# A Linux Implementation of a Differentiated Services Router

Torsten Braun, Hans Joachim Einsiedler[1], Matthias Scheidegger, Günther Stattenberger, Karl Jonas[2], Heinrich J. Stüttgen[2]

Institute of Computer Science and Applied Mathematics, University of Berne
Email: [braun|mscheid|stattenb]@iam.unibe.ch
[1]now: T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Berkom
Email: einsiedler@berkom.de
[2]Computer & Communications Network Product Development Laboratories Heidelberg, NEC Europe Ltd.
Email: [karl.jonas|stuttgen]@ccrle.nec.de

**Abstract.** The Internet Engineering Task Force (IETF) is currently working on the development of Differentiated Services (DiffServ). DiffServ seems to be a promising technology for next-generation IP networks supporting Quality-of-Services (QoS). Emerging applications such as IP telephony and time-critical business applications can benefit significantly from the DiffServ approach since the current Internet often can not provide the required QoS. This paper describes an implementation of Differentiated Services for Linux routers and end systems. The implementation is based on the Linux traffic control package and is, therefore, very flexible. It can be used in different network environments as first-hop, boundary or interior router for Differentiated Services. In addition to the implementation architecture, the paper describes performance results demonstrating the usefulness of the DiffServ concept in general and the implementation in particular.

**Keywords:** Quality-Of-Service (QOS), Internet Protocol (IP), Differentiated Services (Diifserv), Assured Service, Assured Forwarding (AF), Premium Service, Expedited Forwarding (EF), Linux

## 1 Differentiated Services

For scalable QoS support in the Internet, the IETF is developing the Differentiated Services Architecture [2]. The IETF focuses on two services - *Assured Service* and *Premium Service* [3]. For discrimination of these different services from the currently used *Best Effort* service, the IETF proposed a special byte in the Internet Protocol (IP) header, the so-called Differentiated Services Byte (DiffServ byte). This byte contains 6 bits called DiffServ code-point (DSCP). DSCPs describe the so-called per-hop behavior (PHB), which is the externally observable forwarding behavior applied to a DiffServ flow at DiffServ capable nodes [5].

*Premium Service* (Expedited Forwarding, EF [7]) is understood as a Virtual Leased Line service where users cannot exceed the bandwidth. Premium Service is designed in order to achieve low queuing delay, e.g. for real-time applications like IP telephony. *Assured Service* (Assured Forwarding, AF [8]) assures the customer a certain amount of bandwidth. The bandwidth cannot be guaranteed but packets are labeled with higher priority for transmission over the network. Four *Assured Service* classes have been defined [6] with three dropping precedence levels (low, medium and high) each. The

S. Rao and K.I. Sletta (Eds.): INTERWORKING 2000, LNCS 1938, pp. 302-315, 2000.

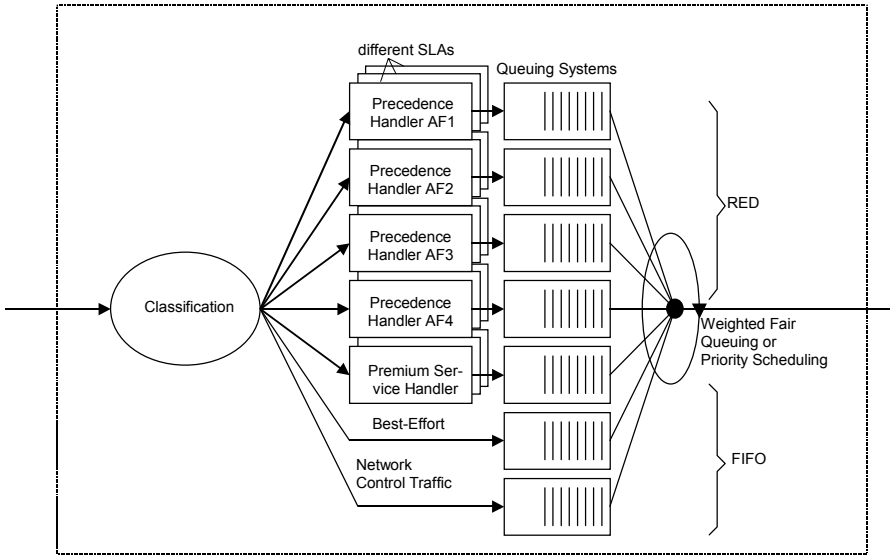


**Fig. 1.** Implementation Architecture of a DiffServ boundary router

dropping precedence levels might be increased but the packet should stay in the same class. The different classes are handled independently from each other, but packets of one micro-flow are mapped to the same class.

