End-to-end Proportional Loss Differentiation

Alok Kumar* **Jasleen Kaur** **Harrick M. Vin**

Technical Report TR-01-33
Department of Computer Sciences
University of Texas at Austin
TAY 2.124, Austin, TX 78712-1188, USA

{alok,jks,vin}@cs.utexas.edu

Sept 26, 2001

Abstract

Service differentiation is at the core of designing next-generation Internet. In this paper, we present a buffer management framework for achieving end-to-end proportional loss differentiation in networks. There are two main facets of our buffer management framework. First, it decouples the decisions of *when* to drop a packet from *which* packet to drop. This allows the framework to utilize existing single-class buffer management techniques—such as RED—to determine *when* to drop a packet; in fact, when instantiated with RED, the framework extends the primary advantages of single-class RED—namely, early notification of congestion and maintenance of average buffer occupancy at low, configurable levels—to a multi-class workload. Second, at each router, the framework governs the selection of *which* packet to drop based on the number of packets of a flow transmitted by its source, rather than the number of packets that arrive at a router. The framework achieves this by encoding information about the losses observed by a flow at a router in packet headers. This allows the framework to provide end-to-end proportional loss differentiation, unlike most existing schemes that provide loss differentiation only on a per-hop basis. We evaluate the efficacy of this approach under various network settings. We describe an implementation of our framework and discuss its complexity.

1 Introduction

The Internet has traditionally supported the *best-effort* service model in which the network offers no assurance about when, or even if, packets will be delivered. With the commercialization of the Internet and the deployment of inelastic continuous media applications, however, the best-effort service model is increasingly becoming inadequate. Hence, development and deployment of network mechanisms that provide different levels of service to different application classes is at the core of designing next-generation Internet.

Over the past few years, several network architectures for providing service differentiation have been proposed [14, 12]. The Differentiated Services (DiffServ) framework is one such architecture [12]. This architecture achieves scalability by implementing complex classification and conditioning functions only at network boundary routers (which process lower volumes of traffic and lesser numbers of flows), and providing service differentiation inside the network for flow aggregates rather than on a per-flow basis [12]. To provide service differentiation to traffic aggregates, the DiffServ architecture defines a small set of behavior (or flow) aggregates (also referred to as Per Hop Behaviors—PHB). Recently, several PHBs—such as the Expedited Forwarding (EF) and the Assured Forwarding

*Currently at Intel Corporation.

(AF) PHB—and several end-to-end services—such as the Virtual Leased Line service [10, 12], Assured service [5], and the Olympic service [8]—have been defined.

Consider, now, the Assured service [5] and the Olympic service [8] definitions. These services are based on the Assured Forwarding (AF) PHB [8]; further, they require the network to provide differentiation in the *loss rates* experienced by packets of flows subscribing to different levels of service. For instance, the Assured service definition partitions the packets of flows requesting the Assured service into two classes—IN and OUT, and require the network to ensure that packets belonging to the IN class always experience lower loss rate than packets belonging to the OUT class. The Olympic service, on the other hand, defines three classes—gold, silver, and bronze. Within each class, three levels of drop precedence—low, medium, and high—may be defined such that packets marked with the low drop precedence experience lower loss rates than those marked with medium, which in turn experience lower loss rates than those marked with high. Observe that both of these service definitions require the network to provide only a *qualitative* differentiation in the loss rates experienced by different classes of packets. Recently, it has been argued that service definitions that export richer semantics—such as quantitative loss differentiation—may be more desirable for network subscribers. Unfortunately, most existing buffer management mechanisms either do not provide any quantitative loss differentiation or do so only on a per-hop basis. In this paper, we propose a buffer management framework for providing end-to-end proportional loss differentiation in networks.

There are two main facets of our buffer management framework. First, it decouples the decisions of *when* to drop a packet from *which* packet to drop. This allows the framework to utilize existing single-class buffer management techniques—such as Random Early Detection (RED)—to determine *when* to drop a packet; in fact, when instantiated with RED, the framework extends the primary advantages of single-class RED—namely, early notification of congestion and maintenance of average buffer occupancy at low, configurable levels—to a multi-class workload. Second, at each router, the framework governs the selection of *which* packet to drop based on the number of packets of a flow transmitted by its source, rather than the number of packets that arrive at a router. The framework achieves this by encoding information about the losses observed by a flow at a router in packet headers (see for example, Dynamic Packet State [16]). This allows the framework to provide end-to-end proportional loss differentiation, unlike most existing schemes that provide loss differentiation only on a per-hop basis. We evaluate the efficacy of this approach under various network settings. We describe an implementation of our framework in routers based on the Intel's IXP1200 network processors. Our preliminary evaluation indicates that EPLD can be implemented in high-speed routers without any degradation in router performance.

The rest of the paper is organized as follows. In Section 2 we formulate the problem of end-to-end proportional loss differentiation and derive our design principles. EPLD is described in Section 3 and an approximation is proposed in Section 4. A comprehensive experimental evaluation is presented in Section 5. In Section 6 we present our implementation design of EPLD in the IXP1200 routers and discuss the storage and computation overheads. We discuss related work in Section 7. Section 8 summarizes our contributions.

2 Problem Formulation

Consider a network that supports N traffic classes, $J_1, J_2, \ldots, J_N$. Let $\alpha_i \neq 0$ be the *loss differentiation parameter* associated with class J_i. Let $\alpha(f)$ denote a function such that $\alpha(f) = \alpha_i$ *iff* f *belongs to class* J_i. Consider two flows, f_j and f_k that share a network path P of one or more hops. Let T be a time-interval of observation during which both the flows are *active* (i.e., both flows are transmitting packets). Let $l_j(P,T)$ and $l_k(P,T)$, respectively, denote the percentage of packets lost on path P during time-interval T, by flows f_j and f_k. Then, we say that the network provides *end-to-end proportional loss differentiation*, if for all j, k, P, and T:

$$\frac{l_j(P,T)}{l_k(P,T)} = \frac{\alpha(f_j)}{\alpha(f_k)} \tag{1}$$

To motivate the design of a buffer management mechanism that can achieve such end-to-end proportional loss differentiation, consider, first, the design of Random Early Detection (RED)—the most popular buffer management

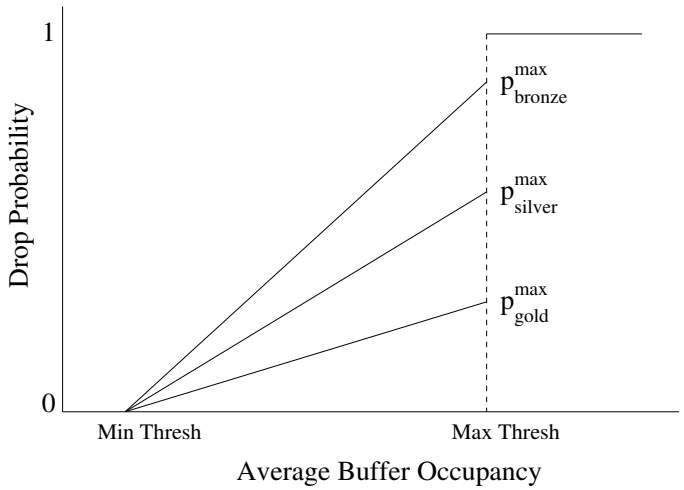


Figure 1: Proportional Loss Differentiation in WRED

scheme for today's networks (that support a single class of service). Routers that implement RED manage the available buffer space in the router as follows. The router maintains a running estimate of the average buffer occupancy. Each incoming packet is dropped with a probability that increases linearly from 0 to p_{max} as the buffer occupancy increases from a minimum threshold (MinThresh) to a maximum threshold (MaxThresh). If the average buffer occupancy exceeds MaxThresh, then all incoming packets are dropped unconditionally. If, on the other hand, the average buffer occupancy is below MinThresh, none of the incoming packets are dropped. The relationship between the drop probability that a packet may observe and the average buffer occupancy is described in a *RED curve*.

There are two primary benefits of RED (over a simple tail-drop buffer management policy). First, by selecting appropriately the parameter values, RED facilitates early congestion notification to the applications. Second, RED enables a router to maintain its average buffer occupancy at low, configurable levels.

Recently, extensions of RED—referred to as multi-class RED or Weighted RED (WRED) [13]—have been proposed to extend the benefits of RED to network environments that support multiple classes of service. These extensions use different RED curves for dropping packets belonging to different service classes [13]. Figure 1 depicts one such setting with three service classes—gold, silver, and bronze. In this configuration, when the average buffer occupancy is in the range [MinThresh, MaxThresh], the probabilities for dropping packets of the gold, silver, and the bronze class are in the ratio $p_{gold}^{max} : p_{silver}^{max} : p_{bronze}^{max}$. Hence, by selecting the values of p_{gold}^{max}, p_{silver}^{max} and p_{bronze}^{max} based on the desired loss ratios, the above configuration can achieve proportional loss differentiation in each router.

