An Adaptive IP Telephony Application over Differentiated Services

Matthias Scheidegger Torsten Braun
Department of Computer Networks and Distributed Systems
Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, 3012 Bern, Switzerland

Abstract

This paper describes a concept based on application adaptivity and Differentiated Services (DiffServ) in order to provide the Quality-of-Service (QoS) required by real-time applications. Assuming the availability of different service classes, we propose that a real-time application always selects the lowest / cheapest service class that still can meet the application requirements. The selection depends on RTP-based QoS monitoring and on additional probing of the quality of the next lower service class. The service class selection algorithm has been implemented within an IP telephony application and evaluated in a test-bed consisting of Linux-PC based DiffServ routers. The evaluation proves the suitability of the approach but also shows several issues for further improvement.

1 Introduction

Real-time applications such as audio/video conferencing are only useful if a certain quality can be provided to the user. Several approaches trying to attain this goal have been investigated. Most of today's real-time applications in the Internet are based on RTP and its companion, RTCP. The QoS feedback information sent from receivers to senders using RTCP messages allows the senders to detect the current transmission characteristics (such as available bandwidth, packet loss and delay jitter) and to modify the transmission parameters (e.g. bit rate, compression level or algorithm). This kind of adaptivity is attractive in scenarios where Best-Effort is the only available service. However, even adaptive applications can't guarantee a quality minimum if the available bandwidth is too low. In those cases, resource reservation is required for achieving the desired transmission. Differentiated Services are a promising approach to provide such a reservation mechanism on the Internet.

In this paper, we combine the two concepts of application adaptivity and DiffServ. We describe a concept that tries to use a better / more expensive service class such as Expedited Forwarding whenever required and a more simple / cheaper service class such as Best-Effort whenever possible. The application can switch between those service classes. The switching decision is based on monitoring the current QoS feedback, thus enabling the user to minimize costs by only using DiffServ in situations where Best-Effort can't provide the desired QoS. Our service class switching approach assumes a strict ordering of the service levels in terms of provided quality. The switching decision is not only based on the current quality of the real-time flow, but also on the quality of a second, low-bandwidth probing flow monitoring the next lower service level. Probing has also been proposed in the DiffServ context for endpoint admission control [1], but in contrast to endpoint admission control we are probing a lower / cheaper service class instead of a higher / more expensive class. Our approach also assumes that prices differ for different service classes (i.e. better service costs more) and are dependent on the traffic volume. Thus, the user should be interested to always use the lowest / cheapest possible service class. Even without traffic volume based pricing the approach might be useful since it helps to better distribute traffic load to the service classes.

Section 2 describes the basic class switching algorithm. This algorithm has been integrated into a complete IP telephony application, whose implementation is described in Section 3. A detailed performance evaluation has been performed, which proves the suitability of the presented concept. The performance measurements and the lessons learned from that are discussed in Section 4. Finally, Section 5 concludes the paper.

2 Switching between service classes

2.1 Ordering of service classes

The ability to order service classes with respect to their QoS guarantees is a prerequisite of the algorithm discussed here. [9] states that "PHBs selected by a Class Selector Code Point SHOULD give packets a probability of timely forwarding that is not lower than that given to packets marked with a Class Selector codepoint of relative order, under reasonable operating conditions and traffic loads." This means that higher order PHBs with a numerically higher DiffServ codepoint should provide a service that is

at least equal or better than lower order PHBs. To express such relations between classes we introduce symbols $>_a$, $\geq_a$, $=_a$. a denotes a specific QoS parameter and can be j for jitter, d for delay, etc. For example, $C_1 \geq_j C_2$ expresses that service class C_1 is better than or equal to C_2, with respect to parameter j, which usually stands for jitter.

