Design and Implementation of Scalable Edge-Based Admission Control

Ping Yuan[†], Julie Schlembach[†], Anders Skoe[‡], and Edward Knightly[†]
[†]Department of Electrical and Computer Engineering, Rice University
[‡]Department of Electrical Engineering, Stanford University

Abstract

While the IntServ solution to Internet QoS can achieve a strong service model that guarantees flow throughputs and loss rates, it places excessive burdens on high-speed core routers to signal, schedule, and manage state for individual flows. Alternatively, the DiffServ solution achieves scalability via aggregate control, yet cannot ensure a particular QoS to individual flows. To simultaneously achieve scalability and a strong service model, we have designed and implemented a novel architecture and admission control algorithm termed Egress Admission Control. In our approach, the available service on a network path is passively monitored, and admission control is performed only at egress nodes, incorporating the effects of cross traffic with implicit measurements rather than with explicit signaling. In this paper, we describe our implementation of the scheme on a network of prototype routers enhanced with ingress-egress path monitoring and edge admission control. We report the results of testbed experiments and demonstrate the feasibility of an edge-based architecture for providing IntServ-like services in a scalable way.

Keywords: admission control, quality of service, scalability, Internet.

1 Introduction

The Integrated Services (IntServ) architecture of the Internet Engineering Task Force (IETF) provides a mechanism for supporting quality-of-service for real-time flows. Two important components of this architecture are admission control [2, 6, 7, 12] and signaling [16]: 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, without further enhancements (e.g., aggregation [10]), the demand for high-speed core routers to process per-flow reservation requests introduces scalability and deployability limitations for this architecture.

In contrast, the Differentiated Services (DiffServ) architecture [1, 9, 14] achieves scalability by limiting quality-of-service functionalities to class-based priority 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, individual flows are not assured of a bandwidth or loss guarantee.

To simultaneously achieve scalability and a strong service model, we have devised Egress Admission Control [3], an architecture and algorithm for scalable Quality of Service (QoS) provisioning. In this scheme, admission control decisions are made solely at egress routers; per-flow state is not maintained in the network core nor in the egress router, and there is no coordination of state with core nodes or other egress nodes. Therefore, admission control and resource reservation are performed in a distributed and scalable way. As a consequence of the scalable architecture, a key challenge is how to assess the network's available resources at an egress point. Our solution is to employ continuous passive monitoring of a path (ingress-egress pair) and assess its available service by measurement-based analysis of the arrival and service times of packets on the path. In this way, an egress router holds implicit control over other paths, by ensuring that all classes on all paths maintain their desired quality-of-service levels. This implicit control factors in effects of cross traffic and prevents other egress routers from admitting flows that would exceed the available bandwidth.

Figure 1 illustrates a simplified model of egress admission control. The figure depicts how an ingress-to-egress path is modeled as a "black box" with an unknown service discipline and cross-traffic that cannot be directly measured. An important part of egress admission control is assessing the available service along this path. We will show how the abstraction of a statistical service envelope [11] provides a general framework for characterizing service, including fluctuating available resources due to varying demands of cross traffic.

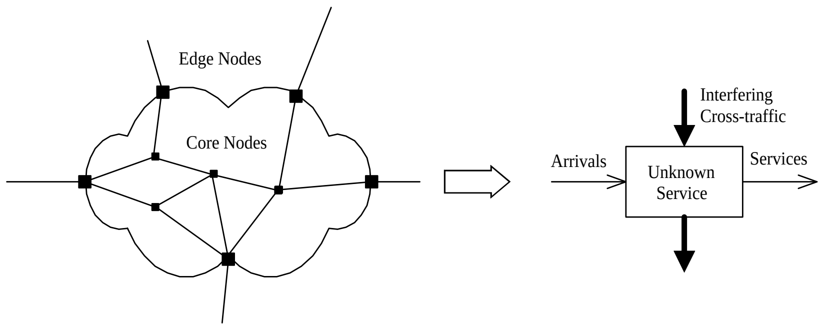

Figure 1: Egress Admission Control System Model

This paper describes our design, implementation, and measurement study of Egress Admission Control. In particular, we illustrate the key design issues that must be addressed for scalable admission control. Compared with an IntServ architecture, our implementation addresses three issues. First, we modify RSVP (Resource reSerVation Protocol) [16] so that only egress nodes process reservation requests. Second, to assess both arrivals and available service at a path's egress node, we insert timestamps and path-level sequence numbers into packets: we describe our implementation of both IPv4 time stamping and Network Time Protocol (NTP) clock synchronization. Finally, we implement an edge based admission control algorithm in which measurements of the service along a path are used to predict and control network-wide QoS.