At a DiffServ node an incoming packet is first classified by a classifier, which identifies the service to be supported. A *Behavior Aggregate* (BA) classifier selects packets based on the DiffServ code-points only. The *Multi-Field* (MF) classifier looks also into other IP or higher layer header fields. The classifier forwards the packet to the service-dependent traffic conditioner which may include a meter, a marker, a shaper and a dropper [10]. With these components several kinds of routers can be built. A Differentiated Services network requires four different kinds of routers:

- The first hop router is placed adjacent to the sender host. Packets are classified (BA or MF) and marked per flow according to a user profile. Service handlers have to ensure conformance of the flows with the predefined profiles.
- Egress routers are located at the border between two DiffServ domains such as Internet Service Providers (ISPs). They have to make sure that the leaving traffic behaves according to the Service Level Agreement (SLA) negotiated with the adjacent domain.
- Ingress routers are also located at the entry points of a DiffServ domain and perform BA/MF classification. Their policing mechanisms restrict the incoming traffic according to the SLAs. Very often, routers at the boundary of a domain (boundary router) work as ingress routers for incoming traffic and as egress routers for outgoing traffic.
- Interior routers within DiffServ domains are responsible for forwarding according to the service requirements of the packets. They have to give higher priority to Differentiated Services packets than to *Best Effort* ones. Interior routers consider aggregated Differentiated Service flows only.

## 2 DiffServ Router Implementation Architecture

Figure 1 shows the developed implementation architecture of our DiffServ boundary router. After classification, the traffic is processed by the corresponding service handlers such as *Premium Service* shapers, policers, or *Assured Service* dropping precedence handlers and then forwarded to the associated queuing systems. For *Premium Service* traffic, network control traffic and *Best Effort* traffic, the queuing system can be a simple FIFO queue, while *Random Early Detection* (RED) is proposed for *Assured Service*. An output scheduling mechanism such as Priority Scheduling or Weighted Fair Queuing is required for selection of packets to be sent via the outgoing interface.

After classification at interior routers the packets are immediately directed to the outgoing queuing systems. While in a first hop router and in an ingress router each connected customer needs a BA/MF classifier and a conditioner, we only have BA classification in an egress router. There is only the need for one classifier and one conditioner per class.

**Table 1.** Structure of the DiffServ table

| source address | source address mask | source port | destination address | destination address mask | destination port | protocol | input DSCP | output DSCP | Bandwidth AF medium dropping precedence | Bandwidth EF / AF low dropping precedence |
|---|---|---|---|---|---|---|---|---|---|---|
| 32 bits | 32 bits | 16 bits | 32 bits | 32 bits | 16 bits | 8 bits | 8 bits | 8 bits | 32 bits | 32 bits |

A DiffServ profile can be implemented based on the IP header fields such as the IP version, source /destination address including network masks, protocol type, source/destination port and DSCP. In addition to the IP header fields, a profile table (Figure 2) includes the parameters such as bandwidth values. The profile influences marking and dropping packets within DiffServ nodes. A *Premium Service* (EF) profile requires a peak bandwidth value which is used to compute the amount of tokens that are left for the service. For *Assured Service* (AF) bandwidth values for low and medium dropping precedence are necessary.

Figure 3 shows the logical structure of an egress router or a first hop router for *Premium Service*. After classification, the packets are stored in a queue until tokens become available. Then, the packet can be sent to the outgoing queue. In the case that more packets arrive in the buffer than packets can be sent, packets must be discarded. In egress routers, it is expected that the senders (or the upstream routers) will send with the agreed rate so that the queue can be kept small [13]. In first hop routers, the buffers have to be bigger because of bursty traffic from non-DiffServ clients.

In a Premium Service ingress router, the packets are discarded as shown in Figure 4. There, we have no buffer (queue) in the traffic conditioner. The arriving packets are stored in the output queuing system, if tokens are available. Otherwise, they are discarded immediately after classification.

Another important difference between the various DiffServ router types is that in an egress router, we usually have BA classification only while first hop or ingress routers have BA/MF classification because they are connected to external customers, which have negotiated SLAs with the provider of the DiffServ domain [11].

Figure 5 shows the architecture of the Assured Service Handler which is an implementation of the three color marking concept [1]. Token buckets support the decision, whether the dropping precedence of a packet must be modified before forwarding to the *Assured Service* queuing system. The *High Dropping Precedence* packets are handled as *Best Effort* traffic but are still marked as *Assured Service* traffic. The restriction of the high dropping precedence bandwidth will be done in the outgoing queuing system. This has to provide some kind of policing functionality. The *Assured Service* queuing mechanism is an extended RED mechanism [9] with three dropping curves for each dropping precedence as depicted in Figure 6. The dropping probability is calculated by the following formula:

$$dp^{drop} = \begin{cases} 0 & for \quad ql < th_{\min}^{drop} \\ \dfrac{ql - th_{\min}^{drop}}{th_{\max}^{drop} - th_{\min}^{drop}} & for \quad th_{\min}^{drop} \le ql \le th_{\max}^{drop} \\ 1 & for \quad ql > th_{\max}^{drop} \end{cases} \tag{1}$$

