Enhancing the DiffServ Architecture of a Simulation Environment

Christos Bouras[1,2], Dimitrios Primpas[2], Afrodite Sevasti[1,2], Andreas Varnavas[2]
[1]*Research Academic Computer Technology Institute-RACTI, Kolokotroni 3, 26221 Patras, Greece*
[2]*Dept. of Computer Engineering and Informatics, University of Patras, 26500, Patras, Greece*
{bouras, sevastia}@cti.gr

Abstract

Simulation has always been a valuable tool for experimentation and validation of models, architectures and mechanisms in the field of networking. In the case of the DiffServ framework, simulation is even more valuable, due to the fact that an analytical approach of mechanisms and services is infeasible because of the aggregation and multiplexing of flows. In this work, we have extended the functionality of a widely used simulation environment towards the direction of realistic traffic generation and a series of mechanisms defined by the DiffServ architecture. The modules created are being presented and a case study of a simulation scenario that exploits the functionality provided by them is described.

1 Introduction

The importance of simulating environments to research conducted in the area of computer and telecommunication systems is undoubted. A series of networking protocols and models have been studied on and evaluated in simulation environments, before tested in practice and delivered to production settings. In the past years, a lot of research work on telecommunications and more specifically in contemporary, transport or backbone IP networks has focused on the introduction and exploitation of the DiffServ framework towards the direction of building advanced networking services as an evolution to the traditional best-effort model.

The DiffServ framework ([17]) proposes the provision of service differentiation to traffic in a scalable manner, by suggesting the aggregation of individual application flows with similar quality needs. It introduces the definition of different service classes to which such aggregates are appointed and the implementation of mechanisms for differential treatment by network elements of the packets belonging to each service class. In compliance with the aggregation model, packets entitled to a certain service or belonging to a specific aggregate are marked with a distinctive value of the so-called Differentiated Services CodePoint (DSCP), a single packet header field, on which routers are based for providing differentiated services. In this work we are presenting a series of modules built on one of the most widely used simulation platforms, the implementation of which provides several tools for efficient simulation of DiffServ mechanisms and DiffServ-based services in exceptionally realistic conditions.

The modules presented here have been built within the environment of the ns-2 simulator ([12]) and comprise self-contained components, each one of which provides an additional traffic generation or DiffServ mechanism functionality ([16]). There are a number of groups working on the ns-2 simulation platform worldwide. The DiffServ functionality supported b the current version of ns-2 is described in [18]. In [1], a module for the functionality of Weighted Fair Queuing (WFQ) in the ns-2 environment is provided. In [2], an implementation of MPLS functionality is provided for the ns-2 platform.

In [3], in the framework of testing a proposed measurement based admission control algorithm, a number of realistic source models are implemented. The authors anticipate for long-range dependence (LRD) of network traffic. They present two types of LRD traffic simulation models, one based on Pareto ON/OFF processes, the superposition of which generates LRD series, and another one based on the fractional autoregressive integrated moving average process for the calculation of the number of fixed-sized packets to be sent back-to-back in each ON period of an ON/OFF source. According to relevant bibliography, the superposition of sources of the latter model effectively simulates aggregated VBR video traffic.

Our approach differs from this one, because it anticipates for simulation of realistic aggregated background traffic, while at the same time providing modules for the generation of isolated flows in what we call ‘foreground traffic’. Thus, we propose the use of

distinguishable traffic flows that need to be individually monitored in a DiffServ environment, in order to observe how DiffServ mechanisms affect the characteristics of traffic and quality per application instead of per aggregate. Aggregated traffic in our model is only simulated to act in the background, in order to efficiently reproduce a realistic DiffServ environment where best-effort traffic co-exists with quality-demanding flows.

The 'cross traffic' model of inserting traffic in a simulation topology is presented in [13]. The module for realistic background traffic generation that we will present in the sequel, can be used to produce 'cross traffic', taking as input parameters two adjacent nodes of a topology and generating a realistic traffic mix on the link between these nodes.

In [4], two modules for the generation of self-similar network traffic (behaviour observed in network traffic according to which the autocorrelation between different times is very large) are presented. From these two approaches, we have chosen to adopt the superposition of fractal renewal processes (SupFRP) methodology in order to reproduce the self-similarity of HTTP traffic in our traffic generation module. In [5] the authors present the characteristics of Internet traffic, from the perspective of the traffic mix on the links of backbone networks. Finally, it must be noted that in [12], several modules on DiffServ functionality are provided as a contribution by several research teams worldwide.

