Performance Evaluation of a Linux DiffServ Implementation

Günther Stattenberger [a] Torsten Braun [a] Marcus Brunner [b]
Heiner J. Stüttgen [b]

[a] *Institute of Computer Science and Applied Mathematics,University of Berne, Neubrückstr. 10, CH-3012 Bern, Switzerland*

[b] *NEC Europe Ltd., Adenauerplatz 6, D-69115 Heidelberg, Germany*

Abstract

We evaluate the beaviour of a implementation of a Differentiated Services router based on a Linux PC. Our results show, that both per-hop-behaviours — expedited and assured forwarding — are able to provide a guaranteed bandwidth together with a very small delay and jitter even in a highly congested network.

Key words: Quality of Service, Differentiated Services, Performance Evaluation

1 Introduction

Differentiated Services (DiffServ) is a well known concept to support Quality of Service in the Internet. It is — unlike Integrated Services — based on flow aggregation and therefore without the need of multifield classification at each hop. Resources are reserved for any flow aggregation e.g. for all flows between two subnets. These reservations are rather static since dynamic reservations for single connections do not scale.

IP packets are marked with different priorities, either by an end system or a router. According to the different priority classes, the DiffServ routers reserve corresponding shares of resources, i.e. bandwidth and buffer space. Packets are marked by writing a DiffServ Codepoint (DSCP) into the Type of Service - byte (ToS byte) of the IP header. Currently the first six bits of the ToS byte are used for the DSCP [NBBB98]. DiffServ routers map the packet's DSCP to a per-hop behaviour. A PHB (per-hop behaviour) is a forwarding behaviour which a router performs on a packet [JNP99,HBWW99]. In DiffServ such a forwarding behaviour is built of a combination of several components. These components are discussed in section

1.2. All router implementations should support the recommended DSCP-to-PHB mapping. A chain of routers supporting the same PHB will provide an end-to-end Quality of Service.

In this paper we will give a short introduction to the DiffServ concepts we used to implement Differentiated Services on a Linux Router. The performance evaluation of this implementation is the main topic of the paper. We will give a description of the test scenario we built in our computer networks laboratory, as well as an overwiew of the evaluation methods. Finally we will present the results regarding bandwith, delay and jitter of 5 MBit/s UDP and TCP traffic using expedited and assured forwarding and compare those results to the performance of best effort traffic forwarding.

1.1 Per Hop Behaviour (PHB)

Several services and their corresponding DSCPs have been defined in the Internet community. Nowadays, DiffServ is mainly based on the two traffic classes (called expedited and assured forwarding) defined in [JNP99] and [HBWW99].

Expedited Forwarding (EF) is also known as premium service [JNP99]. This service shall provide low delay, low loss and low jitter at a fixed rate. It will appear to the endpoints like a “virtual leased line”. To fulfill those requirements, traffic marked for expedited forwarding has to meet very short queues. Therefore, one has to ensure that there is not more EF traffic arriving at one router than the router’s configuration allows to be transported. The departure rate of a traffic flow at each hop should be independent of the characteristics of any other traffic arriving at the router. DiffServ routers will give EF packets priority over other traffic but strictly police any traffic exceeding the negotiated limit to prevent EF traffic to starve out other traffic. Another very important issue is that reordering EF packets must not appear.

Assured Forwarding (AF) defines a service assuring a high probability to transfer the traffic through the network as long as the bandwidth does not exceed the negotiated limit. Traffic exceeding the profile will be forwarded too, but it will be dropped with higher probability in case of congestion. It is also very important, that reordering of packets of the same microflow is strictly forbidden again.

Four different AF classes are defined, each allocationg a specific share of resources and thus having a different level of forwarding assurance. Within these classes packets can be marked with three possible drop precedence values. In case of con-

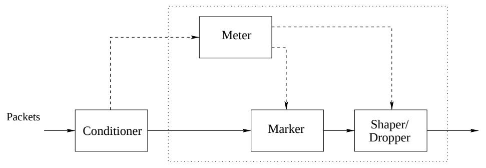

Fig. 1. Combination of DiffServ Components in a Router

gestion the in-profile traffic will be protected by dropping packets with higher drop precedence.

1.2 DiffServ Components

Each DiffServ router implementation consists of of different components (see Figure 1), which interact in a certain way to ensure the proper forwarding of traffic according to the requirements of the individual PHBs. Not all components are required in each DiffServ node. It depends on whether the concept is required in a DiffServ node or not. Those components provide different traffic conditioning functions that range from simple marking to complex shaping and policing actions. The specific parts of a DiffServ router — also called traffic conditioners — are

Classifier: This component forwards the traffic to different service handlers. Classification can depend on multiple IP-header fields, such as adresses / port numbers or protocol IDs (multi-field classifier) or the DSCP only (behaviour aggregate classifier).

Meter: The meter measures the bandwidth of the incoming traffic aggregates and provides this information to the marker or the shaper/dropper.

Marker: The marker writes a specific DSCP into the IP header. This depends either on a static mapping described in the traffic profile or some dynamic input from the meter.

Shaper: This component stores and forwards the incoming traffic. It is responsible for the compliance to the traffic profile and assures that by delaying or dropping packets. A shaper therefore provides some burst protection for the network behind it.

1.3 DiffServ Router Types

The combination of the different traffic conditioners presented in section 1.2 allows us to build several DiffServ router types (see [BBC$^+$98]).