To evaluate the scheme, we performed an extensive measurement study on a testbed consisting of prototype routers equipped with our implementation of Egress Admission Control. The experiments indicate that the algorithm is able to control the network's admissible region within a range such that the requested quality-of-service parameters can be satisfied. This demonstrates a key component of scalable admission control, namely, that flow-based QoS can be achieved without signaling each traversed node for each reservation request.

2 Design and Implementation

Our design consists of three inter-related components: (1) the signaling protocol by which new flows are established, (2) the traffic measurement module, including time stamping and loss detection, and (3) the admission control module which accepts or rejects requests to establish new real-time flows.

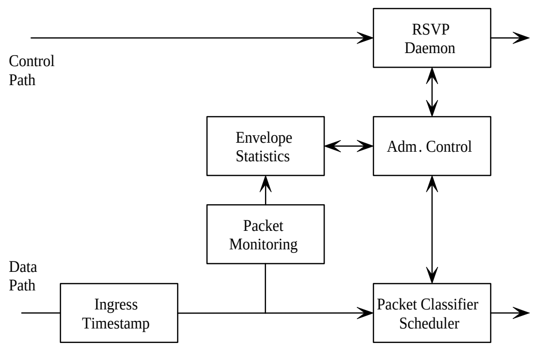

Figure 2: Edge Router Architecture

Figure 2 depicts the relationship among these system components. As shown, when a user desires to initiate a new QoS session such as streaming video or voice over IP, the host sends a signaling message to determine if the requested service is available. The request handler then calls the admission control routine to determine whether the new flow may be admitted while satisfying all flows' statistical service guarantees. Meanwhile, the statistical properties of a path are monitored in real-time at the egress node. This information is accessed by the admission control algorithm in its decision. Each of these edge-router components communicate via shared memory.

2.1 Flow Establishment and Signaling

In order to perform flow-based admission control decisions, a signaling protocol is required to communicate a resource reservation request from a host to a domain's egress router. Consequently, we modified RSVP to establish flows in the Egress Admission Control architecture.

In general, RSVP's functionality as defined in [16] can be described as follows. When a host's application requests a specific QoS for its data stream, the host calls the RSVP daemon to deliver its request to routers along the data stream's path. Each RSVP router has several local procedures for reservation setup and enforcement. Policy control determines whether the user has administrative permission to make the reservation. Admission control keeps track of the system resources and determines whether the node has sufficient resources to supply the requested QoS. The RSVP daemon invokes both procedures before accepting the new flow. If either test fails, the RSVP daemon returns an error notification to the application that originated the request. If both checks succeed, the RSVP daemon sets parameters in the packet classifier and packet scheduler to obtain the requested QoS. The packet classifier determines the QoS class for each packet and the packet scheduler orders packet transmission to achieve the promised QoS for each flow.

In our implementation of Egress Admission Control, the requirements of RSVP are significantly simplified. Foremost, RSVP reservation messages need only be processed by a requesting flow's egress router (or each domain's egress router in the case of multiple domains). Since a prominent feature of RSVP is that it provides transparent operation through non-supporting regions, the RSVP daemon is simply not running on the core routers, and reservation requests are merely forwarded by core routers as normal IP packets. Therefore, it is only necessary to run the RSVP daemon on edge routers where admission control decisions are made.

The next step is for an edge node to identify that it is in fact the egress node and not an ingress node. This is quite simple to achieve based on the static configuration of the edge router itself: if it receives an RSVP message on its core-node interface, it is the egress node for the flow and should process the request; otherwise, the router is an ingress node and should simply forward the packet.

Our second alteration to RSVP is in the admission control algorithm. We modified the RSVP daemon to call our egress admission control program (described below) upon receipt of a new reservation request. This algorithm is invoked as an alternative to IntServ-style per-link measurement based admission control such as described in [2].

Finally, we do not send explicit tear down messages upon completion of a real-time session, as per-flow state is not maintained anywhere in the system, and all admission decisions are based on measurements. (We note however, that tear down messages may be desirable for other purposes such as resource usage accounting).

Regardless, the RSVP messages themselves are not modified. Excluding the changes made to the RSVP daemon, the protocol is compliant with the standard, and the host and user interfaces are left unaltered.

2.2 Measuring Path Traffic and Service

In order for an egress router to assess even simple characteristics of a path such as ingress-to-egress queueing delays, the egress router must measure both the egress router's service time of packets, and the times that packets entered the ingress node. Consequently, to reveal such entrance times to the egress nodes, we insert time stamps for real-time packets at ingress nodes.[1]

[1]A modified design that would avoid time stamping would be for ingress nodes to collect arrival statistics and periodically transmit the aggregate information to the egress nodes. However, such coordination of state among edge nodes is not a part of our current design.