Unfortunately, such extensions of RED have the following two limitations.

1. *Inconsistent congestion notification*: Ideally, loss differentiation should be provided by determining the class from which a packet should be dropped, in the event that a router needs to drop a packet. In the above scheme, however, loss differentiation is achieved by determining *when* an incoming packet is dropped; the selection of the packet to be dropped is implicit. This approach has two undesirable effects.

   - Consider two routers R_1 and R_2 with the WRED buffer management scheme (as depicted in Figure 1). Let R_1 and R_2 receive packets at the same rate. Now, if all of the traffic entering R_1 and R_2, respectively, belongs to the bronze and the gold class, then router R_1 will drop a much larger number of packets as compared to router R_2 (even though both routers are receiving packets at the same rate, and hence are experiencing the same level of congestion). Since packet losses indicate congestion to applications, the WRED configuration described in Figure 1 delivers inconsistent congestion notifications to the applications.

- With the WRED configuration described in Figure 1, the average buffer occupancy in routers is not just a function of the RED parameters, but also the traffic composition. For instance, in the above example, since router R_2 drops a much smaller number of gold packets, it experiences a much higher level of average buffer occupancy (as compared to router R_1, which, by virtue of dropping a larger number of bronze packets, maintains the average buffer occupancy at lower levels). Hence, WRED does not preserve a key property of RED—by selecting appropriately the RED parameters, average buffer occupancy can be *configured* to remain below certain levels [7].

These observations lead us to our first design principle.

Principle 1 *Buffer management schemes should decouple the decisions of when to drop a packet from which packet to drop* [1].

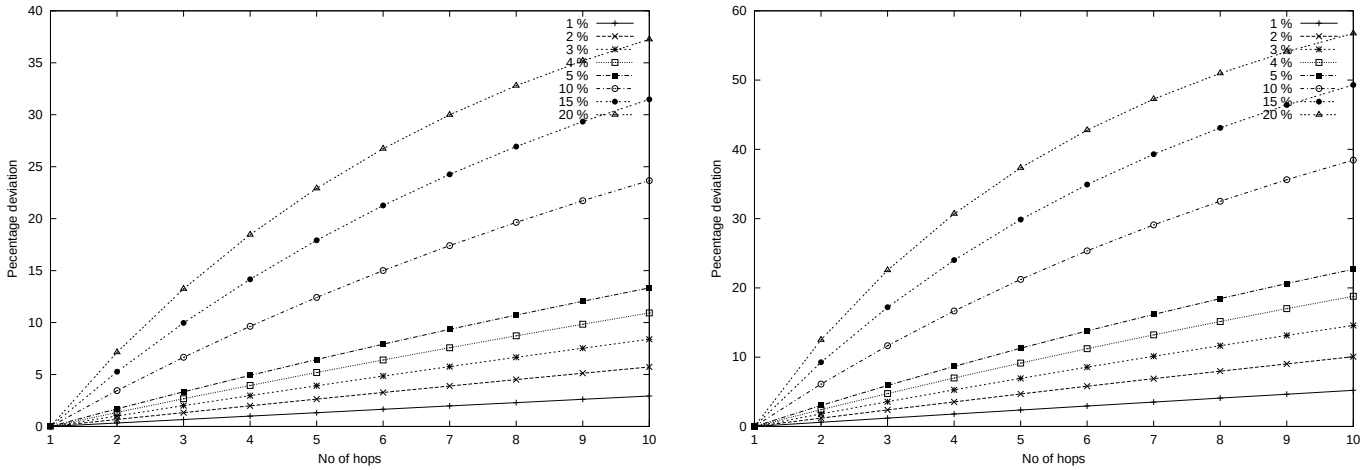
2. *Lack of proportional differentiation in multi-hop networks*: A network of WRED routers does not provide the desired loss differentiation in a multi-hop network. To demonstrate this, consider two flows f_j and f_k that share a network path P consisting of two routers. During an observation interval of length T, let flows f_j and f_k be active simultaneously. Further, let the loss rates experienced by packets of the flows at each router be proportional to the loss differentiation parameters of the classes the flows belong to. Thus, for some k_i (which depends on the level of congestion at router i), packets of flow f_j experience a loss rate—with respect to the incoming traffic at that router—of $\alpha(f_j)k_i$ at the i^{th} router on the path. Then, the ratio of packet losses incurred by the two flows in a two-hop network is given by:

$$\begin{aligned}
\frac{l_j(P,T)}{l_k(P,T)} &= \frac{\alpha(f_j) \times k_1 + (1 - \alpha(f_j) \times k_1) \times \alpha(f_j) \times k_2}{\alpha(f_k) \times k_1 + (1 - \alpha(f_k) \times k_1) \times \alpha(f_k) \times k_2} \\
&= \frac{\alpha(f_j)\ (k_1 + k_2) - k_1 k_2\ \alpha(f_j)^2}{\alpha(f_k)\ (k_1 + k_2) - k_1 k_2\ \alpha(f_k)^2} \qquad (2) \\
&\neq \frac{\alpha(f_j)}{\alpha(f_k)} \qquad (3)
\end{aligned}$$

This illustrates that proportional loss rates at individual routers fails to translate to the same ratio for end-to-end loss rates. Observe that the end-to-end loss ratio is a function of the *sum* and *product* of the loss rates observed at individual routers (Equation (2)). While the *sum* of loss rates over all routers on a shared path is in the desired ratio, the *product* is not.

Figure 2 depicts the expected deviation from the per-hop loss ratios with increase in the number of hops in a shared path. Each line in Figure 2 represents a level of congestion in the routers (which in turn translates to a certain percentage of packets dropped at the routers); further, the graph considers a setting with multiple routers along the path experience the same level of congestion. Figure 2 shows that if the level of congestion, indicated by the packet loss rate, is high, then the end-to-end loss differentiation deviates significantly from the desired ratio even when flows share path with a small number of hops. On the other hand, for networks with small packet loss rates, the deviation is significant only if flows share a path with a large number of congested routers. Thus, the deviation from the desired ratio of proportional loss differentiation becomes significant when (1) flows encounter multiple congested links on their path, and (2) the loss rates experienced at the congested links are high.

[1] The need for decoupling the decision of *when* to drop a packet and *which* packet to drop was also identified in [6]. In [6], the authors motivate the need for decoupling by arguing that with WRED, the ratio of losses for observed by two different classes depends on the traffic composition. However, the WRED configuration shown in Figure 1 does not have this limitation. Even for this configuration, we have argued that decoupling is necessary for a completely different set of reasons.



(a) $\alpha(f_1)/\alpha(f_2) = 2$ (b) $\alpha(f_1)/\alpha(f_2) = 4$

Figure 2: Computed Deviation from Desired Loss Rate Ratio on Multiple Links

A recent study reported in [4] measures loss rates of around 10% on sample paths through the Internet. Thus the Internet today satisfies condition (2), but it is not known whether these losses occur at a single link or accumulate over multiple links that are simultaneously congested[2]. In the absence of any conclusive evidence, it can be argued that since today's networks carry predominantly TCP flows (which drastically reduce transmission rate upon the first indication of congestion), in the steady state, the flows experience significant packet losses at only one link (the most severe bottleneck along the path). Unfortunately, with the predicted increase in streaming media traffic [3]—most of which use UDP or its variants for network transmission and perform rate control at the application level—a greater fraction of future Internet traffic would either not respond to the first indications of congestion or would respond at much slower time-scales. Thus, in future networks, flows may experience significant packet losses at multiple congested links along their path. In such scenarios, per-hop differentiation would not extend to end-to-end loss differentiation.

A natural way to address this limitation is to eliminate the *product* terms from the *end-to-end* loss rate computation. Observe that the product term in Equation (2) arises because the loss rate at every router is defined (and provided) as a percentage of packets lost with respect to the number of packets that *reach that router*. If instead, each router defines (and provides) the loss rate as the percentage of packets lost with respect to the total number of packets *transmitted by the source*, then the *end-to-end* loss rate would simply be an *additive* function of the individual loss rates. This observation leads us to our second design principle.

Principle 2 *At each router, the selection of which packet to drop should be based on the number of packets of a flow transmitted by its source, rather than the number of packets that arrive at a router.*

In what follows, we present the design of End-to-end Proportional Loss Differentiation(EPLD) which is based on these two principles.

[2]In [4], based on the wide variation in the round-trip times observed for packets, authors hypothesize that: *"It seems likely that rather than a single standing queue (where packets are delayed waiting to be processed by some router on the path), this path is subject to one (or more) changing queues"*.