| | |
|---|---|
| $dp$ | Dropping probability with $dp^{highDrop} \le dp^{mediumDrop} \le dp^{lowDrop}$ |
| $drop$ | Dropping precedences (low, medium and high) |
| $th_{min}$ | Minimum threshold of the queue |
| $th_{max}$ | Maximum threshold of the queue |

$$th_{Start\,drop} = th_{\min}^{HighDrop}$$

$$th_{Hard\,drop} = th_{\max}^{LowDrop}$$

In our implementation, we try to protect the network control traffic against dropping in the way that its priority is just below the priority of *Premium Service*. This is valid for both Priority Scheduling or Weighted Fair Queuing output queuing, which are the two options implemented for output queuing (see Section 3.2)

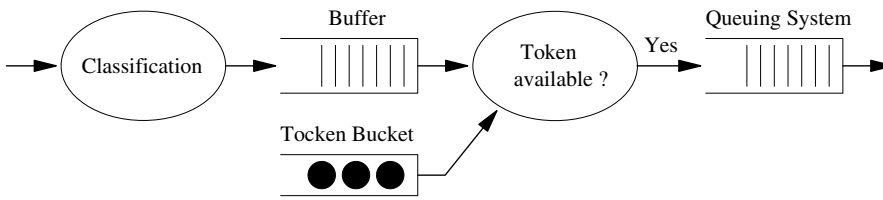


**Fig. 2.** Egress or first hop *Premium Service* route

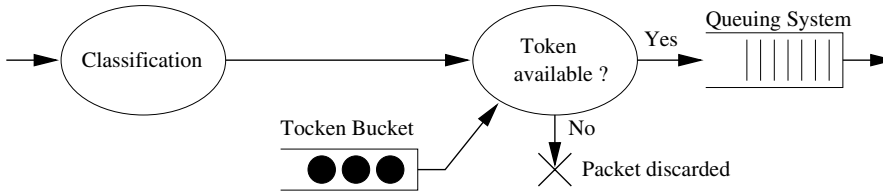


**Fig. 3.** Ingress *Premium Service* router

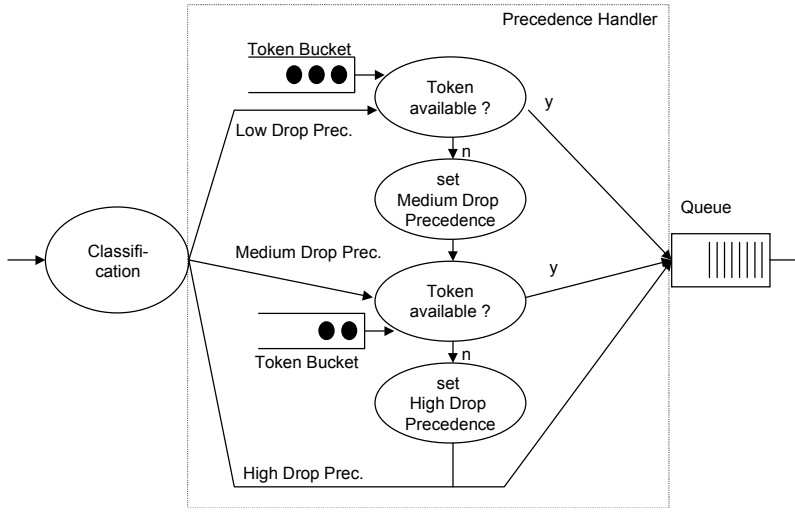


**Fig. 4.** Functionality of the precedence handler

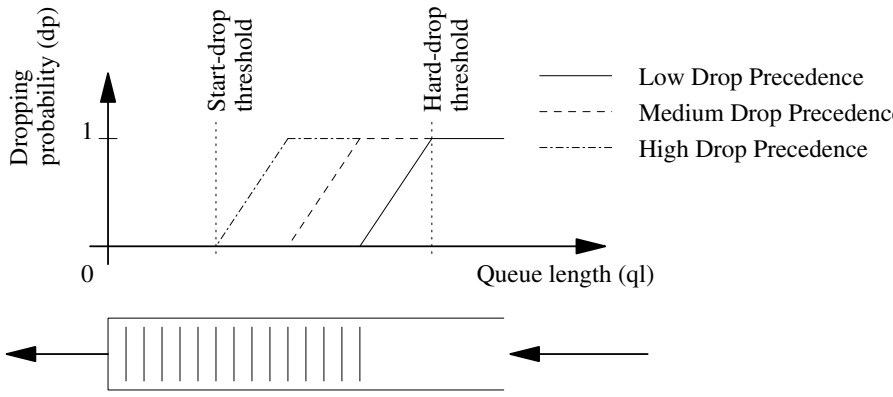


**Fig. 5.** RED queue for three dropping precedence

## 3 Linux Implementation of DiffServ Routers

Linux kernels allow a wide variety of traffic control functions [13]. Several DiffServ modules have been made available for Linux [14, 15]. The traffic control functions are either hard-coded during compiling the kernel or they can be loaded dynamically after system initialization or during run-time. Our implementation takes advantage of the second option and allows to change the configuration without resetting the node. New modules such as shapers, markers, meters and droppers can be added or removed via the command line. The traffic control allows to compile a single kernel for a node, which can be configured as DiffServ first hop, ingress, egress and interior routers, supporting flexible SLAs.

