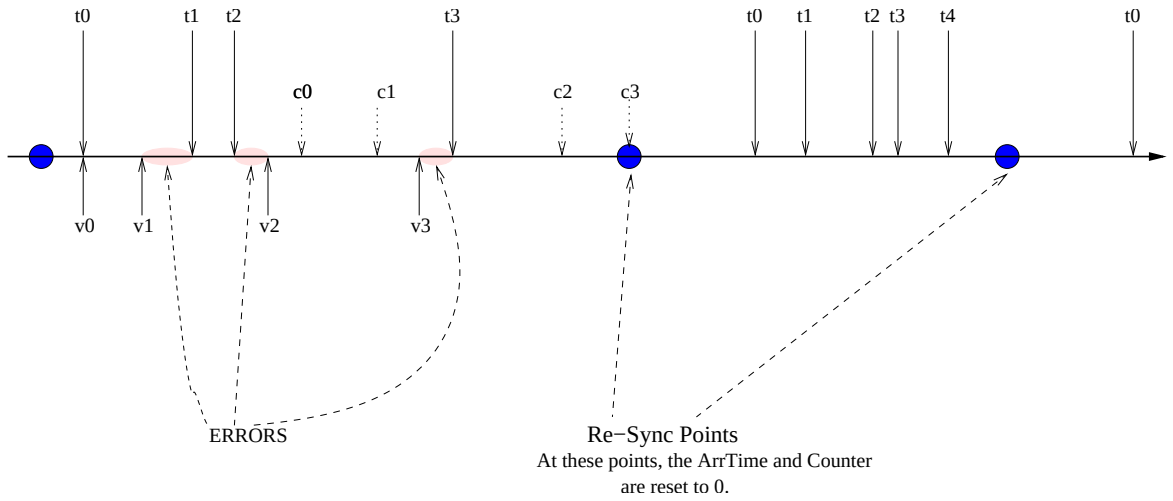


Figure 5: Estimate arrival times by using virtual clock

for sequential 4 packets, (c_0, c_1, c_2, c_3) are the actual departure times and (v_0, v_1, v_2, v_3) are the estimated arriving times according to Equation 1. And the packet delay can be calculated by applying Equation 2. As we can see, not surprisingly, errors exist. Fortunately, we have *Re-Sync Points (where "ArrTime" and "Counter" are reset to 0)* to

re-synchronize virtual arriving times to real arriving times, which helps to reduce the error. We hope that after a long run, the estimated delay values will converge to the real ones. The simulation results presented in Section 4 show that this estimation method is valid.

Another problem we encounter during solving this problem is that when the virtual time increases, the tokens in "Counter" may overflow. We believe that in Mobius, $Counter \rightarrow Mark()$ is implemented as an integer or short which can not hold large number of tokens. Since there is no way to declare local or temporary variables, we have to use global ones. Details are provided in Section 4.

We also use the similar method to measure the packet delay for both the assured traffic and the best-effort traffic in the DiffServ model which combines all three service classes.

Problem 3: How to get effective throughput for assured traffic and best-effort traffic in the RIO-queue model?

Solution: In our RIO-queue SAN model, it is not difficult to get the overall effective throughput for the combined traffic of both classes, say, ϕ. Meanwhile, we can easily measure the loss probability for each class. Assume the loss probabilities of assured packets and best-effort packets are q_a and q_b respectively. Given the offered load λ and the fraction of assured packets p, we can compute the effective throughput for individual class traffic as follows

$$\phi_a = \frac{p(1-q_a)}{p(1-q_a)+(1-p)(1-q_b)} \cdot \phi \tag{3}$$

$$\phi_b = \frac{(1-p)(1-q_b)}{p(1-q_a)+(1-p)(1-q_b)} \cdot \phi \tag{4}$$

where ϕ_a and ϕ_b are effective throughput for assured class and best-effort class respectively. The simulation results shown in Section 4 verify the correctness of the above equations since the simulation results nicely match the theoretic results.

A Suggestion: In Mobius, we can provide a way to declare local or temporary variables in simulations, which will facilitate our modeling a lot.