There are four design components to our traffic and service measurement methodology: inserting timestamps at ingress nodes, synchronizing edge-router clocks, capturing and reading timestamps along with other packet header information at the egress node, and calculating traffic envelopes from this data.

2.2.1 Timestamping

In order to communicate packet ingress entrance times to egress routers, we insert information into fields of the IP header at the ingress node, which is ignored by the core nodes and read at the egress node. With an approach analogous to the Dynamic Packet State code [13], we utilize 18 bits from the ip_tos field and ip_off field to transmit a 10-bit timestamp as depicted in Figure 3.

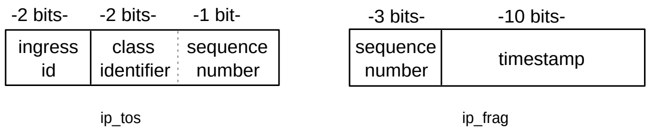

Figure 3: IP Header Insertions for Path Monitoring

The ingress node transmits the ten most significant bits of the fraction of the second that just passed in Universal Coordinated Time when the call to insert the arrival timestamp is summoned in the ip_input() routine. At the egress node, the service time is registered when the packet leaves the router at the outgoing interface. This ensures that when calculating the envelopes, we also account for queuing delays at both edge routers.

In our particular implementation, inserting the timestamps into the ip_tos and ip_off fields of the IP packet header ensures that the intermediate routers forward the packets without any additional processing overhead. Had the IP timestamp options been used, additional packet-processing overhead would have occurred at the intermediate nodes. This type of implementation does assume that no packets are fragmented, but this is simple to control in a laboratory setting.

The aspect of the implementation illustrates the current limitations of IP options, as the existence of the IP_OPTIONS flag invokes the function ip_dooptions(). This then incurs significant overhead since the function call places the packet on the “slow path”. A more efficient solution could utilize a flag indicating whether a packet should be simply be forwarded by an interior node, or whether it needs to be processed further. Such a flag would easily be detected in hardware, and hence be compatible with high-speed core routers.

2.2.2 Time Synchronization of Edge Routers

As explained above, in order to compute one-way delays and service envelopes at the egress point, the arrival times of packets at the ingress node must be communicated to the egress node. For this to occur, the clocks in the various edge routers must be synchronized within a value that guarantees delay bounds on the order of tens of msec.[2] We implemented two solutions to achieve synchronization: a local hardware solution based on Code Division Multiple Access (CDMA) pilot signals and a remote solution synchronizing to other Internet hosts which themselves have hardware support such as CDMA or Global Positioning System (GPS). In both cases, the Network Time Protocol (NTP) protocol is used to synchronize the router's clock with either the local hardware or the remote host.

In our testbed implementation, we used a CDMA time device (Pracis CT) on each router that receives signals broadcast from CDMA base stations and synchronizes its system time and frequency using the pilot and sync channel data just as a CDMA phone handset does. NTP is then used to synchronize the system clock with the local CDMA device's clock. With this approach, we found that the time offset between edge routers is less then 0.3 milliseconds even under heavy load. As queuing delays and transmission delays are

[2]Observe that an alternate solution is for core routers to append a cumulative queueing delay to a field in the packet header so that egress nodes can deduce the entrance time. However, we do not consider such solutions which require special-purpose functionality of core routers and limit our study to pure edge-based solutions.

significantly larger than this offset, the timestamps are accurate enough to calculate the path characteristics at the egress router using the timestamp information from the ingress router.

In addition to this local hardware solution, we also used NTP to synchronize with remote hosts that are not part of the testbed. In particular, NTP operates by sending its own messages, containing the four most recent timestamps between a client and another server or reference time source, such as a radio or satellite receiver or modem, at polling intervals between 64 and 1024 seconds, depending on the stability of the two clocks. The round-trip propagation time and offset are calculated from these four timestamps. Then the computer specified to adjust its time does so by gradually skewing its clock to the "correct time" of the other clock using a phase lock loop clock discipline algorithm described in detail in [8]. This algorithm alters the computer clock time, while compensating for the intrinsic frequency error and dynamically adjusting the poll interval. The measured time errors discipline a feedback loop, which controls the phase and frequency of the clock oscillator. This is done with the aid of the clock adjust process, which runs at intervals of one second.

However, we found that using such NTP synchronization with a remote public time server available over the Internet can be problematic. In particular, we found that under heavy loads, the time accuracy can degrade to as large as 100 milliseconds, which adversely effects the QoS control mechanisms. Such clock skew is due primary to the large queueing delays and losses incurred by the NTP messages themselves under heavy load. While this issue can be addressed with additional modifications to the routers (e.g., a priority class for NTP packets), we found the local hardware solution to be superior, and use it for all measurements reported in this paper.