2.2 Switching between service classes

Assuming that it is possible to order the available DiffServ classes, the following algorithm can be used to always select the lowest (cheapest) class being able to provide the desired service. Note that the decision strategies are open and need to be defined more accurately. The decisions may also be non-deterministic.

```
FOREVER
    wait for new QoS monitoring diagnosis
    create list L of requirements that cannot be met
    IF L empty
        create QoS monitoring diagnosis for next lower class
        IF diagnosis is sufficient
            select next lower class
        ELSE
            keep current class
        ENDIF
    ELSE
        search next higher class with respect to L
        IF found
            select this class
        ELSE
            keep current class
        ENDIF
    ENDIF
LOOP
```

Since we can assume that a higher service class provides the same or a better service than any lower service class, the decision whether to switch from the current to a higher service class only depends on the monitored performance of the current service class. If the desired QoS parameters aren't met the next higher available service class is selected for the application's data flow.

In the other direction, however, it is not sufficient to base the decision on the current class' performance only. Switching back to a lower service class only makes sense if its QoS parameters would meet the desired levels. However, we don't know any more about the situation in the lower service class than that it is equal or worse, compared to the current class. One approach might be to periodically switch back to a lower service class and see whether it provides the desired service. Unfortunately, short-term QoS degradation may occur if the service in the lower class turns out to be insufficient. Voice over IP applications are particularly sensitive in that respect. Thus, it is necessary to probe the lower service class in parallel to the regular real-time data flow. If the probing indicates that the QoS requirements might also be met in the lower class, switching back may be considered safe.

An important issue are the parameters of the probing flow, in particular its bandwidth and probing period length. We propose to use a certain fraction of the data flow's bandwidth for the probing traffic, similar to RTP/RTCP. The probing period must be long enough to get a good estimate of the QoS in the lower service class.

One can certainly think about probing a higher service class, too. However, there are two reasons not to consider this option further in our case: First, probing causes additional network load and, in order to minimize that, we decided to use probing for the next lower class only. The second reason is that a temporary switch to the higher class does not lead to a worse service according to our assumption and does, therefore, not cause QoS degradation. The only scenario where probing for higher classes would be reasonable would be to probe multiple classes simultaneously, with the goal to select the most suitable one directly. Although that would accelerate convergence of the algorithm, the bandwidth consumption would be significantly higher. One negative point of the approach is the possibility of network oscillations when it is applied by many users at the same time. However, that effect may be reduced by varying the probing time randomly. It should also be mentioned that our approach is not intended to be used by applications generating the bulk of network usage, but rather as a way to optimally exploit a given service level agreement.

3 DSPhone implementation

To evaluate the approach a simple RTP based VoIP application has been implemented. It is called DSPhone, which stands for DiffServ Phone. Instead of live audio recording and playback the program supports audio file I/O to make tests reproducible. The RTP implementation of DSPhone includes a significant part of the features required by [2]. Since it is focused on point-to-point connections the application only supports sessions with two participants, which reduces the program's complexity. Although the detailed design of DSPhone is beyond the scope of this paper, an introduction of the central concepts is necessary.

Transmission parameters like the currently used DiffServ class are stored in an instance of the *Policy* class. To be able to experiment with different approaches it is necessary to isolate the algorithms calculating these parameters and make them exchangeable. The abstract class *Policer* (not to be confused with a router's policing functionality) does that by defining a common interface for these algorithms. At the beginning of a session and on arrival of new transmission quality measurements the active Policer de-

cides whether and how the parameters in Policy are to be changed. Simple Policers only take SR/RR (RTP Sender and Receiver Reports) packets into account, more complex ones also use the values supplied by a *Scout* object, which conducts and evaluates probing of another service class.

Each Policer determines the values of the parameters contained in Policy. These include the DS codepoint of outgoing packets and the packets' payload format and size. The DS codepoints correspond to the ones from [13] and select either Best-Effort, four Assured Service classes (aka Assured Forwarding) with increasing QoS guarantees, or Premium Service (aka Expedited Forwarding). Additionally, the pseudo codepoint `IGNORE` can be used, causing DSPhone to use the system default.

All measurements and statistics related to the current session are stored in the RTPStats class. Policers use those attributes to assess the situation, focusing mainly on jitter, delay and packet loss ratio. These values must be normalized to the interval [-1,1] by *evaluation functions* before they can be used in a logical expression (-1 stands for "insufficient", 0 for "sufficient" and 1 for "exceeded"). Depending on the decision algorithm, a parameter exceeding the expectations can compensate for the slightly insufficient behavior of another.