3 End-to-end Proportional Loss Differentiation

In this section, we describe our notation, a conceptual design of our buffer management framework, and then a scalable instantiation of the framework in a DiffServ network.

3.1 Notation

Consider a snapshot of the queue at any router x. Let F^x be the number of flows, $f_1, \ldots, f_{F^x}$, that are actively transmitting data through this router. We refer to the k^{th} packet, in the increasing order of packet arrival times, in the router queue as p^k. We refer to the k^{th} packet *of flow* f_i in the router queue as p^{i_k}. Let n_i be the number of flow f_i packets in the queue, and $n = \sum_{i=0}^{F^x} n_i$ be the total number of packets in the router queue. Then, $\{p^0, p^1, \ldots, p^n\}$ represents the router queue, and $\{p^{i_0}, p^{i_1}, \ldots, p^{i_{n_j}}\}$ represents the *logical* queue of packets belonging to flow f_j. Let $\alpha(f_i)$ represent the loss differentiation parameter of the class that flow f_i belongs to.

3.2 Buffer Management Framework: Concepts

Our buffer management framework decouples the decision of *when* to drop a packet from *which* packet to drop (and thereby adheres to our Principle 1). Further, it does not specify an algorithm for determining *when* to drop a packet; the EPLD algorithm (described below) for selecting *which* packet to drop can be used in conjunction with a variety of policies—such as tail-drop and RED—for determining *when* to drop a packet. In fact, when instantiated with RED, the framework extends the primary advantages of single-class RED—namely, early notification of congestion and maintenance of average buffer occupancy at low, configurable levels—to a multi-class workload.

Once a decision to drop a packet is made, our EPLD algorithm selects a packet to be dropped. To provide end-to-end proportional loss differentiation, at each router, EPLD governs the selection of *which* packet to drop based on the number of packets of a flow transmitted by its source, rather than the number of packets that arrive at a router. A router can obtain information about the total number of packets transmitted by the source, only if it has information about the number of packets dropped at previous routers on the path.

EPLD achieves this by encoding information about the packet losses observed by a flow at a router in packet headers (see for instance, Dynamic Packet State [16]). Specifically, EPLD associates, with each packet of flow f_i, a quantity *Virtual Count* VC_i that accounts for the number of packets that belong to flow f_i that have been dropped in front of the packet at any router queue. VC_i is initialized to the *packet size* when the packet is generated at the source. For simplicity, if we assume that a flow transmits all packets of the same size[3], then VC_i is initialized to 1. This quantity is encoded in the header of a packet and is carried across routers along its path. Whenever a packet is dropped, its VC_i value is added to the VC_i value of the next packet in the queue that belongs to the same flow.

Now, for each flow f_i at router x, define *Virtual Length* $VL^x(f_i) = \sum_{k=0}^{n_j} VC_i(p^{i_k})$. Then $VL^x(f_i)$ represents the actual queue length of flow f_i in the queue of router x *if* none of its packets had been dropped at previous routers in the network. Thus, $VL^x(f_i)$ represents the number of packets transmitted by the source and not just those that reach the router.

EPLD uses the VC_i values to select a packet to be dropped as follows: Select with probability proportional to $VC_i(p^{i_k}) \times \alpha(f_i)$ a packet to drop. Such a selection ensures that flow f_i is selected with probability proportional to $VL^x(f_i) \times \alpha(f_i)$.

We now show that EPLD , by using the above mechanism, ensures that the loss rates of flows are in the ratio of their loss differentiation parameters. Consider a time interval $[t_0 \ldots t_0 + \Delta t]$ during which the sending rates of f_i and f_j are r_{f_i} and r_{f_j}, respectively. The number of packets generated by these sources during this time interval are in proportion to their sending rates. This means that when the packets generated in this interval are in router x, for

[3]The description of EPLD can be easily generalized to the case when flows transmit packets with different sizes; simply substitute the number of packets by the sum of their sizes.

some k^x,

$$VL^x(f_i) = k^x \; r_{f_i} \tag{4}$$

When the packets generated in $[t_0 \ldots t_0 + \Delta t]$ are in router x, suppose the router drops a total of d_x packets. Out of these, the *expected* number of drops in flow f_i is:

$$\frac{VL^x(f_i) \times \alpha(f_i)}{\sum_{m=0}^{F^x} (VL^x(f_m) \times \alpha(f_m))} \times d_x$$

Suppose flows f_i and f_j share the path P through the network. Then, the *expected* end-to-end loss rate experienced by flow f_i on path P would be:

$$\begin{aligned} l(f_i) &= \frac{\sum_{x \in P} \left(\frac{VL^x(f_i) \times \alpha(f_i)}{\sum_{m=0}^{F^x} (VL^x(f_m) \times \alpha(f_m))} \times d_x \right)}{\Delta t \times r_{f_i}} \\ &= \sum_{x \in P} \left(\frac{d_x}{\Delta t \; \sum_{m=0}^{F^x} (VL^x(f_m) \times \alpha(f_m))} \times \frac{VL^x(f_i)}{r_{f_i}} \times \alpha(f_i) \right) \end{aligned}$$

Using Equation (4),

$$l(f_i) = \sum_{x \in P} \left(\frac{d_x \times k^x}{\Delta t \; \sum_{m=0}^{F^x} (VL^x(f_m) \times \alpha(f_m))} \right) \times \alpha(f_i) \tag{5}$$

Similarly, the *expected* loss rate experienced by flow f_j is:

$$l(f_j) = \sum_{x \in P} \left(\frac{d_x \times k^x}{\Delta t \; \sum_{m=0}^{F^x} (VL^x(f_m) \times \alpha(f_m))} \right) \times \alpha(f_j) \tag{6}$$

Dividing (5) by (6), we get,

$$\frac{lp(f_i)}{lp(f_j)} = \frac{\alpha(f_i)}{\alpha(f_j)}$$

This illustrates that the expected value of the end-to-end loss rates of flows f_i and f_j are in the desired ratio.

Observe that, in the above description, when a packet of flow f_i is dropped, its VC_i value is added to that of the next packet belonging to the *same flow*. This requires routers to perform packet classification on a per-flow basis, as well as maintain packet classification state. Maintenance of such per-flow packet classification state can limit the scalability of routers. In what follows, we describe a technique for scalable instantiation of the EPLD concepts in a DiffServ network.

3.3 Scalable Instantiation of EPLD in DiffServ Network

The DiffServ architecture achieves scalability by implementing complex per-flow classification and conditioning functions only at network boundary routers (which process lower volumes of traffic and lesser numbers of flows), and providing service differentiation inside the network (i.e., at core routers) for *flow aggregates* (also referred to as service classes) rather than on a per-flow basis [12]. The conceptual description of EPLD can be adapted for the DiffServ architecture as follows:

1. The edge routers perform per-flow functionality, and initialize each packet with an appropriate VC_i .

2. Core routers differentiate among packets belonging to different service classes; they do not distinguish between individual flows. Hence, when a packet of class J_i is dropped, its *VC*$_i$ value is added to the next packet belonging to the *same class*.

Further, at core router, EPLD defines the *virtual length of a class*, *VQL*$_i^x$, as *VQL*$_i^x$= $\sum\{VC_i(p^k) \mid p^k \; belongs \; to \; class \; i; \; 0 \leq k \leq n\}$; and then selects with probability proportional to $VQL_i^x \times \alpha_i$ the class from which a packet is dropped.

Observe that, with the above adaptation, the *VQL*$_i^x$ is decremented only when a packet of class J_i gets dequeued (packet drops simply result in addition of their *VC*$_i$ values to the next packet of class J_i). Hence, in the event that most packets of a class J_i are dropped, and hence rarely dequeued the corresponding *VQL*$_i^x$ value could see an unbounded growth. Additionally, increase in *VQL*$_i^x$ in turn increases the likelihood of selecting a packet from class J_i for future drops, worsening the situation further.

To eliminate such unbounded growth of *VQL*$_i^x$, EPLD includes a mechanism to *flush-out* *VC*$_i(p)$ whenever a dropped packet p of class J_i would have been dequeued had it not been dropped. Specifically, with each packet p of class J_i in the router queue, EPLD associates an N-dimensional vector of *Virtual Count*s: $\{VC_j(p) : 1 \leq j \leq N\}$ [4], where N is the number of service classes supported in the router. When a packet belonging to class J_i arrives at a router, the *VC*$_i$ value associated with that packet in the router is initialized to the *VC*$_i$ (p) value carried in the packet header from the previous router. All the other components of the N-dimensional vector are initialized to 0.

$$VC_j(p) = 0, \qquad 1 \leq j \leq N, j \neq i \tag{7}$$

When a packet p^k belonging to class J_i is dropped from the router queue, the N-dimensional vector of *Virtual Count*s maintained for the next packet in the router queue is updated as follows:

$$VC_j(p^{k+1}) \; = \; VC_j(p^{k+1}) + VC_j(p^k), \qquad 1 \leq j \leq N$$

EPLD then redefines $VQL_i^x = \sum_{k=0}^{n} VC_i(p^k)$. With this, whenever a packet is to be dropped, EPLD first selects a candidate class J_i with a probability proportional to $VQL_i^x \times \alpha_i$, and then selects a candidate packet p^k from class J_i with a probability proportional to $VC_i(p^k)$.

