Introducing Size-oriented Dropping Policies as QoS-supportive Functions

Stylianos Dimitriou, *Member, IEEE*, Ageliki Tsioliaridou, and Vassilis Tsaoussidis, *Senior Member, IEEE*

***Abstract*—The continuous increase of Internet users worldwide, as well as the extensive need to support real-time traffic and bulk data transfers simultaneously, has directed research towards Service Differentiation schemes. These schemes either propose techniques that provide users with the necessary quality guarantees or follow a "better-than-best-effort" approach to satisfy broadly the varying needs of different applications. We depart from our new service principle called Less Impact Better Service (LIBS) and propose a novel Service Differentiation method, namely Size-oriented Dropping Policies, which uses packet size to categorize time-sensitive from delay-tolerant flows and prioritize packet dropping probability, accordingly. Unlike existing proposals, the distinction of flows is dynamic and the notion of packet size is abstract and comparative; a packet size is judged as a unit within a dynamic sample space, that is, current queue occupancy. We evaluate Size-oriented Dropping Policies both analytically and experimentally; we observe a significant increase on the perceived quality of real-time applications. Delay-sensitive flows increase their bandwidth share, to reach a state of system fairness, regulating the dominant behavior of bulk-data flows.**

***Index Terms*—Active Queue Management, Fairness, Service Differentiation**

I. Introduction

THE diversity of Internet applications along with the increased service expectations of modern Internet users call for networks with diverse service capacity. Due to the limited management capability of Internet flows on a user- or application-oriented basis, services and requirements cannot form a one-to-one corresponding relation; instead, they can provide the distinctive input to a broader and abstract Service Differentiation scheme. This allows for preserving the distributed management structure of the Internet and satisfies broadly user requirements as well. Hence, the real issue in Quality of Service supportive schemes is their capability to provide better service without increasing the management complexity of the Internet and without damaging its main properties, which is resource sharing, utilization efficiency and system fairness.

The Size-oriented Dropping Policies (SDP) [12] scheme that we propose, analyze and evaluate here promotes further a class of services that we defined in [21] as Less Impact Better Service (LIBS). The LIBS discipline imposes that traffic that causes only minor delays should enjoy increased privileges over the rest of the traffic and in this context, it defines a delay-oriented (instead of throughput) metric of fairness. This is a rather logical expectation in many aspects: one that feeds a network with a few bytes cannot tolerate huge delays; a user that transfers huge files is prepared to tolerate more delays; the impact of network delay is crucial for real-time applications; the impact of small delays on long-lasting applications may even not be recognized by the user. LIBS relies on that last idea, precisely: it exploits the time that is statistically insignificant for delay tolerant applications to promote the service of delay-intolerant application. Deploying LIBS practically means that the transmission of a SYN message should be favored at the expense of an FTP transmission. LIBS philosophy can be implemented either by scheduling or by dropping. One way to apply LIBS by scheduling is to favor high priority packets and forward them to their destination, immediately upon their arrival. The NCQ [20] algorithm, which incorporated the LIBS discipline into packet scheduling, promotes small packets in the queue and increases their chances of successful arrival. NCQ and its ancestor, NCQ+ [23], distinguish traffic into non-congestive (small and tiny packets) and congestive (big packets). Non-congestive data, which includes VoIP and sensor traffic, is considered to have small impact on contention and receives special service.

While LIBS was realized using scheduling disciplines, here we exploit the possibility of realizing LIBS by differentiating dropping policies. Furthermore, we also highlight the possibility of combining both techniques, which are complementary indeed.

SDP is implemented using the experience gained by the RED scheme. Minimum and maximum thresholds define the regions where unforced and forced dropping occurs. In addition, SDP records the size of each incoming packet to calculate an average packet size which serves temporarily as a rough guide to differentiate small and big packets comparatively and dynamically. Clearly, a comparative distinction is vital for the efficiency of the proposed scheme:

S. Dimitriou is with Democritus University, 67100, Greece (phone: +30-2541079556; fax: +30-2541079554; e-mail: sdimitr@ee.duth.gr).

A. Tsioliaridou is with Democritus University, 67100, Greece (e-mail: atsiolia@ee.duth.gr).

V. Tsaoussidis is currently with the Massachusetts Institute of Technology (e-mail: tsaoussi@mit.edu).

This work was funded by the European Commission and the project PENED 2003 of GSRT.

whether or not the next arriving packet will be dropped will depend on this average size. If the next packet exceeds the average size, it will be dropped with the same probability as imposed by RED. If, however, the next packet is smaller than the average size, then its dropping probability will be smaller and proportional to the packet size. This adaptive behavior of SDP allows it to define traffic classes dynamically, depending on the circumstances. We cannot characterize a given packet as small or big in advance; SDP will decide how to classify it based on recent arrival history.

So, does packet size suffice as a criterion to differentiate application services? Small packets usually correspond to real-time applications, such as VoIP or video streaming, whose successful and timely delivery affects significantly the user-perceived quality. Real-time data is sent over UDP or TCP-friendly protocols; these are typically non-, or less-responsive and may fail to satisfy the efficiency objective, let alone the fairness objective [25]. On the other side, bulk-data applications, such as FTP or BitTorrent [1], use big packets and TCP. They are characterized by loss and delay tolerance and their behavior is responsive; losses determine their sending rate. A network that consists of both real-time and bulk-data traffic exhibits inevitably service bias when resources are exhausted: responsive flows can and will exploit any available bandwidth, compelling unresponsive flows to a small link share. In SDP gateways, small packets experience less dropping than in RED, enforcing intentionally dynamic reallocation of resources. Real-time applications are therefore allowed to increase their sending rates up to some threshold that guards prioritized operations within the confines of fairness. How much priority should we grant to small flows? As long as bulk-data transmissions dominate network contention, we should promote time-sensitive applications; when more real-time flows populate the network and their service impact becomes significant on other flows, LIBS services should be canceled.

SDP exhibits three main characteristics:

1. Dynamic Management. Packet classification is performed on-the-fly and the network dynamics are reflected into the classification per se. Static quantitative thresholds (e.g. fixed packet sizes), or dubious qualitative criteria, such as the underlying transport protocol, limit system flexibility, including service requirements of future applications.

2. Cost-effective administration. SDP has trivial memory and processor requirements. It is semi-stateless since it uses only a single variable and the information needed from the packet (i.e. size) is easily extractable.

3. Ease to deploy. Its simple design makes SDP easily deployable. It does not require end-user modifications and its algorithm can be easily integrated in routers functionality as a policy furnishing.

During the experimental evaluation of our method, we tested SDP in a wide spectrum of network topologies, using various metrics; we compared it with Droptail, RED and NCQ+. We show that SDP outperforms current implementations without giving critical flows more resources than their fair share. SDP not only increases system Fairness but also Goodput and channel utilization.

The structure of the paper is as follows: In section 2 we discuss the related work. In section 3, we present our algorithm analytically and discuss its advantages over other proposals. In section 4 we analyze the impact of SDP on packet loss rate and queuing delay and in section 5 we examine how SDP deters users from misbehaving with packet fragmentation. In section 6, we describe our evaluation methodology including the simulation setup and, in section 7, the evaluation metrics. Sections 8 and 9 outline and analyze the experimental evaluation of SDP. Finally section 10 outlines our conclusive remarks and future work.

II. Related Work

Service differentiation has been developed on the basis of resource reallocation in line with the corresponding service requirements of diverse applications. Relevant applications are those with strict delay, jitter or loss constraints, which can be satisfied, typically, by prioritizing real-time data over bulk data transfers. The DiffServ [2] approach enables prioritization by relying on marking with corresponding service identifiers, whereas the IntServ [4] approach reengineers the architecture itself to allow for guarantees through signalling and reservation. However, a significant number of proposals have emerged on the basis of router enhancements in order to support service differentiation without affecting the end-nodes. Among those, the LIBS principle shares a common objective and relies on common packet-oriented policies; however, LIBS redefines fairness in terms of delay and the fair-share in reverse terms of contribution to delay. The NCQ mechanism [20], which deployed LIBS using a scheduling-oriented approach, distinguishes packets into big packets that cause significant delays and eventually inflict congestions (congestive) and small packets that cause minor delays (non-congestive). Non-congestive packets are prioritized in the queue as long as their number does not exceed a threshold. In turn, this threshold is adjusted in accord with the fairness performance of participating flows. Although the concept is generally known to networking and operating system communities, the dynamic resource management techniques within the framework of delay-oriented fairness has not been previously analysed. NCQ, as a product of LIBS philosophy, manages to increase fairness among congestive and non-congestive applications.