In this paper, we present a realistic background and foreground traffic simulation module in section 2 while section 3 outlines a series of DiffServ mechanisms' simulation modules. We also outline a case study for the exploitation of the simulation environment in section 4. The paper is concluded with our intended future work in this area.

2 Traffic generation

As part of our extensions to the ns-2 simulator, we implemented modules for the simulation of:

- Background traffic, simulating the aggregated IP traffic existing in the backbone links of a MAN/WAN.
- Foreground traffic, simulating individual flows or a traffic mixture of bandwidth and delay critical IP multimedia applications, the latter created by multiplexing flows belonging to simulated VoIP transmissions, streaming video and H.323 conferencing.

2.1 Background traffic

Simulated background traffic corresponds to the four basic types of traffic in a real network, SMTP, FTP, TELNET and HTTP. In order for the background traffic to be produced, the topology depicted in Figure 1 was used.

The traffic sources depicted in Figure 1 are being dynamically created on each node and the resulting mixture of packets on the backbone link adheres to relevant bibliography ([5]), as presented in Table 1.

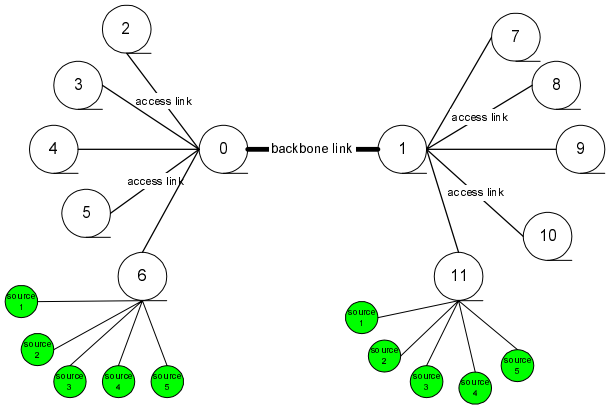


Figure 1. Topology for background traffic generation

The mixture of traffic preserves a relation of a%-(100-a)% between TCP and UDP packets. This relation is configurable and is implemented by selecting the type of traffic generated by each source created and attached to a node through a uniform random variable.

Table 1. Synthesis of background traffic

Traffic type	Packets' percentage
FTP	32-34.4%
HTTP	21-22%
SMTP	13-18%
TELNET	27-32%

The synthesis of packet sizes is implemented in such a way that it resembles the mixed Internet traffic as defined in [7] (50 % of 40-byte packets, 33 % of 552-byte packets and 9 % of larger packets, with maximum packet size that is configurable), again by selecting the packet size of traffic generated by each source created and attached to a node through a uniform random variable. TCP aggregated traffic was simulated with FTP data transmissions ([8]), while the UDP aggregated traffic was reproduced according to the distributions of [5], presented by Table 2.

Table 2. Background traffic simulation of inter-packet transmission times

Traffic type	Simulation of packet inter-arrival times
SMTP	Pareto distribution
FTP	Pareto distribution
Telnet	Pareto distribution
HTTP	Self-similar (Superposition of fractal processes-SupFRP)

The module works as follows: A configurable number of peripheral nodes are created and connected on each of

the two backbone link nodes (Figure 1). On each peripheral node, a configurable number of agents (TCP or UDP) for every type of traffic (SMTP, FTP etc) and a source for every agent are attached. Sources are modelled according to the ON/OFF model.

A number of input parameters are required for the module's operation. The major ones are the duration of the simulation (in seconds), the bandwidth (bw) and propagation delay (pd) attributes of the backbone ($bw_{backbone}$, $pd_{backbone}$) and access links (bw_{access_i}, pd_{access_i}) as well as the transmission rates of the TCP and UDP sources. In order for the load on the backbone link to be adjustable according to the transmission rates of the TCP and UDP sources and not limited by the access links' capacities, it is suggested that input parameters are such that:

$$bw_{backbone} < bw_{access_i}, \forall i$$

Based on the input parameters, the TCP window for all TCP agents created during the simulation time is set to:

$$round(\frac{2\times pd_{backbone}\times bw_{backbone}}{8\times TCPpktsize_{\max}})$$

Thus, the TCP window of all TCP agents used is configured to follow relative recommendations for optimal performance ([9]).