A **Boundary Router** is located at the borders of a DiffServ domain. Several subtypes can be specified according to the location of the router in the forwarding path:

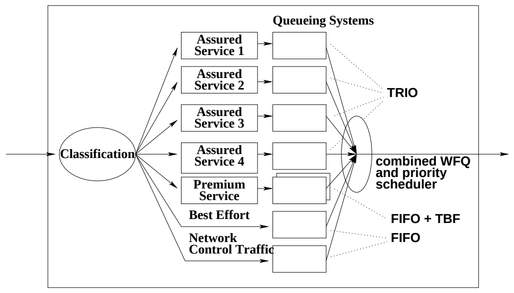

Fig. 2. Ingress Router Implementation Architecture

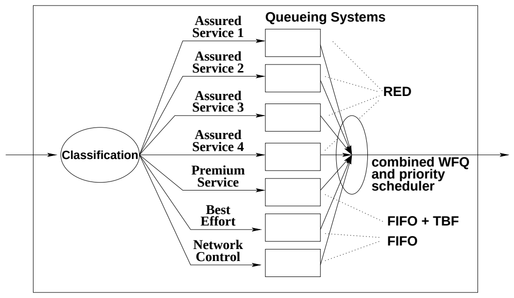

Fig. 3. Interior Router Implementation Architecture

An **Ingress Router** (see Figure 2) is located at the entry point of a traffic stream into the domain. It carries most of the traffic conditioning work, because the traffic has not been aggregated yet. Normally an ingress router architecture consists of

- a classifier, that forwards the packets to the correct service handler,
- several service handlers for the different PHBs (expedited and assured forwarding),
- queueing disciplines to perform shaping and policing to the flows and
- a scheduler, that has to ensure, that each single service class has enough bandwidth available, but does not exceed the negotiated limits.

An **Egress Router** is located at the exit point of a traffic stream. It has to ensure by traffic conditioning that the agreement about the amount of traffic leaving the DiffServ domain is met.

Interior Routers can take full advantage of the traffic aggregation. The architecture is much simpler than the architecture of a boundary router (see Figure 3). Interior routers simply handle the traffic according to the configuration concerning the service classes. As the traffic should be correctly shaped by the ingress routers, the core routers directly store the traffic in the queues. Traffic conditioning is only performed to detect possible misconfiguration of the border routers and to minimize the damage caused by that.

2 Configuration of a Linux DS-Router

The implementation of Differentiated Services on a Linux router [BSE$^+$00] provides a full set of traffic conditioning modules enabling a user to set up any kind of DiffServ router (i.e. boundary and core routers). Those modules include a marker, a classifier / scheduler, service handlers for EF and AF and several queueing disciplines such as token bucket filters, FIFO and TRIO (see section 2.1) queues. All traffic conditioners have been implemented as kernel modules that can be activated by the `tc` command, which is part of the `iproute` package [Kuz]. This command can set the parameters of the queueing disciplines (bandwidth, buffer space ...) and can combine the traffic conditioners to form the configuration of a particular

DiffServ router interface (boundary or interior router, see also Figures 2 and 3).

The information, which flows will get a certain service, is stored in a table within the router's kernel-memory. A kernel module together with an user interface program has been developed to support creating and changing those tables during runtime. The tables contain the source and destination subnet addresses of each privileged flow, marking information (the DSCP) and metering limits (bandwidth values) for the service handlers. IPv4 as well as IPv6 addresses are supported.

2.1 Traffic Conditioner Modules

The **service_handler** is the marking module of our implementation. It compares all incoming packets to the flows held in its table and writes the according DSCP into the IP header. Since this module has no metering functionality the dropping probabilities of AF packets are set by the `precedence_handler` module (see below).

The **dsclsfr** module is a combination of a BA classifier and a scheduler. The classification procedure is executed when enqueueing a packet and forwards the packets according to their DSCPs to one of seven traffic conditioners. Those conditioners are intended to handle the four assured forwarding classes, expedited forwarding traffic, network control traffic and best effort traffic. The scheduling performed by the `dequeue` function is a combination of priority scheduling and weighted fair queueing. The highest priority is assigned to expedited forwarding traffic, the second highest priority to network control traffic and the third priority to the remaining five traffic classes. Those five classes — four for assured forwarding and one for best effort — are handled by the weighted fair queueing algorithm. The weights are configurable and can be specified via the command line.

The **precedence_handler** is a color - aware two-rate three color marker [HG99]. The AF-PHB defines four independent service classes, each operating at three levels of dropping probability. Traffic below the negotiated bandwidth limit has the lowest probability of getting dropped ("is marked green"). A packet is marked "yellow" (to a higher dropping probability), if it does not exceed a certain exceed-bandwidth. All other traffic is marked "red". The `precedence_handler` specifies the color-part of the AF-DSCP (see [HBWW99]), while preserving the color of already marked incoming packets.

The **premium_shaper** is a conditioner for metering and classifying expedited forwarding traffic in ingress routers. According to [JNP99] each expedited forwarding traffic must be shaped to the negotiated rate. This module offers the possibility of shaping a certain number of flows (currently up to 256) independently to different bandwidth values by offering multi-field classification based on IP adresses, port numbers and protocol ID. For each flow a separate token bucket filter that is configured to the maximum bandwidth / burst size parameters has to be installed.

The **TRIO** queue is a modification of the well-known RED queueing algorithm [FJ93]. While the RED algorithm uses one dropping probability function, the TRIO queue uses three dropping functions, one for each color of the three color marker described above. By combining a properly configured TRIO queue with the `precedence_handler` we can ensure increasing dropping probability for yellow and red packets while sustaining the order of the packets within the AF flow.

