DiffServ over MPLS: Tuning QOS parameters for Converged Traffic using Linux Traffic Control

Sundeep .B.Singh and Girish.P.Saraph

Indian Institute of Technology Bombay, Powai, Mumbai-400076, India

***Abstract*— MPLS over DiffServ couples the diffserv's per hop guarantees with MPLS traffic engineering capabilities to provide better quality of service (QoS) guarantees than that provided by MPLS or DiffServ alone. Linux provides powerful traffic control features to ensure proper traffic classification and differentiation. However, the current implementation of the traffic filter, used for MPLS over DiffServ architecture on Linux, exhibits certain drawbacks. This paper discusses the shortcomings of the current filter implementation and proposes a new filter that is capable of alleviating the problems.**

***Index Terms*— DiffServ over MPLS, Linux traffic control, RSVP-TE.**

I. Introduction

THE continuing exponential growth of the Internet has placed a tremendous strain on the service provider networks. Coupled with this increase, there has been introduction of newer applications like e-commerce, voice and multimedia services that require higher bandwidth and stricter quality guarantees. Augmenting the capacity of a network is a costly proposition, so the service providers are looking at architectures that gives them greater control on the traffic passing through their domains and hence ensure optimal performance with minimal increase in network resources.

MPLS is a label-switching technology with powerful traffic engineering features like explicit route specification, path pre-emption and fast reroute. Differentiated services, on the other hand, provide quality of service (QoS) at per node level by aggregating the traffic in classes or behaviour aggregates (BAs) that receive proper treatment within the DiffServ domain.

Although MPLS and DiffServ are independent technologies, they complement each other quite well. Looking collectively, MPLS is used to move packets from one place to another through a series of hops along a pre-selected path, while, DiffServ is used to ensure that the packet receives proper treatment at each of these hops. Hence, DiffServ over MPLS [1] can provide overall better guarantees by providing QoS, on both, per node and end-to-end path basis.

RSVP-TE daemon for the DiffServ over MPLS [2] has been implemented on Linux. The queuing and scheduling information is read from the ds_config file and installed on the system. Service differentiation depends on how well the traffic filters are implemented.. Linux provides a powerful set of traffic control (TC) utilities. However, the current traffic filter implementation does not provide proper differentiation for different classes of traffic as shown by the simulation results presented here. Hence there is a need to understand the short-comings and build a better filter for supporting DS-MPLS.

The rest of the paper is organized as follows: Section II gives an overview of the DiffServ over MPLS and section III presents the architecture of the RSVP-TE daemon for Linux. Section IV discusses the queuing schemes provided by Linux traffic control and the logical structure of the current filter implementation. Section V presents simulation studies that demonstrate the drawbacks of the filter. Section VI details the newly proposed traffic filter and the reasons for its better performance. Finally, the conclusions are presented.

II. Overview of DiffServ over MPLS

Traffic entering a differentiated services (DiffServ) domain is classified into behaviour aggregates (BAs), based on differentiated services code point (DSCP), which comprise of six bits of DS/TOS (Differentiated Services/Type of Service) field. Inside the DiffServ domain, a packet is forwarded according to its BA. This forwarding treatment applied to a BA at a DS node is defined as per-hop behaviour (PHB). There are two standard PHB groups: Expedited Forwarding (EF) PHB and Assured Forwarding (AF) PHB, with EF having the highest priority for low latency service. Within the AF, there are four classes, class AF1 has the highest priority while AF4 has the lowest. Within each class AFx, there are three drop precedence (DP) levels, the subclass DP1 has the lowest dropping probability while DP3 has the highest.

MPLS or Multi-protocol label switching simplifies packet-forwarding by forming label-switched paths or LSPs that act like virtual tunnels. Inside the MPLS domain, a packet is first classified into a forwarding equivalence class (FEC), based on it, is assigned a locally significant label at the ingress. At the subsequent hops, there is no further inspection of the network layer header and the packet is forwarded by label switching.