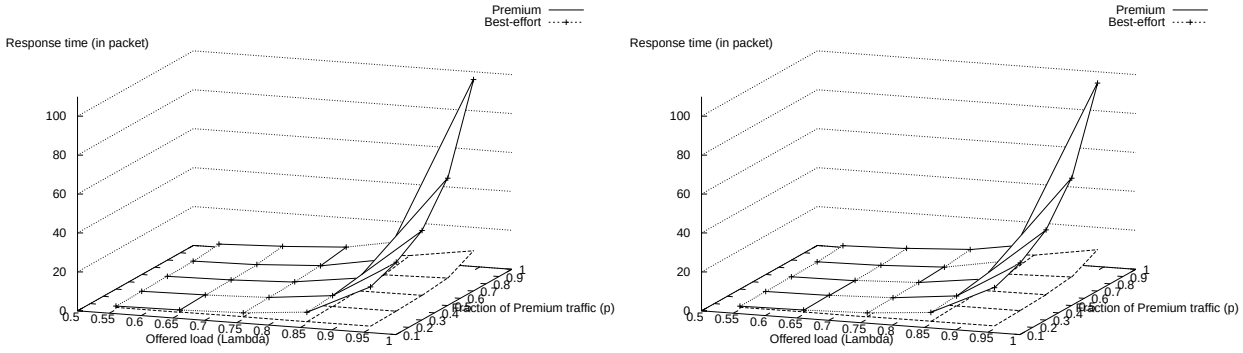
4 Experiments and Results

Basically we develop SAN models respectively for 1) PS-queue, 2) RIO-queue and 3) Combined model of both PS-queue and RIO-queue for all three service classes. In the first case (PS-queue), performance is evaluated in terms of the response time (queueing delay plus service time), and the loss probability. In the second case (RIO-queue), effective throughput for both assured class and best-effort class are measured, as well as the loss probabilities. In the third case (combined model), delay and loss probability are evaluated. For PS-queue and RIO-queue, we also carry out simulations using some general distributions for the arrival process and the service time, other than Poisson process and exponential distribution. Results are analyzed and compared with each other.

4.1 PS-Queue

For PS-queue, we develop two different models for preemptive model and non-preemptive model respectively. In [4] and [7], only the preemptive model was used and analyzed. But in reality, most of the switches and routers in the Internet are using non-preemptive model. Thus we model both scheme and compare the difference between them. The results show that *in normal cases the difference between them is negligible*.

4.1.1 Preemptive vs Non-Preemptive



(a) Preemptive

(b) Non-Preemptive

Figure 6: Mean response time (packet delay) for preemptive and non-preemptive PS-queues

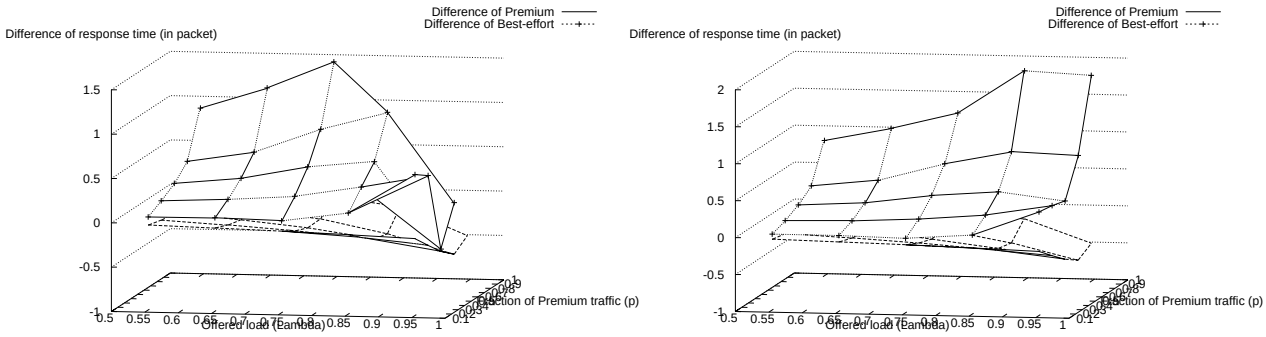
The simulation results are demonstrated in Figure 6(a) for the preemptive PS-queue scheme. The size of the high priority queue is 100 and the size of the low priority queue is 1000. Service time is normalized to 1. The total offered load λ varies from 0.55 to 0.95 on the y-axis, the fraction of premium class traffic p varies from 0 to 1 on the x-axis. Not surprisingly, the mean response time (delay) for premium packets is always smaller than the delay experienced with best-effort packets. And when the offered load is high and the fraction of premium traffic is high, the best-effort packets suffer very large delay and the difference between two classes increases. The reason behind it is that when the offered load is high and the fraction of premium traffic is high, the best-effort packets have to wait for and even preempted by more number of premium packets. Comparing this results with the results in [4], we find that they totally agree.

