An Experimental Study of Probing-Based Admission Control for DiffServ Architectures

Susana Sargento[1], Roger Salgado[1], Miguel Carmo[1], Victor Marques[2], Rui Valadas[1], and Edward Knightly[3]

[1] University of Aveiro/Institute of Telecommunications, 3810 Aveiro, Portugal, susana@ua.pt, roger@av.it.pt, etmac@ua.pt and rv@ua.pt
[2] Portugal Telecom Inovação, 3810 Aveiro, Portugal, victor-m-marques@ptinovacao.pt,
[3] ECE Dept., MS380, Rice University, Houston, TX 77005, USA, knightly@ece.rice.edu

Abstract. Probing is a well-known admission control technique that can achieve high utilization and per-flow quality of service in a scalable way. We have recently introduced an extension to the basic probing technique, called ε-probing, to overcome a resource stealing problem that impairs the use of probing in systems with multiple service classes. In this paper we describe an experimental system that was designed to evaluate the effectiveness of both probing and ε-probing techniques. We have developed a software module that implements the probing functionality, which can be inserted in end hosts or edge routers. Several tests were carried out to study the effect of various system parameters in the performance of the probing techniques. The results clearly show that both probing techniques are able to accurately perform admission control while achieving high utilization. Moreover, they also show that in environments with multiple service classes such as DiffServ, ε-probing can eliminate the resource stealing problem, providing an effective solution to support per flow QoS without signaling and without maintaining flow state at core routers.

Keywords: Call Admission Control, DiffServ, QoS, Test-bed.

1 Introduction

The Integrated Services (IntServ) architecture of the IETF provides a mechanism for supporting quality-of-service for real-time flows. Two important components of this architecture are admission control [3], [10] and signaling [5]: the former ensures that sufficient network resources are available for each new flow, and the latter communicates such resource demands to each router along the flow's path. However, the demand for high-speed core routers to process per-flow reservation requests introduces scalability limitations in this architecture.

In contrast, the Differentiated Services (DiffServ) architecture [6], [2] achieves scalability by limiting quality-of-service functionalities to class-based priority

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 49–61, 2002.

mechanisms together with service level agreements. However, without per-flow admission control, such an approach necessarily weakens the service model as compared to IntServ, namely bandwidth or loss guarantees are not assured to individual flows.

A key challenge addressed in recent research is how to simultaneously achieve the scalability of DiffServ and the per-flow QoS assurance of IntServ. Towards this end, several novel architectures and algorithms have been proposed, which require always some specific functionality to be employed at edge and/or core nodes. In probing schemes ([1], [8], [9]), these functionalities are not required: there is no signaling protocol and no special packet processing within core nodes, and still a per-flow QoS is assured. With such a scheme, the endpoints perform admission control by assessing the congestion state of the network, transmitting a sequence of probe packets and measuring the corresponding performance. If the performance (e.g., loss ratio) of the probes is acceptable, the flow is admitted; otherwise it is rejected. More specifically, to establish a real-time flow between two hosts, the sender host transmits a sequence of probes into the network at the desired rate and flow behavior. If the loss ratio of the probes is below a pre-established threshold for the traffic class, then the flow is admitted, and otherwise it is rejected. Scalability is achieved in such a framework by pushing all quality-of-service functionality to end-hosts, indeed removing the need for any signaling or storage of per-flow state. Moreover, [4] found that such an architecture is indeed able to provide a single controlled-load like service as defined in [11].

However, when host-controlled probing schemes are generalized to support multiple service classes, a resource stealing problem, first described in [4], may occur. To illustrate the resource stealing problem, consider the example of a Class-Based Weighted Fair Queuing (CBQ) scheduler, where each of two classes is assigned a weight of 50%. Assume that the offered load is initially $0.8C$ in class 1 and $0.2C$ in class 2, where C is the link capacity. Due to the work conserving nature of the scheduler, class 1 can borrow class 2 resources and utilize up to 80% of the link capacity without loss. If now class 2 probes the link for an additional offered load of $0.3C$, class 2 flows will be admitted and served without loss. However, the service rate of class 1 will decrease to $0.5C$ and 30% of class 1 packets (which belong to already admitted flows) will be dropped. Thus the admission of new flows in class 2 forced class 1 into a situation of QoS violations that can not be detected by the probing flow. Such resource stealing arises from a fundamental observability issue in a multi-class system: the performance isolation property provided by CBQ schedulers also inhibits flows from assessing their performance impact on other classes.