To complete the description of the implementation of EPLD in a DiffServ architecture, we provide the following details.

1. When a packet p of class J_i is dequeued and hence transmitted, it does not carry the $N-1$ quantities, $VC_j(p)$ $j \neq i$, with it. Recall that $\forall j \neq i; VC_j(p)$ represents a temporary store for a component of VC_j for the first packet of class J_j behind it. To carry forward these values, EPLD defines a set of N variables—called *Accumulate*. When packet p of class J_i gets dequeued, EPLD updates these variables as well as the $VC_i(p)$ value carried in the packet header as follows:

$$\begin{aligned} VC_i(p) &= VC_i(p) + Accumulate_i \\ Accumulate_i &= 0 \\ Accumulate_j &= Accumulate_j + VC_j(p), \qquad 1 \leq j \leq N, j \neq i \end{aligned}$$

2. If the packet selected to be dropped happens to be the last packet in the router queue, there its *VC*$_i$ values cannot be carried forward to the next packet. In general, the probability of dropping the last packet is likely to be small, and hence this boundary case may not be very significant. However, in the approximate, more efficient, version of EPLD that we discuss in Section 4, the last packet may be dropped with a reasonable probability. To take care of this situation, EPLD maintains another set of N per-class variables, $Backlog_j$, which serve as

[4]Note that out of these, $N-1$ counts are used *only* within the router queue, and are NOT carried to the next router. Therefore, they are not encoded in the packet header. The only quantity carried to the next router is *VC*$_i$. Hence, this mechanism in EPLD does not result in a significant increase in packet length/network load.

a temporary store of the $VC_j(p)$ values if the last packet p in the router queue is dropped. Whenever a new packet p arrives, the values assigned in Equation (7) are incremented as: $VC_j(p) = VC_j(p) + Backlog_j,\ 1 \leq j \leq N$

A pseudo-code for the implementation of EPLD is presented in Figure 3.

3.4 Discussion

Observe that the aggregation of per-flow state into per-class state inherent in the DiffServ architecture introduces the following error: the value of VQL_i^x at a router may *not* represent exactly the number of packets of all flows belonging to class J_i that would have reached the router had there been no losses of class J_i packets in any of the previous routers. This is because, with the aggregation of flows into classes, $VC_i(p^k)$ may in fact count the drops of packets that belong to flow f_i (or a group of flows) that may share only partially the path with the remaining flows of that class.

To estimate the significance of this error, consider a network path P that includes router x. Consider all packets of class J_i at router x that traverse P, and define $VL_i^x(P) = \sum\{VC_i(p^k)|\ p^k\ traverses\ P;\ 0 \leq k \leq n\}$ and $VL_i^x(\hat{P}) = \sum\{VC_i(p^k)|\ p^k\ does\ not\ traverse\ P;\ 0 \leq k \leq n\}$. Let $N_i^x(\hat{P})$ and $N_i^x(P)$, respectively, denote the total number of class J_i packets in the router queue destined for paths $\hat{P}$ and P. Then, the probability that the drop of a packet destined on path P gets recorded on a packet destined for path $\hat{P}$ is given by $\frac{N_i^x(\hat{P})}{N_i^x(P)+N_i^x(\hat{P})}\frac{VL_i^x(P)}{VQL_i^x}$, where $\frac{VL_i^x(P)}{VQL_i^x}$ denotes the probability that a packet destined for path P is dropped, and $\frac{N_i^x(\hat{P})}{N_i^x(P)+N_i^x(\hat{P})}$ denotes the probability that the next packet in the queue belongs to a flow traversing path $\hat{P}$. Similarly, the probability that the drop of a packet destined to path $\hat{P}$ is recorded on a packet destined to path P is given by $\frac{N_i^x(P)}{N_i^x(P)+N_i^x(\hat{P})}\frac{VL_i^x(\hat{P})}{VQL_i^x}$. If these two probabilities are equal at all routers x, then the *expected* value of VQL_i^x at any router is as desired. These two probabilities are equal if $\frac{N_i^x(P)}{VL_i^x(P)} = \frac{N_i^x(\hat{P})}{VL_i^x(\hat{P})}$; that is, if flows on path P interact with cross-traffic belonging to the same class that has seen similar loss rates.

In Section 5, we conduct experiments in which traffic on path P interacts with cross-traffic which has already experienced a wide range of loss rates in previous routers. We observe from our simulation results that any such errors do not significantly affect the end-to-end loss differentiation. We also show that this mechanism of selecting packets in EPLD does reasonably approximate per-flow differentiation.

4 Approximate Scheme

The most expensive operations in EPLD are invoked when a packet has to be dropped. The scheme requires that a packet be dropped randomly (weighted by the virtual count it carries) from the logical queue of a selected class – this implies that the router may have to traverse the logical queue of a class completely before it selects a packet to drop. Traversal of each element of the queue incurs an overhead of multiple memory accesses. A drop operation would be invoked only when the router queues are reasonably large – therefore, such an overhead could further slow down the already overloaded router. If we drop packet from the end of logical queue, then fairness across flows within a class is not ensured. A flow with very high VC_i can result in high loss in other flows in the same class.

Schemes like WRED do not incur this overhead because they adopt a policy of dropping the packet at the tail (or head) of the FIFO queue. However, this leads to the coupling of the decision of *when* to drop a packet with *which* packet to drop – In Section 2, we have seen that this is undesirable. Hence, we introduce the following approximation to achieve a balance in the tradeoff between traversal of just a single element of the class queue and the ideal scheme that traverses the class queue completely. We select a packet randomly (weighted by its virtual count) from only among the last l packets in the logical queue of a class. The smaller the value of l, the more unfair is the scheme across flows within a class, and the larger the value of l, the closer it would be to ideal EPLD. In order to ascertain

Var Backlog[$1 \ldots N$];
Accumulate[$1 \ldots N$];
Deficit[$1 \ldots N$];
VL[$1 \ldots N$];
Queue[$1 \ldots N$];

Initially $\forall i :: (Backlog[i], Accumulate[i], Deficit[i], VL[i], Queue[i] := 0, 0, 0, 0, \phi)$;

Enque(p) {Enque the packet p}
i := Class(p);
Queue[i] := Queue[i] + $\{p\}$;
$\forall j :: (VC_j(p) := Backlog[j]; Backlog[j] := 0)$;
$VC_i(p) := VC_i(p) + marked(p)$;
{marked(p) is VC_i value stored in the header of packet p}
$VL[i] := VL[i] + VC_i(p) + Deficit[i]; Deficit[i] := 0$;
$\forall j \neq i :: (Deficit[j] := Deficit[j] + VC_j(p))$;

Drop(p) {Remove packet p from the queue}
if $(p.next \neq nil)$ then $\forall i :: (VC_i(p.next) := VC_i(p.next) + VC_i(p))$;
else $\forall i :: (Backlog[i] := Backlog[i] + VC_i(p))$;
i := Class(p);
Queue[i] := Queue[i] - $\{p\}$;

Service(p) {Service the packet p (which is head of queue)}
$\forall i :: (Accumulate[i] := Accumulate[i] + VC_i(p))$;
$\forall i :: (VL[i] := VL[i] - VC_i(p))$;
i := Class(p);
$marked(p) := Accumulate[i]; Accumulate[i] := 0$;
{marked(p) is VC_i value stored in the header of packet p}
Queue[i] := Queue[i] - $\{p\}$;

Figure 3: Pseudo-code for EPLD

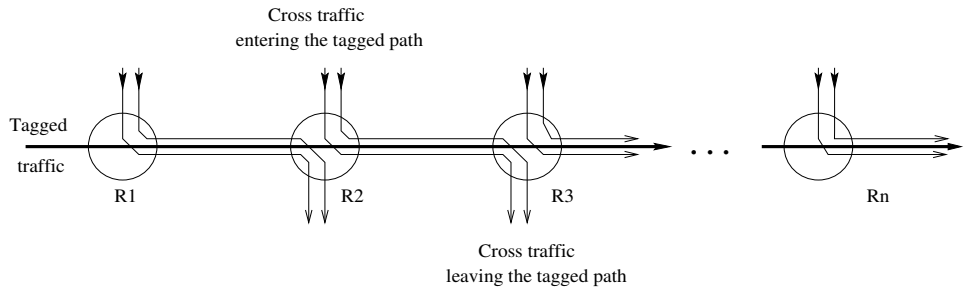


Figure 4: Network Topology

a suitable value of l, we conduct an extensive set of experiments, the results of which are presented in Section 5.3.2. From the experiments (Figure 10(b)), we find that an approximate scheme with $l = 4$ functions very similar to the ideal scheme.