The functions used in DSPhone are e_j, e_d and e_{pl}, for "Evaluate Jitter", "Evaluate Delay" and "Evaluate Packet Loss", and have been chosen to represent the effect of the respective parameter on the transmission quality. They are defined as follows:

$$e_j(x) = \begin{cases} 1 - \frac{1}{1+(\frac{x \cdot tp}{s})^4}, & x > 0 \\ 0, & otherwise \end{cases}$$

The constant s (for scale) chooses the turning point of the function. tp is a constant needed to make s correspond directly to that point. The form of the function represents typical properties of jitter: low jitter has small effects, but higher values necessitate larger jitter buffers. Therefore, s should be chosen to be the smallest jitter value where a larger buffer becomes necessary.

$$e_d(x) = \begin{cases} 1 - e^{-\frac{x-s}{s}}, & x > s \\ \frac{c(x-s)}{s}, & otherwise \end{cases}$$

e_d has two constants: s chooses the function's zero point (the desired delay value); c (for compensation) specifies the effect of delay below this value ($e_d(0)$ yields c). This function was chosen because a relatively small increase of delay can have significant effects on the conversation quality in low delay environments, but almost none when the delay is already high.

$$e_{pl}(x) = \begin{cases} m \cdot x, & x \geq 0 \\ 0, & otherwise \end{cases}$$

The simple function e_{pl} only has one constant, the slope m, which is usually equal to 1. Since the loss ratio is already in the interval [0,1] it can be used directly most of the time. If necessary, m can make the evaluation more strict. The functions presented here have been chosen to represent certain properties of the measures they normalize and have shown reasonable performance during the experiments. They will have to be replaced by more suitable ones in order to optimize the approach, however.

The probing mentioned in 2.2 is done by *Scout* objects. They create application specific RTCP packets and send them forth and back between the peers to estimate delay, jitter and loss ratio of a specific service class. Since they resemble ICMP Pings, these probes are called RTCP Ping packets. Once a test series has been completed the Scout object sends the result to the Policer. DSPhone implements the following Policers: *DefaultPolicer* contains no decision algorithm but uses fixed values and is used if no other Policer is selected. *StaticPolicer* is a non-dynamic Policer with selectable transmission parameters, such as the DiffServ class point. It was used to obtain reference values for the experiments below.

Two Policers implement the approach proposed in section 2. The simpler one of them is *SwitchingPolicer*, which can only choose between two service classes. Therefore, it is best suited for situations where there are just two classes, a cheap and an expensive one. In this context they are called "low" and "high" class. To evaluate the situation SwitchingPolicer compares the sum of the single evaluation functions to a threshold value. If the sum exceeds the threshold the situation is judged as bad. If the current class is high and evaluation of the situation yields good results a fallback to the low class is considered, depending on the measurements done by the Scout. Making actions depend on the sum enables us to let good results of one measure compensate for bad ones of another. To avoid changing the class too frequently, evaluations are only done after three RTP SR/RR packets have been received (This value is configurable). The algorithm implemented in SwitchingPolicer follows closely the one from section 2.2:

When new measurement data becomes available, it calculates the evaluations of loss ratio, jitter and delay using the functions e_{pl}, e_j and e_d. Then, if the application is currently sending low priority traffic and the sum of the evaluations is greater than the threshold value, it switches to the high service class and activates a Scout to observe the low class. On the other hand, if the application is sending high priority traffic and if both the evaluation sum is less than the threshold and the Scout object reports good conditions in the low service class, the algorithm switches to the low class and deactivates the Scout.