In [7] we proposed ε-probing as a probing scheme designed to eliminate stealing in CBQ schedulers in a minimally invasive way. The goal of ε-probing is to enable inter-class resource sharing to the maximal extent allowed by the system architecture. In ε-probing, a new flow requesting admission in a class transmits a probe in the desired class and, simultaneously, a probe with a small bandwidth ε in all other classes. The motivating design principle is that the impact of the new flow on all classes must be observed, so that the new flow is only admitted

if all probes, including the ε-probes, are admitted. Consider again the previous example of the CBQ scheduler. With ε-probing, when class 2 is probed for the additional $0.3C$, class 1 is also probed with ε-probes. The probing in class 2 is successful but the ε-probing in class 1 will not, since class 1 is not allowed to use more bandwidth. Class 2 flows will not be admitted until class 1 releases bandwidth and no resource stealing will occur.

We developed an experimental system with a DiffServ architecture that includes both probing and ε-probing admission control algorithms. The performance of these algorithms is studied through a number of experiments.

The paper is organized as follows. In section 2 we present the experimental system architecture. In sections 3 and 4 we describe two software modules, the traffic generator and the probing module, which were developed as part of the overall experimental system. Section 5 presents the actual experimental set-up used to carry out the experiments. In section 6 we discuss the experimental results. Finally, in section 7 we conclude the paper.

2 Experimental System Architecture

In this section, we describe the experimental system that is designed to evaluate the efficiency of the proposed ε-probing technique, while closely replicating an operational DiffServ network.

The goal in our experimental studies is to observe the behavior of the probing and ε-probing techniques on a congested network. It would be impractical to have the overall traffic demand generated by many different hosts, as it will be the situation in an operational DiffServ network. Instead we have developed a traffic generator software module that, for each Class of Service (CoS), generates traffic at both flow level and packet level. Due to performance reasons, in the actual experimental set-up we use one host for each CoS.

The probing functionality was implemented in a probing software module, which probes on behalf of a set of users. The probing module can be inserted in end-hosts or edge routers. In the actual experimental set-up the probing module is installed in a dedicated PC, called the probing server, which is connected to a local network delimited by two routers, an access router and an edge router. In this configuration, it can be seen as extending the capabilities of current low-cost edge routers to support probing based admission control. The probing module operates in promiscuous mode, by listening to all packets injected into this local network. It accepts flow set-up requests and performs admission control by probing the DiffServ network; it is also responsible for marking the data packets sent by the traffic generators according to requested CoS. The edge router performs packet classification and scheduling, functions that are found in current low-cost routers. The access router is only used for traffic isolation. Thus, the set of two routers plus probing server emulates a DiffServ edge router that includes admission control based on ε-probing.

The interaction between the various network elements is performed by special purpose application layer protocols. The exchange of control information

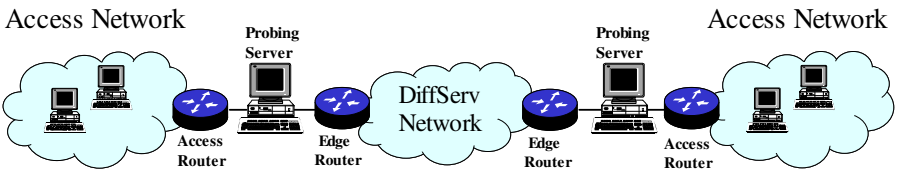


Fig. 1. Experimental system architecture.

between the traffic generator and the probing module is done using TCP. An alternative here could be the use of RSVP. The exchange of control information between probing modules and the data transport is done using UDP.