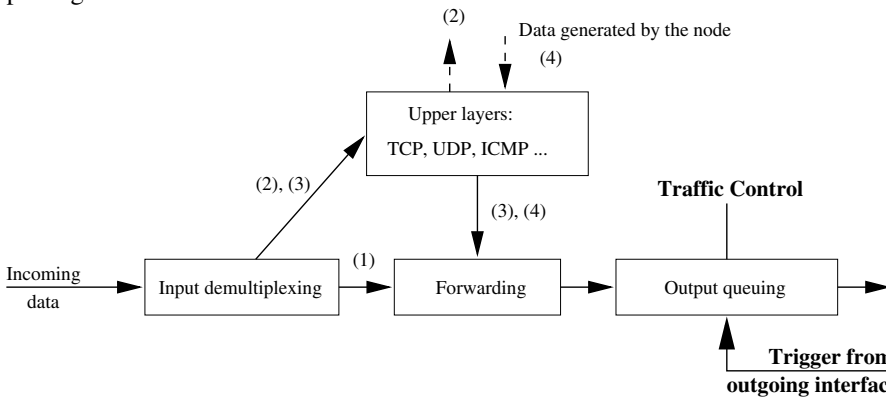


**Fig. 6.** Network packet processing

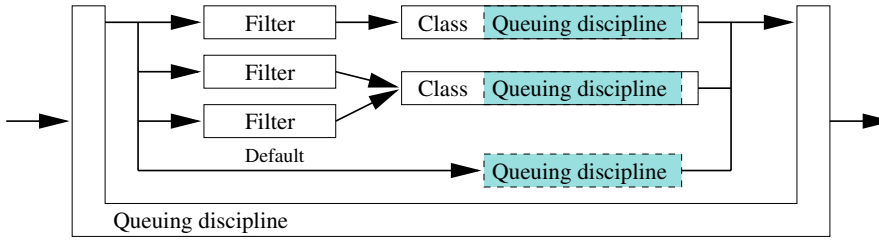


**Fig. 7.** Queuing discipline with filters and classes

### 3.1 Linux Networking Support

Figure 7 shows IP packet processing in the Linux kernel. A router forwards received packets directly to the network, e.g. to another interface (1). If the node is also an end system (server, workstation, etc.) or an application level gateway, the packets destined to it are passed to higher layers of the protocol stack for further processing (2). This can also include manipulation of fields and then forwarding to the network (3) again. An end system can generate packets by itself, which then will be sent through the protocol stack to the forwarding block (4). The forwarding component does not only include the selection of the output interface but also the selection of the next hop, encapsulation, etc.

From there, the packet is queued for the particular interface. This is the point of *traffic control* execution. Manipulation, such as delaying packets, changing header fields, dropping etc. can be done there. After traffic control has released the packet, the particular network device can pick it up for transmission. The output queuing block is triggered by the output interface. For processing the next packet, the interface sends a start signal to the output queuing block.

After compilation of `tc` and loading the modules, the code components can be added via the command line or a management tool (a Shell or Perl script) to the outgoing queuing block. The code consists of queuing disciplines, classes (the identification of a queuing discipline), filters, and policing functions (within filters and classes). Figure 8 shows an example of a queuing discipline. Packets, which are forwarded over the same interface, may desire different treatment. They have to be enqueued into different queuing disciplines. For an enqueued packet the called queuing discipline runs one filter after the other until there is a match with a class. Otherwise, the default queuing discipline is used. In the case of a match, the packet is enqueued in the queuing discipline related to the class for further manipulation of the packet. Different filters can point to the same class. Policing functions are required in the queuing disciplines to ensure that traffic does not exceed certain bounds. For example, for a new packet to be enqueued, the policing component can decide to drop the currently processed packet or it can refuse the enqueuing of the new one. Each network device has an associated queuing discipline, in which the packets are stored in the order they have been enqueued. The packets are taken from the queue as fast as the device can transmit them.

### 3.2 DiffServ Modules for Linux

The framework of our implementation mainly focuses on the enqueue and dequeue components of the queuing discipline structure because these components are the right place for the Differentiated Services implementation. The implementation of six queuing disciplines was necessary to cover all four kinds of DiffServ nodes. Some queuing disciplines work with profiles. During initialization of these queuing disciplines, the information is copied from a file into the memory, from where the queuing discipline can access it.