A more complex algorithm is implemented in *ServicePolicer*. It takes all of the service classes and properties

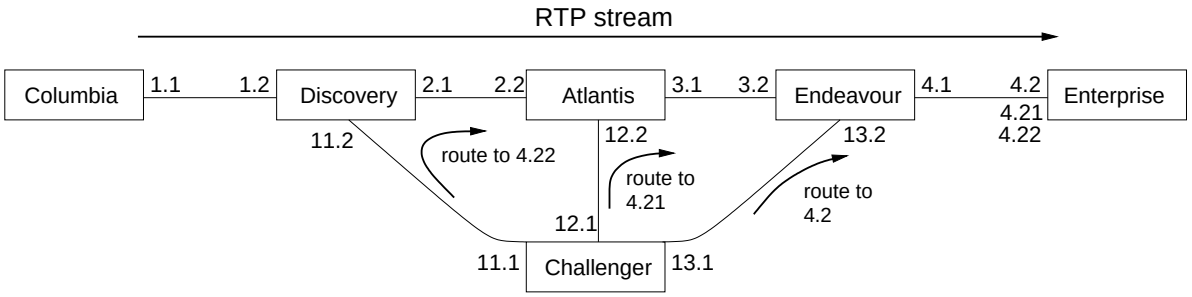


Figure 1: Topology of the test network

defined in [13] into consideration and evaluates jitter, delay and loss ratio independently with three threshold values (t_j, t_d and t_{pl}). As in SwitchingPolicer the situation is only reevaluated after three SR/RR packets have been received. The relations for the single requirements are chosen as follows ($j \rightsquigarrow$ jitter, $d \rightsquigarrow$ delay, $pl \rightsquigarrow$ packet loss ratio): "Premium service $>_{d,pl}$ assured service 4 $>_{d,pl}$ ... $>_{d,pl}$ assured service 1 $>_{d,pl}$ best effort" and "premium service $>_j$ assured service 4 $=_j$ ... $=_j$ assured service 1 $=_j$ best effort" (premium and assured service are synonyms for expedited and assured forwarding). These relations are assumptions and unrealistic if the premium service class is overloaded. The classes are clearly ordered with respect to delay and packet loss. However, the jitter relation forms two equivalence classes, one containing premium service and one containing all other classes. Rising jitter must therefore cause a transition to premium service, irrespective of the currently used service class. This makes it unclear to which class the application should return to once the situation permits it. In this case the original algorithm must be defined more precisely: The "next lower" class to fall back to is determined by the service class observed by the current Scout object. When the algorithm changes to a higher class, such a Scout object gets created to measure the behavior of the class the application is about to leave. Given that ServicePolicer changes from assured service 1 to premium service, it must consequently fall back to assured service 1. After falling back the next class to be measured must be determined. This is uniquely defined by the relation $>_{d,pl}$. In the above example the "next lower" class after falling back would be best effort.

4 Performance Evaluation

4.1 Setup of the experimentation network

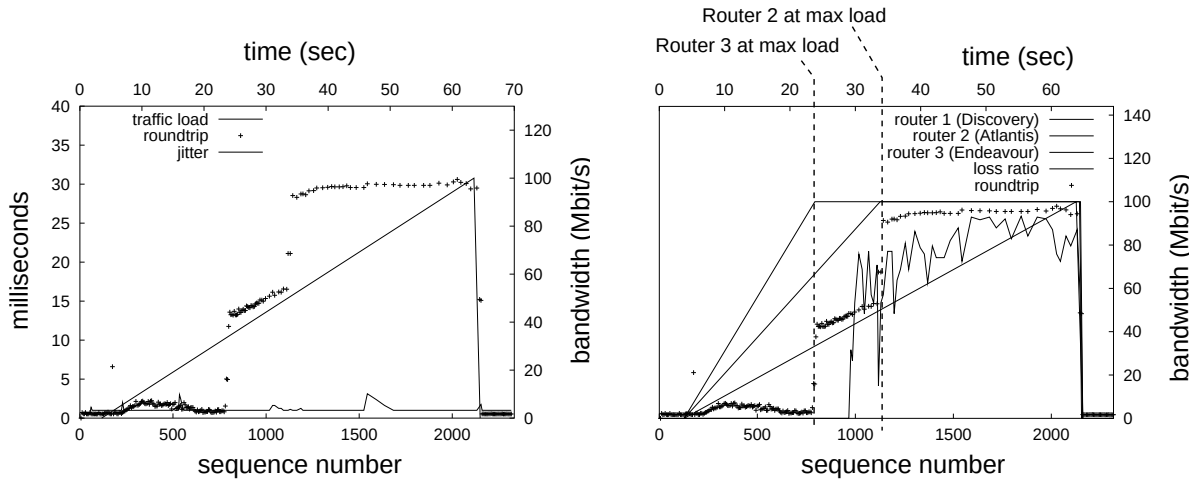
DSPhone was tested in an IP network with the following setup: The endpoints are connected over three DiffServ routers, which in turn are connected to an interfering sender. This sender generates additional network load by sending UDP packets over three different routes to the destination endpoint. Figure 1 shows the topology (The shown IP addresses only include the lower 16 bits). This routing behavior can be achieved by creating two alias addresses on the destination host (Challenger). All links are 100 Mbit/s ethernet in duplex mode.