Several dropping-oriented approaches exist that either provide some service differentiation or manage resources fairly and, occasionally, in an application-oriented manner. Although they do not exhibit the same level of service sophistication as NCQ and SDP, along with their primary objectives, they also achieve some fundamental service differentiation. In [14], Floyd and Fall introduced mechanisms based on the identification of high bandwidth flows from the drop-history of RED. In [7] the authors propose an explicit allocation of bandwidth to various flows based on their respective needs and determine this allocation by modifying accordingly the dropping probability. Unfortunately, both these methods demand costly memory structures. Weighted RED with Thresholds (WRT) [3] calculates a separate average length for the higher-priority packets, preventing starvation for the lower-priority traffic. Flow RED (FRED) [18] uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use. Certainly, extended memory and processor power is required for a large number of flows. On the other hand, RED-PD (Preferential Dropping) [19] maintains a state only for the high-bandwidth flows and drops their packets more frequently than packets from low-bandwidth flows. Still, increased number of flows require memory-demanding approaches. Note that, unlike LIBS, in all the above approaches, bandwidth is considered the scarce resource and mechanisms are designed to manage bandwidth allocation.

Fair Queuing [11] maintains equal queues for each flow and in Weighted Fair Queuing the queues can have different length. Core-Stateless Fair Queuing [24] uses two types of routers; edge and core. Edge routers compute per-flow rate estimates and label the packets with these estimates, whereas core routers drop the packets probabilistically based on these labels. Nevertheless, such techniques involve radical modifications on the network's structure. Finally, the CHOKe mechanism [22] attempts to identify flows that heavily occupy the bottleneck queue by matching every incoming packet against a random packet in the queue and either drop both, if they belong to the same flow, or accept them with a certain probability. The accuracy, however, of this method remains an open issue as it does not take account of real-time traffic.

III. Size-oriented Dropping Policies

A. *Justification*

Application layer protocols define the structure of a packet as well as specific transport details of the traffic pattern that will be followed by the respective flow. Probably, packet size is the most typical and easy-to-extract indicator of the type of the application that created the packet. The transmission delay of a packet is proportional to its size and determines its probability of being accepted successfully by the router. Small packets are utilized by applications that require fast delivery times, constant inter-arrival times, limited packet losses or by applications that do not generate periodically great volumes of data. This is not a product of coincidence or some negotiation; real-time applications rely on sampling techniques for voice or images and hence, packet generation and content is not really an administrative issue. Beyond that, these applications include real-time applications such as audio and, less often, low bit-rate video streaming (VoIP and IPTV) or critical applications, such as DNS and sensor monitoring. Due to the demands of such applications, continuous packet losses usually degrade severely their performance, distort the user-perceived result or cause unnecessary retransmissions that limit the lifetime of battery-powered devices.

On the other hand, big packets are preferred for bulk data transfers as they are characterized by small overhead to payload ratios. In such cases, we can tolerate small delays, as long as the throughput remains greater than an acceptable limit, which certainly may vary from application to application or from user to user but also may vary depending on network contention, time of the day, and end-to-end distance. Big packets are mainly used by file sharing protocols such as FTP or BitTorrent [1]. Dropping such packets affects the application's throughput, since the underlying transport protocol (TCP) will detect the loss and respond; yet data integrity will not be damaged as the packet will be recovered.

It is apparent that a protocol capable of categorizing flows by the size of their packets will potentially provide Service Differentiation. Yet, the characterization of the packet size is (and should remain) flexible. In common networks, real-time applications typically utilize packet sizes bellow 200B (for example VoIP uses 140B packets) and bulk data applications utilize 1KB packets. However, in the next few years as high-speed networks will be further spread and new types of traffic will emerge, it is difficult to predict how the correlation of packet sizes will evolve. Thus, utilizing static thresholds for a packet classification system, would impawn the algorithms adaptability to future applications.

Other packet properties could serve as indicators to identify packets as well, such as Type of Service, Source and Destination Ports (determine the type of application), and Time to Live. However, extracting these properties requires costly packet inspection and large reference tables (for Ports), which should be updated regularly and doesn't necessarily guarantee better results.

B. *The algorithm*

Based on the previous reasoning, SDP uses packet size to classify flows. However, this is not a fixed classification: SDP keeps track of one variable, *sdp_thresh*, which is the moving average of the incoming packets in the queue and which is used to dynamically and comparatively classify big or small packets[1]. If the size of the next arriving packet is greater than *sdp_thresh*, then the packet is classified as big and will be dropped with the same probability as imposed by the RED algorithm. On the other hand, if the size is less than *sdp_thresh*, then the packet is classified as small and the dropping probability will be less than the RED probability and is calculated based on *sdp_thresh* and the packet size (Fig. 1).

In Fig. 1, *sdp_drop* and *red_drop* are the dropping probabilities of SDP and RED respectively, while *pkt_size* is the size of the last packet entered the router. Fig. 1 depicts graphically the following:

in case pkt_size<sdp_thresh:

$$sdp\_drop = red\_drop \frac{pkt\_size}{sdp\_thresh} \quad (1)$$

[1] Note that our work does not require two classes of service necessarily; instead, packet classification may integrate more application categories.

in case pkt_size≥sdp_thresh:

$$sdp\_drop = red\_drop \quad (2)$$

finally *sdp_thresh* is calculated as follows:

$$sdp\_thresh = (1-a)\cdot sdp\_thresh + a\cdot pkt\_size \quad (3),$$

where the weight factor α can take small values, below 0.1, that can capture router's state. During the initialization, *sdp_thresh* is equal to the size of the first arriving packet.

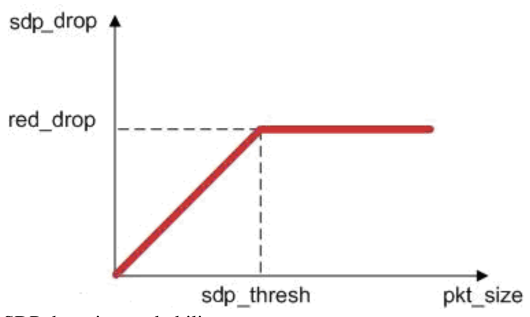

Fig. 1. SDP dropping probability.

We demonstrate the pseudo-code for the SDP algorithm. We consider a router that accepts packets noted as *pkt*. We use the following variables and functions (Table 1);

TABLE 1
Pseudo-code variables and functions

NAME	DESCRIPTION
size(pkt)	returns the size of the packet *pkt* in bytes
red_drop	the dropping probability of RED; computed elsewhere in the code
rand(x,y)	returns a random number between *x* and *y*
enqueue(pkt)	enqueues the packet *pkt* in the queue
drop(pkt)	drops the packet *pkt* from the queue

```
pkt_size=size(pkt)
sdp_thresh=0.9*sdp_thresh+0.1*pkt_size
if (sdp_thresh<pkt_size)
  then sdp_drop=red_drop*(pkt_size/sdp_thresh)
  else sdp_drop=red_drop
prop=rand(0,1)
if (prop<sdp_drop)
  then drop(pkt)
  else enqueue(pkt)
```

As we see, *sdp_thresh* depends more on the sizes of recently arrived packets and less on the packets that have recently departed - this renders *sdp_thresh* an implicit measure of the network's activity. Moreover, by giving equal priority to all packet sizes above *sdp_thresh*, we manage to serve smaller packets more effectively but still confine their service with the bandwidth restriction of the fair share. We elaborate on SDP functionality below, based on some selected scenarios.

C. *Case studies*

Managing effectively small packets

First consider a router mainly occupied by 1kB-packets when a 100B packet arrives. Since *sdp_thresh* is near 1kB, the probability of rejecting the last small packet is almost 90% less than dropping any other big packet (see (1), (2)), thus we can almost guarantee that this packet will be forwarded. After some time, small packets start gradually populating the queue. *Sdp_thresh* is decreased (see (3)) until small packets do not enjoy the same privilege, since they now contribute to the router contention.

Fair treatment for big packets

Next assume that the router serves mainly 100B-packets and a 1kB packet arrives. According to (2), this last big packet will have the same priority as the smaller ones, since *sdp_thresh* is near 100B. Although this might seem absurd, due to the predominance of small flows, the buffer space occupied by a sole big packet will cause only a small additional queuing delay, compared to the total. Dropping this packet will have a significant negative impact on its generating flow, but only a minor positive impact on the rest of the flows. We remind that a RED gateway operating in byte-mode (see [8]) would have probably blindly dropped this packet.

Adaptive behavior

Consider now a router that serves only 100B-packets until 1kB-packet flows begin their transmission. *Sdp_thresh* starts increasing and small packets enjoy comparatively less proactive dropping. As 1kB packet flows end their transmission, *sdp_thresh,* will again decrease, restoring the dropping probability at the previous levels. Throughout the entire time, SDP will eventually manage to maintain the same service quality for small packets.

Serving effectively multiple traffic classes

Finally, assume a queue of 1kB packets, where one 100B and one 500B packet arrive. Although they are both below *sdp_thresh*, they will not be dropped with the same probability. The 500B packet will be assigned bigger dropping probability than the 100B packet, but smaller than the 1kB packets.

D. *The significance of the weight factor α*

An important component of SDP that we analyze last is the α variable. We explained earlier why *sdp_thresh* should reflect the router's current state. As packets of different size populate the queue, *sdp_thresh* should be able to adjust fast enough to reflect the new state.