To support DiffServ over MPLS, the six-bit DSCP in Layer 3 DS field is mapped to the three-bit EXP field present inside the MPLS shim layer header or encapsulated into the Layer 2 header itself. This is needed because MPLS routers do not examine the Layer 3 header during forwarding. The standards define two types of label switch paths (LSPs):

EXP inferred LSP or E-LSP in which, the differential PHB is inferred from the EXP bits,

Label inferred LSP or L-LSP in which, the MPLS label itself signifies DS treatment and determines the PHB applied to all packets in the FEC.

III. RSVP-TE Daemon For DiffServ Over MPLS Under Linux

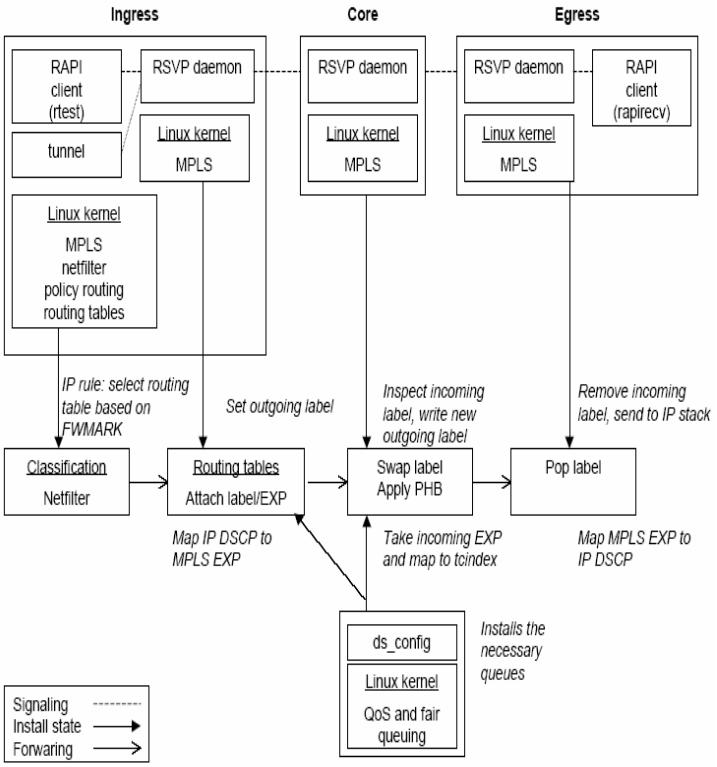

Fig. 1. DiffServ over MPLS using RSVP-TE under Linux [3]

The RSVP-DS-TE daemon has components in both user and kernel space. The components in kernel space are: MPLS kernel component, netfilter and the QoS and fair queuing component. The components in user space are RSVP daemon for the signaling and maintaining of the MPLS state and for distribution of labels. "Rtest", on the ingress, issues request for the LSP with resource allocation to the RSVP daemon. Once the signaling is complete and path is set-up, "rapirecv" on the egress, sends acknowledgement back to the ingress. "tunnel" is used to map specific traffic from ingress onto the LSP.

When a packet enters the ingress, it is marked and its FEC is identified using netfilter. Then, the packet is labeled with DSCP is mapped to the EXP bits and forwarded to the proper MPLS tunnel interface. The "QoS and fair queuing" module reads the ds_config file to determine the structure of the traffic filter to be installed. At the intermediate nodes, the incoming label is swapped with a new label while, the EXP value in the MPLS header is mapped to "tcindex" to determine its PHB treatment based on its behaviour aggregate.

IV. Traffic Control And Current Filter Implementation

A. Linux Traffic Control (TC)[4]:

Linux provides powerful traffic control features using TC. Traffic control consists of shaping, scheduling and policing. Queuing disciplines (qdiscs) are the building blocks of traffic control system. A lot of qdiscs are available on Linux, but the ones that have been used in implementing the traffic filter are:

1. Classless qdiscs: These qdiscs do not have classes.

Packet FIFO (PFIFO): This is a simple FIFO qdisc where the buffer size is specified in packets.