All hosts run the same DiffServ-enabled Linux kernel 2.2.17 with different configurations: The second and third hop routers, Atlantis and Endeavour, are configured as interior nodes. Discovery is configured as ingress or as interior node, depending on the test. On the source and destination hosts no DiffServ modules are loaded. During the tests an unidirectional flow was sent from Columbia to Enterprise. It consisted of μ-Law encoded audio data with 8000 Hz sample rate and had a packetization interval of 30 ms. Interfering UDP traffic was sent from Challenger to all three intermediate hops using a program called UDPgen, which is able to send UDP packets with programmable, time variable bandwidth usage. UDPrcv measured the resulting bandwidth on the destination host.

4.2 Best-effort performance

In order to be able to correctly analyze the following tests, we analyzed the behavior of DSPhone without DiffServ using DefaultPolicer. This was done in three experiments with different network loads, i.e. without interfering traffic, with a fully congested network and with slowly rising traffic going from zero to full in 60 seconds.

The results were mostly predictable. Packet loss, delay and jitter all increased sharply once the network entered a congested state. However, the experiments also revealed a few peculiarities. First, jitter calculation showed rounding errors when the delay was very low. Second, roundtrip time during full congestion was about 5 ms higher than the calculated one-way delay from source to destination (1400 bytes of payload per packet, router queues storing 100 packets each), which is obviously due to the routers working at maximum load. The third experiment showed steep slopes in roundtrip time progression, followed by almost constant periods. Figure 2 indicates a strong correlation



(a) Delay and jitter

(b) Loss ratio and roundtrip time in comparison to router throughput

Figure 2: Best-Effort with slowly rising congestion

between these slopes and the development of router load: Every time a router reaches its maximum throughput the queue starts to grow, leading to a steep increase in roundtrip time. Once the queue is full and packets get dropped, the loss ratio increases also. Obviously the delay begins to rise some time before the routers start dropping packets. Hence, by including the change in delay in the parameters considered by SwitchingPolicer and ServicePolicer (see below), one could avoid a significant part of the packet loss. At the time the routers start dropping packets the application would already have changed the service class and could thus avoid any packet loss.

4.3 Expedited forwarding performance

The next step was to evaluate DSPhone's behavior when using a statically selected service class. The goal was to determine how much the use of higher order PHBs can protect a flow from the adverse effects of congestion in a lower PHB. As in the third test of DefaultPolicer, all three routers were put under load using UDP traffic with slowly increasing bandwidth. However, the RTP packets were tagged with the expedited forwarding codepoint while the interfering traffic remained best effort. As could be assumed because of earlier measurements [14] no packet loss occurred. Jitter remained low and stable. On the other hand the graphs show a remarkable increase in roundtrip time starting approximately when the third router reaches its load maximum (Figure 3). A possible explanation is a timing problem on the destination host. When the rate of arriving packets is very high the active UDPrcv processes can delay the scheduling of DSPhone. Arriving SR packets can't get accepted on time, causing higher roundtrip times.