The message flow is the following (Figure 2). The traffic generator asks for the admission of a new flow by opening a TCP connection with the probing module and sending a REQUEST message. The REQUEST message includes the source and destination IP addresses, the source and destination UDP/TCP ports, the protocol type and the desired class of service. This information is required in order to completely identify the flow at the probing module. Upon receiving the REQUEST message, the ingress probing module initiates the probing process. It sends a PROBE START message, followed by several probe packets, ending with a PROBE STOP message. All these messages are addressed to the destination host and transported over UDP. As mentioned before, there are two types of probe packets: regular probes, sent on the desired class of service, and ε-probes sent on the remaining classes. The egress probing module listens promiscuously to these control messages and probing packets, and counts the number of probes received in each class between PROBE START and PROBE STOP. When it hears the PROBE STOP message it sends a STATISTICS message back to the ingress probing module with this information. If the STATISTICS message is not received within a pre-defined timeout the flow is rejected, and the TCP connection with the traffic generator is closed. Otherwise, the probing module performs an admission control decision based on the counts of probes and ε-probes carried in the STATISTICS message and on the target loss ratio. If the flow is accepted it sends an AUTHORIZE message and closes the TCP connection with the traffic generator; otherwise it sends a REJECT message, also closing the TCP connection. If the flow is accepted the traffic module starts sending data packets (transported over UDP). To signal the end of data transmission, the traffic generator module opens a new TCP connection with the ingress probing module and sends a END SESSION message.

The REQUEST and END SESSION messages have the same format, and are identified by a flag. The AUTHORIZE message corresponds to "0" and the REJECT message to "1", both coded as unsigned int. The probe control messages, PROBE START, PROBE STOP and STATISTICS, include three fields: the first field identifies each message; the second indicates the CoS; the third is used to transport, in the STATISTICS message, the counts of probes in each class. Note that the information exchanged at the application layer is not

sufficient to completely identify a flow. The IP addresses and UDP/TCP ports are also required. This option had the purpose of minimizing the overhead.

All sockets used in the communication between probing modules are of type raw sockets. As will be detailed in section 4, this type of sockets allows operation in promiscuous mode and manipulation of the header fields from lower layers.

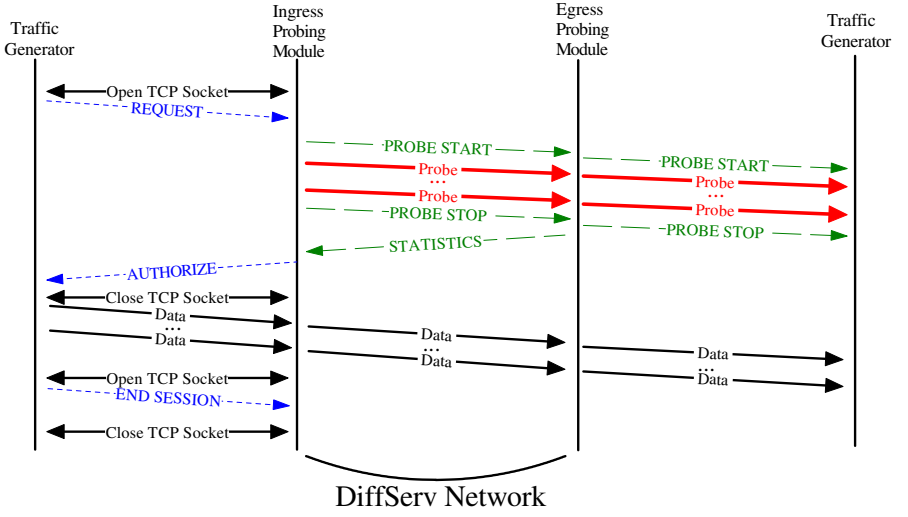


Fig. 2. Message flow between traffic generator and probing modules.

We use the IP TOS byte field to differentiate among classes of service and priorities. It is assumed that control messages injected into the DiffServ network have higher priority. The precedence bits of the TOS byte are used to differentiate between control, probe, ε-probe and data packets. Specifically, we assign 110 to control packets, 010 to probe packets, 100 to ε-probe packets and 000 to data packets. The differentiation between CoS is carried out using the TOS bits of the TOS byte. We leave to the probing module the role of manipulating the TOS byte. All data packets sent by the traffic generator have a TOS byte of zero and are marked according to their class of service at the probing module.