The selection of sink nodes for each source is made by a random number generator. When a source is 'active' or 'ON', it is scheduled to produce packets according to one of the distributions for packet inter-arrival times of Table 2. The transmission duration and the intervals between transmissions ('OFF' intervals) for all sources are produced according to the distributions presented in Table 3.

Table 3. Background traffic simulation distributions

Traffic type	Transmission duration	Interval between transmissions
SMTP	Log-normal	Exponential
FTP	Exponential	Exponential
TELNET	Log-normal	Exponential
HTTP	Exponential	Exponential

These distributions are based on the implementation of [6] with minor differences mainly in the log-normal distribution implementation. A similar set of distributions' table is presented in [5], however in this work active state intervals are calculated according to the log-normal distribution for all types of traffic.

Focusing on the implementation, four procedures, one for each type of traffic (SMTP, FTP, TELNET and HTTP), have been implemented. Each one of them is responsible for creating traffic sources for the corresponding traffic type, defining the transport protocol used (TCP or UDP) and the produced packets' size, creating the required agent to which the traffic source is attached and attaching the agent to a peripheral node of the topology. Finally, the traffic source is triggered to start generating packets towards an elected the sink. The input parameters for each of these procedures are the pointers to the source and sink nodes as well as and the duration of simulation.

Operation of sources is controlled by another family of procedures, one for each traffic type, that are responsible for defining the details of transmission for each traffic source. More specifically, they define the idle and active state intervals for each source, according to the distributions of Table 3, as well as the packet inter-arrival times when each source is active according to Table 2. It is here that the destination node or sink for each source's transmission is randomly selected from the group of nodes on the other end of the backbone link to which the current source is attached. The selected sink differs between consecutive active states of the source. These two groups of procedures presented are mainly responsible for traffic generation over the already described topology.

Before concluding with the implementation of background traffic, some differences in the simulation of SMTP traffic are briefly mentioned. Since SMTP-based traffic has different characteristics than the other types of traffic (exchange of smaller data quantities of a 50KB average size with -usually- more than one recipient) the implementation anticipates for shorter active state intervals for SMTP sources, equal to the time required to transmit a 50Kb mail message over a link with bandwidth $bw_{backbone}$, and for multiple sinks for each SMTP traffic source.

The background traffic module is created so as to function, as already explained, according to the 'cross traffic' model. As such, the module can be used on a simulation topology by making a call:

generate-backgr-traffic <node1> <node2> $< bw_{backbone} >< pd_{backbone} >< bw_{access_i} >< pd_{access_i} >$<link>

where node1, node2 are the nodes between which cross-traffic has to be created and link is the topology link between the two nodes. This call can be used to fill each link of a simulation topology with background traffic.

2.2 Foreground traffic

The foreground traffic simulation module has been created by the creation of flows belonging to isolated VoIP transmissions, streaming video and H.323 conferencing traffic flows.

For the simulation of VoIP data transmission an exponential ON/OFF distribution was used, with an average duration of ON periods equal to 1.004 sec, average duration of OFF periods (idle-time) equal to 1.587 sec, packet size of 188 bytes (8 byte UDP header+20 byte IP header+160 byte voice data) and a

transmission rate during the 'on' period of 80Kbps. These values obey to the principles of [10]. For the realistic reproduction of aggregated VoIP traffic multiple flows with these characteristics can be created simultaneously.

For the simulation of streaming transmissions, a number of MPEG video traces as well as VBR sources were used. The video traces were measured to have an average transmission rate of 333Kbps, maximum burst time of 0.9895ms at the rate of 8.085 Mbps (1000 bytes) and packet size of 200 bytes. The VBR sources have been based on the VBR traffic generator provided by the ns-2 environment and configured with an average transmission rate of 448Kbps and variation of 0.25, an average burst duration of 0.1 sec at a rate of 648 Kbps and average number of rate changes during a burst equal to 10.

For the simulation of H.323 flows, traces with an average transmission rate of 321Kbps, maximum burst of 4Kbytes and a distribution of packet sizes shown in Table 4 (results from measurements of H.323 transactions made within the scope of [11]), were used.