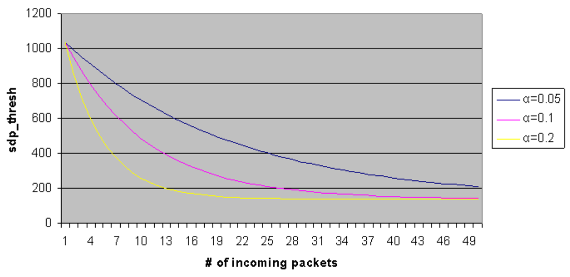

Fig. 2. Convergence of *sdp_thresh* with different values of α.

We assume that *sdp_thresh*=1040B and 140B packets arrive at the router. By setting *a*=0.1, *sdp_thresh*'s value will be below 150B in 43 steps, or 43 packets. If the buffer's capacity is near 140·43≈6kB then *sdp_thresh* will reflect only the packets currently in the queue, otherwise a bigger or smaller *α* value might be more appropriate (this is not exactly accurate however, since *sdp_thresh* takes account also the sizes of the packets that have been dropped). Different *α* values result in different convergence times (Fig. 2). For now we consider *α* static and equal to 0.1.

IV. Impact Analysis

We will now examine the impact of SDP on packets. Being SDP a dropping-oriented protocol, it affects both the Packet Loss Rate and the Queuing Delay. The aim of our analysis is to confirm that SDP achieves successfully Service Differentiation and whether there are cases that lead to unfairness or underutilization. Since RED and SDP are based on the same core-algorithm, we will study the impact of SDP on packets, in contrast to the behavior of packets served by RED gateways. For the readers' convenience, we will refer to flows that generate big packet sizes as big flows and to flows that generate small packet sizes as small flows.

During our analysis we use some variables (Table 2). When these variables have the letter R subscripted, they refer to RED, whereas when they have the letter S subscribed, they refer to SDP.

TABLE 2
Analysis variables

Name	Description
Db, Ds	the dropping probability of big and small packets, respectively
x	the *sdp_thresh* variable
sb, ss	the size, in bytes, of big and small packets, respectively. Unless stated otherwise, we assume that all big and all small packets have the same size
T_{D1B}	the transmission delay of 1 byte in the given link conditions, used as delay unit
Qd	the queuing delay of a packet

A. Packet Loss Rate

SDP aims at minimizing the loss of small packets. Since small packets usually characterize real-time traffic that is not typically governed by the AIMD principle, the effect on decreased dropping is related directly to the lost data. Moreover, since SDP does not penalize big packets more than RED, its impact on loss rate due to dropping is zero[2]. To estimate the packet loss rate, we first calculate the dropping probability.

Big packets:

$$Db_R = red\_drop$$
$$Db_S = sdp\_drop = red\_drop$$
$$impact = Db_S - Db_R = 0 \quad (4)$$

Small packets:

$$Ds_R = red\_drop$$
$$Ds_S = sdp\_drop = \frac{ss}{x} \cdot red\_drop$$

[2] In fact, we have increased loss due to the increased queue length which we consider trivial

$$impact = Db_S - Db_R = red\_drop \cdot \left(\frac{ss}{x} - 1\right) < 0 \quad (5)$$

Equation (4) shows that SDP does not increase loss rate of big packets more than RED. Equation (5), on the other hand, shows that SDP decreases the PLR of small packets providing them with increased privileges. For small packets, the loss rate is a function of *ss* and *x*. Smaller values of *ss* and/or bigger *x* signify less proactive drops. The packet size is determined by the corresponding application, while *sdp_thresh* is calculated by the packet sizes that populate the queue. In general, the packet size is predefined; there are, however, cases that we may wish to adjust it dynamically for better service. As this is not uncommon, we demonstrate later how SDP can overcome such actions.

B. Queuing delay

We consider a router where big and small packets (noted as '*b*' and '*s*', respectively) have arrived. Some of them have been accepted and some others have been dropped. A packet arrives. Regardless of its size, this packet will experience some queuing delay. This queuing delay will depend on the dropping probabilities of the packets that arrived previously in the queue. For simplicity, we assume that the dropping probability is independent of the packet's position in the queue.

$$\begin{aligned}
Qd_R &= b \cdot bs \cdot T_{D1B} \cdot (1 - red\_drop) + s \cdot ss \cdot T_{D1B} \cdot (1 - red\_drop) = \\
&= T_{D1B} \cdot (b \cdot bs + s \cdot ss - red\_drop \cdot (b \cdot bs + s \cdot ss)) \\
Qd_S &= b \cdot bs \cdot T_{D1B} \cdot (1 - sdp\_drop) + s \cdot ss \cdot T_{D1B} \cdot (1 - sdp\_drop) = \\
&= T_{D1B} \cdot \left(b \cdot bs \cdot (1 - red\_drop) + s \cdot ss \cdot \left(1 - \frac{s}{x} \cdot red\_drop\right)\right) = \\
&= T_{D1B} \cdot \left(b \cdot bs + s \cdot ss - red\_drop \cdot \left(b \cdot bs + \frac{1}{x} \cdot s \cdot ss^2 \right)\right)
\end{aligned}$$

$$\begin{aligned}
impact &= Qd_S - Qd_R = T_{D1B} \cdot \\
&\left(b \cdot bs + s \cdot ss - red\_drop \cdot \left(b \cdot bs + \frac{1}{x} \cdot s \cdot ss^2 \right)\right) \\
&- T_{D1B} \cdot (b \cdot bs + s \cdot ss - red\_drop \cdot (b \cdot bs + s \cdot ss)) \Leftrightarrow
\end{aligned}$$

$$impact = T_{D1B} \cdot red\_drop \cdot ss \cdot s \cdot \left(1 - \frac{ss}{x}\right) > 0 \quad (6)$$

since *ss<x*.

Equation (6) demonstrates the impact on the queuing delay regardless of the packet size. All packets will experience increased delay since some small packet that would have otherwise been dropped from the queue, now contribute to delay cumulatively. Big flows, that generally use TCP, will respond to this delay and will decrease the rate with which they increase their sending windows. Moreover, big packets increase the risk to be dropped due to the increasing competition in the queue that might result in an average queue length more than the *maximum threshold.* On the other hand, whereas small packets experience also bigger delays, they have more chances to survive eventually from proactive dropping. Concluding, we prove that SDP can achieve Service Differentiation and that it adjusts its behavior to different packet sizes.

V. Developing Individual Strategies

SDP makes the assumption that small packet sizes correspond only to applications that require special service. However, users might try to fragment their bulk data into smaller pieces in order to promote themselves and gain from decreased dropping. We prove in this section using the basic principles of game theory that in SDP, the result of such an action depends on the behavior of the rest of the flows and that it is uncertain whether fragmentation is a winning or losing strategy.

We assume a bulk data application that sends packets of a specific size in a single router network in presence of other flows. After some time, only the aforementioned application changes its attitude and fragments its data into smaller packets.

We will use the variables cited in Table 3. When accentuated, they will refer to variables after fragmentation.

TABLE 3
Analysis variables

Name	Description
x	the *sdp_thresh* variable before fragmentation
x'	the *sdp_thresh* variable after fragmentation
ps	the total size of the application's packet before fragmentation
ps'	the total size of the application's packet after fragmentation
pl	the payload of the packet
od	the overhead of the packet
k	the fragmentation factor of the packet
ABL	an "average bytes lost" index which is the packet size of a packet multiplied by its dropping probability. If $ABL'/ABL > 1$, we lose from fragmentation, otherwise we win

Based on our previous assumptions we examine three main cases which can be concluded in Table 4. This table presents the possible cases from a single-user perspective, before and after fragmentation. For example, case (2) means that the user's packet size before fragmentation was bigger than *sdp_thresh*, whereas after fragmentation the new packet size is smaller than the new value of *sdp_thresh*.

TABLE 4
Possible outcome for a user, before and after fragmentation.

		Before fragmentation	
		$ps > x$	$ps < x$
After fragmentation	$ps' > x'$	(1)	(4)
	$ps' < x'$	(2)	(3)

The fourth case although objects to our assumptions, is possible in practical conditions and thus it will be examined separately.

1) $ps > x$, $ps' > x'$

In this first case, even though we fragment, the packet size is still bigger than *sdp_thresh*.

$$ABL = ps \cdot sdp\_drop = (pl + od) \cdot red\_drop$$

$$ABL' = k \cdot ps' \cdot sdp\_drop = k \cdot \left(\frac{pl}{k} + od\right) \cdot red\_drop =$$

$$= (pl + k \cdot od) \cdot red\_drop$$

$$\frac{ABL'}{ABL} = \frac{(pl + k \cdot od) \cdot red\_drop}{(pl + od) \cdot red\_drop} = \frac{pl + k \cdot od}{pl + od} > 1 \quad (7)$$

In this case, the more we fragment, the more we lose.