Token Bucket Filter (TBF): This has functionality of the token bucket /leaky bucket queuing model.

Random Early Detect (RED): This has RED scheme[5] based probabilistic packet drop mechanism.

2. Classful qdiscs: These qdiscs contain classes which may contain other qdiscs. The different qdiscs are as follows:

Priority Queuing Discipline (PRIO): In PRIO, packets are first classified using the filter and placed into different priority queues, which by default are three. Packets are scheduled from the head of a given queue only if all queues of higher priority are empty. Within each of the priority queues, packets are scheduled in FIFO order.

Generalized RED (GRED): [6] It was specifically designed keeping in mind the need for implementing multiple drop precedence in Assured Forwarding Per Hop Behaviour (AF PHB). GRED can have a maximum of 16 RED virtual queues (VQs) each of which can be configured independently. GRED takes advantage of the probabilistic dropping mechanism in RED, to implement AF classes. The drop priority among the VQs is chosen using the four least significant bits (0…15 for 16 VQs) in the tc_index field (discussed in DSMARK qdisc).

DSMARK qdisc: [6] This qdisc is used for marking the Differentiated Services Code Point (DSCP) field of the packets and/or with tcindex classifier set, it may also be used for forwarding the packet to the appropriate AF queue in the GRED qdisc. The operation is as follows: with the "set_tc_index" option set, the TOS field is copied to a 16 bit filed "tc_index". Next, the tcindex classifier reads the tc_index field and performs the following operation:

(tc_index & mask)>>shift;

mask and shift are the parameters that can be passed with the tcindex classifier using TC. The value returned after this operation (16 bit value) is then re-written by the qdisc back to tc_index. This tc_index value may now be used for remarking the packet. In fig.2., the four rightmost bits of the tc_index field is used by the GRED qdisc to select a VQ for the packet (as discussed above in GRED).

Hierarchical Token Bucket (HTB): [7] HTB is based on TBF and Deficit Round Robin (DRR), which is a modified version of Weighted Round Robin (WRR). The term "hierarchical" implies that the filter allows hierarchical link sharing (such as a tree) on an interface. HTB allows sharing of bandwidth between its subclasses, each of which can be configured independently.