Figure 6(b) shows the results for a non-preemptive system. Not surprisingly, the results are almost the same as those of a preemptive system. Comparing Figure 6(b) and Figure 6(a), it is difficult to tell the difference between them. In the following subsection, we will plot the real difference between them by subtracting the delay values of the non-preemptive system from their counterpart values of the preemptive system.

4.1.2 Comparison

The figure on the left hand side shows the delay difference between the preemptive and non-preemptive schemes by using the *Place* method and the figure on the right hand side illustrate the same result but using the *Global Variable* method. We can see that the left hand side graph is surprising or even unreasonable because the delay difference suddenly drops when offered load and fraction of premium traffic increase to some level. Actually, this was exactly the reason we suspected the results we got by using the *Place* method and re-built the model by using the *Global Variable* method. The right hand side graph looks more reasonable, which implies that the *Global Variable* method is better than the *Place* method. The reason why the *Place* method does not work well is that when the offered load is high, the number of tokens held in the "Counter" place may be too large and incurs "Counter→Mark()" to overflow.

From the results we can validate that the difference between *preemptive* and *non-preemptive* is not significant. Even when the offered load is high and the fraction of premium traffic is high, although the absolute value of difference goes



(a) Place Method

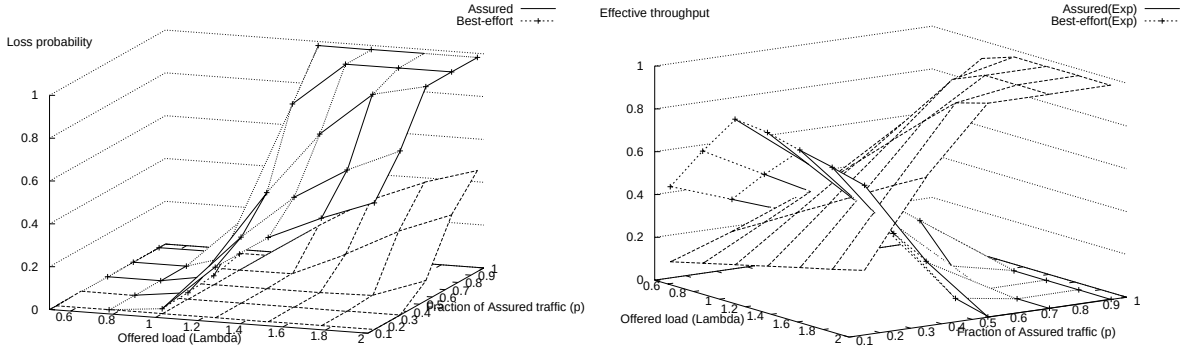
(b) Global Variable Method

Figure 7: Difference of mean response time between preemptive and non-preemptive PS-queues

up, but compared with the delay value itself, the difference accounts for only very small part of it.

From both graphs we find that the premium packet delay of a preemptive system is smaller than that of a non-preemptive system (the delay difference is negative). On the other hand, the best-effort packet in a preemptive system suffers larger delay than that in a non-preemptive system. This agrees with our expectation and proves from another perspective that our models are correctly built.

4.2 RIO-Queue



(a) Packet Loss Probability

(b) Effective Throughput

Figure 8: Loss Probability and Effective Throughput for Assured and Best-effort Traffic in a RIO-queue (With Exponential Service Time)

In this subsection, we focus on the performance measurement of the RIO-queue with respect to packet loss

probability and effective throughput. In order to validate the use of Poisson arrival and the exponential service time, more general arrival processes and service time distributions are tested. Results are compared and analyzed. We concentrate on the study of service time instead of arrivals. Several non-exponential distributed service time models are tested, such as deterministic service time model, uniformly distributed service time model Erlang distributed service time model etc. The results show that various service time models have very limited impact on the system performance in terms of the loss probability. Therefore, we prove that the traditional analysis based on the assumption of exponential service time are valid. In [4], the authors proved that different arrival processes (such as On/Off Exponential model and On/Off Pareto model) have small impact on overall system performance too. Due to lack of time and space, we did not carry out complete sets of simulations to study the impact of various general arrivals. But we did conduct some individual simulations and the results match what is given in [4]. We do not present such individual results in this report.