2.2.3 Capturing Traffic at Egress Nodes

In order to collect timestamps of all real-time packets traversing a path, we modified the packet sniffer tcpdump[3], which uses the libpcap library to read header information from IP and TCP/UDP headers. While tcpdump is not the ideal way to perform this task, since it is performed at user level, it is sufficient for the 10 Mb/sec routers employed in the testbed. For high-speed implementation, this functionality could be integrated into the kernel or supplemented with hardware support.

In addition to timestamps, the egress router records the packet length, an ingress node identifier, a class identifier, and a four-bit per-path sequence number (as depicted in Figure 3). The class and ingress node identifier will allow admission control on a per-class, per-path (ingress-egress pair) basis.

The per-path sequence number is used to identify loss on the path at the egress point. Using a 4-bit sequence number ensures that up to 15 consecutive packet losses can be detected. In our current implementation, we have not used this loss detection, as the class QoS requirements are stringent enough to keep loss sufficiently low that dropped packets can be ignored. However, for less stringent QoS requirements, such loss would need to be incorporated: otherwise egress routers may over-estimate the available service along a path, as not all arriving traffic is incorporated into the measurement.

2.2.4 Computing a Path's Available Service

To assess the available service on a path, we measure a path's statistical service envelope [11], a general characterization of the end-to-end service received by a traffic class. This service abstraction can incorporate the effects of interfering cross traffic without explicitly measuring or controlling it. Moreover, the service envelope exploits features of the backbone nodes' schedulers and the effects of statistical resource sharing at both the flow level and the class level. For example, if a class is provided a circuit-like service without sharing among traffic classes, the service envelope will measure a simple linear function. In contrast, if the network performs scheduling similar to weighted fair queuing, the service envelope will reflect the available capacity beyond the minimum "guaranteed rate" which can be exploited by the class, i.e., the excess capacity which is available due to fluctuating resource demands of cross traffic and other traffic classes. Finally, by limiting a class' traffic through controlling admission of flows into the class, we can ensure that the class' predicted quality-of-service is within its requirements. When all edge routers perform the algorithm, the scheme ensures that all classes of all paths receive their desired service levels.

[3]www.tcpdump.org

There are two methods for calculating the traffic envelope. One approach is to use the method where a rate-based envelope is considered; that is, calculate the peak rate for a given interval length. A second, and analogous method, is to calculate the total number of bytes that arrived in an interval - that is, measure the minimum interval length over which a certain number of bytes is transmitted. In our implementation, we use the latter approach, as it removes a number of divisions by the interval size from the algorithm.

In other words, instead of determining the maximum number of bytes to arrive in an interval of given size (via a sliding window) or discretizing the time scale, we calculate the minimum time required for a certain number of bytes to arrive/ be serviced or discretize the scale for the number of bytes that arrived. The primary motivation for this design decision is the fact that the flow requests specify a statistical delay bound, and following the latter scheme ensures that the variance remains in the time-domain. Consequently, the admission control equation (described below) can be applied directly without undergoing computationally expensive conversions.

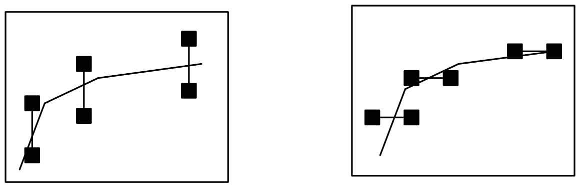
(a) Interval Discretization (b) Data Discretization

Figure 4: Envelope Illustrations of Bytes vs. Interval Length

2.2.5 Admission Control

An egress router's admission control decision is described as follows and is illustrated in Figure 5 (see [3] for further details). Consider a system where a traffic class between a particular ingress-egress pair has a measured peak rate arrival envelope with mean $\bar{R}(t)$ and variance $\sigma^2(t)$. In other words, over successive measurement windows, the average maximum number of arrivals is given by $t\bar{R}(t)$, and its variance is given by $t^2\sigma^2(t)$. Similarly, the class' minimum service envelope has measured average $\bar{S}(t)$ and variance $\Psi^2(t)$. The new flow with peak rate P is admissible with delay bound D if

$$t\bar{R}(t) + Pt - \bar{S}(t+D) + \alpha\sqrt{t^2\sigma^2(t) + \Psi^2(t+D)} < 0$$

where α is set according to the required violation probability [3]. Moreover, we ensure the stability condition that

$$\lim_{t\to\infty} \bar{R}(t) < \frac{\bar{S}(t)}{t}.$$

Notice that for a first-come-first-serve server with link capacity C, $\bar{S}(t)/t = C$. Figure 5 illustrates that for a given threshold (in bytes) $\bar{R}(t)$ is obtained by normalizing the maximal arrival envelope, and $\bar{S}(t)$ is computed as the maximum time for service for the particular threshold. The algorithm considers each threshold and determines whether the network can satisfy the new flow's quality of service requirements. If both of the above requirements are satisfied, a message to admit the new flow is relayed by the request handler and the user may begin sending the traffic.