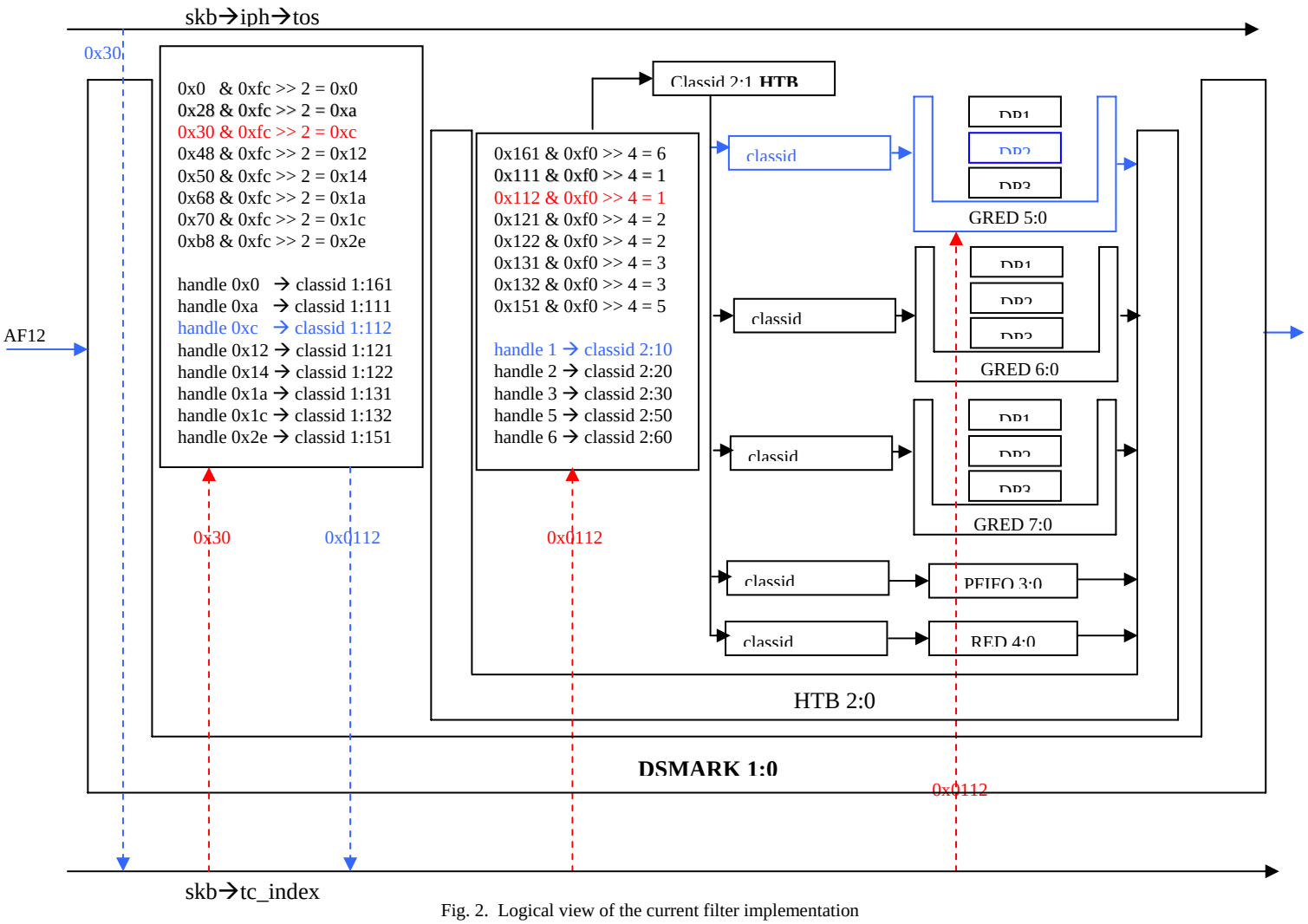

Fig. 2. Logical view of the current filter implementation

B. *Filter Structure*

The logical representation of the current TC filter is shown in Fig. 2. Its operation is explained here by using AF12 type flow as an example. When this TC filter is used at the ingress node, the Type Of Service (TOS) byte of the incoming packet (say, the packet is marked AF12) is copied into to tc_index. Then the DSMARK filter operates on the tcindex and the following operation is done:

(tc_index & mask) >> shift

For this DSMARK filter, mask=0xFC and shift=2, so for an AF12 (TOS=0x30) packet:

(0x30 & 0xFC) >> 2 = 0xC

which is the corresponding value of the DSCP field. Now corresponding to this value 0xC, the tc_index classifier's element returns the minor-value of the class identifier, in this case 0x112, to the DSMARK qdisc.

After passing through the DSMARK qdisc, the packet again encounters the tc_index classifier/filter at the HTB qdisc, with mask=0xF0 and shift=4. The tc_index value is used for this operation and the following results for the AF12 packet:

(0x0112 & 0xF0) >> 4 = 1

which signifies that the packet belongs to AF1 and is passed to class-id 2:10 which is for AF1 packets. Note that this time, the result of the bitwise operation is not returned back to the tc_index (that can only be done inside a DSMARK qdisc). After this, the packet goes through the corresponding GRED qdisc which learns about the packet's drop precedence by checking out the last four bits of the tc_index; since for AF12 it is 0x0112 and the AF12 packet gets a drop precedence of 2.

This explains the logical passage of the packet through the traffic filter.