5 Experimental Evaluation

We have implemented EPLD in the *NS-2* network simulator [2]. Using this simulator, we evaluate EPLD in the following two settings.

1. **Single-hop topologies**: We evaluate the efficacy of EPLD in achieving proportional loss differentiation among traffic aggregates that share a single network node. We investigate whether this also results in loss differentiation across individual flows within the traffic aggregates. We then demonstrate the benefits of the decoupling of the decisions of *when* to drop a packet from *which* packet to drop by comparing the performance of EPLD with WRED.

2. **Multi-hop topologies**: We demonstrate that a network deploying EPLD can provide proportional loss differentiation across any shared portion of multi-hop paths. We then conduct a series of experiments to evaluate the effects of the approximations introduced in Sections 3 and 4.

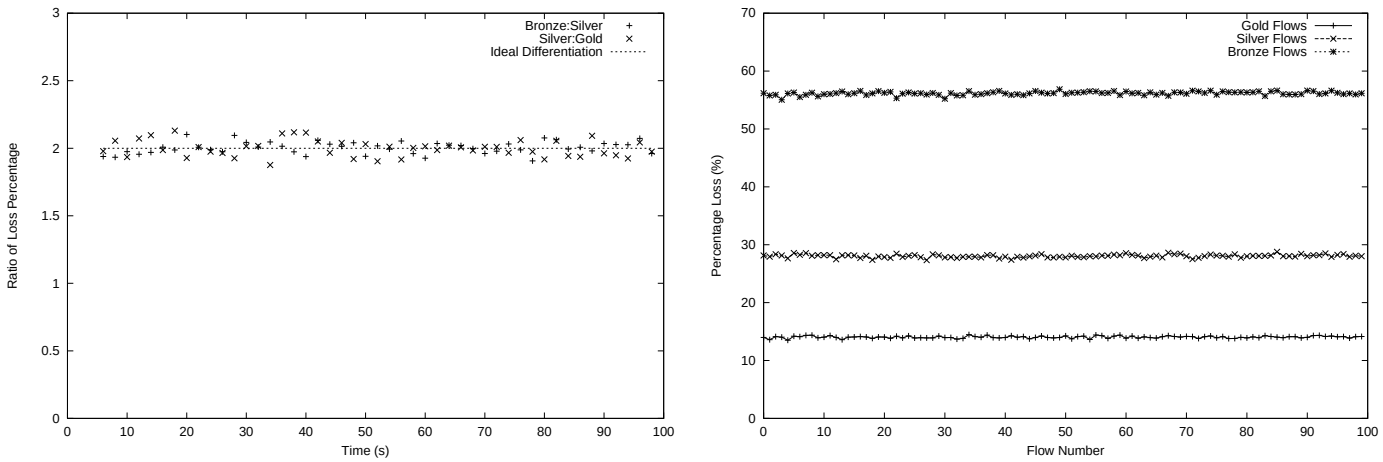
In what follows, we describe our simulation setup and discuss the experimental results.

5.1 Experimental Setup

For all experiments, we consider a linear network topology consisting of n nodes as shown in Figure 4. Each link in the network has a capacity of $40Mbps$ and an propagation delay of $5ms$. All packets are $500B$ in size. The buffers at each link are provisioned to hold upto 100 packets, which is equal to the *delay-bandwidth product* of the link. Each experiment is run for a simulation duration of $100s$.

The network is assumed to offer 3 service classes with respect to proportional loss differentiation – we label these as the *gold*, *silver*, and *bronze* classes. Unless specified otherwise, the ratio of loss differentiation parameters of the gold, silver, and bronze classes is assumed to be $\alpha_{gold} : \alpha_{silver} : \alpha_{bronze} = 1 : 2 : 4$. We use RED to decide when to drop a packet. The values of Minthresh, Maxthresh, and p_{max} parameters of RED are set to 5, 80, and $\frac{1}{3.42}$ respectively. When RED indicates that a packet needs to be dropped, EPLD is used to decide which packet to drop. For WRED, we select the values of the parameters as: $p_{max}^{silver} = \frac{1}{4}$, $p_{max}^{gold} = p_{max}^{silver}\alpha_{gold}/\alpha_{silver}$, and $p_{max}^{bronze} = p_{max}^{silver}\alpha_{bronze}/\alpha_{silver}$. These values are chosen so that if the total traffic is composed of equal amount of traffic from the three classes, then the average loss rates seen with WRED and RED are equal.

All traffic entering the network at the first node and leaving at the last node in the linear topology is labeled as the *tagged traffic*. At every node in the linear topology, traffic other than the tagged traffic enters the network and exits the network at the next node – we label all such traffic as the *cross-traffic*. The tagged traffic shares the bandwidth



(a) Across Aggregates

(b) Across Individual Flows

Figure 5: Proportional Loss Differentiation in EPLD

of each outgoing link with the cross-traffic that enters the corresponding node on its path. All traffic is generated by the aggregation of exponential *on-off* sources with an average on- and off-duration of $0.5ms$ each–the number of aggregated sources are specified for each experiment.

5.2 Evaluation in Single-hop Topologies

5.2.1 Validation of EPLD

In the first experiment, we simulate a one-hop network, with an average aggregate bit-rate of the gold, silver, and bronze traffic of $10Mbps$ each. The traffic aggregate in each class is generated by the aggregation of 100 exponential *on-off* sources. The average bit-rate of each of the 100 sources within a service class is chosen uniformly in the range $0.5Mbps$ to $0.15Mbps$. We measure the loss rates seen by the aggregate gold, silver and bronze traffic in observation intervals of duration $2s$ each. In Figure 5(a), we plot the ratio of these loss rates for the silver versus gold traffic and the bronze versus silver traffic. We observe that the ratios are very close to the desired value of 2.

This verifies proportional loss differentiation with respect to the losses seen by traffic aggregates – the next interesting question that arises is whether this results in differentiation across individual flows subscribing to the traffic classes. To see this, in Figure 5(b), we plot the loss rates observed by each of the 300 flows over the entire run of the simulation in the above experiment. We find that the ratio of loss rates of the individual flows are also in the ratio of the class loss differentiation parameters – hence, loss differentiation is seen across individual flows within traffic aggregates.

5.2.2 Insensitivity to Traffic Composition

In order to investigate whether loss differentiation in EPLD is sensitive to traffic composition, we next consider a network in which we vary the ratio of sending rates of the traffic aggregates. The aggregate bit-rate of the gold and silver traffic is set to $10Mbps$ each, but the aggregate bronze bit-rate is varied from 0 to $18Mbps$ to generate different network scenarios. For each such scenario, we measure the ratio of loss rates seen by the traffic from the silver versus gold class and the bronze versus silver class. Figure 6 plots these ratios against the aggregate bronze bit-rate. We find that the loss differentiation provided by EPLD is independent of the service class composition of the total traffic. The loss ratios deviate when the bronze load is very low – this is because there are very few packet losses (around 0.001%) at this level of utilization.