This approach pushes the task of network resource management to the edge of the system, and does not require that interior nodes perform per-flow or per-class bandwidth reservation. Instead, edge nodes exclusively handle admission control decisions and signaling messages. The available service in the contiguous interior of the network is inferred by inserting timestamps into packet headers at the ingress nodes and statistically analyzing the path's characteristics.

3 Experimental Design and Measurements

In this section, we present the results of a measurement study obtained using our implementation of egress admission control in a network of prototype routers.

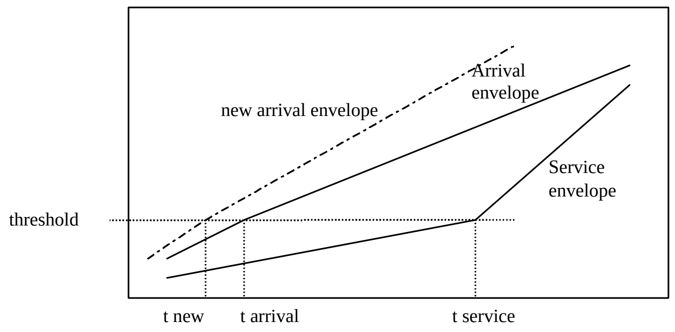

Figure 5: Illustration of Admission Control

3.1 Scenario

In order to study the performance of the scheme, we perform five different experiments, each building upon the previous one. All routers and hosts run the FreeBSD v3.2 operating system and are connected via 10 Mb/sec links. The buffer size of the routers is 250 packets, which for a link capacity of 10 Mb/sec and maximum packet size of 1500 bytes corresponds to a maximum queuing delay of 300 milliseconds. We begin with a baseline experiment depicted in Figure 6, in which a single node functions as both the ingress and egress router. This router resides between the source machine, which generates the traffic, and the destination machine, which receives the traffic. In all cases, each host will generate multiple flows, and hosts are connected to routers via 100 Mb/sec links so that no queueing occurs in hosts.

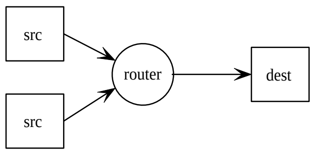

Figure 6: Baseline Experiment

The remaining experiments are performed with the configuration depicted in Figure 7. Here, three routers interconnect five hosts. Depending on the experiment, these routers function as ingress, egress and/or core routers. Experimental results are reported between the hosts labeled "src" and "dest" whereas the two hosts labeled "cross traffic src" 1 and 2 function as cross-traffic generators to congest routers and test the egress admission control algorithm's ability to infer the available service along a path with unmeasured cross traffic.

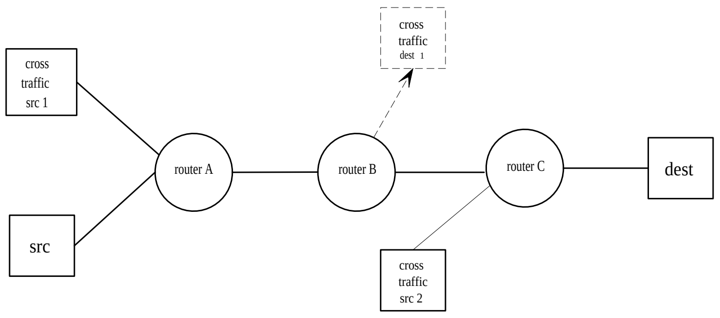

Figure 7: Ingress/Egress Pair Experiments

3.2 Traffic Generation

To emulate the behavior of realistic real-time flows, we designed a Pareto on-off traffic generator that transmits packets only after the QoS request is admitted by the egress router's admission control test. It consists of two components: the actual traffic generation and communication with RSVP. Packets are generated according to the Pareto on-off model with the following parameters: packet size 1000 bytes, mean burst time 250 msec, mean idle time 250 msec, and peak rate 400 kb/sec. The Pareto shape parameter is 1.9 (recall that a Pareto shape parameter less than 1 results in an infinite mean while a shape parameter less than 2 results in an infinite variance) and the flow lifetime is 5 minutes. Thus, this traffic generator produces highly bursty traffic which, when aggregated, forms a flow that exhibits self-similarity [15].