2.2 *Example of an Linux Ingress Router Configuration*

There is a major difference between the implementation architecture of the DiffServ routers shown in Figures 2 and 3 and the actual arrangement of traffic conditioners within a Linux router: While in Figures 2 and 3 all conditioners are passed only once by each packet, the arrangement of traffic conditioners by the `tc` command is a tree (see Figure 4), and therefore a packet has to pass each conditioner twice. Additionally the classifier has to perform the scheduling simultaneously (see the description of the `dsclsfr` module above). The different service classes are handled by appropriate service handlers (i.e. `precedence_handler` or `premium_shaper`) or directly forwarded to queues (see Figure 4). To install a traffic conditioner at the correct location within the forwarding path the `tc` command needs three parameters: Each traffic conditioner has a unique ID (called `handle`), a `parent` (the `handle` of the parent conditioner in the tree), and a `class_id`, which indicates the position the conditioner occupies within multiple child conditioners. For example, if the `dsclsfr` in Figure 4 has `handle` 1, the `premium_shaper` would have `parent` 1 and `class_id` 5. Those three parameters, together with other, conditioner-specific parameters (e.g. queue length) are passed to the `tc` command.

Flow description information cannot be passed to the traffic conditioners via the command line. This information is stored in tables, allocated by the `dstable` module. This module can handle several table IDs and the conditioner modules can access the tables by using the correct ID. It is therefore possible to install a table with all EF flows and a second one with all AF flows and pass the table IDs to the correct service handlers. The communication between the tables and the traffic conditioning modules is done by a API provided by the `dstable` module.

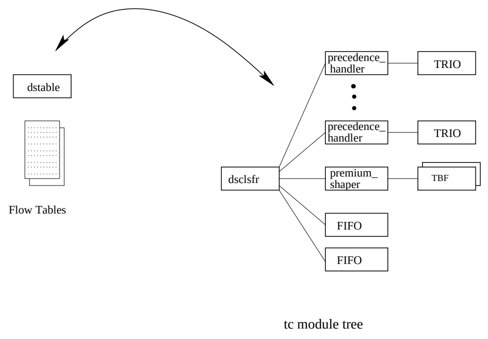

Fig. 4. Ingress Router Configuration

3 Evaluation and Tests

3.1 *Testnetwork Design*

The main problem of designing a test network for our implementation is the limitation of hardware resources (Linux PCs, Ethernet cards, ...) in a laboratory, while trying to show the availability of end-to-end quality of service in large backbone networks.

If we look to typical ISP network topologies, we recognize, that usually there are only very few backbone routers (mainly 3 – 6), between an ingress and an egress router. Therefore we set up a test network consisting of three routers: an ingress router, an interior router and an egress router (see Figure 5).

To simulate a highly congested router our small “backbone” has been flooded by an aggressive UDP sender, that sends full 100 MBit/s traffic to the receiver on each of its three links. At each outgoing router interface the DiffServ traffic from the sender has therefore to be protected against a heavy background traffic load.

Since the number of hops does not influence the service provisioning significantly we can estimate the behaviour of the DiffServ traffic classes (i.e. the provision of bandwidth and certain delay and jitter limits) for large-scale backbones.

3.2 *Performance Measurement Procedures*

In our test network we used full duplex 100BaseTx connections. As mentioned in section 3.1 the Host A sends UDP and TCP traffic to Host B, that is marked by the expedited and assured forwarding DSCP. The background traffic from Host C has three different routes to Host B. Therefore each DiffServ router has to drop 100 MBit/s (= 50%) of the incoming traffic at the outgoing interface (see Figure 5).

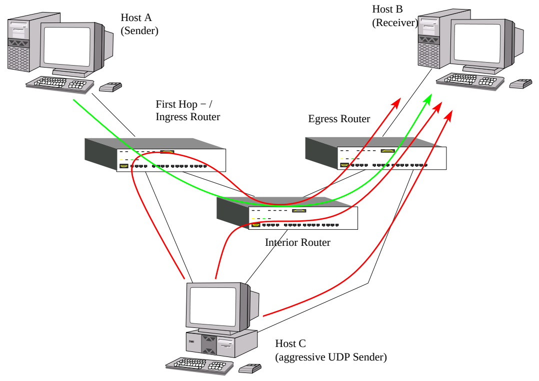

Fig. 5. Testnetwork topology

A couple of self-programmed tools were used for the load generation. These allow to set various parameters, including ports, bandwidth and packet size (only for UDP traffic). The sender program transmits a packet of the specified size, then waits for a time t

$$t = \frac{\text{packet size}}{\text{bandwidth}}$$

and transmits the next packet. Waiting can be implemented in two ways: The first possibility is to get the actual time again and again, and send the packet as soon as the calculated time interval has passed (busy waiting). This consumes a lot of computing power, but gives an accuracy of about 10 μs. The second possibility is to wait for an operating system call to resume. The main drawback of this method is, that the accuracy of the Linux operating system is limited to 10 ms – 1 ms.

Both waiting algorithms have been implemented for the UDP sender, whereas only the second has been implemented for TCP. Since the TCP implementation queues the packets in order to form a traffic stream, higher timing accuracy would be lost in this case.

The packet payload consists of an identification number to calculate loss rates and two timestamp fields, that have been used to calculate the end-to-end delay and jitter as shown in Figure 6. Before starting the tests, a two minute period of delay measurements without background traffic have been performed. The sender writes its local time t_0 to the first field, the receiver writes its local time T_0 to the second field and sends the packet back to the sender. Together with the arriving time of the packet at the sender t_1 and assuming that the transmission time is equal in both directions (we may assume that, since there is no background traffic and no congestion) we can easily calculate the delay

$$d = \frac{1}{2}(t_1 - t_0)$$

and the clockskew

$$c = T_0 - t_0 - d.$$

Figure 7 illustrates the drift between sender and receiver clocks before the first