Table 4. Synthesis of simulated H.323 traffic

Packet size	Percentage
64	< 1%
65 - 127	< 1%
128-255	38 %
256-511	8 %
512-1023	18 %
1024-1518	34 %

For each module of the foreground traffic simulation, a flow of a traffic type can be generated by making a call:
create-<type of traffic> <source>< sink>

where source and sink are the source and sink nodes between which foreground traffic is generated.

3 Implementation of DiffServ modules

3.1 Leaky Bucket Shaping

Leaky Bucket is a shaping algorithm, introduced in [15], according to which packets arriving with random rate are shaped to a configurable constant rate. The Leaky Bucket anticipates for a buffering capacity with constant size. When packets arrive when the "bucket" is full, they are dropped. We have implemented the Leaky Bucket algorithm within the ns-2 environment, so as to provide the shaping functionality for use within DiffServ experiments.

The basic idea of our implementation is to delay each packet (if needed) for the appropriate time in order to shape traffic in a constant rate. The best way to do that is to delay the packets immediately when they enter a network node (the router), before the routing and scheduling processes take place. The critical point is to calculate the right delay time.

The existent implementation of ns-2 did not allow handling of packets when entering a router (edge or core) due to the fact that the implementation of the router itself lacks a receive function of its own and it inherits the receive function from an ancestor class. So we overloaded the receive function so as to call the Leaky Bucket function if configured to do so by the simulation script. In this way, the shaping functionality is inserted before the routing process.

The Leaky Bucket module requires the specification of three input parameters, namely the DSCP of the packets to shape, the rate to which traffic is shaped (in bits/sec) and the depth of the Leaky Bucket used to accumulate packets during shaping (in bytes). The Leaky Bucket module can be used by making the call:
<router_interface> **Leaky-Bucket** <DSCP> <shaping_rate> <bucket_depth>

where router_interface is the router port where the Leaky Bucket shaper is applied to all packets carrying the specified DSCP value.

Focusing on the implementation, the LeakyBucket module can be addressed via a simulation script during topology generation. A series of variables are then initialised so that entries for the DSCPs, the rate, the maximum size and the current size of Leaky Bucket are created. Each packet arriving at an interface for which a shaper is configured is examined with respect to its DSCP value. If a shaping entry for the packet's DSCP exists, the available buffer space of the shaper is examined. If the packet cannot be accommodated in the buffer it is dropped, otherwise the available buffer space and number of packets in the buffer are updated.

The interval for which each packet must be delayed in order for the required shaping to be achieved is also calculated. This calculation is based on the number of the packets that are already delayed in the shaper's buffer and the time of the last packet's departure. Each packet is scheduled to be released towards the transmission queue after the calculated period of time. At that time the available buffer space and number of packets in the buffer are updated once again.

The algorithm for the implementation of the Leaky Bucket module is presented in Figure 2. A function that prints the statistics of the Leaky Bucket shaper, namely the number of shaped and dropped packets has also been implemented and embedded in the Leaky Bucket module.

3.2 DiffServ-based tracing

In the original version of ns when trace-all for tracing of all events in the entire simulated topology is enabled, all packets are traced unconditionally. This leads to very large trace files, which demand a lot of time to be

processed. Especially in a realistic DiffServ simulation environment, where the traffic generation modules that we have already presented are used, interest is focused on tracing only foreground traffic in order to observe the quantitative and qualitative characteristics of foreground traffic for a variety of DiffServ mechanisms and topologies. Therefore, monitoring of background traffic is time and space consuming, as well as not necessary. In order to reduce the size of trace files, tracing only of packets that belong to a specific service class (according to their priority field or DSCP value) or to specific flows was implemented. In this way, classes of packets (such as background traffic) for which detailed tracing is not necessary, can be avoided.