2) $ps > x$, $ps' < x'$

$$ABL = ps \cdot sdp\_drop = (pl + od) \cdot red\_drop$$

$$ABL' = k \cdot ps' \cdot sdp\_drop = k \cdot \left(\frac{y}{k} + od\right) \cdot \frac{\left(\frac{y}{k} + od\right)}{x'} \cdot$$

$$red\_drop = \frac{k}{x'} \cdot \left(\frac{y}{k} + od\right)^2 \cdot red\_drop$$

We remind that in order to win, we want $\frac{ABL'}{ABL} < 1$.

$$\frac{ABL'}{ABL} = \frac{\frac{k}{x'} \cdot \left(\frac{pl}{k} + od\right)^2 \cdot red\_drop}{(pl + od) \cdot red\_drop} = \frac{k \cdot \left(\frac{pl}{k} + od\right)^2}{x' \cdot (pl + od)} < 1 \Leftrightarrow$$

$$x' > \frac{k \cdot \left(\frac{pl}{k} + od\right)^2}{(pl + od)} \quad (8)$$

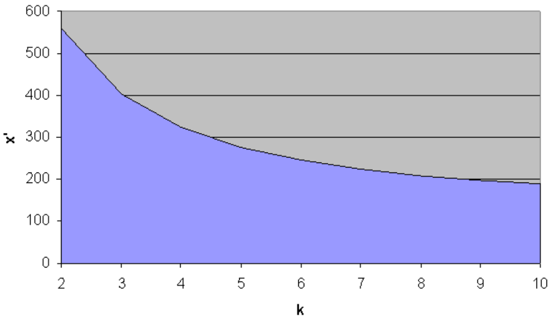

Fig. 3. x' as a function of the fragmentation factor.

For *pl*=1000B and *od*=40B, if *x'* lies in the blue area in Fig. 3 then we lose, else we win. For relatively small values of *k*, the above function is constantly decreasing. In this case, we increase our probabilities of winning by increasing *k* (that is the fragmentation), thus decreasing the packet size. However, for bigger values of *k*, the function has a negative peak (Fig. 4).

For given *pl* and *od* the lower peak is unique. This lower peak defines the point where *x'* has its lower value. At this point (k=25 in Fig. 4), we have the biggest possibility of winning from fragmentation. However, since we do not know the current value of x', x' may have any value. If x' is either in the blue or the red zone then we lose from fragmentation. The red zone defines the cases where the packet size is bigger than x', thus we fall back to the first case.

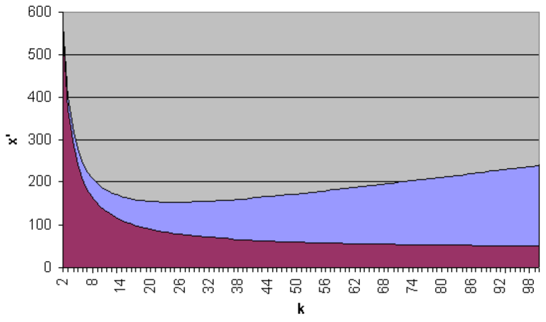

Fig. 4. x' and packet size as a function of the fragmentation factor.

3) $ps < x$, $ps' < x'$

$$ABL = ps \cdot sdp\_drop = (pl + od) \cdot \frac{(pl + od)}{x} \cdot red\_drop =$$

$$= \frac{1}{x} \cdot (pl + od)^2 \cdot red\_drop$$

$$ABL' = k \cdot ps' \cdot sdp\_drop = k \cdot \left(\frac{pl}{k} + od\right) \cdot \frac{\left(\frac{pl}{k} + od\right)}{x'} \cdot$$

$$red\_drop = \frac{k}{x'} \cdot \left(\frac{pl}{k} + od\right)^2 \cdot red\_drop$$

$$\frac{ABL'}{ABL} = \frac{\frac{k}{x'} \cdot \left(\frac{pl}{k} + od\right)^2 \cdot red\_drop}{\frac{1}{x} \cdot (pl + od)^2 \cdot red\_drop} = \frac{x}{x'} \cdot \frac{k \cdot \left(\frac{pl}{k} + od\right)^2}{(pl + od)^2} \quad (9)$$

In order to win we must have $\frac{x'}{x} > \frac{k \cdot \left(\frac{pl}{k} + od\right)^2}{(pl + od)^2}$ which is a similar function as in the second case. For pl=1000B and od=40B we get the graph in Fig. 5.

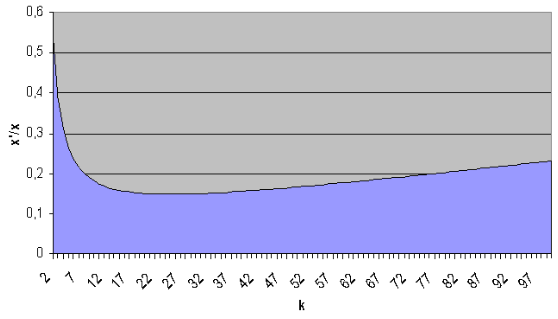

Fig. 5. x'/x as a function of the fragmentation factor.

Since we assumed that x' is modified only by one flow, we expect that x'/x is near 1. Hence in this case we can be sure that we always win, even though there is a specific packet size that we have the most benefits.

If we cancel the assumption that only one flow alters its stance, or in case that more flows enter the network, then the fourth case is possible.

4) $ps < x$, $ps' > x'$

$$ABL = ps \cdot sdp\_drop = (pl + od) \cdot \frac{(pl + od)}{x} \cdot red\_drop =$$

$$= \frac{1}{x} \cdot (pl + od)^2 \cdot red\_drop$$

$$ABL' = k \cdot ps' \cdot sdp\_drop = k \cdot \left(\frac{pl}{k} + od\right) \cdot red\_drop =$$

$$= (pl + k \cdot od) \cdot red\_drop$$

$$\frac{ABL'}{ABL} = \frac{(pl + k \cdot od) \cdot red\_drop}{\frac{1}{x} \cdot (pl + od)^2 \cdot red\_drop} = x \cdot \frac{(pl + k \cdot od)}{(pl + od)^2} < 1 \Leftrightarrow$$

$$\Leftrightarrow x < \frac{(pl + od)^2}{(pl + k \cdot od)} \quad (10)$$

which is impossible, since we supposed that $x > pl + od$. Thus, no matter what we do, we always lose.

We can summarize the previous analysis by saying that if $ps' > x'$ then we certainly lose, otherwise, we may either win or lose. The entire problem resembles the ‘prisoner’s dilema’; if only one user fragments its data he wins, while the other loses, if they both fragment their data, they both lose. We note that the loss is not due to the decreased packet size but because of the increased number of packets the user has to generate to maintain the same sending rate.

VI. Simulation Setup

A. AQM mechanisms setup

During the experimental evaluation we compare SDP with three other AQM mechanisms: DropTail, RED and NCQ+. We use the following sets of RED, NCQ+ and SDP parameters:

RED: The RED parameters are set according to the recommendation in [13]. That is, we use the “gentle” mode, the maximum threshold is set to three times the minimum threshold, and the minimum threshold is set to 1/8 of the buffer size. We use RED in byte-mode unless it is stated otherwise. The difference is that while classic RED drops randomly packets with the same probability regardless of their size, RED in byte-mode increases the probability as packet size increases.

NCQ+: NCQ+ parameters are set according to the recommendations in [20] and [23]. That is $ncqthresh_1$ is set to 0.05 and α is always equal to 0.1.

SDP: The weight factor α is set to 0.1.

Additionally, we measure the buffer space allocated for the queue in bytes when we use RED and SDP but we measure it in packets when we use NCQ+ and DropTail. The difference is that a queue measured in bytes might accept a small packet and drop a bigger when there is lack of buffer space, whereas a queue measured in packets will treat all the incoming packets equally regardless of their size. The reason for using two different ways of measuring the queue is that NCQ+ was originally analyzed and evaluated in [20] in packet mode. We follow this principle, and in order to ensure fairness during the evaluation, we compare the average values of the evaluation metrics (see Section 8).

B. Application setup

We simulate three types of traffic; FTP which consists of bulk data traffic and usually corresponds to big packet sizes, VoIP which consists of real-time traffic and small packets and Sensors which also consist of real-time traffic, however, their packet sizes are smaller than VoIP. The characteristics of each type of traffic are as follows:

FTP Traffic: FTP packets are carried by the TCP NewReno version. Packet size is always mentioned in each experiment.

VoIP Traffic: VoIP packets are carried by UDP. During a conversation, speakers alternate between activity and idle periods. Taking into consideration the ON and OFF periods [5], as well as the heavy-tailed characteristics and self similarity of VoIP traffic [10], we used the Pareto distribution for modeling the call holding times. We configure Pareto with a mean rate that corresponds to the transmission rate of 64kbps and the shape parameter is set to 1.5. In accordance with [5], we distribute the ON and OFF periods with means of 1.0sec and 1.35sec, respectively. We simulate VoIP streams of 64kbps (following the widely-used ITU-T G.711 [9] coding standard) and we set packet sizes at 160 bytes (each packet has 40-byte packet header).

Sensor Traffic: We simulate Sensor flows by sending periodically packets of 40 bytes (20 bytes of sensor data plus a 20-byte packet header) carried by UDP. The interval between two consecutive sensor transmissions is set to 50ms.

VII. Evaluation Metrics

Goodput: Goodput is used to measure the overall performance of the network in terms of effective bandwidth utilization.

$$Goodput = \frac{OriginalData}{TransmissionTime}$$

where *OriginalData* is the number of bytes delivered from a sender to the corresponding receiver during their connection (*TransmissionTime*), excluding the retransmitted data and the overhead induced by packet headers.