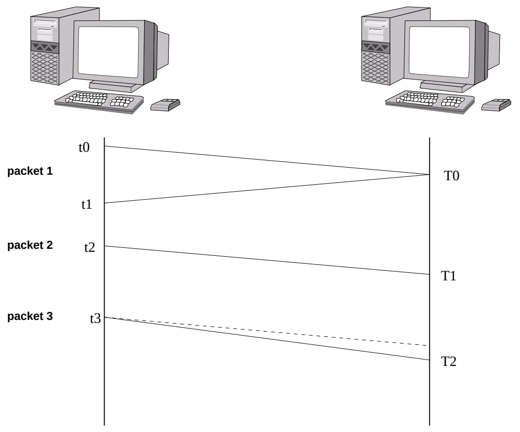

Fig. 6. Delay and jitter measurements

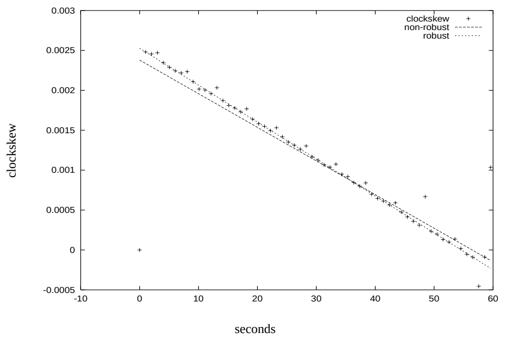

Fig. 7. Clockskew Variation

test. An obvious linear trend can be observed and assuming that this linear trend is constant during the measurements, we can compute the delay of each single packet. To estimate the trend, two different statistical methods were used: a standard least squares and a M-estimate method (minimizing absolute deviation).

The least squares method is obviously more sensitive to data that shows a large deviation from the linear trend because the error in this case is not normally distributed. Therefore, the results of the more robust method have been used in future calculations.

Now we can measure the delay and jitter during the tests (see packets 2 and 3 in Figure 6): estimating the clockskew

$$c(t) = a \cdot t + b$$

the delay of a packet is calculated by

$$d = T_1 - t_2 - c(t_2).$$

The jitter is calculated using two subsequent packets: according to [JNP99] jitter is defined by

$$j = |T_2 - T_1 - (t_3 - t_2)|.$$

This can be explained by Figure 6: packet 3 is a little bit slower (solid line) than packet 2 (dashed line). The difference of those two packets' speed is exactly the expression in the definition above. Taking the absolute value we ensure, that slower and faster packets do not average the jitter to zero.

The delay and jitter values are measured by the receiver during a configurable timescale and the average of that values is used in the graphs shown in section 3.3. The timescale used in the results section was 100 ms, which is a good compromise between the need of an accurate image of the behaviour of the DiffServ network and the limited timing accuracy of the Linux routers.

The overall duration of each test was 10 minutes to get a statistically significant amount of data, that allows us to predict the behaviour of a Linux DiffServ router as accurately as possible.

3.3 Results

Tests without DiffServ

In order to be able to compare our results to the results achieved with best effort forwarding, we performed some measurements without DiffServ. The throughput, the delay and the jitter have been measured for a 5 MBit/s UDP flow from host A to host B (see Figure 5).

During the first test there was no background traffic at all. Under these circumstances we can see, that the backbone of three routers creates a delay of just 1 ms (see Figure 9). No packet loss can be recognized in Figure 8, and the jitter is at the lowest limit of the computer's time accuracy. For most of the time we get a jitter lower than 2 μs. There is no other way to explain the variance of the jitter than by the randomness of the router's interrupts.

The second initial test was designed to test unprotected UDP traffic from Host A to Host B against the 100 MBit/s background traffic generated from Host C. The flow's bandwidth was again 5 MBit/s.

In Figure 11 we can see, that the background traffic of Host C causes serious losses even to the connectionless UDP traffic. Approximately 1 MBit/s of the sender traffic can cross the network only. This is a loss of approximately 80%. Also, the variance of the througput is considerably large.

The delay of the flow is approximately 28 ms (see Figure 12). Obviously, the traffic always meets full queues at each router due to the constant background traffic. Therefore, a packet has to wait until the maximum queue length has been forwarded. Since the default queue length of the FIFO queues in the routers is 100 packets we expect a delay of

$$d = 3 \cdot \frac{100 \cdot (1024 + 8 + 20 + 122) \cdot 8 \text{ Bit}}{100 \text{ MBit/s}} = 28.2 \text{ ms}$$

which shows the excellent accuracy of the delay measurement procedure. The values in the parentheses are the payload size (1024), the UDP header size (8), the IP header length (20) and the size of the Ethernet 802.3 header and tail (122) in bytes.

The jitter of this flow has a lower bound of 2 ms and a high variance (see Figure 13). Compared to the jitter values of the last series which has been lower by three orders of magnitude (Figure 10) we can see the big randomness of the packets