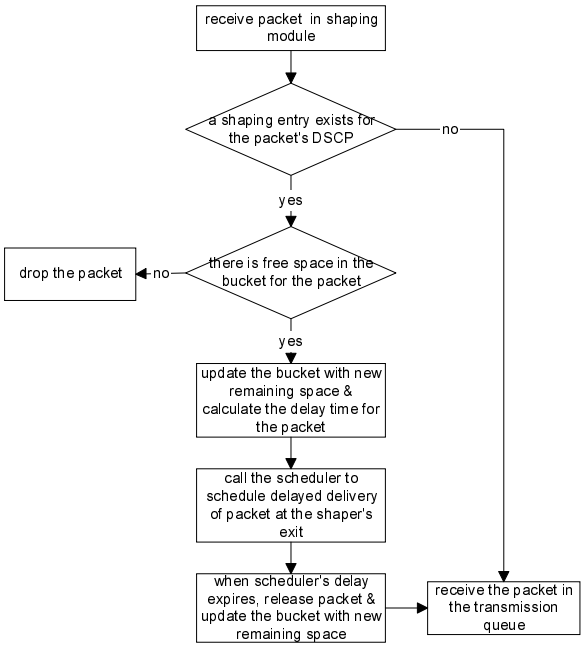


Figure 2. Flow chart of Leaky bucket shaping implementation

In our implementation, the individual flow identification numbers (flow-ids) and the value of the priority fields (or DSCPs) of the flows and classes of packets that we were interested in tracing are stored in appropriate data structures. The receive function of tracing objects has been updated so as to always consult the list of flow-ids and priority field values for which tracing has been explicitly enabled and forward to the tracing process only matching packets. The remaining packets are forwarded directly to the next downstream object of the ns topology, escaping the tracing process. Explicit definition of flow-ids and priority fields for which tracing must be enabled can be performed by:

set_DSCP_to_trace<DSCP1><DSCP2>.. <DSCPn>

and/or

set_fid_to_trace <fid1> <fid2> … <fidn>

A packet the flow_id or DSCP value of which has been included in a command such as these is always traced.

3.3 Scheduling at the ingress interface

The purpose of the implementation of the module for packet scheduling at ingress interface queues is to create a queue mechanism at the routers, where the packets are put in different queues depending on the incoming link that brought them there and are scheduled according to their priority towards a transmission queue of an egress interface. Ingress scheduling is a functionality that a series of Cisco Gigabit Switched Routers (GSRs) offer and has been introduced to solve the head-of-line problem for high-priority traffic in large speeds.

The basic class of the module (Queue_input) is derived from the original Queue class of the ns-2 architecture, inheriting all necessary functionality for a router queue. Queue_input was augmented with a list of queues (Queue_per_link objects) where the packets are enqued depending on the incoming link that brought them to the router. This is achieved by checking the field of IP packet headers that denotes, when tracing is enabled, the upstream node for each packet. Scheduling between the 'Queue_per_link' queues is performed in a round-robin fashion.

Each Queue_per_link object is a Random Early Detect (RED) Queue of the original ns-2 DiffServ architecture that simulates the queuing mechanism at an egress interface of a real router. Packets are enqued in different sub-queues depending on the priority field or DSCP value. For scheduling between the physical sub-queues of the Queue_per_link object, any of the supported scheduling algorithms inherited from the original ns-2 RED Queue can be used. The architecture implemented is shown in Figure 3.

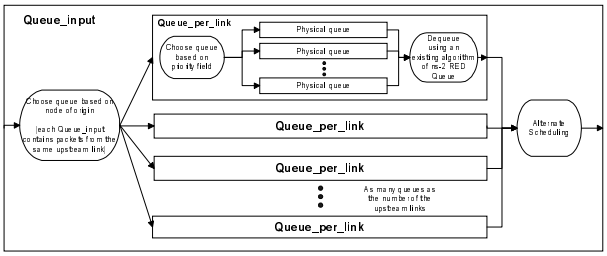


Figure 3. Architecture of the ingress interface scheduling module

In the example of Figure 4, a sample configuration for ingress interface scheduling at a router with two upstream links and three outgoing links is presented.

for each outgoing link i of a simulated router {
create a Queue_input(i) object
set number of Queue_per_link of Queue_input(i) = 2
set size of each Queue_per_link of Queue_input(i) = 5000pkts
arrange so that packets coming from upstream node s(0) to go to Queue_per_link 0
arrange so that packets coming from upstream node s(1) to go to Queue_per_link 1
set number of physical queues of each Queue_per_link of Queue_input(i) = 3
set scheduling algorithm between physical queues of each Queue_per_link of Queue_input(i) to be e.g. Round Robin
for each Queue_per_link of Queue_input(i) {
for all possible DSCP values of packets entering Queue_per_link(j) {
arrange the packets with DSCP= k to be enqued to an underlying physical queue, according to the local DiffServ policy
}
}
connect Queue_input(i) to the ns topology before the out-going link(i) of the router
}