Meangoodput: Meangoodput is the average of the Goodput values of the individual flows.

$$Meangoodput = \frac{\sum_{i=1}^{n} Goodput_i}{n}$$

Lossrate: Lossrate is the number of lost packets divided by the total number of transmitted packets. Packets can be lost due to buffer overflows or proactive dropping.

$$Lossrate = \frac{NumberOfLostPackets}{NumberOfTransmittedPackets}$$

Meanlossrate: Meanlossrate is the average of all individual lossrates in the network.

$$Meanlossrate = \frac{\sum_{i=1}^{n} Lossrate_i}{n}$$

Fairness: We use the Chiu's fairness index [6], which captures the bandwidth allocation among competing flows.

$$Fairness = \frac{\left(\sum_{i-1}^{n} Throughput_i\right)^2}{n\sum_{i-1}^{n} Throughput_i^2}$$

Application Satisfaction Index: The Application Satisfaction Index (ASI), which was introduced in [21] by L. Mamatas and V. Tsaoussidis, captures the delay fair share per application on the basis of the delay impact of each application on others. It is defined as:

$$ASI = 1 - \frac{\sum_{i=1}^{n} \left| Delay_i - \frac{Data_i}{TotalData} Delay_{\max} \right|}{nDelay_{\max}}$$

where *n* is either the number of active nodes or the number of different traffic classes; *Datai* the total transmitted data of the ith node to the receiver; *TotalData* the total transmitted data of all nodes; *Delayi* the average queuing delay of the ith node and *Delaymax* the maximum queuing delay of the system.

R-Factor: We characterize the quality of voice communication using the R-Factor, which is included in the E-Model ([15], [16]), an ITU-proposed analytic model of voice quality. R-Factor captures voice quality and ranges from 100 to 0, representing best and worst quality respectively. R-Factor incorporates several different parameters, such as echo, background noise, signal loss, codec impairments and others. In [17], R-Factor is defined as:

$$R = a - \beta_1 d - \beta_2 (d - \beta_3) H (d - \beta_3) - \gamma_1 - \gamma_2 \ln(1 - \gamma_3 c)$$

where $\alpha = 94.2$, $\beta_1 = 0.024\text{ms}^{-1}$, $\beta_2 = 0.11\text{ms}^{-1}$, $\beta_3 = 177.3\text{ms}$, expresses the mouth-to-ear delay and the packet loss rate. For the G.711 codec, *d2e*1γ= 0, γ = 30, 3γ= 15.

VIII. Simulation Results

For the experimental evaluation we will use two cross-traffic topologies (Fig. 6 and 7) which incorporate two bottlenecks. The difference between these two topologies is that in the second topology, the VoIP data is transported via wireless links and we have an additional type of traffic; small FTP flows. This modification matches better the way VoIP is transferred over the Internet and allows us to have a more realistic scenario.

Throughout our experiments, we attempt to address four specific issues:

1) Prove that SDP succeeds to distinguish and subsequently

classify big from the various degrees of packet sizes and accomplish Service Differentiation (Scenario 1 - Scalability of SDP).

2) Evaluate the applicability of SDP for VoIP traffic (Scenario 2 - Impact on the VoIP load).

3) Show the impact of SDP on sensor-based applications (Scenario 3 - Impact on the sensor generated traffic).

4) Evaluate SDP in a more complex scenario, where different types of traffic, such as big FTP flows, small FTP flows, VoIP and sensor traffic, coexist in the same network. (Scenario 4 - Impact on various traffic classes).

5) Examine, in a simple topology, if SDP manages to increase the ASI [21] compared to RED (Scenario 5 - Impact on ASI).

We discuss each scenario with its corresponding results.

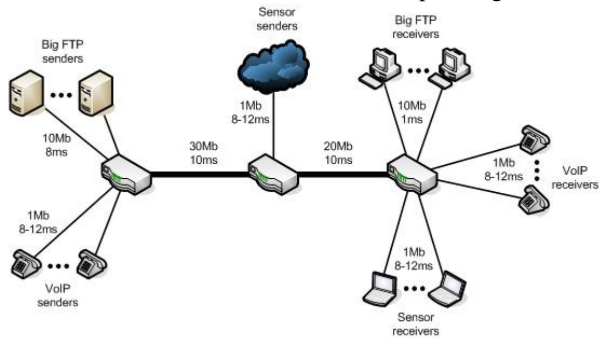

Fig. 6. Cross-traffic topology.

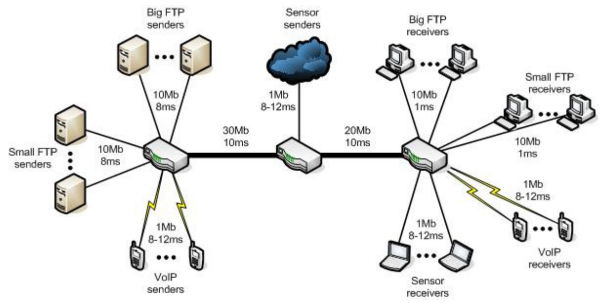

Fig. 7. Cross-traffic topology with wireless VoIP users.

1) *Scenario 1 - Scalability of SDP*

In this first scenario, we vary the total number of flows from 40 to 240, while maintaining constant percentages for the different types of flows. Specifically, 5% of the flows involve VoIP calls, 5% sensor data transmission, and the remaining 90% big FTP flows with 1KB packets. This scenario allows us to draw conclusions on the scalability of the proposed mechanisms in low and high contention of the link.

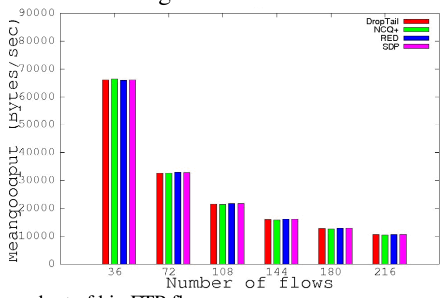

Fig. 8. Meangoodput of big FTP flows.

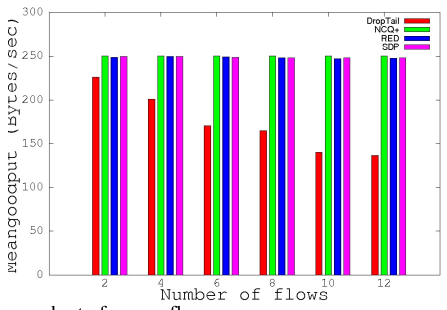

Fig. 9. Meangoodput of sensor flows.

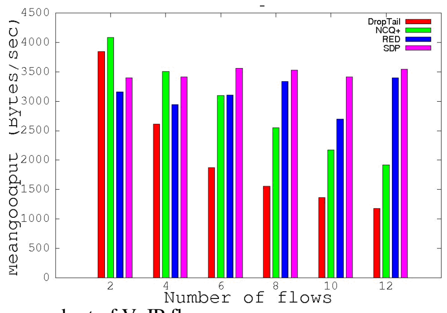

Fig. 10. Meangoodput of VoIP flows.

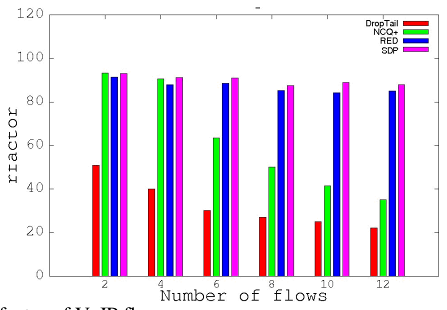

Fig. 11. R-factor of VoIP flows.

Fig. 8, 9 and 10 show that SDP succeeds to apply service differentiation among various flows better than DropTail, RED and NCQ+. While both RED and NCQ+ are capable of providing quality guarantees for sensor packets successfully (Fig. 9), only SDP increases the *meangoodput* of VoIP (Fig. 10), without affecting big FTP flows (Fig. 8). This increase signals a corresponding increase in the user-perceived quality, expressed by *R-factor* (Fig. 11). The most interesting point is that SDP has a consistent behavior as the number of contention increases; Fig. 9 and 10 show that as flows increase, the *meangoodput* for small packets remains the same. In fact, the same applies for *R-factor* (Fig. 11); the quality of voice remains the same although more calls are initiated. Fig. 10 exhibits an interesting illation; as long as the number of VoIP flows is small, NCQ+ strategy results in more Goodput than SDP. SDP manages to serve better VoIP flows when their number increases. We demonstrate that dropping- and scheduling-oriented LIBS are indeed complementary and that different network conditions impose different approaches.

2) *Scenario 2 - Impact on the VoIP load*

In this scenario we attempt to capture the impact of SDP on VoIP applications considering different levels of VoIP traffic. We set the total number of flows to 100. Sensor flows consist of the 6% of the total number of flows (6 flows) and VoIP flows vary from 6% to 36% (from 6 to 36 flows).