Both the traffic generator and probing modules are developed to run under Microsoft Windows 2000. The software is developed using Microsoft Visual C++, Windows Sockets 2.0 and resorts to multi-thread programming techniques. In our implementation each flow is a thread and, inside each flow's thread, tasks that can be executed concurrently give rise to new threads. The use of Windows Sockets 2.0 and Microsoft SDK make possible the implementation of the promiscuous mode operation at the probing module. Note that the same type of facilities were available for a Unix development.

In the next two sections we will describe with more detail the traffic generator and the probing modules.

3 Traffic Generator Module

The traffic generator module generates the traffic of each CoS at two levels, flow and packet level. It generates new flows according to a Poisson process. The admitted flows have a duration characterized by an exponential distribution. For each flow, the traffic generator creates the corresponding packet stream. Several models are available for the packet arrival process and for the packet length. The arrival process can be Constant Bit Rate (CBR) or ON-OFF with exponential or Pareto ON and OFF durations. CBR sources are only characterized by the packet arrival rate. ON-OFF sources require the specification of the average ON and OFF times and of the packet arrival rate in the ON state. The Pareto distribution requires an additional parameter called shape. The packet length may be fixed, exponential or Pareto.

The traffic generator handles two types of sockets: a TCP socket for the exchange of control information with the probing module, and a UDP socket for data transmission. There is also a thread per CoS that schedules the arrival of the next flow and determines its duration. When a flow starts, another thread is created, which is responsible for the generation of packets for that flow, and of the control messages exchanged between the traffic generator and the probing module.

4 Probing Module

The probing module is responsible for handling the probing process and for packet marking. As mentioned above, the probing module listens promiscuously to the packets that are injected into its local network. This mode is implemented using raw sockets, which allows the manipulation of the IP header fields. At the ingress side, the probing module captures the data packets and re-injects them into its local network after changing the TOS and checksum fields of the IP header. Since Microsoft Windows 2000 does not support natively the manipulation of the TOS byte, we developed a patch for this purpose. Besides the raw sockets, the probing module handles a TCP socket for the exchange of control information with the probing module and UDP sockets for the transmission of data packets, probes and ε-probes and probe control messages. There is a thread permanently listening for new flow set-up requests, at a specific port. When the probing module receives a request from the traffic generator, this thread will produce a new one that will handle the flow. To increase the performance of the system, we use asynchronous UDP sockets to prevent the permanent polling of the socket state. The TCP sockets used in the implementation are of blocking type. In this case, the program suspends the execution of other tasks until the socket operation is finished.

The main window allows the configuration of several parameters: the server port for communication with the traffic generator, the gateways towards the access network or the DiffServ network, the probing duration and timeout, and the link capacity. Note that the link capacity is only required for the computation of

some parameters (wrong decisions and stolen bandwidth). The timeout indicates the maximum time interval after sending PROBE STOP that the module waits for the STATISTICS message.

Two other windows can be opened from the main one, called probing traffic and statistics, respectively. There is one statistics window for each CoS. The probing traffic window (Figure 3) is where the traffic models and parameters for the generation of probes and ε-probes are configured. It also includes the target loss ratio for probes and ε-probes, and an option for deactivation of ε-probes. The statistics windows includes, for each CoS, statistics such as the number of data packets, probes and ε-probes received and sent, the number of blocked and accepted flows, the number of wrong decisions and the percentage of stolen bandwidth. The window also displays a curve of the evolution of the blocking probability over time. All these parameters are updated in real time. Also, the configuration of the experiment's length and of the warm-up time for statistics collection are performed in this window.