Figure 4. Sample configuration required in simulation scripts for using the ingress interface scheduling module

3.4 MDRR scheduling

In the DiffServ module of ns-2 one can choose one of the following algorithms for scheduling at the edge and core routers: Weighted Round Robin (WRR), Weighted Interleaved Round Robin (WIRR), Round Robin (RR), and Priority (PRI). We implemented two more algorithms, Modified Deficit Round Robin – Strict (MDRR_STR) and Modified Deficit Round Robin – Alternate (MDRR_ALT), that are used by Cisco GSRs ([14]).

In the MDRR scheduling algorithm, all the queues, except for the low latency one (LLQ), are served in a deficit round robin (DRR) fashion. Each one of DRR-served queues can be configured with a weight, according to which an initialisation quantum (in bytes) defining the maximum number of packets that can be uninterruptedly served by the queue is calculated. A deficit (initially equal to the corresponding quantum) is applied to each DRR queue and is decreased by the size of a packet, each time a packet exits the queue. The scheduler moves on to the next queue to be served when the current queue's deficit becomes zero or negative.

When all DRR queues have been served, completing a round of service, all DRR queues' deficits are augmented by the corresponding quantum values, depending on the configured queues' weights.

As far as serving the LLQ is concerned, two alternative modes exist:

- MDRR strict priority scheduling and
- MDRR alternate priority scheduling

In both modes the weights for each physical queue on a router interface can be set by calling:
<router_interface> **addQueueWeights** <physical queue> <weight>

3.4.1 MDRR strict priority scheduling. In Strict Priority mode, the LLQ is always served if packets are queued. Thus, every time the queue to be served has to be elected, the LLQ is examined. If a packet is ready (at the head of the LLQ) to be transmitted, then it is immediately placed on the simulated transmission medium. Otherwise, the round robin-fashion of serving the DRR queues picks up from where it was interrupted, when a previous packet appeared at the head of the LLQ queue.

The MDRR strict priority scheduling algorithm was implemented so as to enhance the scheduling alternatives of the Random Early Detect (RED) Queue of the original ns-2 DiffServ architecture. It can be activated by defining the scheduling mode for each router's interface during the simulation topology set-up:
<router_interface> **setSchedularMode** MDRR_STR

3.4.2 **MDRR alternate priority scheduling.** In alternate priority mode, service alternates between the LLQ and the other DRR queues. The LLQ now obtains its own weight, quantum and deficit values, the latter of which is updated every time an LLQ packet is served and augmented by a value equal to the current queue's quantum at the end of each LLQ-DRR1-LLQ-DRR2-…LLQ-DRRn round. Again, the MDRR alternate priority scheduling was implemented as an additional module to the scheduling alternatives of the Random Early Detect (RED) Queue of the original ns-2 DiffServ architecture.

Figure 5 presents the flowchart that was used for the implementation of the MDRR alternate priority scheduling module. MDRR alternate priority scheduling algorithm can be activated by defining the scheduling mode for each router's interface during the simulation topology set-up:
<router_interface> **setSchedularMode** MDRR_ALT

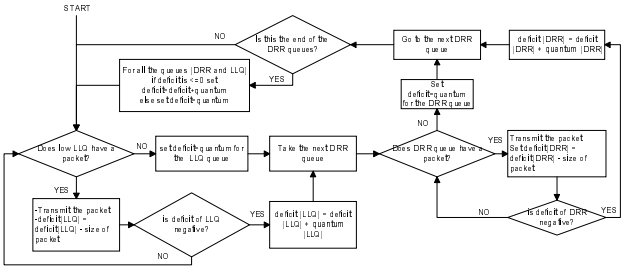
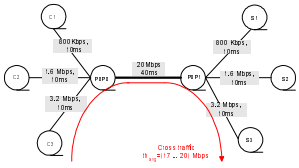


Figure 5. Flowchart of MDRR-Alternate scheduling algorithm

4 A case-study

In this section, a case-study demonstrating the use of the modules already described is presented. Our aim was to evaluate in a realistic simulation environment, two scenarios for serving Quality-of-Service (QoS)-demanding traffic (VoIP, MPEG video streaming and H.323 flows) according to the DiffServ principles. More specifically, we were interesting in testing two different scenarios over a topology simulating the backbone link of a MAN that serves aggregated traffic from adjacent domains. The topology and synthesis of foreground traffic used are depicted in Figure 6.