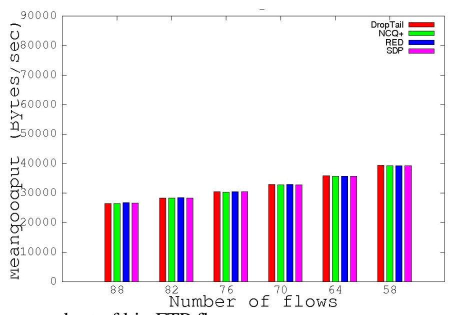

Fig. 12. Meangoodput of big FTP flows.

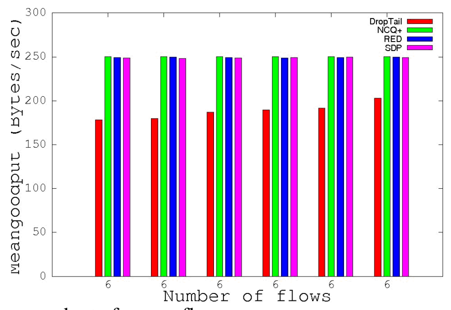

Fig. 13. Meangoodput of sensor flows.

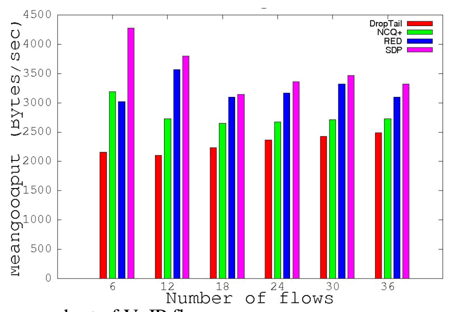

Fig. 14. Meangoodput of VoIP flows.

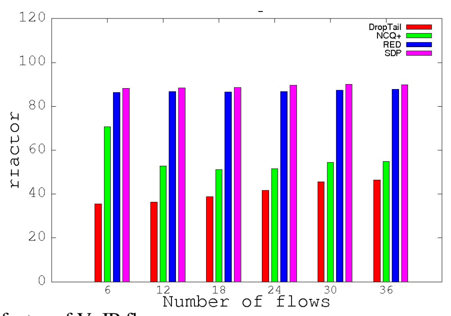

Fig. 15. R-factor of VoIP flows.

The results are similar to the results in the first scenario, however there are some differences. This time, we decrease the number of big FTP flows and increase the number of VoIP flows. Fig. 12 shows how all schemes allocate more bandwidth for each big flow. Fig. 13, also shows that NCQ+, RED and SDP provide the same quality guarantees for sensor traffic. However, Fig. 14 and 15 depict SDP's superiority over the other AQM schemes since SDP promotes VoIP packets more effectively and increases the R-factor. Nevertheless, we do not experience the same behavior as in the Scenario 1, as the number of VoIP flows becomes significant and comparable to the number of big flows. *Sdp_thresh* is decreased and the dropping probability for VoIP packets is increased.

3) *Scenario 3 - Impact on the sensor-generated traffic*

We repeat the previous scenario, only now we vary the number of sensor flows. Once again we set the total number of flows to 100, the number of VoIP flows to 6% of the total number of flows and we vary the sensor flows from 6% to 36%. We study the impact on sensor-generated traffic.

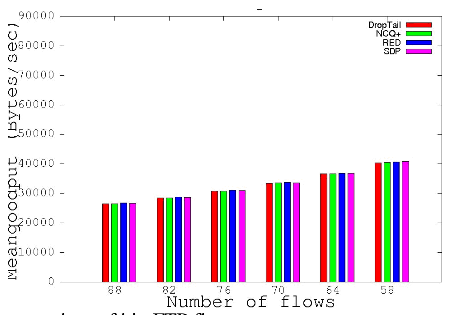

Fig. 16. Meangoodput of big FTP flows.

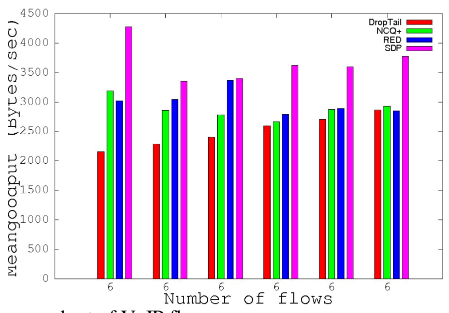

Fig. 17. Meangoodput of VoIP flows.

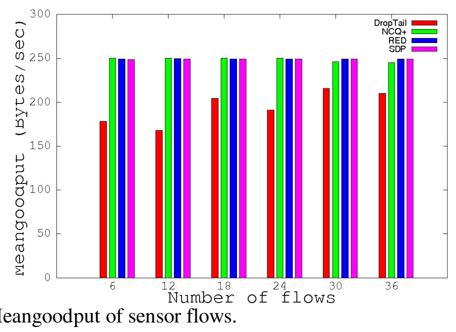

Fig. 18. Meangoodput of sensor flows.

Apart from Droptail, all AQM schemes provide almost equal service to sensor packets (see Fig. 18). However, SDP also favors VoIP packets as more packets arrive successfully at the receivers (see Fig. 17). Remarkably, this superiority of SDP is not at the cost of big FTP flows (see Fig. 16), but it comes as a result of better packet classification.

4) *Scenario 4 - Impact on various traffic classes*

In this last scenario we will use the topology of Fig. 18, with all two types of FTP flows, big FTP (1KB packet sizes) and small FTP (500B packet sizes). We increase the total number of flows in the network and study the performance of each type. 10% of the total number of flows is VoIP traffic, 10% is sensor traffic, and each of the small and big FTP flows consist of the 40% of the total traffic. This scenario is more close to real-life networks where different types of applications compete for some limited common resources.

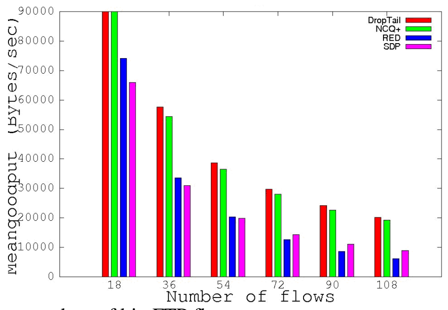

Fig. 19. Meangoodput of big FTP flows.

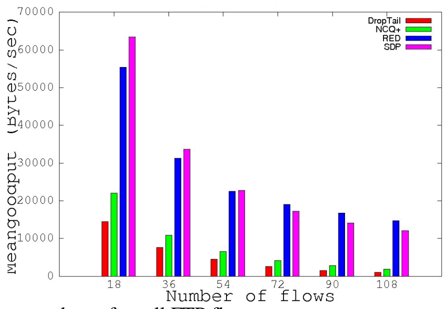

Fig. 20. Meangoodput of small FTP flows.

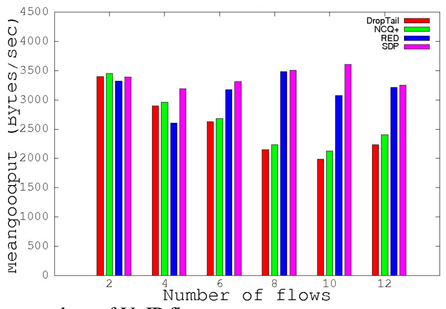

Fig. 21. Meangoodput of VoIP flows.

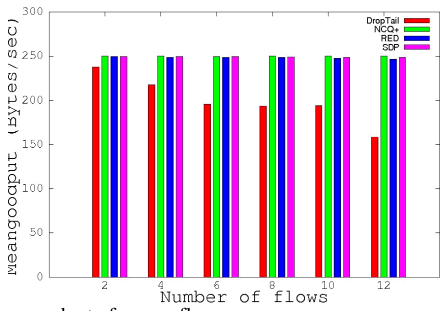

Fig. 22. Meangoodput of sensor flows.

This last scenario proves not only that SDP is scalable but also adaptive. SDP does not allocate blindly buffer space to small FTP flows; as their number increases *sdp_thresh* decreases and they tend to get equal priority to big FTP flows. The bandwidth allocated for VoIP and sensor traffic is the same, since their contribution to the total contention remains trivial all the times. While NCQ+ and RED may have similar results to SDP in Fig. 22, SDP achieves a better and fairer distribution of bandwidth, leading to a more effective AQM approach.

5) *Scenario 5 - Impact on ASI*

In [21], the ASI depicted NCQ functionality effectively and reliably, as NCQ aims to minimize the delay of non-congestive traffic. SDP however, adjusts dropping probability that generally does not affect queuing delay, thus we do not expect a significant ASI increase. We conducted an experiment on a dumbbell topology (Fig. 24). The number of the flows varies from 100 to 500 and small flows consist the 10% of the total flows. Small flows consist 100B packets and big flows 1kB.

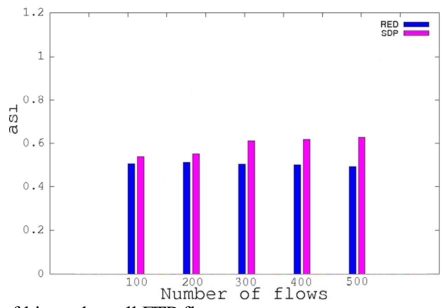

Fig. 23. ASI of big and small FTP flows.

The results of the simulation are depicted in Fig. 23. Even though SDP is designed to decrease the queuing delay of packets, it achieves to increase ASI slightly, compared with RED, due to the fact that it increases small FTP flows satisfaction by assigning less dropping probability.