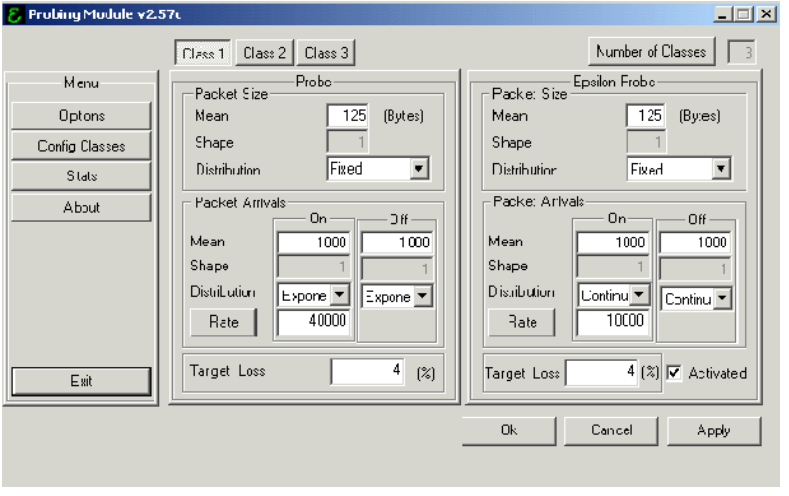


Fig. 3. Probing traffic window of the probing module.

5 Experimental Set-Up

We perform experiments both with two CoS. All sets of experiments resort to the set-up depicted in Figure 4. Each source host A and B generates traffic in a different CoS. Traffic generator modules are plugged in both source hosts A and B. Hosts A and B are 120 MHz Pentium PCs with 64 Mbytes of RAM. Because of performance reasons two probing servers are used at the ingress side. Probing

server A is a 350 MHz Pentium II with 128 Mbytes of RAM, and probing server B is a 733 MHz Pentium III with 256 Mbytes of RAM. The probing server at the egress side is a 933 MHz Pentium III with 256 Mbytes of RAM. The Operating System (OS) of the source and destination hosts A and B is Windows NT 4.0, and the OS of the probing servers is Windows 2000 Professional.

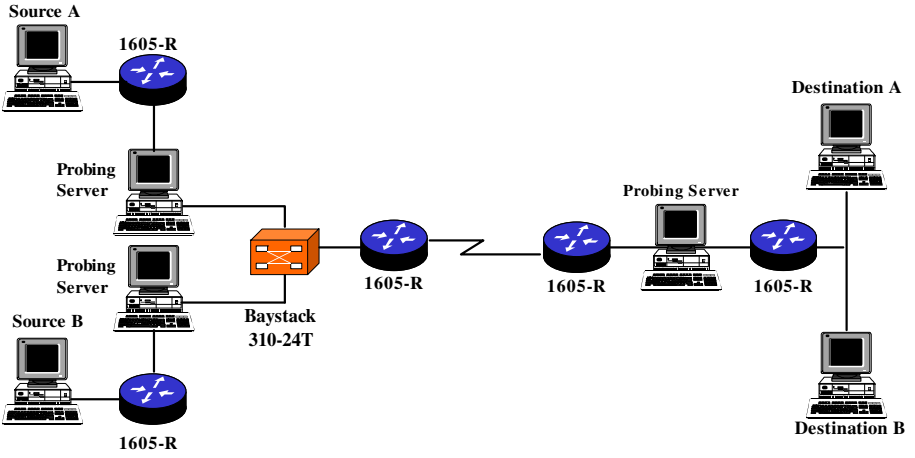


Fig. 4. Experimental Set-Up.

All routers used in the experiments are Cisco 1605 R, running IOS version 12.0(7)T. The ingress and egress edge routers are connected through a serial link, because it offers great flexibility in controlling the link's bandwidth. Our experiments with two CoS resort to Cisco's Custom Queuing. This mechanism works with a maximum of 16 queues, that can be divided in two groups, where one group uses strict priority scheduling and the other uses deficit round-robin scheduling; the latter group has a lower strict priority. In our case, we configure one queue with strict priority (for the control traffic) and two queues with deficit round robin (for the data and probing traffic). Classification at the edge routers is based on the analysis of the precedence and TOS bits and resorts to Cisco's Access Lists. An Ethernet switch (Baystack 310-24T) is used to multiplex the traffic from hosts A and B at the ingress side.

As referred in [4], the probing schemes are able to guarantee a per-flow QoS to controlled load services. The best-effort and the guaranteed services are not considered here, because there will be a different priority for each type of services and a rate limiter will be associated with the guaranteed traffic. Then, our study can be based only on the controlled load services.

6 Experimental Results and Discussion

In this section we present and discuss two sets of experiments. The first set considers two CoS and a constant offered load. The second set considers also two CoS but a time-varying offered load.