The second part of the traffic generator handles communication with RSVP. As RSVP is a receiver oriented reservation protocol, it needs both the path information from the source host and the QoS request information from the destination host. In order to communicate the user's request to the egress router, a module in the traffic generation program on the source host side generates a path message, while a module on the destination side sends a reserve message with the QoS request. By calling the RSVP application interface function, the sender side receives the admission control result.

3.3 Measurements

Experiment 1: One Node

The first experiment consists of a single router, one class, and no cross traffic. Pareto on-off traffic is sent from the main source host to the router, and then to the destination host as depicted in Figure 6. The peak rate of each flow is 800 kb/sec and the mean rate is 400 kb/sec. The link capacity is manually configured to be 9 Mb/sec using Alternate Queueing (ALTQ) [4]. Thus, under a peak rate allocation scheme, 11 flows would be admitted, and to ensure stability, no more than 22 flows can be admitted. In this simplified scenario, the service envelope is simply S(t) = 9t, and we configured S(t) manually (rather than measuring it) in order to establish a performance benchmark in the case in which service is known and largely deterministic. Moreover, with the single router configuration, timestamps are not required as the router monitors the original packet entrance times (similar to the case of IntServ measurement-based admission control [2]). For the experiments, α, the parameter which controls the fraction of packets violating the class delay bound (and hence controlling the loss probability as some of these packets will be dropped when the buffer is full) is 1.0. The arrival envelope is computed by evaluating the output of tcpdump at the end of each one-second interval.

Delay Reqst (msec)	Number of Flows	Mean Delay (msec)	Maximum Delay (msec)	% Outside Bound
5	16.0	1.52	17.9	1.25
10	16.3	1.83	22.5	1.20
20	18.0	2.35	36.8	0.56
60	21.1	12.85	124.7	6.16

Table 1: Single Node Baseline Experiments

We make the following observations about the experimental results reported in Table 1.[4] First, the algorithm has exploited statistical multiplexing gains, even in this scenario of a moderate number of traffic flows. The average link utilizations are in the range of 67% to 94%. As a consequence of overbooking, violations of the target delay occur and the fifth column indicates that the violations range from 0.56% to 6.16% of packets. Second, observe that assigning different delay targets has the desired impact on measured performance, allowing mean delays in the range of 1.52 msec to 12.85 msec, and maximum delays in the range of 17.9 msec to 124.7 msec. Hence, the algorithm provides the basic mechanisms for performance differentiation in multi-class networks. Finally, we observe that the targeted violations due to statistical multiplexing are not precisely met, as the percentage of violations differs in the four cases despite having the same α of 1.0 for all experiments. The differences arise from a number of factors: the quantization of

[4]All reported measurements refer to average results from at least three experiments.

the measured arrival process; the discrete nature of flows themselves (a discrete number of flows is admitted, whereas to achieve the precise QoS target may require between N and $N+1$ flows); the strong impact on QoS for each new flow in the regime of a moderate number of flows; and the extreme burstiness of the traffic itself.

Experiment 2: Ingress-Egress Pair

In the second set of experiments, we consider multiple routers but without cross traffic. As depicted in Figure 7, the system contains an ingress, core, and egress router (A, B, and C respectively), and traffic is transmitted between the two hosts at ingress node A and a single destination host attached to egress router C.

Here, we establish Pareto on-off flows with peak rate 800 kb/sec, mean rate 400 kb/sec, and a link capacity of 9 Mb/sec, so that the range of admissible flows is 11 to 22. The target delay bound is 20 msec and $\alpha = 1.0$.

Figure 8 displays an example arrival and service envelope used to make an admission decision at a particular time instance of the experiment. Observe that the arrival and service envelopes have crossed indicating that the stability condition is satisfied. Moreover, observe the general concavity of the arrival envelope and convexity of the service envelope. Convexity of service envelopes is normally evident in multi-class scenarios, for if a flow remains continually backlogged over longer interval lengths, it attains a greater service on average, due to the fluctuating demands of other flows in other classes. In this case with a single class and no cross traffic, one may expect the service to be closer to linear, i.e., S(t) = 9t. However, there are two reasons for its convexity. First, other traffic is still traversing the links, including RSVP messages, NTP traffic, and other minor but noticeable background traffic such as NFS (Network File System) traffic. Second, the empirical service envelope is actually an approximation to the true available service. For example, while an infinite rate input flow would indeed measure a service envelope of 9t, a "minimally backlogged" flow, such as described in [11] would measure a lower service envelope due to its own rate variations.