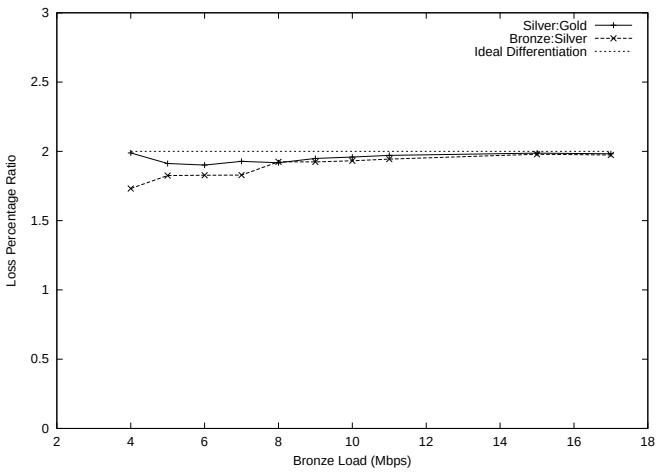


Figure 6: Insensitivity to Traffic Composition

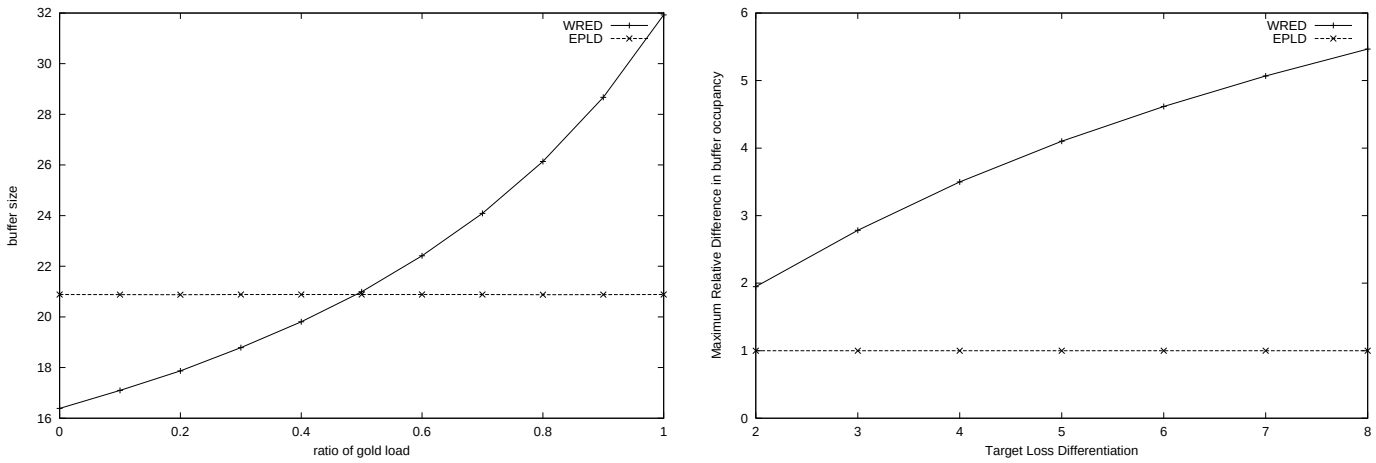
5.2.3 Consistency in Congestion Notification

As argued in Section 2, a scheme like WRED may provide inconsistent notification of congestion to the end hosts. Depending on the traffic composition, two networks that differ substantially in their WRED buffer occupancy could drop packets and give similar indications of congestion to an end host. Since EPLD decouples the decision of when to drop a packet from which packet to drop, it does not suffer from this limitation and preserves the semantics of RED in giving consistent congestion notification to end hosts. To demonstrate this effect, we generate a set of network scenarios, where the aggregate utilization of the network is held constant, but the traffic composition is varied. The aggregate bit-rate of the silver traffic is set to $20Mbps$. The sum of the aggregate bit-rates of the gold and bronze traffic is set to $25Mbps$. We vary the service class composition of the traffic by varying the fraction of this sum occupied by the gold traffic from 0 to 1 – these two extremes represent scenarios with no gold and bronze traffic respectively. Figure 7(a) plots the buffer occupancy for different fractions of the total traffic occupied by gold flows. We see that in scenarios where the traffic mainly belongs to the bronze class, buffer occupancy is low for WRED – implying that WRED drops more packets in these scenarios although the overall load on the network is held constant. Note that depending on the traffic composition, the buffer occupancy with WRED can differ by even 100% under the same overall load in the network. EPLD maintains the buffer occupancy at the same level for different traffic composition settings.

This effect can get highlighted by the specific choice of the desired loss rate ratio. To demonstrate this, we repeat the above experiment for different ratios of the loss differentiation parameters (as against a value of 2 in the above experiment). We vary the desired ratio of loss rates across the silver-gold or the bronze-silver classes, $\alpha_{silver}/\alpha_{gold} = \alpha_{bronze}/\alpha_{silver}$, from 2 to 8. For each of these settings, we measure the buffer occupancy in the two cases when the fraction of gold traffic in the sum of the gold and bronze traffic is either 0 or 1. Figure 7(b) plots the ratio of these two values of buffer occupancy as a function of the desired loss differentiation ratio. In these experiments, we observe differences in buffer occupancy as high as even 400%.

5.2.4 Insensitivity to Parameter Setting

We next demonstrate that as a result of the coupling of the *when* and *which* decisions, WRED may fail to achieve loss differentiation for some choices of RED parameters. In this next experiment, the aggregate bit-rate of the gold, silver, and bronze traffic is set to $15Mbps$, $15Mbps$, and $10Mbps$ respectively. We vary the MinThresh parameter of the RED curves from a factor of 0.8 to 1.0 of the MaxThresh parameter. For each of these parameter settings, we simulate two kinds of networks – one that employ EPLD on top of RED with this setting, and the other that



(a) Variation in Buffer Occupancy (b) Effect of Desired Ratio in Loss Rates

Figure 7: Inconsistent Congestion Notification in WRED

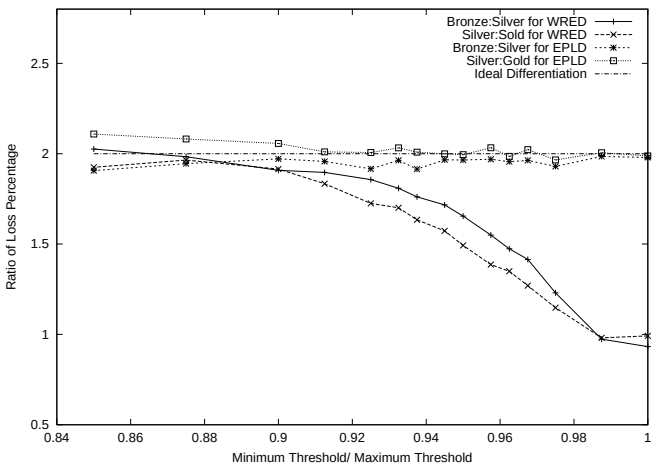


Figure 8: Sensitivity of WRED to Parameter Settings

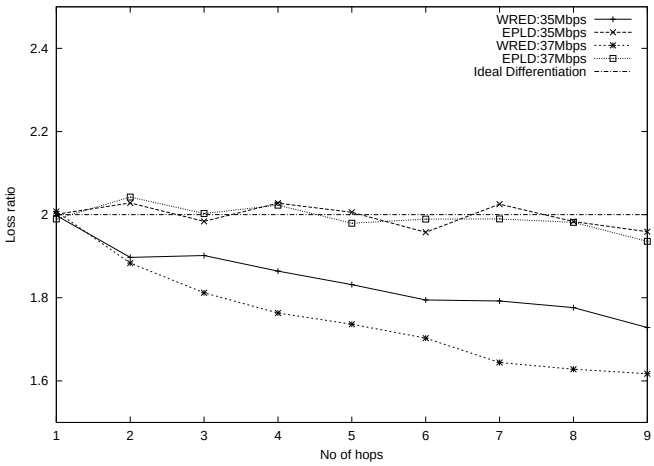


Figure 9: Loss Differentiation Across Multiple Hops

employ WRED with the above setting. In Figure 8, we plot the ratio of loss rates seen by the silver versus gold traffic and the bronze versus silver traffic for the different settings of MinThresh. We observe that as the value of MinThresh approaches that of MaxThresh, the ratio of loss rates across classes deviates from the desired value. Loss differentiation in EPLD is however independent of the value of this parameter.

5.3 Evaluation in Multi-hop Topologies

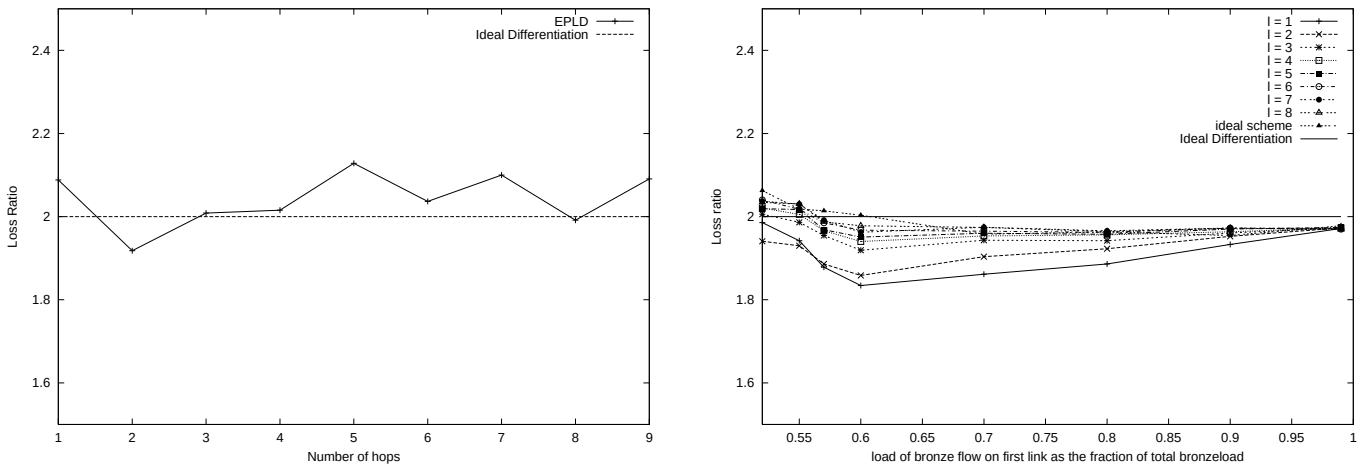
5.3.1 End-to-end Scope of Loss Differentiation