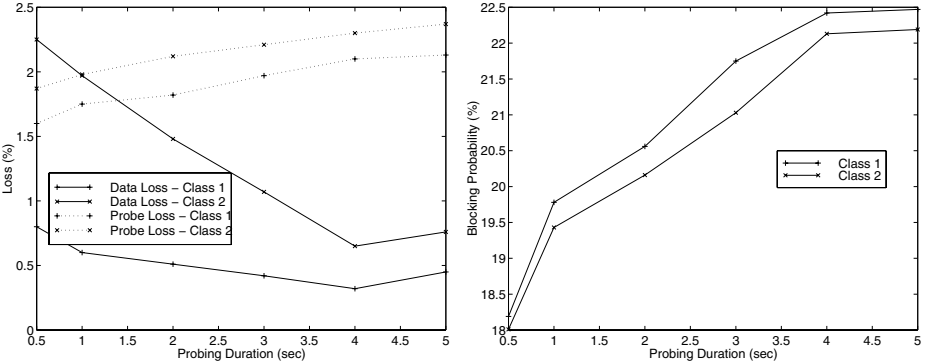
The traffic sources used in the experiments are always CBR. The arrival and departure rates are adjusted to give blocking probabilities near 0.2 (corresponding to an offered traffic that is approximately 120% of the link capacity). The mean number of active flows in a traffic class is $\rho = \lambda/\mu$, where λ and μ are respectively the mean flow arrival and depart rates. The offered load is given by the mean number of flows (ρ) multiplied by its bandwidth. Unless otherwise specified, the link capacity is 1 Mb/sec, the packet length is 125 bytes, the buffer size of the queues is 24000 bytes and the length of each experiment is 1200 seconds. A warm-up time is used in all experiments, which is at least two times the highest value of the flow's average duration. Note that the probing bandwidth always equals the flow bandwidth. Each experiment described bellow is repeated five times and the results represent the corresponding average values.

6.1 Experiments with two CoS and a Constant Offered Load

This set of experiments addresses two traffic CoS and a constant offered load, i.e., the arrival rate and mean duration of the flows do not vary during the experiments. The goal here is to address the resource stealing problem and analyze the behavior of ε-probing. In all experiments, the target loss ratio of the probes and ε-probes is 5%. The flow's bandwidth is 40 Kb/sec in class 1 and 64 Kb/sec in class 2. The flow's ρ is 4 in class 1 and 11 in class 2. The weight assigned to class 1 is 20% and the weight assigned to class 2 is 80%.

Probing duration. In this experiment the bandwidth of the ε-probes in both classes is 20 Kb/sec and the bandwidth of the probes in each class equals that of the flows requesting admission. Figure 5(a) shows the data and probe loss ratios in each class, as a function of the probing duration. Figure 5(b) shows the corresponding blocking probabilities. The data and probe loss ratios are always below the target, showing that the probing-based admission control operates correctly. The blocking probabilities increase and there is also a slight increase in the probe loss ratio, as the probing duration increases. Except for small probing durations, the data loss is always below the probe loss since not all flows are admitted. For small probing durations, the data loss in class 2 is larger than the corresponding probe loss, which can be explained by lack of accuracy due to insufficient probing duration. The data loss decreases for probing durations between 0.5 and 4 seconds. One might expect that a larger probing time would produce a more accurate estimation of the data loss ratio, i.e., a measured data loss ratio closer to the target (which is 5% in both classes). However, due to the overhead introduced by longer probing times, the effect is the opposite. The same behavior is observed via discrete-event simulation in [4]. For probing durations

greater than 4 seconds the data loss ratio increases because the probing traffic gets significant contributing itself to the degradation of the data loss ratio. The loss ratio in class 2 is higher because since class 2 flows have higher bandwidth more probes are generated in the same probing duration.



(a) (b)

Fig. 5. Effect of probing duration on (a) data and probe loss, and (b) blocking probability.