The ***DiffServ Service Handler*** sets the Class Selector Codepoint according to a profile. Packets that do not match with the profile are forwarded as *Best Effort* packets. Network control traffic is forwarded untouched as well. During enqueuing a function is called, which compares the packet header with the profile and then marks the packet with the respective service.

The ***DiffServ Classifier*** splits the traffic into the seven service branches (*Premium Service*, network control traffic, four *Assured Service* classes and *Best Effort* service) according to the DSCP. For dequeuing packets from the different queues, priority scheduling or a weighted fair queuing variant can be used.

- For priority scheduling, the packets are dequeued depending on the priority parameters given to the queuing discipline during initialization. The queue with the highest priority is the first queue that will be accessed. A queue can send only if all queues with a higher priority are empty. The recommended default priority sequence is Premium Service, network control traffic, Assured Services Classes 1-4, Best-Effort.
- The Weighted Fair Queuing (WFQ) variant gives always highest priority to Premium Service packets over network control packets. The remaining bandwidth is shared among the Assured Service and Best-Effort packets according to the configured parameters.

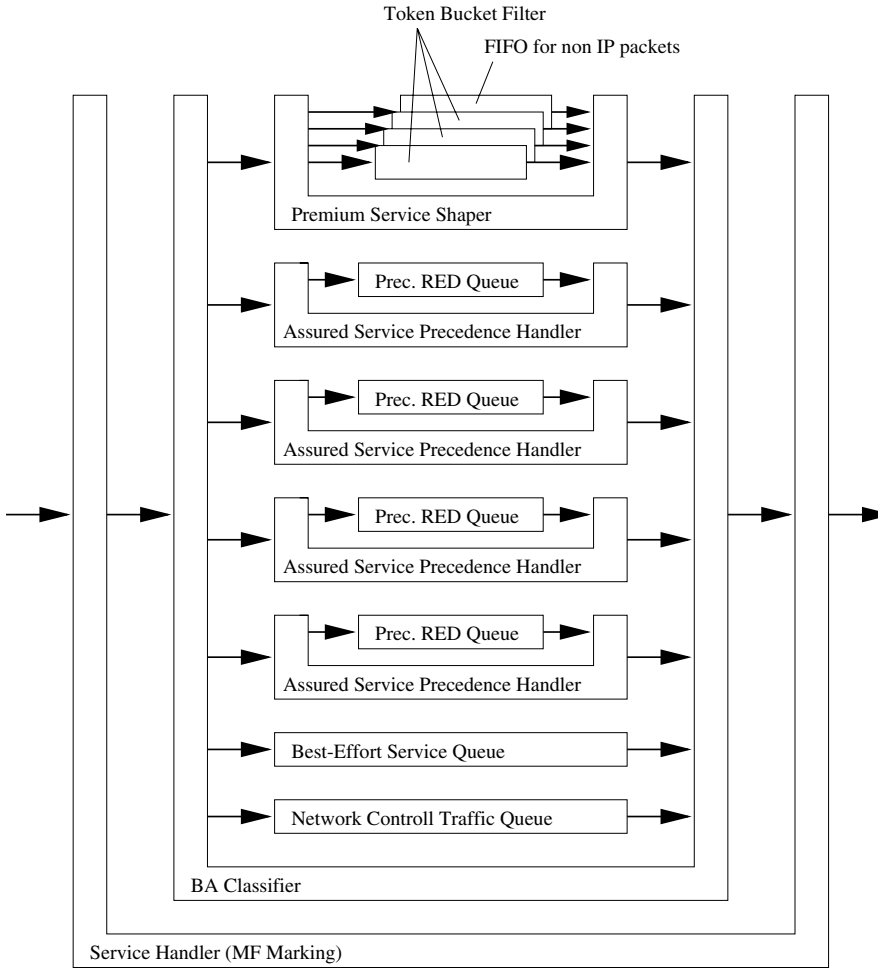


**Fig. 8.** First hop router

The ***Assured Service Precedence Handler*** checks incoming packets whether they are in-profile by measuring the packet size (in bytes) against a token bucket. An in-profile packet is forwarded and the tokens are decremented by the size of the packets. Otherwise, the packet is re-marked with a higher dropping precedence. In the case of medium dropping precedence, the packet is measured against the respective token bucket. If no tokens are available, the packet is forwarded directly into the outgoing queue and marked with high dropping precedence. The marked packets might then be discarded in the following Three Way RED queue.