Results for UDP without DiffServ and background traffic

Results for UDP without DiffServ but with background traffic

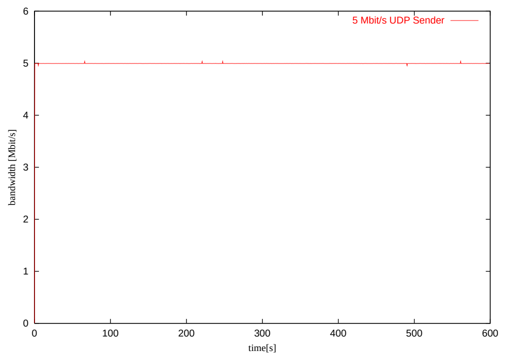

Fig. 8. Bandwidth for 5 Mbit/s UDP

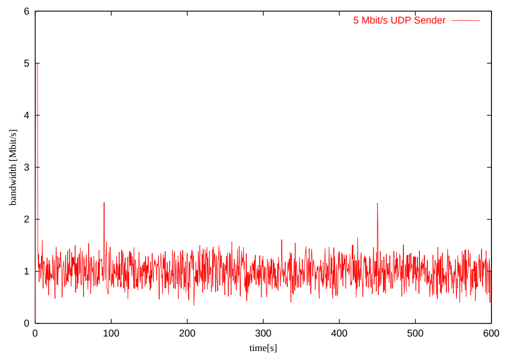

Fig. 11. Bandwidth for 5 Mbit/s UDP

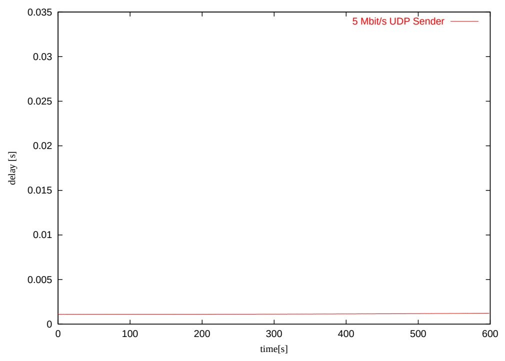

Fig. 9. Delay for 5 Mbit/s UDP

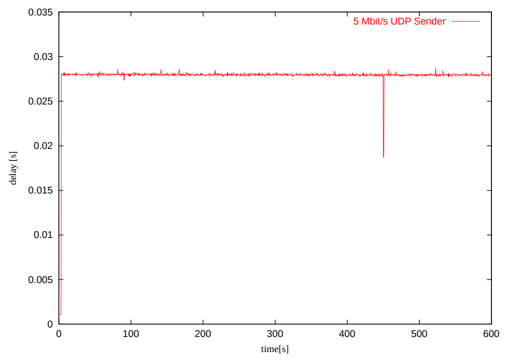

Fig. 12. Delay for 5 Mbit/s UDP

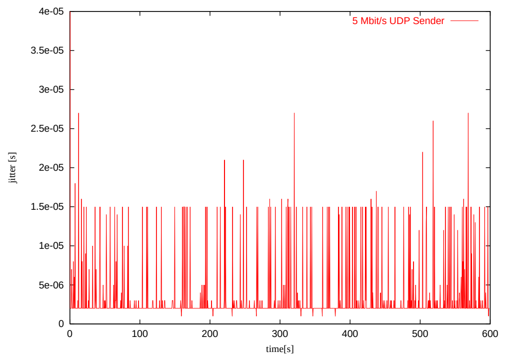

Fig. 10. Jitter for 5 Mbit/s UDP

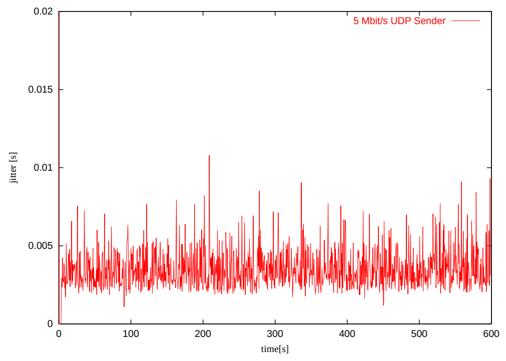

Fig. 13. Jitter for 5 Mbit/s UDP

being enqueued or dropped in the routers. This randomness results from the way, how the Linux-kernel internally handles incoming packets and forwardes them to the outgoing queue.

UDP

The UDP tests have been performed with a reservation of 5 MBit/s. We performed four different runs, sending 2, 4, 5 and 10 Mbit/s from host A to host B (see Figure

5) with UDP payload size of 200, 400, 500 and 1000 bytes. The results presented here have been achieved by using the busy-waiting UDP sender (see section 3.2). Together with the varying UDP payload this ensures a constant bit rate sender measured over very short time intervals together with a constant packet frequency. This is especially important for explaining the results for assured forwarding below.

Expedited Forwarding During the tests of the EF PHB all queues have been configured to a maximum throughput of 5 MBit/s and a maximum delay of 25 ms. This results in an increasing queue length for faster UDP senders, but this approach ensures an equal burst protection for each test run and makes it easier to compare the results.

Figure 14 shows the throughput for all runs (2, 4, 5 and 10 MBit/s). We can see, that in any case the bandwidth is surely provided up to the negotiated limit, regardless of the amount of EF traffic that flows through the router. This means, that the 2 and the 4 Mbit/s sender suffers almost no loss, but the 5 MBit/s sender looses 7.66% of it's packets, while the 10 MBit/s sender looses even 52.1% of it's packets. The reason why the 5 Mbit/s sender looses packets, although 5 MBit/s have been reserved, is, that the token bucket filter has a very small queue and therefore a bursty sender may fill up the token bucket filter and some packets are dropped.

The delay of those four flows is shown in Figure 15. As we can see, DiffServ provides a delay of less than 5 ms as long as the sender meets the bandwidth limitations.

It is rather difficult to deduct the delay of 5 ms. The only thing we can say is, that it is not due to the TBF queue, because logging the fill-state of the queue showed, that there were never more than 2 packets in the queue (=0.17 ms delay). Therefore most of the delay comes from the processing inside a DiffServ router, which can hardly be measured with the required accuracy.

If the sender tries to transmit more traffic than allowed, the traffic shaper in the ingress router will create a delay according to the size of its token bucket filter. In the case of the third test (5 MBit/s Sender and 5 MBit/s reserved) it is the restrictive burst protection of the token bucket filter, that is responsible for the high delay, as already discussed above.

The jitter has a sharp lower bound of approximately 0.6 ms for the bandwith values less than 10 Mbit/s and a large variance which is decreasing when sending at a higher rate. For the 10 MBit/s sender the jitter increases to 0.8 ms but at a smaller variance. We assume, that the token bucket filter cannot dequeue packets in equidistant timesteps — especially at higher rates — and so causes a higher jitter. However, it can dequeue a packet each time it is called, as it's queue is always full, and so the variance of the jitter is small.