In the experiments, the maximum number of admitted flows is 19, corresponding to an average link utilization of 84%, quite similar to utilizations obtained in theory for similar types of flows (see [7] for example).[5] In this scenario, we measured a mean delay of 2.45 msec and a maximum delay of 33.9 msec, with the percentage of packets exceeding the delay bound of 20 msec measured to be 0.81%. (Refer to Table 2 below for summary results.)

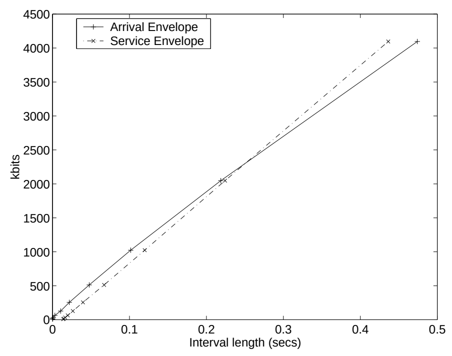

Figure 8: Measured Arrival and Service Envelopes

Experiment 3: Cross Traffic- Congested Ingress Router

In these experiments, we introduce cross traffic between the host labeled "cross traffic src 1" and "cross traffic dest 1." Thus, the cross traffic flows traverse routers A and B whereas the test flows traverse routers A, B, and C. Consequently, egress router C has no explicit measurements of the cross traffic, and must rely on its own path measurements to assess the available capacity on the link between routers A and B. We

[5] Unfortunately, no precise theoretical multi-node admission control algorithm yet exists for comparison.

establish 10 on-off flows as described above for cross traffic, which correspondingly reduces the available capacity along path A-B-C. As in Experiment 2, the delay request for users' traffic is 20 msec and α in the algorithm is again set to 1.0. With the 10 cross-traffic flows, 7.7 of the A-B-C source's flows were admitted on average. Thus, the egress node has inferred the reduction in available service as compared to experiment 3 and significantly reduced its number of admitted flows. Regardless, the 7.7 admitted flows correspond to 17.7 flows on link A-B, one less than allowed in Experiment 2, indicating that the un-measured cross traffic has caused the algorithm to slightly under admit. In the experiments, the mean packet delay is 2.2 msec, and the maximum delay is 51.4 msec, which led to 0.69% of the packets exceeding the requested target.

Experiment 4: Cross Traffic- Congested Egress Router

For the final experiments, we establish cross traffic sessions through the egress router rather than the ingress router. While node C is the egress point for both the A-B-C flows and the cross traffic, these flows do not share the same path, and hence are treated separately by node C. Thus, egress router C must again implicitly discover the cross traffic's effect on the available service. Again, 10 flows of the cross traffic were established, with α set to 1.0 and a delay request of 20 msec.

In the experiments, 7.3 flows from the primary host on path A-B-C were admitted by the egress router, totaling 17.3 flows when combined with the cross traffic. This closely approximates the 83% utilization achieved in Experiment 2. In the experiments, the mean packet delay is 3.21 msec, the maximum delay is 30.8 msec, and the percentage of packets exceeding 20 msec delay is 0.27%.

Exp. No.	Number of Flows	Mean Util.	Mean Delay	Maximum Delay	% Outside Bound
2	18.7	0.830	2.45 msec	33.9 msec	0.81
3	10+7.7	0.785	2.20 msec	51.4 msec	0.69
4	10+7.3	0.770	3.21 msec	30.8 msec	0.27

Table 2: Multiple Router Experiments

Comparing experiments 2, 3, and 4, variations in the total number of admitted flows should be expected, as the scenarios have different queueing and multiplexing characteristics. However, the percentage of packets outside the targeted delay bound would ideally be nearly identical in all three cases, at least to within the granularity of a traffic flow. Figure 9 further illustrates performance differences in the three experiments by depicting the different delay histograms in each case. While these measurements illustrate the difficulties of achieving precise control of end-to-end quality-of-service measures, the experiments do indicate that the algorithm can control admissions so that empirical quality-of-service have a strong correspondence to the targeted values.

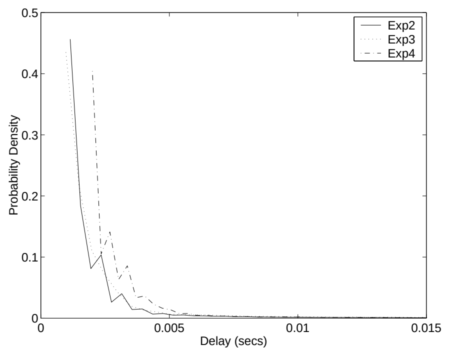

Figure 9: Delay Distribution

Experiment 5: Multi-class

In this set of experiments, we consider two traffic classes sharing a link with capacity 9 Mb/sec. Each link (edge and core) is scheduled according to Class-Based Queueing (CBQ) [5] using ALTQ with the classes having equal bandwidth allocations of 4.5 Mb/sec each and identical priorities for borrowing excess bandwidth. Class 1 has a 40 msec delay bound and $\alpha = 1.0$ whereas Class 2 has a 10 msec delay bound and $\alpha = 3.0$. We consider the same topology as in Figure 7 with Pareto on-off flows with peak rate 800 kb/sec and mean rate 400 kb/sec.