IX. Alternative Strategies On Packet Fragmentation

Assuming that a LIBS-based mechanism is deployed, an application may follow alternative strategies in order to have performance gains. We use a simple dumbbell topology as shown in Fig. 24 and we experiment with FTP traffics that use different packet sizes. We consider 100 FTP applications and two distinct types of classes, one that utilizes big packet sizes (1KB) and the other that utilizes small packet sizes (100B) or medium packet sizes (500B). In the first set of our experiments, we assign the transmission of 5MB data to each of the flow and calculate the average completion time of tasks. Next, we implement experiments where FTP flows have unlimited data to send and study how the available bandwidth is shared among flows that use different packet sizes. In both sets of experiments, we adjust the percentage of big and small flows to study the scalability of the proposed mechanism. We assume that each packet has 40 bytes overhead.

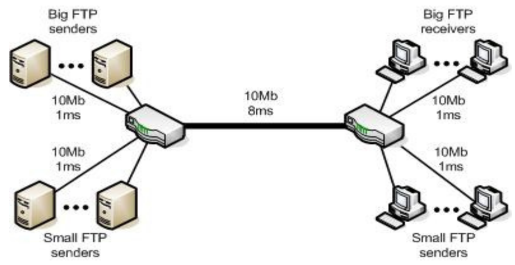

Fig. 24. A dumbbell topology with FTP flows that generate big and small packets.

A. *Fixed Data Transmission*

In this scenario, we compare SDP with RED in byte mode. They both attempt to distinguish small packets and minimize their chances to be dropped. However, RED in byte mode has two functions i) it increases the probability of a big packet to be dropped ii) it decreases the probability of a small packet to be dropped. The size is characterized as "small" or "big" compared to the *meanpacketsize* parameter which is fixed. Packets whose size is equal to *meanpacketsize* are dropped with the probability calculated by the original RED algorithm

We will study two cases: (1) Small FTP flows that use 100B packets and (2) small FTP flows that use 500B packets.

1) *Small FTP flows that use 100B packets.*

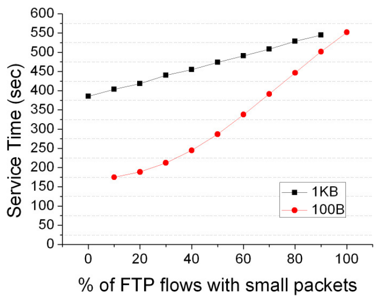

Fig. 25. Service Time with RED.

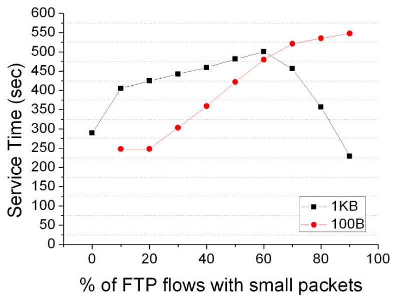

Fig. 26. Service Time with SDP.

As we can see in Fig. 25 and 26 when the percentage of small flows is less than the percentage of big flows, small flows benefit more from the RED mechanism than from SDP and finish their tasks earlier. Similar things apply for big flows. As more flows change their strategy (70-100%) and attempt to benefit from the SDP mechanism, SDP seems to benefit big flows rather than the small ones. This is due to the dynamic classification of SDP, which takes into consideration the proportion of big and small packets in the queue and treats them the same if many small packets populate the queue. On the contrary, RED keeps dropping big packets with higher probability than small packets, although small packets' contribution to congestion is significant.

This trade-off resembles a game theory problem. If all FTP flows play fair the game and use 1KB packets then they will all finish their tasks in 298 sec. If a small amount of them try to cheat they will benefit but as their number is increased, they will all lose and finish their tasks in more than 298 sec. Moreover, when the percentage of misbehaving flows exceeds the one that play the game fairly, not only will they extend the time to finish their task but, beyond that, will also boost the performance of fair flows that now finish their tasks in less than 298 sec.

2) *Small FTP flows that use 500B packets*

We implement the same set of experiments as before, however small flows have 500B packet size rather than 100B.

Fig. 27 and 28 show that even with 500B packets, small flows are "punished"; it takes more time for then to finish their tasks with SDP rather than with RED. Moreover, as the number of small flows is increased significantly, SDP treats better big flows and their service time becomes stable. Fig. 5 shows that in presence of RED, small flows are getting better service, while the service time of big flows is worse and is increased as more flows decide to change their strategy.

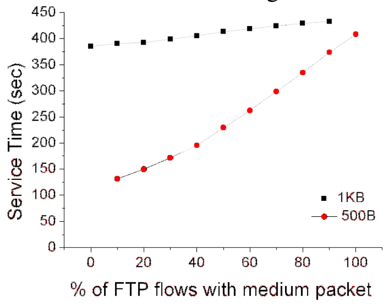

Fig. 27. Service Time with RED.

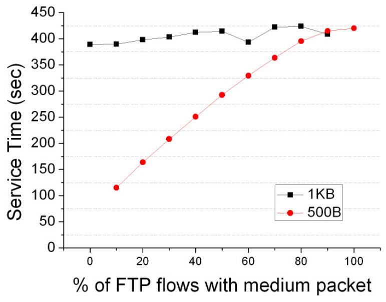

Fig. 28. Service Time with SDP.

B. *Fixed Time Transmission*

Next, we use the same topology as before, however now flows have always data to send. We attempt to study how the available bandwidth is allocated among competing flows. We aim to exploit as well as possible the available resources, as well as to achieve a fair bandwidth distribution.

At first, we consider big flows that generate 1KB packet sizes and small flows that generate 100B packet sizes.

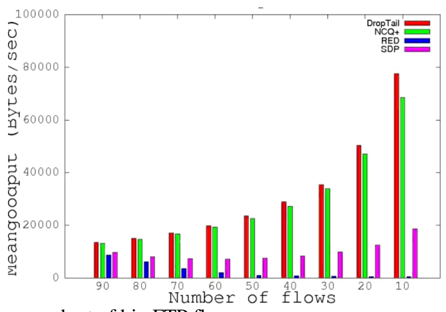

Fig. 29. Meangoodput of big FTP flows.

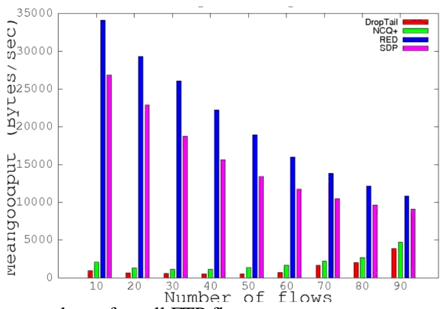

Fig. 30. Meangoodput of small FTP flows.

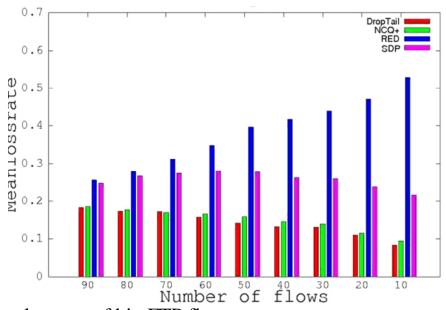

Fig. 31. Meanlossrate of big FTP flows.

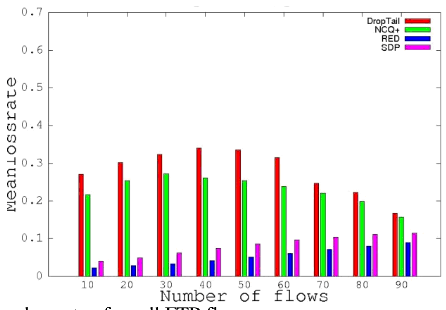

Fig. 32. Meanlossrate of small FTP flows.

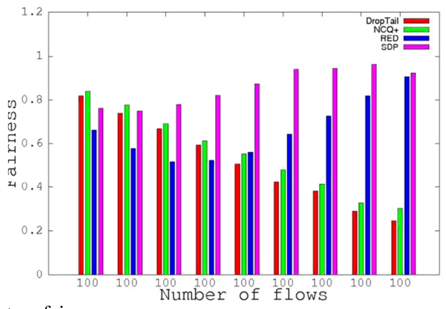

Fig. 33. System fairness.

Fig. 29 and 30 show how SDP reallocates bandwidth among flows in order to achieve a fair distribution. Small flows increase their transmission, while big flows back off. Fig. 31 and 32 depict that by decreasing dropping on small flows, meanlossrate is decreased and thus fairness is increased (Fig. 33). While SDP achieves a good reallocation of network resources, NCQ+ promotes less small packets than necessary while RED promotes more packets; both increasing inequalities among different packet sizes. The only exception when SDP decreases fairness is when we have little number of small flows. This is the only case that we tolerate unfairness as we would like to ascertain that when we have few small packets, they have maximum priority and the best treatment possible. Next, we consider big flows that generate 1KB packet sizes but small flows that generate 500B packet sizes.