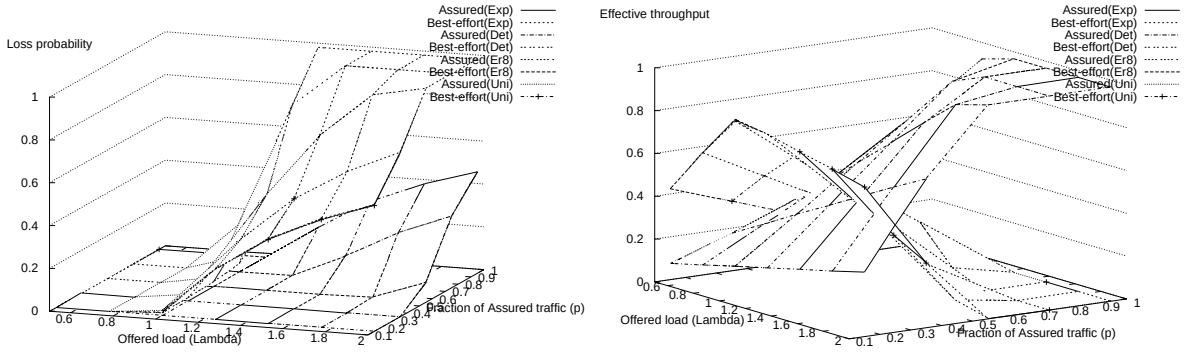
4.2.1 Exponential Service Time

Figure 8(a) illustrates the packet loss probability for assured traffic and best-effort traffic, with exponential service time. The top surface is for the best-effort traffic and the bottom surface is for the assured class. As we can see from the graph, for both assured and best-effort traffic, the packet loss probabilities increase when the offered load λ goes up and / or p goes up. The loss probability for best-effort traffic increases more rapidly than assured traffic does.

Figure 8(b) shows the effective throughput for assured traffic and best-effort traffic, with exponential service time.

Both the loss probability results and the effective throughput results perfectly agree with our expectation and are consistent with the results presented in [4].

4.2.2 General Service Time and Comparison



(a) Packet Loss Probability

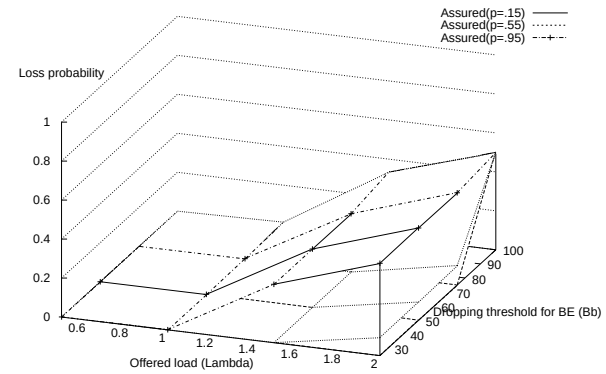
(b) Effective Throughput

Figure 9: Loss Probability and Effective Throughput for Assured and Best-effort Traffic in a RIO-queue (Comparison With General Service Time)

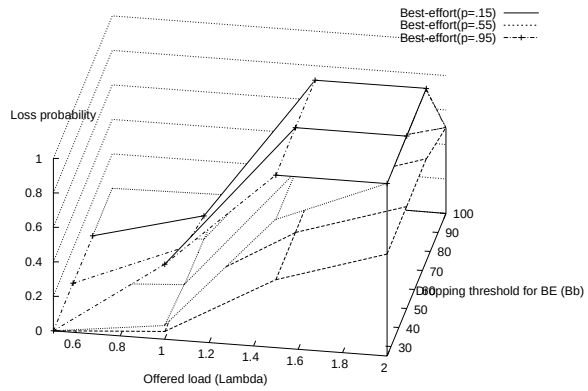
Figure 9 shows the packet loss probability and the effective throughput for both assured traffic and best-effort traffic under different service time assumptions. As we can observe from the graph, all the surfaces for the assured traffic are so close together that they are almost indistinguishable. So are the best-effort ones. From these results we

can conclude that both arrival process and distribution of service time have limited impact on the system performance in terms of the loss probability and effective throughput. So almost all the traditional theoretical and analytical results under Poisson and exponential assumptions are still useful in a more realistic environment.