As illustrated in Section 2, a scheme that provides proportional loss differentiation at a given node would not provide such a differentiation on an end-to-end basis without the introduction of additional mechanisms. EPLD instantiates a set of mechanisms to achieve this. In this set of experiments, we compare EPLD to WRED for the loss differentiation achieved when packets get dropped at more than one node in the network. In this set of experiments, we introduce 2 tagged flows belonging to the gold and silver classes, with a bit-rate of $500Kbps$ each. At each node, the cross-traffic in each class is modeled as an aggregate of 10 exponential on-off sources. We simulate two networks by introducing at each node, a cross-traffic with an aggregate bit-rate of $35Mbps$ and $37Mbps$ respectively. This models networks running at high level of utilization–such conditions are true during network congestion at busy times of the day. Such a choice of traffic load also ensures that the congestion at subsequent nodes is not affected (reduced) by the drops in the tagged flows. Figure 9 plots, for these two settings of the cross-traffic load, the ratio of end-to-end loss rates observed by the silver versus gold tagged flows for different values of n. We observe that EPLD achieves loss differentiation even when packets get dropped at more than a single node, while the deviation of WRED from the desired ratio increases for larger values of n. This deviation also increases with increase in the average loss rate (or utilization) in the network.

5.3.2 Efficacy of Approximate Scheme

Next, we conduct experiments to verify the validity of the approximations that were incorporated to make the scheme scalable.

In Section 3.4, we have seen that when the tagged traffic interacts with cross-traffic that has seen very different loss rates, then VQL_i^x at subsequent nodes may not correctly represent the total number of packets transmitted by the sources of flows in class J_i. In the next experiment, we investigate the impact that this may have on loss differentiation. We simulate a 9-hop topology. A tagged gold flow with a bit-rate of $4Mbps$ is introduced at the first



(a) Effect of Diversity in Losses of Merging Flows

(b) Efficacy of Approximate Scheme

Figure 10: Validity of Approximations

node. At each node, the cross-traffic in each class is modeled as an aggregate of 10 exponential on-off sources. In addition, the cross-traffic consists of a silver flow with a bit-rate of $1Mbps$. The aggregate bit-rate of the cross-traffic is set to $35Mbps$. With this setting, the tagged gold flow experiences packet losses at each node, and interacts with gold cross-traffic that has seen no losses. This may lead to increasing errors in the *Virtual Count* of the tagged traffic at subsequent nodes. At each node, we measure the ratio of loss rates *at that node* of the tagged gold flow and the silver flow in the cross-traffic, with respect to their respective source rates. Figure 10(a) plots this ratio as a function of the number of the node in the topology. From the figure we observe that errors in the *Virtual Count* of the tagged gold flow do not adversely impact the loss differentiation.

In the next experiment, we evaluate the efficacy of the approximate scheme introduced in Section 4 in achieving proportional loss differentiation for various values of the approximation parameter l. Recall that in the approximate scheme, the router considers only the last l packets of a class to select a packet to drop. We simulate a linear topology of 2 hops where the tagged traffic consists of flows from the gold, silver, and bronze classes. Cross-traffic—that belongs to the bronze class—is introduced only at the second node. With this setup, we ensure that the bronze cross-traffic has seen no losses before entering the second node, while the bronze tagged traffic may have experienced losses at the first node before entering the second node. The aggregate bit-rate of the tagged gold and silver traffic are $11Mbps$ each. The sum of the tagged bronze traffic and the bronze cross-traffic entering the second node is kept fixed at $25Mbps$. The larger the fraction of the tagged bronze traffic in the total bronze traffic, the larger the probability for it to experience packet losses at the first node–this is so because this would increase the utilization of the first link, leading to packet losses. In turn, this would mean that at the second node, the tagged bronze traffic that interacts with the bronze cross-traffic typically has much larger values of *Virtual Counts* than the latter. On the other hand, that tagged silver and gold flows do not interact with cross-traffic of their respective classes that has typically different *Virtual Counts*.

We vary the fraction of the tagged bronze traffic in the total bronze traffic and measure the ratio of end-to-end loss rates of flows in the tagged silver and tagged bronze traffic respectively. Figure 10(b) plots these ratios for different values of l, the approximation parameter. As expected, we observe that the larger the value of l, the closer is the performance to the ideal scheme. However, the performance improvement reduces for larger values of l. In particular, we find that an approximate scheme with $l = 4$ functions very similar to the ideal scheme.

6 Implementation Issues

We are currently implementing EPLD in Intel's IXP1200-based programmable router [1]. In this section, we discuss some of the issues involved in implementing EPLD in routers and discuss the corresponding space and computational overhead.

6.1 Overview of IXP1200

IXP1200 is Intel's Network Processor designed for high-speed programmable routers. IXP1200 offers an open architecture; it contains 7 parallel processors—6 microengines that perform packet processing functions in the data path, and a StrongARM core that generally executes code for the control path of a network protocol. Each processor on the IXP1200 has its independent set of registers. Our development prototype board offers a three-level memory hierarchy: (1) a fast, small, on-chip scratchpad memory; (2) an off-chip SRAM-based cache; and (3) an off-chip SDRAM module as the main memory store.

The IXP1200 processor architecture is highly pipelined; it executes most integer operations in a single cycle. Off-chip memory accesses, on the other hand, can cause several cycle stalls. Hence, the main objective of our implementation is to efficiently utilize the three-level memory hierarchy; this involves minimizing the accesses to the slow SDRAM while managing efficiently the contents of SRAM and the scratchpad memory.

6.2 Implementing EPLD on the IXP1200 Platform

Our implementation of EPLD enhances Intel's *reference design*. In Intel's reference design, packets arriving at the router are stored in SDRAM and the packet descriptors are stored in SRAM. Packets in SDRAM are accessed during the receive and transmit operations; packet descriptors are accessed for all the packet processing and scheduling operations. The on-ship scratchpad is used to maintain condensed and frequently accessed information—such as bit vectors indicating packet queue status. Separate microengines perform the receive and transmit operations in parallel.

Implementation of EPLD involves maintenance of additional information for each service class and for each packet in the router. Specifically, for each service class, EPLD maintains 4 variables: (1) Backlog, (2) Accumulate, (3) Deficit, and (4) VL. Further, for each packet, EPLD maintains N VC values, where N Is the number of service classes. In our implementation, we maintain the 4 per-class variables in the on-chip scratchpad memory, and the N per-packet counters in the SRAM. In what follows, we estimate the storage space and computational overhead introduced by our implementation.

6.3 Storage Space Overhead of EPLD

A naive implementation may represent each of the 4 per-class variables and the N per-packet counter as an integer, and hence incur a storage space overhead of a word (i.e., 4 bytes) for each value. However, a closer analysis of the values taken by these variables reveals that it is possible to encode these values in a smaller number of bits, and hence compact the total storage space requirement for these variables.

Figure 11 illustrates that for a wide range of loss percentage values, observed values for the per-class and per-packet variables does not exceed 128, which is half the value we can fit in 8 bits. Hence, each variable can be safely encoded using a single byte.

With this optimization, the 4 per-class variables can be encoded in a single word (or 4 bytes); hence, to support N service classes, EPLD incurs the overhead of N words of additional scratchpad memory as compared to FIFO scheduling with tail-drop buffer management. Further, the N per-packet *count* values can be encoded using $\left\lceil \frac{N}{4} \right\rceil$ words of SRAM memory. To place the overhead incurred in maintaining these per-packet count values in context, consider our IXP1200 platform configuration. Intel's reference design allows, at any time, at most 250 packets to be

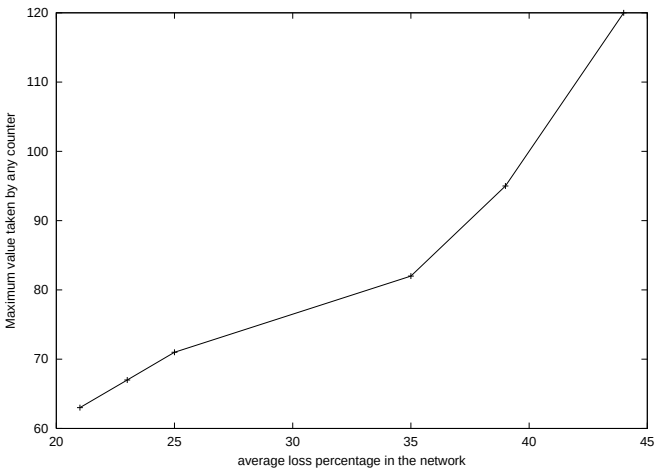


Figure 11: Maximum value taken by a variable at varied loss percentage.

maintained in the SDRAM. For each packet stored in the SDRAM, the EPLD implementation maintains N bytes of information in SRAM. Our current IXP1200 platform contains 8MB of SRAM; assuming that the router supports 4 service classes (i.e., $N = 4$), the per-packet count values needed by EPLD will, in the worst case, occupy at most 0.01% of the available SRAM. Hence, the overall storage space overhead for EPLD is negligible.