Results for UDP traffic with 5 MBit/s EF reservation

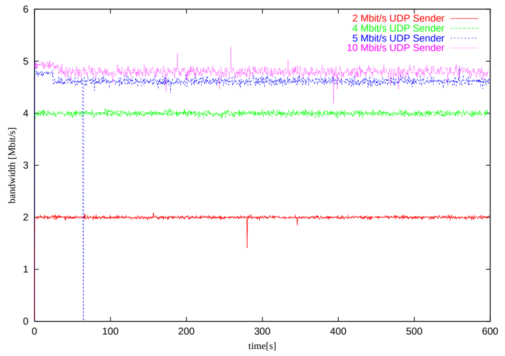

Fig. 14. Bandwidth for 5 Mbit/s EF reservation

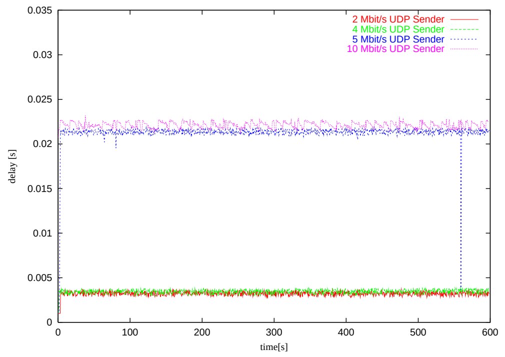

Fig. 15. Delay for 5 Mbit/s EF reservation

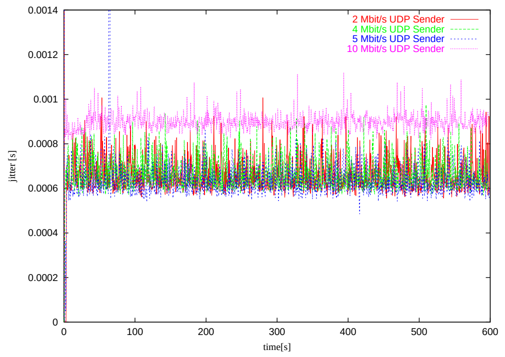

Fig. 16. Jitter for 5 Mbit/s EF reservation

Results for UDP traffic with 5 MBit/s AF reservation

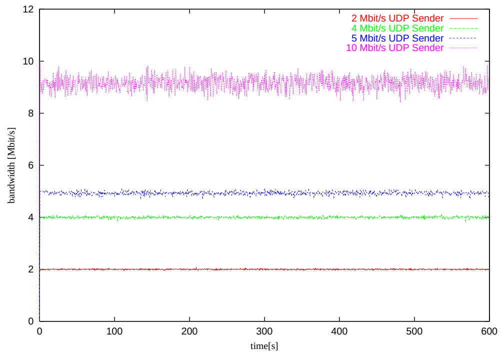

Fig. 17. Bandwidth for 5 Mbit/s AF reservation

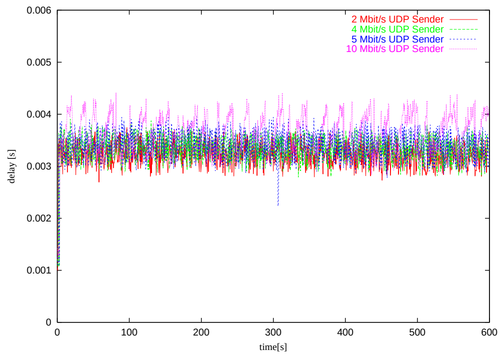

Fig. 18. Delay for 5 Mbit/s AF reservation

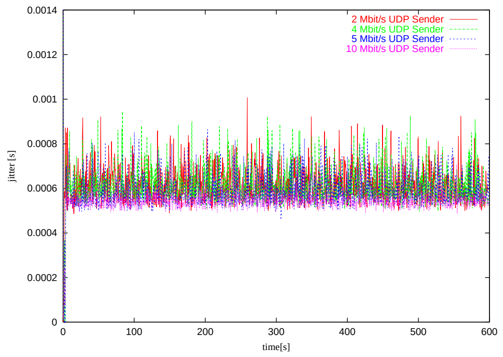

Fig. 19. Jitter for 5 Mbit/s AF reservation

Assured Forwarding The parameter settings for the four AF test runs have been chosen to allow a simple estimation of the results. This prediction is very difficult because of the random behaviour of the TRIO queue. Only in case of one AF flow of a certain class and using only two of the three possible colors an easy prediction is possible. Therefore the `precedence_handler` has been configured to mark all exceeding traffic as red. The dropping of red traffic started at the beginning of the queue and stopped at 20% queue length (i.e. all red packet are dropped), whereas green traffic was dropped at 80% to 100% queue length. The queue length was 200 packets.

Figure 17 shows a similar behaviour for the flows below the limit but also clearly shows the difference between EF and AF in handling out-of-profile traffic. While being strictly shaped in the former case (see Figure 14) we observe a significant amount of out-of-profile traffic crossing the routers in the latter case. This amount heavily depends on the configuration of the TRIO queue and the behaviour of the sender.

We can assume, that the fill-state of the TRIO queue is very small, because we assigned a 40% weight to the assured service class and we transmitted with a low rate. Therefore, a sender with no or just little bursts has a higher probability of seeing no or little queues. Its high dropping precedence traffic will also not be dropped at a very high rate although the TRIO queue starts dropping packets at the very beginning. It is therefore possible for the 10 Mbit/s sender to get almost 9 MBit/s across the network with a reserved rate of 5 MBit/s (see Figure 17).