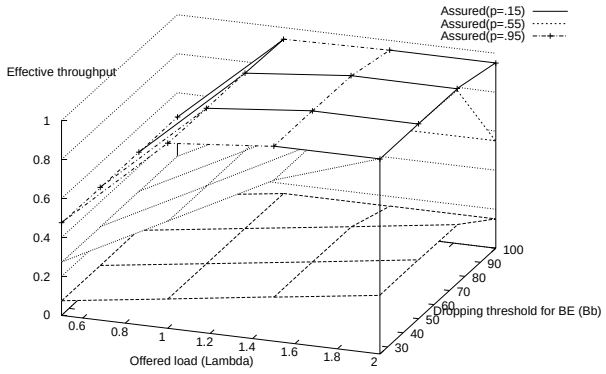
4.2.3 Tuning the Dropping Thresholds for Best-Effort B_b



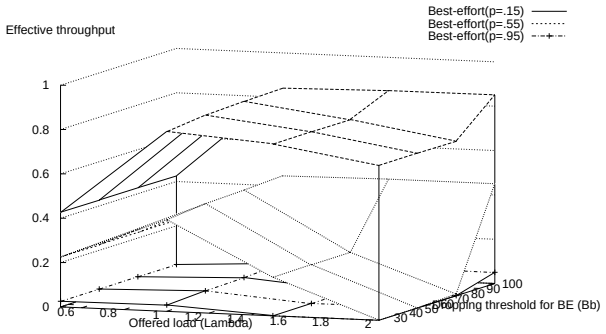
(a) Loss Probability for Assured Class



(b) Loss Probability for Best-Effort Class



(c) Effective Throughput for Assured Class



(d) Effective Throughput for Best-Effort Class

Figure 10: Loss probability and effective throughput vs dropping threshold B_b for a RIO-queue (Comparison With p)

There are some important parameters in DiffServ queueing system, such as the dropping threshold for best-effort traffic in the RIO-queue. By applying different values to these parameters, different QoS performance measures are observed. In this set of simulations, we vary the best-effort traffic dropping threshold B_b and study the impact on QoS performance measures.

The upper two figures in Figure 10 show the packet loss probability with respect to the offered load and best-effort traffic dropping threshold B_b. Figure 10(a) is for assured class and Figure 10(b) is for best-effort class respectively. Several surfaces are shown by varying the value of p (fraction of assured traffic).

The lower two figures in Figure 10 depicts the effective throughput with respect to the offered load and best-effort traffic dropping threshold B_b. Figure 10(c) is for assured class and Figure 10(d) is for best-effort class respectively. Several surfaces are shown by varying the value of p too.

From all above graphs, we can see clearly that by tuning the threshold B_b, different QoS requirements (such as drop probability and effective throughput) can be met. How B_b affects the performance not only depends on the value of itself, but also depends on the network dynamics such as the current offered load λ and the fraction of the traffic from each class. From the results we conclude that in order to provision certain QoS guarantees, it is very necessary for a network administrator to take all three classes traffic into consideration and dynamically adjust the parameters according to the network dynamics.

4.3 Combined Queueing Model for DiffServ

In this subsection, we conduct simulations to study the overall performance of DiffServ system by combining both PS-queue and RIO-queue together, therefore, all three service classes (premium, assured and best-effort) are taken into consideration.

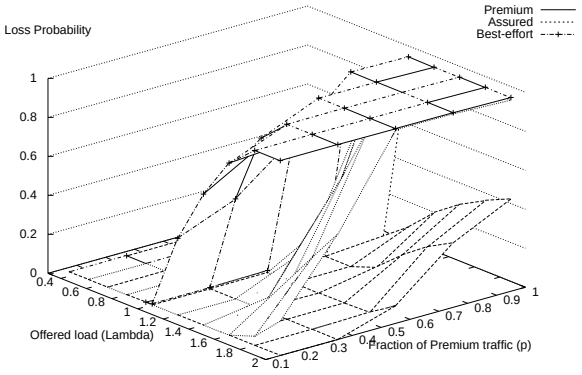
In this set of simulations, we set the maximum queue size for PS-queue and RIO-queue to 100 and 1000 respectively. And the best-effort dropping threshold B_b is set to 500 (half of the RIO-queue size). Here p stands for the fraction of premium class traffic. The assured and best-effort traffic share the rest of the bandwidth equally, i.e., if premium class traffic accounts for p fraction of total bandwidth, the assured and best-effort traffic will occupy $(1-p)/2$ of total bandwidth each.