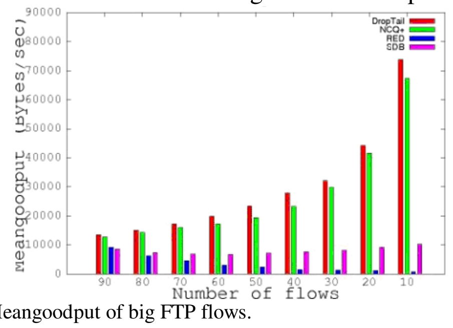

Fig. 34. Meangoodput of big FTP flows.

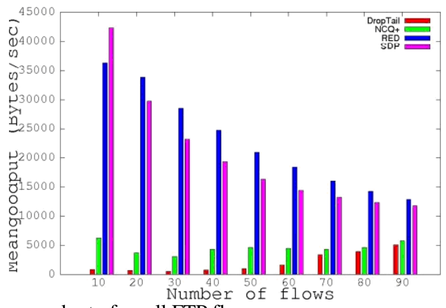

Fig. 35. Meangoodput of small FTP flows.

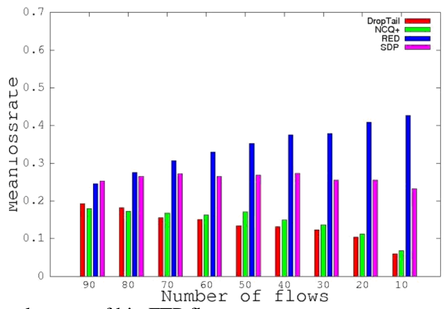

Fig. 36. Meanlossrate of big FTP flows.

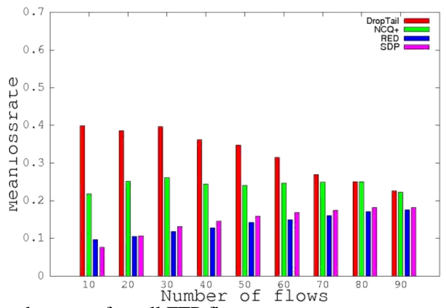

Fig. 37. Meanlossrate of small FTP flows.

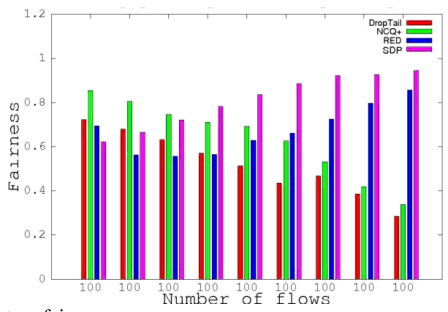

Fig. 38. System fairness.

X. Conclusion And Future Work

We proposed a new Service Differentiation scheme, based on packet dropping, which classifies packets according to their size. Due to its self-adaptable structure, the proposed mechanism does not drop packets blindly; instead packet dropping probability depends on the comparative packet sizes in the queue. During the experimental evaluation we showed that SDP: (i) manages to increase Goodput and link utilization, (ii) achieves a better system Fairness compared to other proposals and (iii) increases the perceived quality on real-time applications. Despite SDP efficiency, its algorithm is light-weight, fast and does not require memory-consuming procedures or large reference-tables.

Although packet dropping proves to be effective for both time-sensitive and delay-tolerant applications, there are cases where a packets need to be promoted through scheduling as well. Future extension of the algorithm will incorporate a time-scheduling mechanism, similar to the NCQ+ principle, that allows special priority for selected packets that require small delivery times. We plan to study the effects on delay, jitter and perceived quality.

References

[1] BitTorrent, http://www.bittorent.com

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “RFC2475 - An Architecture for Differentiated Services”, December 1998.

[3] U. Bodin, O. Schelen, and S. Pink, “Load-tolerant Differentiation with Active Queue Management”, *ACM SIGCOMM Computer Communication Review*, Volume 30, Issue 3, July 2000.

[4] R. Braden, D. Clark, and S. Shenker, “RFC2475 - Integrated Services in the Internet Architecture: an Overview”, June 1994.

[5] P. Brady, “A Statistical Analysis of On-Off Patterns in 16 Conversations”, *The Bell System Technical Journal*, 47 73-91, 1968.

[6] D.-M. Chiu and R. Jain. “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks”, *Computer Networks*, ISDN Syst., 17(1):1-14, 1989.

[7] D. Clark and W. Fang, “Explicit Allocation of Best-Effort Packet Delivery Service”, *IEEE/ACM Transactions on Networking*, August 1998.

[8] S. Cnodder, O. Elloumi, and K. Pauwels, “Effect of Different Packet Sizes on RED Performance”, *Proceedings of ISCC 2000*, July 2000.

[9] R. Cole and J. Rosenluth, “Voice over IP Performance Monitoring”, *ACM SIGCOMM Computer Communications Review*, 31 (2) (2001) 9-24.

[10] T. Dang, B. Sonkoly, and S. Molnar, “Fractal Analysis and Modelling of VoIP Traffic”, *Proceedings of International Telecommunications Network Strategy and Planning Symposium 2004*, June 2004.

[11] A. Demers, S. Keshav, and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm”, *Proceedings of SIGCOMM 1989*, September 1989.

[12] S. Dimitriou and V. Tsaoussidis, “A New Service Differentiation Scheme: Size Based Treatment”, *Proceedings of ICT 2008*, June 2008.

[13] S. Floyd, “RED: Discussions of Setting Parameters”, November 1997.

[14] S. Floyd and K. Fall, “Promoting the use of end-to-end congestion control in the Internet”, *IEEE/ACM Transactions on Networking*, May 1999.

[15] ITU-T Recommendation G.107, “The E-Model, a Computational Model for Use in Transmission Planning”, December 1998.

[16] ITU-T Recommendation G.113, “General Characteristics of General Telephone Connections and Telephone Circuits - Transmission Impairments”, February 1996.

[17] ITU-T Recommendation G.711, “Pulse Code Modulation (PCM) of Voice Frequencies”, November 1988.

[18] D. Lin, and R. Morris, “Dynamics of Random Early Detection”, *Proceedings of SIGCOMM 1997*, September 1997.

[19] R. Mahajan, and S. Floyd, “Controlling High Bandwidth Flows at the Congested Router”, *Proceedings of ICNP 2001*, November 2001.

[20] L. Mamatas, and V. Tsaoussidis, “A new approach to Service Differentiation: Non-Congestive Queueing”, *Proceedings of CONWIN 2005*, July 2005.

[21] L. Mamatas and V. Tsaoussidis, “Differentiating Services with Non-Congestive Queuing (NCQ)”, *IEEE Transactions on Computers*, 2009

[22] R. Pan, B. Prabhakar, and K. Psounis, “CHOKe: a stateless AQM scheme for approximating fair bandwidth allocation”, *Proceedings of INFOCOM 2000*, March 2000.

[23] G. Papastergiou, C. Georgiou, L. Mamatas and V. Tsaoussidis, “On Short Packets First: A delay-oriented prioritization policy”, *Technical Report TR: DUTH-EE-2008-8*

[24] I. Stoica, S. Shenker, and H. Zhang, “Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks”, *IEEE/ACM Transactions on Networking*, February 2003.

[25] V. Tsaoussidis and C. Zhang, “The Dynamics of Responsiveness and Smoothness in Heterogeneous Networks”, *IEEE Journal on Selected Areas in Communications*, June 2005.

Stylianos Dimitriou (M’05) received his Diploma of Electrical and Computer Engineering from Democritus University of Thrace, Greece in 2006. He is currently a PhD candidate in Electrical and Computer Engineering in the same institution. His work includes AQM schemes, QoS and Delay-Tolerant Networking. Up to 2008, he has published 4 papers and he was awarded for his diploma dissertation with the Ericsson Award of Excellence in Telecommunications

Ageliki Tsioliaridou received her Diploma of Electrical and Computer Engineering from Democritus University of Thrace, Greece in 2005. She is currently a PhD candidate in Electrical and Computer Engineering in the same institution. Her research interests include congestion control in packet networks, smooth data transmission algorithms and AQM based algorithms. She participated in the Organizing Committee of WWIC 2005

Vassilis Tsaoussidis (M’96-SM’03) has a B.Sc in Applied Mathematics from Aristotle University, Greece and a Ph.D in Computer Networks from Humboldt University, Berlin, Germany (1995). He held appointments, and a faculty appointment at the Computer Science Department of SUNY Stony Brook (until 2000) and at the college of Computer Science of Northeastern University, Boston (until 2003). He returned to Greece in May 2003 to join the Faculty of the Department of Electrical and Computer Engineering of Democritus University, where he is now full Professor. He also held appointments as Visiting Professor at TU Berlin and MIT Boston

Prof Tsaoussidis was/is editor for IEEE Transactions on Mobile Computing, the Journal of Computer Networks the Journal of Wireless Communications and Mobile Computing the Journal of Mobile Multimedia and the International Journal of Parallel, Emergent and Distributed Systems. He participates(d) in several Technical Program Committees in his area of expertise, such as INFOCOM, GLOBECOM, ICCN, ISCC, EWCN, WLN, and several others.

Vassilis graduated 5 Ph.D students and around 30 Masters.