V. Drawbacks With The Filter

In this section, we demonstrate the short-comings of the above TC filter implementation based on our experimental studies. The results show priority reversal could take place leading to unacceptable performance. The experiments are carried out using a simple network topology with fixed source and destinations, as shown in Fig.3.

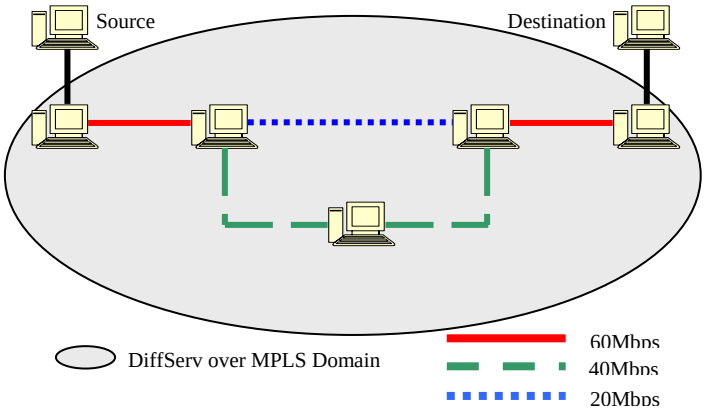

Fig. 3. Network Topology

The experimental observations with the present filter are presented for three cases:

Case I: Three UDP flows with same packet-size and rate.
Flow1 was given EF service,
Flow2 was given AF11 service,
Flow3 was given BE service.

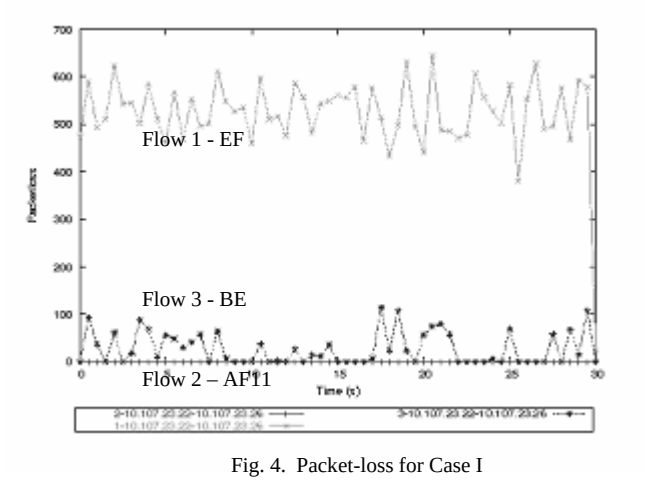

Fig. 4. Packet-loss for Case I

Case II: Two UDP flows with same packet size and rate.
Flow1 was given AF11 service,
Flow2 was given AF12 service.

Case III: Three UDP flows with same packet-size and rate.
Flow1 was given AF11 service,
Flow2 was given AF21 service,
Flow3 was given AF31 service.

From the above it can be seen that only in case II, where BAs share an ordering constraint (PHB Scheduling Class), does the filter show expected behaviour. While in cases I and II, the filter shows anomalies by giving higher preference in terms of lower packet-loss/delay to lower priority BAs. It was further noticed that:

1. PFIFO qdisc implemented for EF PHB, dropped considerable amount of packets and hence the QoS deteriorated even below than that of BE traffic.