Figure 11 depicts packet loss probability, effective throughput and packet delay for all three classes. Figure 11(a) shows the packet loss probability surfaces. We can observe that as offered load λ increases, best-effort traffic experiences the highest drop probability. Next is the assured class. Since the premium class packets have the highest priority, they do not have very high drop probability even when λ reaches 2.0. Figure 11(b) presents the effective throughput for three classes. As we can see, when the traffic load is light in the network, the best-effort traffic enjoys high effective throughput. But as the offered load goes high and the fraction of higher priority traffic increases, best-effort effective throughput deteriorates very quickly. The measure of the packet delay is depicted in Figure 11(c). The top surface is for the best-effort traffic, the middle one is for the assured packets. Compared to the delay experienced by these two classes, the packet delay for the premium class is negligible (the surface is almost on the ground in the figure). If we zoom into the interval between $\lambda = (0.3, 1.0)$, as elaborated in Figure 11(d), we can find that in this interval, the assured packets and best-effort packets have almost the same packet delay. And the graph is almost the same as the result for the PS-queue (Figure 6). While during the $\lambda = (1.0, 2.0)$ interval, surprisingly, the best-effort packets experience smaller delay than the assured ones. The reason is that when λ increases, the RIO-queue becomes longer. Since the best-effort dropping threshold B_b has been set to 500 (half of the size of the RIO-queue), more and more best-effort packets get dropped. But for those admitted best-effort packets, since the queue they entered has only half size of the whole RIO-queue, they should have smaller packet delay. Therefore, the results are reasonable.

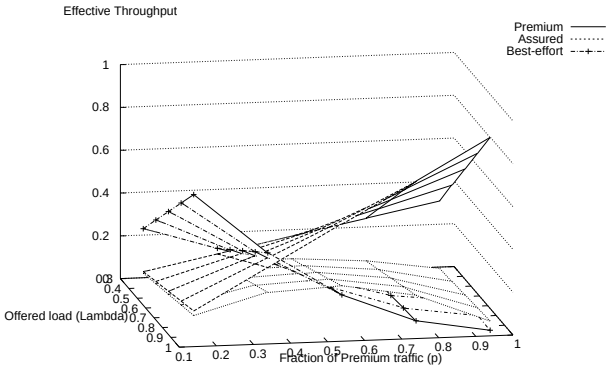
As we can see, the comprehensive SAN model and simulations for DiffServ can give us useful guidance on QoS performance measures for the overall system, as well as the inter-class effects and relationships between these service classes.

5 Conclusion

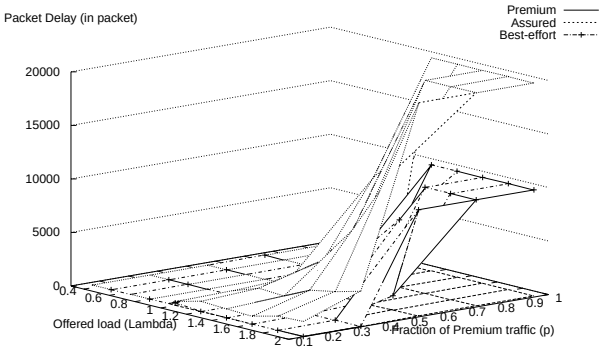
In this work, we modeled and studied five different SAN models for PS-queue, RIO-queue and the combined



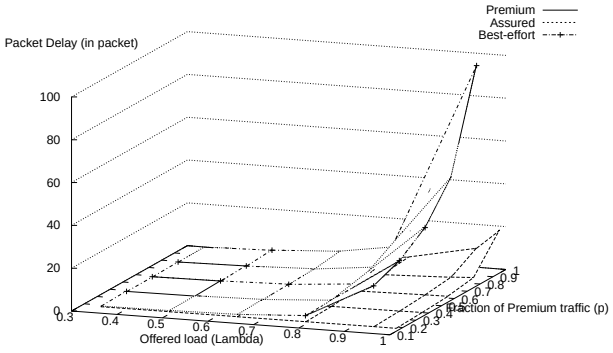
(a) Packet Loss Probability



(b) Effective Throughput



(c) Packet Delay (in packet, large scale)