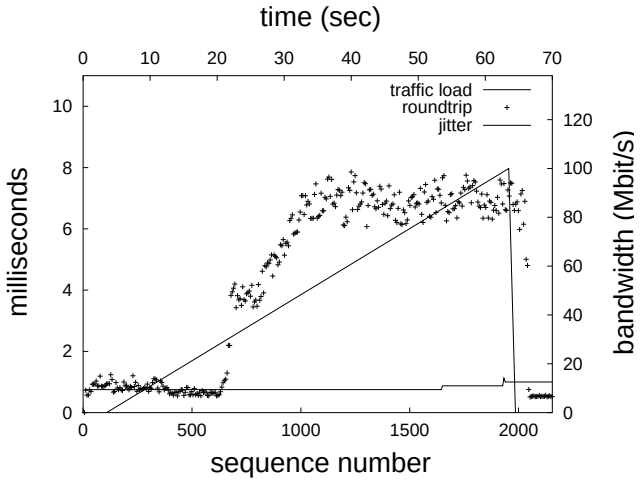


Figure 3: Expedited Forwarding with slowly rising congestion

4.4 Switching between Best-Effort and Expedited Forwarding

Next, we look at the performance of service class switching. In the first experiment we used simple switching between Best-Effort and Expedited Forwarding. We have generated three aggressive traffic peaks, which flow across

three different routes from Challenger towards the destination, 10, 15 and 40 seconds after the experiment started.

Class switching was correctly performed. Shortly after the interfering peaks became active, the application decided to switch to the high service class. The first peak has a strong impact on the loss rate of our real-time flow (Figure 4(a)), while the second peak does not have any impact on it due to the protection by EF service. The third peak has a less significant impact on the flow's packet loss since switching to the high service class occurred directly after the third peak became active. The reason for this is that the decision is being performed periodically and, in the first case, the peak became active directly after an evaluation of the flow's QoS (QoS evaluations are indicated by points on the graph). Consequently, packet loss was already high when the next evaluation took place. A more frequent QoS evaluation would further minimize the delay of service class changes and thus minimize the QoS degradation. Please note that QoS evaluations are more frequently performed while using the low service class since RTCP diagnostics are more frequent than Scout evaluations.

Another possibility to minimize the switching delay would be to take an increasing round-trip delay as a congestion indication and switch to the higher class without waiting for packets to be dropped. The progression of round-trip time in comparison to packet loss (Figure 4(b)) shows why this is reasonable. Alternatively, using active queuing strategies such as RED [12] could also improve the switching behavior, because packet loss increases more slowly, thus leaving the algorithm time to switch classes.

The figures also illustrate the decisions whether to fall back to the low service class (Figures 4(a) and (b)). These decisions are based on probing. The scout thread periodically evaluates the probing results of the low class and decides if it would be reasonable to move back to Best-Effort.

4.5 Switching between several service classes

As an example for switching among more than two service classes we flooded the paths Challenger → Atlantis → Endeavor → Enterprise and Challenger → Endeavor → Enterprise with UDP Best-Effort traffic, starting 3 seconds after the real-time flow. 30 seconds later an AF Class 1 peak followed. This scenario should force the algorithm to move from BE to AF1, and later to AF2.

The class switching behaviour turned out as expected. Figure 5(a) shows two short packet loss peaks at the beginning of a congestion period. However, they are very short, due to the quick reaction time of the application. The figure also shows the measured packet loss rate of the scout thread and illustrates how the application returned to the lower service class as soon as the measurements permitted it. Figure 5(b) additionally shows the corresponding round-trip delay and jitter measurements. Again, the packet loss rate was the dominating factor in the service class switching decision. Delay and jitter were too small to have significant impact on the switching decision.

5 Conclusions

This paper presented an approach to provide QoS for real-time applications. We implemented a Voice over IP application that was able to adapt to varying network conditions by selecting an appropriate DiffServ service class in order to get the desired QoS. The strategy is to always select the lowest (cheapest) service class which is sufficient to deliver that QoS. The performance of the proposed concept has been evaluated in a Linux router based DiffServ experimentation network. The performance measurements proved that an adaptive application can react fast enough to network congestion situations and can minimize service degradations. By using a probing mechanism the application can switch back to a lower service class as soon as congestion disappeared. Although the results are very encouraging there are several issues for further improvement. In particular, the network congestions could be discovered earlier if not only delays, jitter and packet loss are considered but also a significant increase of delay. Another improvement can be expected by the deployment of active queuing mechanisms for Best-Effort traffic.