6.4 Time Complexity of EPLD

To determine the time-complexity of implementing EPLD , we determine, for each of the *enqueue*, *dequeue*, and *drop* functions described in Figure 3, the the number of additional (as compared to the FIFO queue implementation with drop-tail buffer management policy) memory and register accesses as well as computational instructions required in the IXP1200 architecture. Note that the *enqueue* and *dequeue* functions are performed for each packet, while the *drop* function is invoked only when a packet is dropped from the router queue. Table 1 summarizes the number of memory references and computational instructions required to update the variables maintained by EPLD .

Our preliminary evaluation indicates that these functions can be implemented in the IXP1200 platform without adversely affecting the router performance (i.e., the router will continue to receive and transmit packets at the maximum link speeds). Recently, it was also reported in [15] that the current IXP1200 platform offers a 400 cycle extra capacity on packet receives (i.e., additional processing that consume upto 400 cycles per packet can be added without slowing down the router). These 400 cycles can accommodate 28 SRAM accesses [15]. If we assume $N = 4$ and hence $W = 1$ word, then our implementation of EPLD incurs an overhead of 6 SRAM word accesses for the *drop* function, and 1 SRAM word accesses and 6 scratchpad word accesses for the *enqueue* function. Hence, even if we assume that each scratchpad access is as expensive as an SRAM access, EPLD adds an overhead of at most 13 SRAM accesses, leaving sufficient room for the execution of the additional computational instructions. We expect to report detailed measurements on the impact of EPLD on the receive and transmit performance in the final version of this paper.

7 Related Work

The Assured service [5] and the Olympic service [8] definitions within the Differentiated Services Architecture [12], require the network to provide qualitative differentiation in the loss rates experienced by packets of flows subscribing to different service classes. In this paper, we have described a mechanism for providing end-to-end proportional loss differentiation in networks; this mechanism will enable the design of network services that export richer semantics—

Function	Variable	Scratchpad		SRAM		SDRAM		Computational Instructions
		Read	Write	Read	Write	Read	Write	
Enqueue	Backlog	W	W					
	Accumulate							
	Deficit	W	W					
	VL	1	1					1
	VC				W			W+1
	Total	2W+1	2W+1	0	W	0	0	W+2
Dequeue	Backlog							
	Accumulate	W	W					W
	Deficit							
	VL	W	W					W
	VC			W			1	
	Total	2W	2W	W	0	0	1	2W
Drop	Backlog							
	Accumulate							
	Deficit							
	VL							
	VC			$(l+1)$W	W			W
	Total	0	0	$(l+1)$W	W	0	0	W

Table 1: Time Complexity of implementing EPLD (as compared to FIFO with tail-drop buffer management). W denotes $\left\lceil \frac{N}{4} \right\rceil$. For each of the three operations, we have calculated the overheads attached with each of the variables. Blank entries in the table represent no overhead

quantitative loss differentiation—to network subscribers.

Loss differentiation involves buffer management. Conventional single-class buffer Management schemes—such as tail-drop and RED [7]—do not differentiate across packets belonging to different service classes. Several schemes—such as Longest Queue Drop (LQD) [17], Strict Priority Queuing [11], Complete Buffer Partitioning (CBP), Partial Buffer Sharing (PBS) [9], multi-class RED or WRED [13]—have been proposed to provide loss differentiation across different classes. LQD [17] combines buffer management with fair queuing and selects packet to drop from the flow with the longest queue at a router. This schemes imposes the need to maintain per-flow state and perform per-flow packet classification in routers; this limits the scalability of routers that carry many flows. Complete buffer partitioning (CBP) statically partitions the buffer across different flows (or service classes). This limits the gains from *statistical multiplexing*, and leads to wastage of unused buffer space. Partial buffer sharing (PBS) addresses the drawback of CBP by limiting the fraction of the total buffer space that gets statically partitioned across different flows or service classes. WRED is an extension to RED, where different RED curves are used for dropping packets belonging to different service classes. Unfortunately, such simple extensions of RED have drawbacks. In Section 2, we showed that this scheme results in inconsistent notification of congestion to end-applications, and unlike RED, fails to maintain the buffer at a low configurable level independent of traffic composition.

In [6], the authors propose a scheme to achieve proportional loss differentiation coupled with proportional delay differentiation. However, loss differentiation in this and all of the schemes described above is limited to a single congested link in the network. We have shown that such per-hop differentiation does not extend to end-to-end loss differentiation in networks where significant losses could be experienced at multiple links.

8 Conclusions

Service differentiation is at the core of designing next-generation Internet. In this paper, we present a buffer management framework for achieving end-to-end proportional loss differentiation in networks. There are two main facets of our buffer management framework. First, it decouples the decisions of *when* to drop a packet from *which* packet to drop. This allows the framework to utilize existing single-class buffer management techniques—such as RED—to determine *when* to drop a packet; in fact, when instantiated with RED, the framework extends the primary advantages of single-class RED—namely, early notification of congestion and maintenance of average buffer occupancy at low, configurable levels—to a multi-class workload. Second, at each router, the framework governs the selection of *which* packet to drop based on the number of packets of a flow transmitted by its source, rather than the number of packets that arrive at a router. The framework achieves this by encoding information about the losses observed by a flow at a router in packet headers. This allows the framework to provide end-to-end proportional loss differentiation, unlike most existing schemes that provide loss differentiation only on a per-hop basis.

We are currently in the process of implementing and evaluating EPLD in programmable routers based on the Intel IXP1200 network processor. Our initial estimates indicate that EPLD can be implemented in these routers without any degradation in their performance.

References

[1] Intel's IXP1200 Network Processor. http://developer.intel.com/design/network/products/npfamily/ixp1200.h tm.

[2] NS (Network Simulator). 1999. http://www-mash.cs.berkeley.edu/ns.

[3] M Chesire, A Wolman, G M Voelker, and H M Levy. Measurement and Analysis of a Streaming-Media Workload. In *USITS*, 2001.

[4] Kc. Claffy. Internet measurement and data analysis: topology, workload, performance and routing statistics. 1999. http://www.caida.org/outreach/papers/Nae.

[5] D. Clark and W. Fang. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 1(6):362–373, August 1998.

[6] C. Dovrolis and P. Ramanathan. Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping. In *International Workshop on Quality of Service*, June 2000.

[7] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, volume 1, pages 397–413, August 1993.

[8] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. June 1999. Internet RFC 2597.

[9] H.Kroner, G.Hebuterne, P.Boyer, and A.Gravey. Priority Management in ATM Switching Nodes. In *IEEE Journal on Selected Areas in Communications*, volume 9, pages 418–427, April 1991.

[10] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. June 1999. Internet RFC 2598.

[11] A.M. Lin and J.A.Silvester. Priority Queueing Strategies and Buffer Allocation Protocols for Traffic Control at an ATM Integrated Broadband Switching System. In *IEEE Journal on Selected Areas in Communications*, volume 9, pages 1524–1536, December 1991.

[12] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. November 1997. ftp://ftp.ee.lbl.gov/papers/dsarch.pdf.

[13] S Sahu, P Nain, D Towsley, C Diot, and V Firoiu. On Achievable Service Differentiation with Token Bucket Marking for TCP. In *Proceedings of ACM SIGMETRICS*, June 2000.

[14] S. Shenker and C. Partridge. Specification of Guaranteed Quality of Service. Available via anonymous ftp from ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-intserv-guaranteed-svc-03.txt, November 1995.

[15] Tammo Spalink, Scott Karlin, and Larry Peterson. Evaluating Network Processors in IP Forwarding. In *Princeton University Technical Report TR-626-00*, November 2000.

[16] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings ACM SIGCOMM'98*, Sept 1998.

[17] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Design considerations for supporting TCP with per-flow queuing. In *Proceedings of IEEE INFOCOM*, April 1998.