Mismatch between offered load and CBQ weight. In this experiment we introduce a mismatch between the offered load and the CBQ weight of class 1. The weight is kept as before at 20%; the offered load is increased from 20% to approximately 50% of the link capacity (by increasing ρ from 4 to 11). In class 2 we keep everything as before. In Figure 6 we show the data and probe loss ratios versus the bandwidth of ε-probes. With no ε-probing (a null ε-probe bandwidth) the data loss in class 2 is almost 8%, which is larger than the threshold. This behavior is maintained for ε-probe bandwidths bellow 4 Kb/sec, and can be attributed to resource stealing. In fact, whenever class 2 goes into underload, class 1 flows will try to use some of the fair-share bandwidth of class 2 with success. Class 1 flows will then experience resource stealing because in this situation, and since probing is only in the requested class, new requests for class 2 flows will be accepted (at the cost of stealing bandwidth to already accepted class 1 flows). Figure 6 also shows that the probing loss in class 2 increases with the ε-probe bandwidth. This increase is responsible for blocking more class 2 flows when class 1 is using some of the fair-share bandwidth of class 2, which reduces the bandwidth stealing in class 1.

6.2 Experiment with two CoS and a Time-Varying Offered Load

In this experiment we consider a time-varying offered load. The motivation here is to increase the potential for resource stealing, in order to study the effectiveness

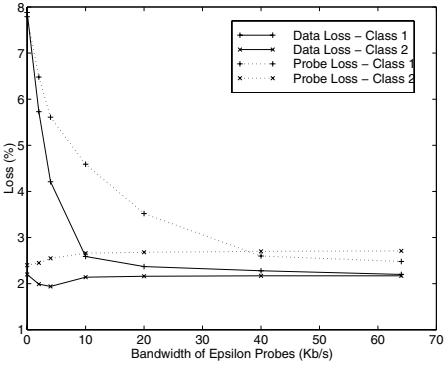


Fig. 6. Effect of mismatch between offered load and CBQ weights on data and probe loss.

of the ε-probing scheme. Specifically, we increase the traffic intensity of class 1 during the experiment at a specific time instant, coinciding with the start of data collection for the purpose of statistics computation. Before this perturbation, the offered load is 20% of the link capacity in class 1 and 80% in class 2. Since each class is assigned a weight of 50%, class 1 will be underloaded and class 2 will be overloaded. The experiment consists in increasing the offered load of class 1, to force the bandwidth stealing of already accepted flows from class 2. The offered load in class 2 corresponds to 64 Kb/s of flow bandwidth and a mean number of flows ρ of 11. Before the perturbation, the offered load in class 1 corresponds to 64 Kb/s of flow bandwidth and a mean number of flows ρ of 4.

The probing duration is kept constant at 2 seconds. The model of the traffic source is CBR in all cases. The target loss ratio of probes and ε-probes is 5%. In the actual experiment, the increase in traffic intensity of class 1 is implemented by two traffic generators. Both generators have a constant offered load, but the second one is only activated later in the experiment. We consider two cases for the perturbation: the second generator has (i) an arrival rate of $0.5sec^{-1}$ and ρ of 10; (ii) an arrival rate of $0.33sec^{-1}$ and also a ρ of 10; the flow bandwidth is kept at 64 Kb/sec in both cases. The goal is to keep the traffic intensity approximately constant while increasing the arrival rate. Given that we want to analyze the transient behavior of the system, i.e., when a perturbation arises, the length of the experiment was constrained to 200 sec (from the start of the second generator), to avoid averaging out the stealing effects.

To analyze the results of the experiment we use two performance metrics: the percentage of wrong decisions and the percentage of stolen bandwidth. The former is the percentage of flows that are accepted when the bandwidth of all admitted flows is higher than the link capacity. The latter is the percentage of bandwidth that is stolen by the admission of new flows when this admission is a wrong decision. The computation of these metrics is done as follows: whenever there is a positive admission decision, we calculate the bandwidth occupied by

all admitted flows, based on the number of flows and on the flow's bandwidth. If this bandwidth is larger than the link capacity (including the tolerance given by the loss target), the decision is computed as a wrong decision. In this case, the difference between the bandwidth of the admitted flows and the link capacity is the stolen bandwidth.