Figure 10 shows the average number of class 2 flows vs. the average number of class 1 flows admitted by the egress admission control algorithm. We make the following observations about the figure. First, the intercepts of the graph indicate that if no class 1 flows are present, 14.5 class 2 flows are admitted, whereas if no class 2 flows are present, 19 class 1 flows are admitted. This illustrates that the QoS differentiation specified by the classes' demands is incorporated into the admission control algorithm and results in a more restrictive admission policy for class 2 to achieve the more stringent performance criteria. Second, we observe that as the number of flows in one class is changed, the other class is able to exploit the available excess capacity and admit additional flows. For example, if 5 class 1 flows are admitted, class 2 can admit 10.5 flows: in this case, class 1 is transmitting at a rate that is less than half of the link capacity, and class 2 infers this additional available capacity and admits 10.5 flows, more than class 2 would be able to admit if it utilized no more than half of the link capacity. Hence, the "borrowing" mechanism of CBQ is exploited by the admission control algorithm to admit more flows than would be possible under a non-work-conserving multi-class scheme in which rate-controllers limit the available bandwidth to each class.

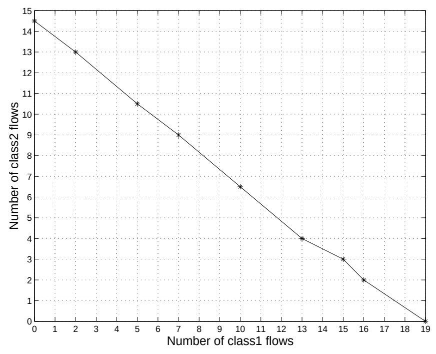

Figure 10: Multi-class Admissible Region

4 Conclusions

This paper describes our design, implementation, and measurements of Egress Admission Control, an architecture and algorithm designed to combine the strong service model of IntServ with the scalability of DiffServ, without sacrificing network utilization. While some aspects of scalability cannot be explored in a laboratory setting, our results demonstrate a key component of scalable admission control, namely, that admission control, signaling, and state management, need not be performed at each node traversed by a flow. Instead, with proper monitoring and control of the available service on an ingress-egress path, network wide quality-of-service can be ensured while signaling only egress nodes.

Acknowledgements

The authors are grateful to Dayong Huang for designing the envelope visualization tools, and to members of the Rice Networks Group for the helpful discussions. We would also like to thank Ion Stoica for the use of his dynamic-packet-state code and Brent Hendricks for sharing his knowledge of FreeBSD.

References

[1] S. Blake et al. An architecture for differentiated services, 1998. Internet RFC 2475.

[2] L. Breslau, S. Jamin, and S. Shenker. Comments on the performance of measurement-based admission control algorithms. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[3] C. Cetinkaya and E. Knightly. Scalable services via egress admission control. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[4] K. Cho. A framework for alternate queueing: Towards traffic management by PC-UNIX based routers. In *USENIX '98 Technical Conference*, New Orleans, LA, June 1998.

[5] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet network. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[6] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Transactions on Networking*, 5(1):56–70, February 1997.

[7] E. Knightly and N. Shroff. Admission control for statistical QoS: Theory and practice. *IEEE Network*, 13(2):20–29, March 1999.

[8] D. Mills. On the accuracy and stability of clocks synchronized by the Network Time Protocol in Internet systems. *Computer Communications Review*, 20(1), January 1990.

[9] K. Nichols, V. Jacobson, and L. Zhang. Two-bit differentiated services architecture for the Internet, 1999. Internet RFC 2638.

[10] P. Pan, E. Hahne, and H. Schulzrinne. BGRP: A framework for scalable resource reservation, 2000. Internet Draft, draft-pan-bgrp-framework-00.txt.

[11] J. Qiu and E. Knightly. Inter-class resource sharing using statistical service envelopes. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[12] J. Qiu and E. Knightly. Measurement-based admission control with aggregate traffic envelopes. *IEEE/ACM Transactions on Networking*, 9(2):199–210, April 2001.

[13] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, August 1999.

[14] B. Teitelbaum et al. Internet2 QBone: Building a testbed for differentiated services. *IEEE Network*, 13(5):8–17, September 1999.

[15] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analyisis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[16] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, September 1993.