References

[1] L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang, *Endpoint Admission Control: Architectural Issues and Performance*, ACM SIGCOMM 2000

[2] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, IETF AVT Working Group 1996

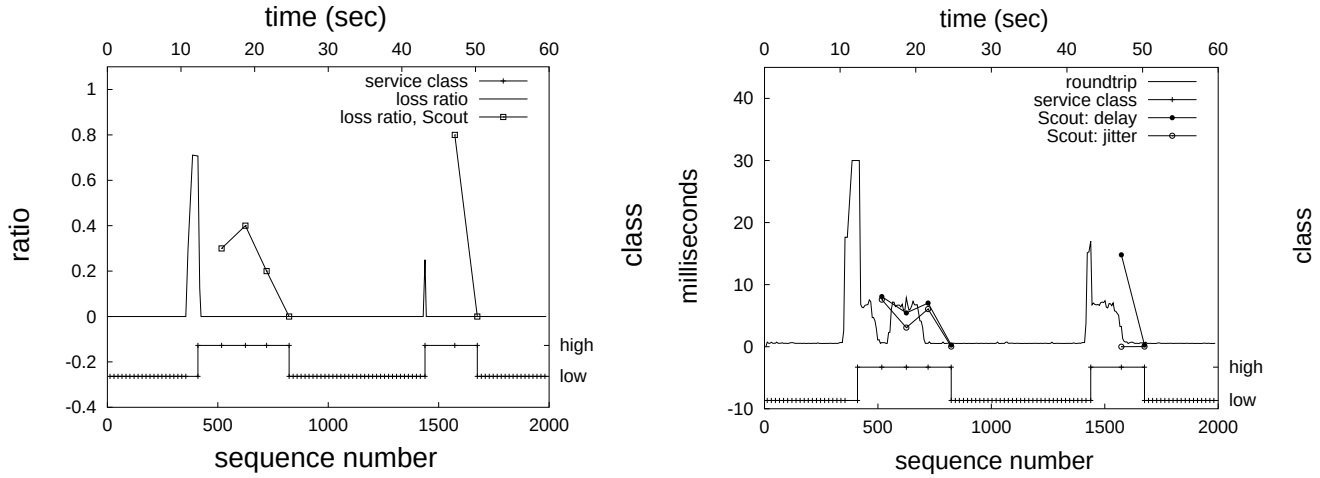
[3] H. Schulzrinne, *RTP Profile for Audio and Video Conference with Minimal Control*, RFC 1890, IETF AVT Working Group 1996

[4] H. Schulzrinne, A. Rao, R. Lanphier, *Real Time Streaming Protocol (RTSP)*, RFC 2326, IETF 1998

[5] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, *Resource ReSerVation Protocol (RSVP)*, RFC 2205, IETF 1997

[6] J. Wroclawski, *The Use of RSVP with IETF Integrated Services*, RFC 2210, IETF 1997

[7] J. Wroclawski, A. Charny, *Integrated Service Mappings for Differentiated Services Networks*, IETF 2001



(a) Packet loss ratio with SwitchingPolicer

(b) Delay with SwitchingPolicer

Figure 4: Results when switching between two classes

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*, RFC 2475, IETF 1998

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

[10] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, *Assured Forwarding PHB Group*, RFC 2597, IETF 1999

[11] V. Jacobson, K. Nichols, K. Poduri, *An Expedited Forwarding PHB*, RFC 2598, IETF 1999