The delays of the four flows in Figure 18 show a timely constant behaviour with values between 3 and 4 ms. The delay for senders with lower rates or exactly at the rate of the precedence handler show a random distribution, whereas the out-of-profile sender at 10 MBit/s has periodical oscillations. We assume, that those oszillations are caused by the random number functions used by the TRIO queue.

The jitter for each test was about 0.5 ms at a large variance which became smaller for higher rates (see Figure 19).

Summary EF is a good mechanism to provide UDP bandwidth under a heavy background load. The packets are forwarded almost without loss ($< 0.001\%$) as long as the sender rate is below the configured rate of the traffic shaper. The behaviour of our sender and the burst protection of the token bucket filter resulted in some packet loss and filled the queues during the test when sending exactly at the TBF's limit.

The delay shows an increase from about 1 ms to about 4 ms compared to the empty network (see Figure 9) but this is a large gain compared to the situation without DiffServ, when we see a delay of about 30 ms (see Figure 12).

The jitter increases by three orders of magnitude when we have additional background traffic (Figures 10 and 13) but DiffServ is also able to provide better results for the jitter by a factor of 5 – 10 (Figures 16 and Figures 13).

AF also proves to be able to protect UDP bandwidth against heavy background traffic. There is no obvious drawback of using assured instead of expedited forwarding for UDP traffic. The delay values are even smaller for AF traffic at a rate directly at the negotiated limit or higher, becuase there is no strict shaping of the flow that could result in filled queues. The jitter values don't even seem to depend on the DiffServ service type, they are almost the same for assured and expedited forwarding.

TCP

The TCP tests were performed with a bandwith reservation of 5 MBit/s, too. Like during the UDP tests we had four test runs with TCP senders restricted to a maximum bandwidth of 2, 4, 5 and 10 MBit/s. The TCP implementation of the sending host A set the payload size of the TCP packets itself.

Expedited Forwarding The queue settings — especially the queue length — had to be adjusted to the specific TCP behaviour. Since the packet frequency cannot be adjusted like for the UDP sender, we had to weaken the burst protection, because otherwise some TCP packets would be dropped causing TCP's congestion control mechanism to reduce the bandwidth. This would severely affect the DiffServ tests. Therefore we used a queue, that was four times larger than the queue used for the UDP tests. The queue length results in a constant maximum delay of 100 ms.

The bandwidth plot for expedited forwarding with TCP shows a very large variation (see Figure 20), especially for values at or higher than the rate limit of the token bucket filter. This behaviour is a result of the TCP congestion control function which decreases bandwidth after a single packet loss. At a timescale of 100 ms those bandwidth breakdowns can be observed in the plot, the use of larger timescales would smoothen the plot. For lower bandwith values, the influence of the congestion control is not very significant.

The delay is approximately 5 ms for the two senders with 2 and 4 MBit/s (see Figure 21). Since the congestion control limits the bandwidth to values below the rate of the token bucket filter, the queue is not filled up and therefore the delay is also approx. 5 ms, even for the sender at the negotiated limit. In particular, TCP senders with higher rates fill up the queue before the congestion control can limit the bandwidth and so the maximum delay of 100 ms can occur in these cases.

The jitter values decrease with increasing bandwidth from 10 to 3 ms. This can be explained by the stream-oriented TCP service. Due to that feature it is not so simple to influence the TCP packet size and the transmitting frequency as it was in the UDP case. The TCP sender will send a few packets at a high rate and then wait, until the average bandwith is below the TBF's limit. Therefore, the queue length a TCP packet will encounter at the router is not constant for all packets, resulting in a higher jitter.