(d) Packet Delay (in packet, small scale)

Figure 11: QoS performance measures for all three classes in a DiffServ system

DiffServ model. Simulations were used to study the performance of these queueing systems in terms of packet response time (delay), effective throughput and packet loss probability. All the results agreed with our expectation and the theoretical or analytical results provided in the literatures. By carefully digesting the results we have got, we found that in a DiffServ domain, the inter-class effects are significant. Therefore, it is very necessary for a network administrator to take all three classes traffic into consideration. Parameters (such as the fraction of each class, and the dropping thresholds for the RIO-queue) can be tuned according to different QoS requirements for different service classes.

References

[1] S.Blake et. al. An Architecture for Differentiated Services. *RFC 2475*, December 1998.

[2] F.Baker, D.Black, S.Blake, and K.Nichols. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. *RFC 2474*, December 1998.

[3] K.Nichols, V.Jacobson, and L.Zhang. A Two-bit Differentiated Services Architecture for the Internet. *RFC 2638*, July 1999.

[4] M.May, J.C.Bolot, C.Diot, and A.Jean-Marie. Simple Performance Models of Differentiated Services Schemes for the Internet. In *Proceedings of IEEE Infocom 1999*, March 1999.

[5] Jay Moorman. *Modeling of the Multiclass Priority Fair Queuing (MPFQ) Algorithm*. University of Illinois, http://ipoint.vlsi.uiuc.edu/wireless/papers.html, 1999.

[6] Jay Moorman. *Simulation of the Multiclass Priority Fair Queuing (MPFQ) Algorithm*. University of Illinois, http://ipoint.vlsi.uiuc.edu/wireless/papers.html, 1999.

[7] S.Sahu, D.Towsley, and J.Kurose. A Quantitative Study of Differentiated Services for the Internet. In *Proceedings of IEEE Global Internet'99, Rio de Janeiro, Brazil*, December 1999.

6 Appendix: Some Algorithms Implemented in the DiffServ SAN Model

We only list the code for "Enable_svc" input gate in the DiffServ SAN model, where priorities for different service classes are enforced and packet delay is traced. For detailed code of other parts of the DiffServ SAN model or other SAN models, please refer to the corresponding SAN models in Mobius directly.

Program 6.1 The function code for the input gate "Enable_svc" in the DiffServ SAN model

```
cnt++;
if (RIOQueue->Mark() == 0) {
  cnt = 0;
  arra = arrb = 0;
}
PSThpt->Mark() = RIOThpt->Mark() = 0;

if (State->Mark() == 1) {  // The packet was from PSQueue
    PSQueue->Mark()--;
} else  if (State->Mark() == 2) {  // The packet was Assured
    RIOQueue->Mark()--;
    DelayAs->Mark() = (int)(cnt - arra + 0.5);
    if (RIOQueue->Mark() == 0) {  // RIOQueue is empty
        cnt = 0;
        arra = arrb = 0;
    } else {  // RIOQueue is not empty, estimate arrtime for next pkt
        if (RIOQueue->Mark() <= BePktNum->Mark()) { // There is no assured pkt i
n the queue
            arra = 0;
        } else {
            arra += (cnt - arra) / (RIOQueue->Mark() - BePktNum->Mark() + 1);
        }
    }
} else { // The packet is best-effort
    RIOQueue->Mark()--;
    BePktNum->Mark()--;
    DelayBe->Mark() = (int)(cnt - arrb + 0.5);
    if (RIOQueue->Mark() == 0) {  // RIOQueue is empty
        cnt = 0;
        arra = arrb = 0;
    } else  if (BePktNum->Mark() == 0) {  // RIOQueue is not empty, but there is
 no besteffort pkt in itestimate arrtime for next pkt
        arrb = 0;
    } else {  // there are besteffort pkt in the queue
        arrb += (cnt - arrb) / (BePktNum->Mark() + 1);
    }
}
```