The ***Three Way RED Queue*** drops packets according to the RED parameter values. This is done in the enqueue component. The dropping probability is calculated depending on the dropping precedence of incoming packets. The packet will then either be dropped or forwarded to the following FIFO queue in which the packet is stored for dequeuing. Two parameter sets are required. A limit defines the maximum number of packets, the

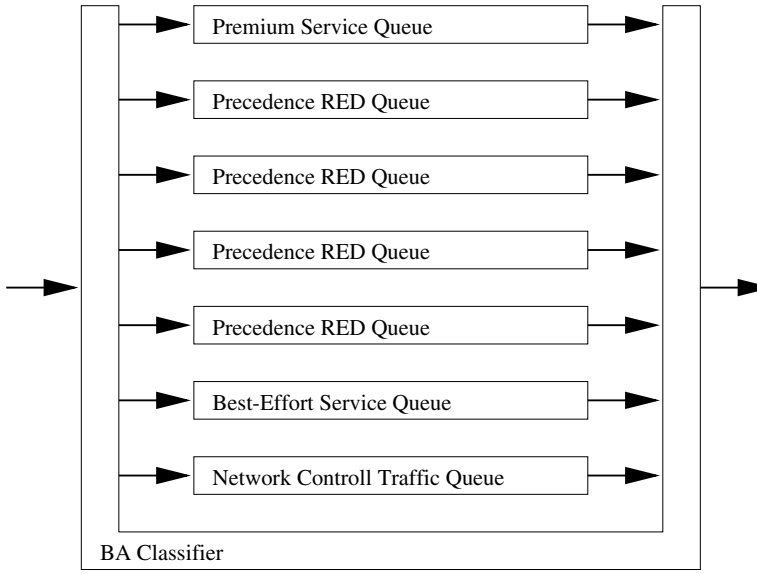


**Fig. 9.** Interior router

Three Way RED queue can buffer, and floating-point numbers define the thresholds of the discarding curves.

The ***Premium Service Policer Handler*** polices the *Premium Service* traffic according to *Premium Service* profiles. An arriving packet will be forwarded to the FIFO queue, if there is a match with the profile and if there are enough tokens available in the respective token bucket. Otherwise, the packet is not conforming to the profile and is discarded.

Since we have to use a shaper for each *Premium Service* flow, there is the need for a ***Premium Service Shaper Handler*** that forwards the packet to these shapers. IP packets matching a profile are forwarded to the respective shaper and are discarded otherwise. This shaper module is part of the standard Linux kernel distribution [20, 22].

### 3.3 Structure of DiffServ Routers

Figure 9 shows the structure of a first-hop router with MF and BA classification. For *Premium Service*, shaping is used, the Three Way RED queue has been selected for the different AF classes. An egress router looks quite similar but does not need a service handler for MF marking. An ingress router differs from the egress router in having policing instead of shaping for *Premium Service* and in having a service handler for BA/MF re-marking.

Figure 10 shows the simplicity of an interior router. The packets are classified in the BA classifier and forwarded to the corresponding queue of each service.

## 4 Performance Measurements

For performance measurements we used the `ttcp` tool and a self-written UDP socket program for the generation of aggressive bursty flows to several destinations. We configured one router as first hop and ingress router in order to test the interoperation between the queuing disciplines. For the measurements we used `tcpdump` and shell scripts processing the `tcpdump` traces.

The following setup has been chosen for the measurements with two flows: The host `elmer` has been the source of the DiffServ flows (*Assured* or *Premium Service*) and `weasel4` has been the destination. The *Best Effort* background traffic has been sent from `elmer` to `weasel`. The background traffic has been generated for each measurement and filled up the rest of the available bandwidth on the 10 Mbps link. In the sessions with three flows, elmer has been the source of the third flow and `weasel5` has been the destination. `weasel`, `weasel4` and `weasel5` were different logical interfaces using the same Ethernet interface of one host. The traffic source `elmer` was connected via an 100 Mbps link. The router had an incoming 100 Mbps interface and formed by the 10 Mbps outgoing interface a bottleneck.

Table 1 shows measurements of a *Best Effort* background flow and a *Premium Service* flow with different bandwidth values for shaping and for policing.

Table 2 shows the measurements of two *Premium Service* flows through two shapers together with a *Best Effort* background flow.

Table 3 shows measurements with an *Assured Service* flow and a *Best Effort* background flow. The *Assured Service* parameters have been as follows:

- Queue length: 10 packets
- Low dropping precedence, Start: 0.9, End: 1.0
- Medium dropping precedence, Start: 0.1, End: 0.5
- High dropping precedence, Start: 0.0, End: 0.1

Table 4 shows measurements with constant Three Way RED queue parameters but different bandwidth values for the Precedence Handler. We had an *Assured Service* flow and a *Best Effort* background flow.

*Assured Service* parameters:

- Queue length: 20 packets
- Low dropping precedence, Start: 0.9, End: 1.0
- Medium dropping precedence, Start: 0.2, End: 0.5
- High dropping precedence, Start: 0.0, End: 0.2

Finally, we mixed *Assured Service* traffic with *Best Effort* traffic and *Premium Service* traffic, the last one was policed (ingress router setup). We had an *Assured Service* flow, a *Premium Service* flow, and a *Best Effort* background flow.