	High-priority traffic	Packet sizes (bytes)
C_1	2 VoIP flows (ON rate: 80 Kbps per flow)	188
C_2	1 MPEG video flow (r_{avg} : 333 Kbps)	200
C_3	1 VBR flow (r_{avg} : 448 Kbps) & 2 voice flows (ON rate: 80 Kbps per flow)	1500 & 188

Figure 6. Topology and foreground traffic synthesis

Background traffic was produced as already described in this paper and was inserted as 'cross-traffic' on the backbone link of our topology. Three adjacent domains with different access link capacities were attached to the PoP0 of our topology and equal sink domains were attached to PoP1. The table of Figure 6 presents the synthesis of foreground traffic as inserted to PoP0 by each adjacent domain C_i.

Our interest focused on DiffServ mechanisms applied for serving QoS-demanding, foreground traffic at PoP0 and PoP1 and how two different scenarios would affect the quality perceived by foreground traffic flows. Comparison of the quality perceived by foreground traffic was made in two different scenarios where PoP0 and PoP1 were configured with MDRR strict priority scheduling so that:

- Case 1: The total of foreground traffic was served by the LLQ both in PoP0 and PoP1
- Case 2: The VoIP flows were served by the LLQ queue and the rest of foreground traffic was served by a DRR queue with a high queue weight value (97), ensuring access to 97% of the available bandwidth in absence of VoIP traffic. Background traffic was served by another DRR queue with queue weight of 3.

Table 5 and Table 6 present the measured throughput values measured in the two different cases.

Table 5. Throughput (in Kbps) of foreground traffic in case 1

Interface	Port rate	Max	Mean	Min
PoP0	20Mbps	1570	1006	606
PoP1->S1	0.8Mbps	161	52	0
PoP1->S2	1.6Mbps	912	436	156
PoP1->S3	3.2Mbps	761	518	420

The total throughput of foreground traffic is increased in case 2, while it is obvious how the LLQ queue 'protects' VoIP traffic. VoIP traffic is observed to reach the maximum rate of 320Kbps at PoP0, which is equivalent to the accumulation of peaks (80Kbps) of the four VoIP flows transmitted through the PoP. This observation also holds for interfaces PoP1-S1 (2 VoIP flows VoIP -165 Kbps maximum throughput) and PoP1-S3.

Table 6. Throughput (in Kbps) of Gold traffic in case 2

Interface	Port rate	Max	Mean	Min
PoP0->total	20Mbps	1623	1039	661
->Voice traffic		320	135	0
->Other fore-ground traffic		1386	903	606
PoP1-S1	0.8Mbps	165	71	0
PoP1-S2	1.6Mbps	912	436	156
PoP1-S3->total	3.2Mbps	750	532	420
->Voice traffic		165	64	0
->Other fore-ground traffic		630	467	420

Table 7 presents the quality metrics' values for the two cases under examination (delay and jitter measured from the egress interface of a domain C_i up to the ingress interface of a destination domain). One can observe how the end-to-end delay perceived by VoIP traffic is improved in case 2 for VoIP transmitted between C3->S3, when compared to case 1 where VoIP shared the LLQ with foreground traffic of another type at PoP1 (2nd line of both tables). Jitter, however, between these two specific cases of line 2 is increased. VoIP traffic which does not share a transmission queue with another type of foreground traffic in case 1 demonstrates a slight decrease in the quality perceived in case2 (1st line of both tables). As far as the other types of foreground traffic are concerned a slight decrease in the quality perceived is noticed. This is due to the fact that this traffic is transferred from the strict priority queue (LLQ) to a regular queue of service.