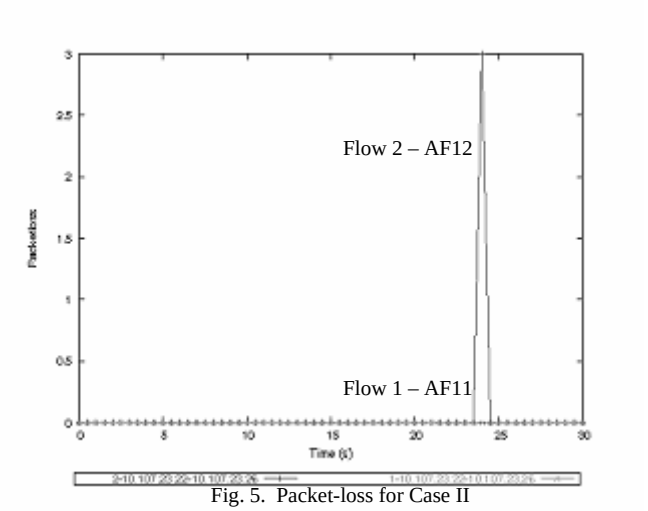

Fig. 5. Packet-loss for Case II

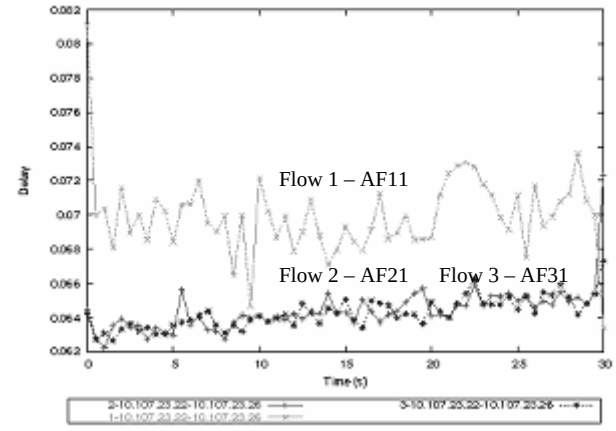

Fig. 6. Delay for Case III

2. On increasing the buffer size for PFIFO, the drops decreased but the delay and jitter increased to high values.

As evidenced from above, the current traffic control filter may not provide proper differentiation and may lead to priority reversals. Thus, the whole idea of the DiffServ over MPLS could be defeated.

VI. Proposed Filter

To over these shortcomings, we have implemented a new filter as shown in Fig. 7. The filter structure shows that, the EF, AF and BE classes are given priority in that order, using PRIO qdisc. In order that the lower classes are not starved, the EF queue uses a TBF qdisc.

The AF traffic has been bundled into the HTB filter so that the excess bandwidth among the AF classes can be shared and bandwidth resources are fully utilized. TBF filter is not needed here since the HTB filter itself has bandwidth restricting mechanisms.

Finally, the BE traffic is given the lowest priority, while still being able to access the excess bandwidth. As in the previous filter, it is still implemented using RED.

The ongoing experiments conducted on this TC filter have shown that it overcomes all the priority reversal problems seen earlier.