*Assured Service* parameters:

- Queue length: 20 packets
- Low dropping precedence, Start: 0.9, End: 1.0
- Medium dropping precedence, Start: 0.1, End: 0.5

High dropping precedence, Start: 0.0, End: 0.1

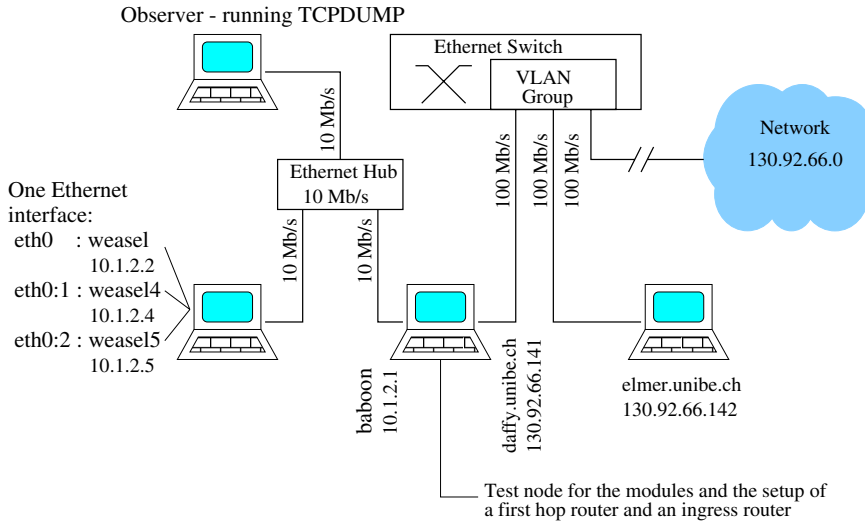


**Fig. 10.** Test network

**Table 2.** *Premium Service* shaping and policing with different bandwidth values

| | *Bandwidth (setup)* | *Number of Packets* | *Achieved Bandwidth* | *Time Period* |
|---|---|---|---|---|
| Shaper | 64 kb/s | 3570 | 65.7 kb/s | 628.39 s |
| | | 4708 | 65.6 kb/s | 830.29 s |
| | 128 kb/s | 6372 | 131.3 kb/s | 561.78 s |
| | | 6350 | 131.4 kb/s | 559.35 s |
| Policer | 800 kb/s | 41408 | 799.8 kb/s | 599.35 s |
| | | 32229 | 799.2 kb/s | 466.80 s |
| | 1.28 Mb/s | 133105 | 1.280 Mb/s | 1203.74 s |
| | | 126654 | 1.279 Mb/s | 1146.25 s |

**Table 3.** Two parallel *Premium Service* shapers in parallel

| | *Bandwidth (setup)* | *Number of Packets* | *Achieved Bandwidth* | *Time Period* |
|---|---|---|---|---|
| Shaper | 128 kb/s | 8404 | 130.8 kb/s | 743.78 s |
| | 64 kb/s | 4217 | 65.6 kb/s | |

**Table 4.** Assured Service Flows

| *Dropping Precedence* | *Bandwidth (setup)* | *Number of Packets* | *Achieved Bandwidth* | *Time Period* |
|---|---|---|---|---|
| low | 800 kb/s | 58201 | 799.5 kb/s | 842.73 s |
| medium | 640 kb/s | 46559 | 639.5 kb/s | |
| high | - | 0 | 0 | |

**Table 5.** Different precedence handler bandwidths

| *Dropping Precedence* | *Bandwidth (setup)* | *Number of Packets* | *Computed Bandwidth* | *Time Period* |
|---|---|---|---|---|
| low | 800 kb/s | 63850 | 798.9 kb/s | 925.17 s |
| medium | 640 kb/s | 51006 | 638.2 kb/s | |
| high | - | 190059 | 2378.1 kb/s | |
| low | 1280 kb/s | 70555 | 1276.6 kb/s | 640.80 s |
| medium | 960 kb/s | 52321 | 945.2 kb/s | |
| high | - | 87669 | 1583.7 kb/s | |

**Table 6.** Ingress router with *Premium*, *Assured* and *Best Effort* Service

| *Service* | *Dropping Precedence* | *Bandwidth (setup)* | *Number of Packets* | *Achieved Bandwidth* | *Time Period* |
|---|---|---|---|---|---|
| Assured Class I | Low | 1280 kb/s | 44812 | 1276.8 kb/s | 406.29 s |
| | Medium | 960 kb/s | 32212 | 917.8 kb/s | |
| | High | - | 13454 | 383.3 kb/s | |
| Premium | Policer | 1280 kb/s | 44813 | 1276.8 kb/s | |
| Assured Class I | Low | 1280 kb/s | 72544 | 1278.8 kb/s | 656.70 s |
| | Medium | 960 kb/s | 52886 | 932.2 kb/s | |
| | High | - | 23740 | 418.5 kb/s | |
| Premium | Policer | 1280 kb/ s | 71349 | 1257.7 kb/s | |