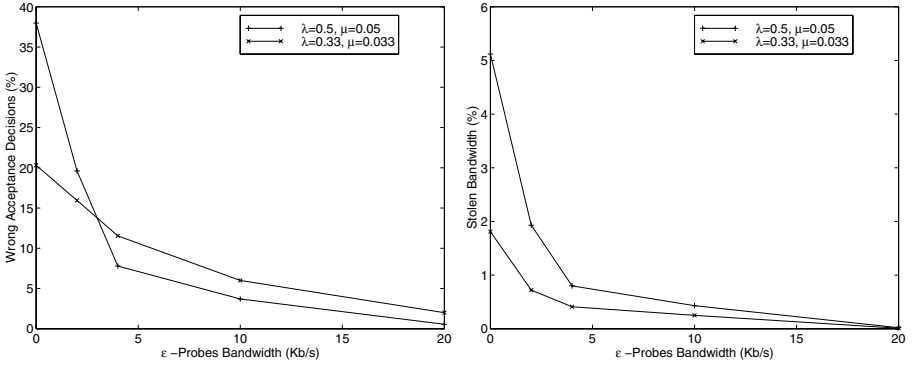


Fig. 7. Effect of time-varying offered load on (a) wrong decisions and (b) stolen bandwidth.

Figure 7(a) and (b) show that without ε-probing the percentage of wrong decisions and of stolen bandwidth is very high (wrong decisions are 38% with the first perturbation and 20% with the second one; stolen bandwidth is more than 5% in the first perturbation and almost 2% in the second one). This is due to resource stealing, when class 1 recovers its bandwidth after the system's perturbation. Both metrics decrease rapidly with the ε-probe bandwidth: with only 2 Kb/sec the stolen bandwidth values decrease almost to one half, and with 10 Kb/sec (less than 1/6 of the bandwidth of admitted flows) the stealing is almost insignificant. A comparison of the two curves in each figure shows that a larger arrival rate provokes more stealing. Thus, the results of this experiment where resource stealing is intentionally aggravated, clearly show that ε-probing is able to eliminate this problem.

7 Conclusions

Placing admission control functions at the network's endpoints has been proposed as a mechanism for achieving per-flow quality of service in a scalable way. In this paper we have described an experimental system with a DiffServ architecture that includes both probing and ε-probing admission control algorithms. The ε-probing technique was introduced to overcome the so-called resource stealing problem that impairs multi-class systems based on simple probing. A number of

experiments was carried out to study the performance of these admission control algorithms. The results clearly show that the probing schemes are able to accurately perform admission control while achieving high utilization. Moreover, they also show that in multi-class environments such as DiffServ, ε-probing can eliminate the resource stealing problem. For example, it was shown that the resource stealing problem can be virtually eliminated by using ε-probes with a bandwidth higher than 1/6 of the flows' bandwidth. Thus, the ε-probing scheme is able to provide an effective solution to support per- flow QoS without signaling and without maintaining any flow state at core routers.

References

1. G. Bianchi et al. Throughput analysis of end-to-end measurement-based admission control in ip. In *Proceedings of IEEE INFOCOM 2000, Tel Aviv, Israel*, March 2000.
2. K. Nichols et al. *Two-bit differentiated services architecture for the Internet.* Internet RFC 2638, 1999.
3. L. Breslau et al. Comments on the performance of measurement-based admission control algorithms. In *Proceedings of IEEE INFOCOM 2000, Tel Aviv, Israel*, March 2000.
4. L. Breslau et al. Endpoint admission control: Architectural issues and performance. In *Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden*, August 2000.
5. L. Zhang et al. Rsvp: A new resource reservation protocol. In *IEEE Network*, volume 7, pages 8–18, September 1993.
6. S. Blake et al. *An architecture for differentiated services.* Internet RFC 2475, 1998.
7. S. Sargento et al. Resource stealing in endpoint controlled multi-class networks. In *Proceedings of International Workshop on Digital Communications (Invited Paper), Taormina, Italy*, September 2001.
8. V. Elek et al. Admission control based on end-to-end measurements. In *Proceedings of IEEE INFOCOM 2000, Tel Aviv, Israel*, March 2000.
9. R. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proceedings of ITC '99, Edinburgh, UK*, June 1999.
10. E. Knightly and N. Shroff. Admission control for statistical qos: Theory and practice. In *IEEE Network*, volume 13, pages 20–29, March 1999.
11. J. Wroclawski. *Specification of the controlled-load network element service.* Internet RFC 2211, 1997.