Results for TCP traffic with 5 MBit/s EF reservation

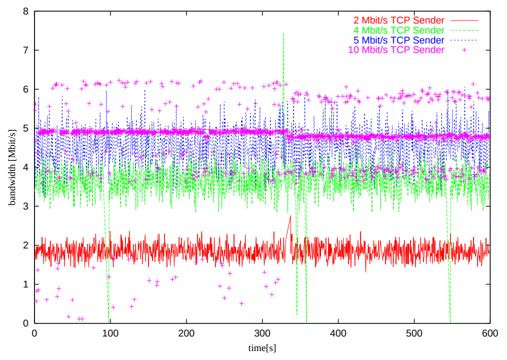

Fig. 20. Bandwidth for 5 Mbit/s EF reservation

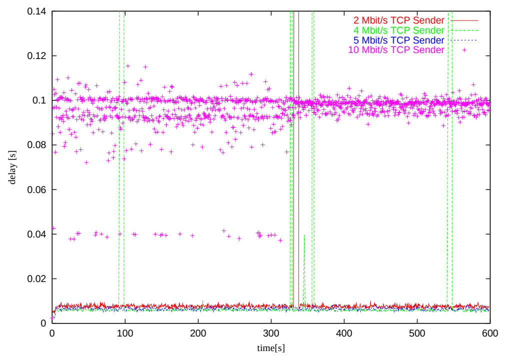

Fig. 21. Delay for 5 Mbit/s EF reservation

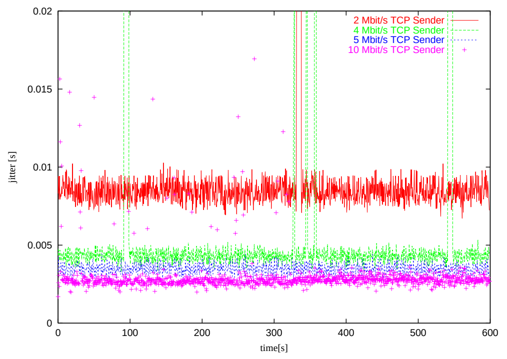

Fig. 22. Jitter for 5 Mbit/s EF reservation

Results for TCP traffic with 5 MBit/s AF reservation

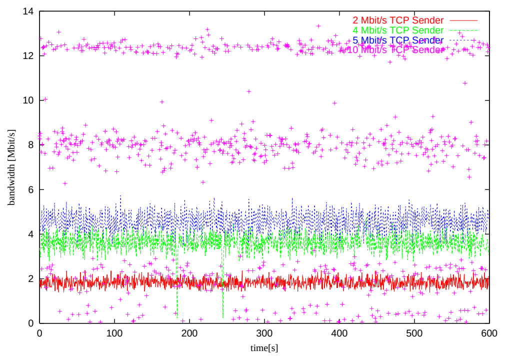

Fig. 23. Bandwidth for 5 Mbit/s AF reservation

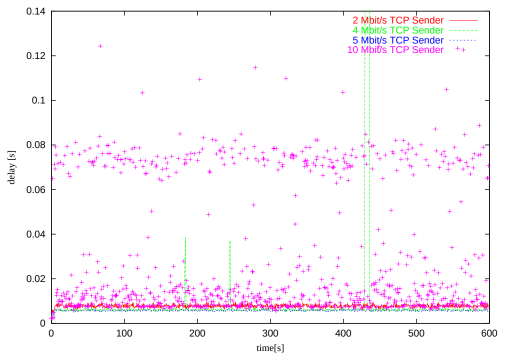

Fig. 24. Delay for 5 Mbit/s AF reservation

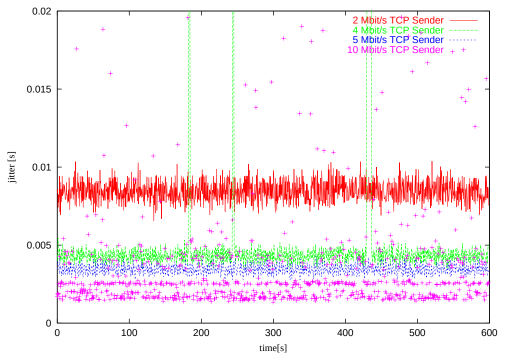

Fig. 25. Jitter for 5 Mbit/s AF reservation

Assured Forwarding The queue settings for assured forwarding did not change, because the TRIO queue length is set in units of packets and not bytes. Therefore, we used the same 200 packet queue as during the UDP test series.

We can see that for AF the bandwidth is not as constant as for EF (see Figure 23). This can be explained by the use of a TRIO queue, that randomly drops packets, even if the buffer is not filled up. Therefore, especially for higher bandwidth values there is a certain probability for packet loss and in that case TCP's congestion control decreases the bandwidth. On the other hand, the sender is now allowed to send more than the negotiated limit. All these issues give the large variation of bandwidth for assured service.

The AF delay is 6 -7 ms as long as the sender meets the rate of the precedence handler, i.e. if the AF traffic is marked with low dropping precedence (see Figure 24). For higher bandwith, some packets will be marked as high dropping precedence and probably get lost. The retransmission of those packets results in a larger delay and a high delay variation. In this case, also the jitter varies significantly (see Figure 25) compared to the jitter during the former tests.

Summary The results show, that even for rates below the limitation of the shaper the TCP bandwidth is much more irregular than for UDP. Averaging at a larger timescale would show better behaviour, but for applications which need a constant bandwidth at a small timescale the congestion control is a serious obstacle.

The delay of TCP is in the same range as the delay for UDP packets, but we have to reserve a buffer space four times as large to weaken the burst protection. This results in high delay for higher rates than negotiated.

The jitter values for TCP are larger than for UDP. This is mainly due to the different algorithms we used for the sender programs. Since TCP does not allow to specify the packet size we could not increase the packet frequency and therefore the time between two subsequent packets could be up to 20 ms (see discussion in section 3.2).

AF is not the optimal choice for TCP, because TCP cannot take advantage of the possibility of sending out-of-profile traffic. This is due to TCP's congestion control, because out-of-profile packets will be dropped with a higher probability, causing TCP to reduce the bandwidth. Therefore, we can see frequent bandwidth breakdowns and because of retransmissions a great variance in delay will occur.

3.4 Conclusion

In this paper we gave a short overview of our implementation of a DiffServ router based on Linux. A short description of the available kernel modules and a description of the router's configuration used during the performance evaluation have been presented. During this performance evaluation we tested the behaviour of UDP and TCP flows, that used either expedited or assured forwarding. Our results show, that it is possible to protect certain flows against aggressive UDP background traffic. Both, expedited and assured forwarding are able to provide a guaranteed bandwidth together with very low delay and jitter. Compared with today's best effort traffic forwarding we have smaller delay and jitter by a factor of 5. We can conclude, that DiffServ is a reliable and scalable concept to support Quality of Service in the Internet.

References

[BBC$^+$98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, December 1998.

[BSE$^+$00] T. Braun, M. Scheidegger, H. Einsiedler, G. Stattenberger, and K. Jonas. A linux implementation of a differentiated services router. In Sathya Rao and Kaare Ingar Sletta, editors, *Next Generation Networks — Networks and Services for the Information Society*, volume 1938 of *Lecture Notes in Computer Science*, pages 302 – 315, October 2000.

[FJ93] S. Floyd and V. Jacobsen. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. RFC 2597, June 1999.

[HG99] J. Heinanen and R. Guerin. A two rate three color marker. RFC 2698, September 1999.

[JNP99] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. RFC 2598, June 1999.

[Kuz] Alexey Kuznetsov. iproute2 release 990824. ftp://ftp.sunet.se/ pub/ Linux/ ip-routing/ iproute2-2.2.4-now-ss990824.tar.gz.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field in the IPv4 and IPv6 headers. RFC 2474, December 1998.