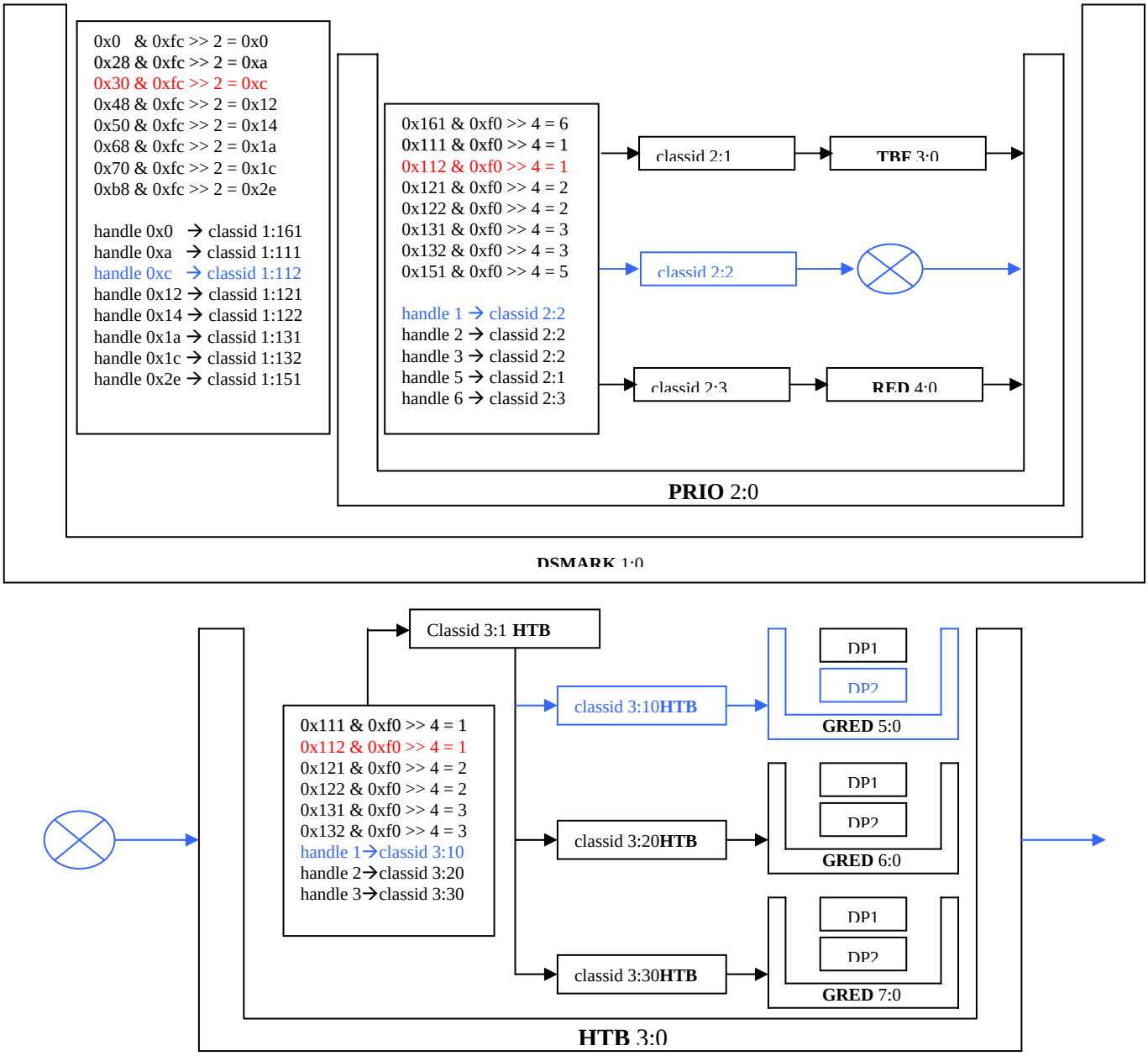

Fig. 7. Logical view of the new filter implementation

VII. Conclusions

This paper has analyzed the logical structure of the traffic filters for implementing DiffServ over MPLS on the Linux platform. The traffic filter is a very important component of this architecture and is essential for providing the expected QoS to different flows.

The existing TC filter implementation for Linux is found to have certain drawbacks, which could lead the priority reversals under certain conditions. A proper differentiation of traffic is important for DiffServ over MPLS services. The new filter has been implemented to mitigate the anomalies of the previous filter. The ongoing experiments on it have shown significantly better results.

References

[1] L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services", *RFC 3270*, May 2002.

[2] For MPLS patch and RSVP-TE daemon, http://dsmpls.atlantis.ugent.be/

[3] Pim Van Heuven, "RSVP-TE daemon for DiffServ over MPLS under Linux", in *9th International Linux System Technology Conference*, September 4-6, 2002 in Cologne, Germany.

[4] Linux 2.4 Advanced Routing HOWTO, referred for Linux qdiscs and netfilter, http://lartc.org/howto/

[5] Floyd, S., and Jacobson, V., "Random Early Detection gateways for Congestion Avoidance", in *IEEE/ACM transactions on Networking*, vol. 1, no. 4, pp 397-413, 1993.

[6] QoS on Linux. http://www.opalsoft.net/qos/

[7] For HTB documentation and HTB patch for Linux kernel 2.4.18 http://luxik.cdi.cz/~devik/qos/htb/