## 5 Related Work

There are currently several DiffServ implementations under Linux being developed, e.g., the KIDS implementation from University of Karlsruhe [16]. The most similar one to our implementation is the implementation described in [14] which we will call the EPFL implementation hereafter. This and our implementation are both based on the Linux traffic control package. While for our implementation sophisticated DiffServ queuing and scheduling components such as the Three-Color-Marking for Assured Service have been developed, the EPFL implementation tries to use more general components not tailored to DiffServ. The classification of the EPFL is more flexible but required to modify core

Linux data structures while our implementation avoided this. For output queuing, we developed a WFQ variant based on the bad performance behavior experienced from other available output scheduling mechanisms. Another significant difference is the kind of configuration of both implementations. The EPFL implementation requires rather long and more complex `tc` configuration scripts. By using ASCII configuration tables we believe that our approach simplifies the configuration of a DiffServ router by human users. Our implementation also allowed to integrate a layer-4 flow detection mechanism [12]. In addition, special queuing disciplines for ATM have been implemented that allow to replace software shaping and policing by ATM hardware [17].

## 6 Conclusions and Outlook

This paper described a DiffServ implementation for Linux performing DiffServ processing at the egress point of a router or an end system. The measurements clearly show the usefulness of our implementation architecture. In addition, some Differentiated Service processing such as policing could be located at the ingress interface of a router. This would allow to perform traffic conditioning functions on the whole traffic received from a single DiffServ domain, e.g. from a single customer. Otherwise, if the traffic is spread over several output interfaces, the aggregate traffic can not be policed correctly. More detailed performance measurements allowing the comparison with the KIDS implementation will be published in a subsequent paper.

## Acknowledgements

The implementation platform used at the University of Berne has been funded by NEC Europe Ltd. the SNF R'Equip project no. 2160-053299.98/1, and the foundation "Förderung der wissenschaftlichen For-schung an der Universität Bern". NEC Europe Ltd. funded several persons involved in this project at University of Berne. The authors are grateful to Werner Almesberger (EPF Lausanne) for constructive discussions on Linux related issues.

## References

1. J. Heinanen, R. Guerin: "A Single Rate Three Color Marker", Internet Draft draft-heinanen-diffserv-srtcm-01.txt, May 1999.
2. IETF-DiffServ-Working Group, "Differentiated Services for the Internet." http://www.ietf.org/html.charters/diffserv-charter.html/.
3. K. Nichols, V. Jacobson, and L.Zhang, "A Two-bit Differentiated Services Architecture for the Internet", Internet RFC 2638, July 1999.
4. K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the Differentiated Service Field (DS Field) in the IPv4 and IPv6 headers," Internet RFC 2474, December 1998.
5. M. Borden and C. White, "Management of PHBs", Internet Draft: draft-ietf-diffserv-phb-mgmt-00.txt, September 1998.
6. J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group", Internet RFC 2597, June 1999.
7. V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB", Internet RFC 2598, June 1999.

8. D. Clark and J. Wroclawski, “An Approach to Service Allocation in the Internet”, Internet Draft draft-clark-diff-svc-alloc-00.txt, July 1997.
9. S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance” in IEEE/ACM Transactions on Networking, Vol.1 N.4, pp. 397-413, August 1993.
10. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An Architecture for Differentiated Services”, Internet RFC 2475, December 1998.
11. Y. Bernet, D. Durham, and F. Reichmeyer, “Requirements of Diff-Serv Boundary Routers”, Internet Draft draft-bernet-diffedge-01.txt, Nov. 1998.
12. T. Harbaum, M. Zitterbart, F. Griffoul, J. Röthig, S. Schaller, H. J. Stüttgen: Layer 4+ Switching with QoS support for RTP and HTTP, Proceedings of IEEE Globecom Conference, Rio de Janeiro, Brazil, December 1999
13. A. Kuznetsov, “Traffic Control Software Package for Linux.” ftp://ftp.inr.ac.ru/ip-routing/.
14. W. Almesberger, J. H. Salim, and A. Kuznetsov, “Differentiated Services on Linux”, Internet Draft: draft-almesberger-wajhak-diffserv- linux-01.txt, June 1999.
15. W. Almesberger, “Linux Traffic Control - Implementation Overview.” ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz, Nov. 1998.
16. K. Wehrle, R. Bless: “Evaluation of Differentiated Services using an Implementation under Linux”, Proceedings of the International Workshop on Quality of Service (IWQOS'99), London, UK, May 31 - June 4, 1999.
17] T. Braun, A. Dasen, M. Scheidegger, K. Jonas, H. Stüttgen: Implementation of Differentiated Services over ATM, IEEE Conference on High-Performance Switching and Routing, Heidelberg, June 26-29, 2000.