Table 7. Quality metrics' values

(case 1)

Traffic aggregate	max delay (ms)	max jitter (ms)
Voice C1->S1	73.98	10.14 (1.9)
Voice C3->S3	70.85	2.5 (0.49)
MPEG	67.04	4.9 (0.5)
H.323	70.85	2.5 (0.49)

(case 2)

Traffic aggregate	max delay (ms)	max jitter (ms)
Voice C1->S1	74.05	10.19 (1.94)
Voice C3->S3	65.29	4.2 (1.07)
MPEG	71.964	9.07 (0.713)
H.323	74.83	6.15 (0.9)

5 Future work-conclusions

In this paper we have presented a series of modules implemented so as to extend the DiffServ functionality of the ns-2 simulator. We have also provided an example of use for most of these modules and an indicative

experimental set-up for the evaluation of different DiffServ mechanisms and scenarios. The importance of our work lies on the fact that the scale of operation in DiffServ environments does not allow for analytical evaluation of models and mechanisms. Therefore, the use of simulation is necessary and particularly useful as a first step before implementation and testing in a real network topology.

Our future work on this area will concentrate on developing and testing new DiffServ-based modules within the ns-2 environment. Such modules include tools for qualitative metrics' measurements (e.g. measurement of packets' delay between two user-configurable network locations), mechanisms for packets' metering and classification, alternative scheduling algorithms and the implementation of integrated service management entities, such as bandwidth brokers.

6 References

[1] R. Wielicki, 'ns-2 ad-ons page', found at: http://thenut.eti.pg.gda.pl/~rafalw/wfq/

[2] G. Ahn, 'MPLS Network Simulator', found at: http://flower.ce.cnu.ac.kr/~fog1/mns/index.htm

[3] S. Jamin, P. B. Danzig, S. Shenker, and L. Zhang, 'Measurement-Based Admission Control for Integrated Services Packet Networks', in proceedings of ACM SIGCOMM'95, pp. 2 – 13, Cambridge, USA, 1995

[4] M. Yuksel, 'Traffic Generator for an On-Line Simulator', Master's Thesis, Department of Computer Science, Rensselaer Polytechnic Institute, 1999

[5] M. Yuksel, B. Sikdar, K.S. Vastola, and B. Szymanski, 'Workload generation for ns Simulations of Wide Area Networks and the Internet', in proceedings of CNDS Conference, part of SCS Western Multiconference, San Diego, USA, 2000

[6] S. Kalyanaraman, K. Vastola and B. Szymanski, 'Traffic generator', result of project 'Network Management and Control Using On-line Collaborative Simulation', funded by DARPA-ITO, found at: http://poisson.ecse.rpi.edu/~olsim/results/results.html

[7] K. Thompson, G.J. Miller, and R. Wilder, 'Wide-Area Internet Traffic Patterns and Characteristics', in IEEE/ACM Transactions on Networking, pp. 10-23, 1997

[8] H. Sawashima. Y. Hori, H. Sunahara and Y. Oie, 'Performance Evaluation of UDP Traffic Affected by TCP Flows', in IEICE Trans. Commun., vol.E81-B, no.8, pp.1616-1623

[9] Data Intensive Distributed Computing Group, 'TCP Tuning Guide for Distributed Application on Wide Area Networks', Lawrence Berkeley National Laboratory, found at: http://www-didc.lbl.gov/tcp-wan.html

[10] ITU-T, P.59, 'Artificial conversational speech', (03/93)

[11] SEQUIN: 'Service Quality across Independently Managed Networks', IST Project IST-1999-20841, project web-site found at: http://www.dante.net/sequin/

[12] S. McCanne and S. Floyd, 'ns Network Simulator', available at: http://www.isi.edu/nsnam/ns/

[13] C. Dovrolis, D. Stiliadis and P. Ramanathan, 'Proportional Differentiated Services: Delay Differentiation and Packet Scheduling', in proceedings of ACM SIGCOMM '99 Conference, Boston, USA, 1999

[14] Cisco 12000 Series Internet Router: Frequently Asked Questions, found at: http://www.cisco.com/warp/public/63/gsrfaq_11085.shtml

[15] ATM Forum, 'Traffic Management Specification', Version 4.0, af-tm-0056.00, April 1996

[16] C. Bouras, D. Primpas, A. Sevasti and A. Varnavas, 'DiffServ functionality patches developed for the ns-2 simulator', found at: http://ouranos.ceid.upatras.gr/diffserv/nspatches/description.htm

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

[18] K. Fall and K. Varadhan (editors), 'The ns Manual-Chapter 9: Differentiated Services Module in ns', 2001, available at: http://www.isi.edu/nsnam/ns/