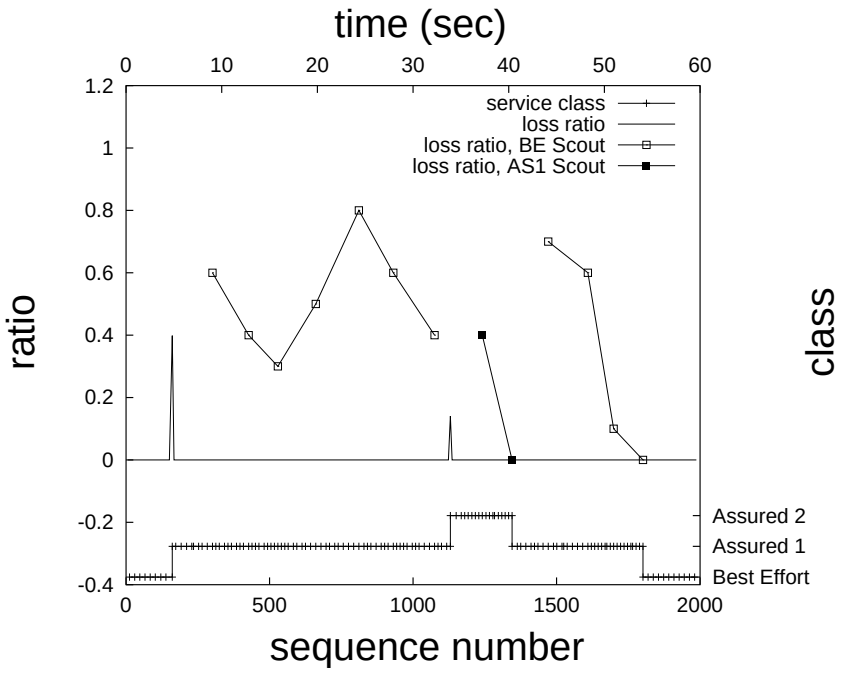
[12] S. Floyd, V. Jacobson, *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 397-413, 1993

[13] T. Braun, M. Scheidegger, H.J. Einsiedler, G. Stattenberger, K. Jonas, H. Stüttgen, *A Linux Implementation of a Differentiated Services Router*, Networks and Services for Information Society (Interworking'2000)

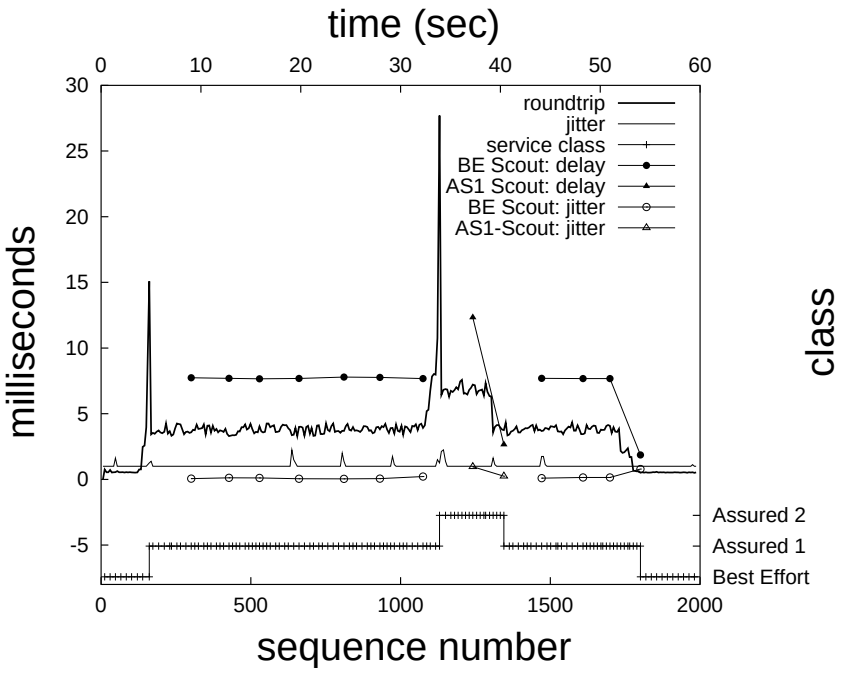
[14] G. Stattenberger, T. Braun, M. Brunner, H. Stüttgen, *Performance Evaluation of a Linux DiffServ Implementation*, 2001, submitted for publication

[15] G. Stattenberger, T. Braun, *Implementation and Configuration of a Linux Differentiated Services Router*, Technical Report, IAM-00-010, November 2000

[16] T. Braun, A. Dasen, M. Scheidegger, K. Jonas, H. Stüttgen, *Implementation of Differentiated Services over ATM*, Conference on High Performance Switching & Routing (Joint IEEE ATM Workshop 2000 and 3rd International Conference on ATM), 2000 Heidelberg, Germany



(a) Packet loss ratio with ServicePolicer



(b) Delay and jitter with ServicePolicer

Figure 5: Results when switching between several classes