Weighted Fair Early Packet Discard at an ATM Switch Output Port

András Rácz
Traffic Analysis and Network Performance Lab., Ericsson Kft.,
H-1300 Budapest 3. P.O.Box 107, Hungary, Andras.Racz@lt.eth.ericsson.se
Gábor Fodor
Mobile Networks and Systems Research, Ericsson Radio Systems,
Torshamnsgatan 23, Kista, SE-164 80 Stockholm, Gabor.Fodor@era-t.ericsson.se
Zoltán Turányi
Traffic Analysis and Network Performance Lab., Ericsson Kft.,
H-1300 Budapest 3. P.O.Box 107, Hungary, Zoltan.Turanyi@lt.eth.ericsson.se

Abstract— **In this paper we consider an output port of an ATM switch, where cell streams belonging to different TCP and UDP sessions arrive. During congestion the switch applies** *Early Packet Discard* **in order to reduce bandwidth waste. In order to guarantee fairness in the sense that only misbehaving sources get affected by the packet drop mechanism and to ensure high server utilization we propose an algorithm which also features simplicity. A salient feature of this algorithm is that it allows a predefined share α_i to be associated with stream i. The algorithm (which we call the Weighted Fair EPD, WFEPD) attempts to provide this weighted share of the bandwidth for the streams in the long time average. A sliding window implementation of WFEPD allows us to demonstrate its efficiency in terms of fairness and bandwidth utilization.**

I. Introduction

It has been observed by many papers that in layered packet data networks, where longer packets of a higher layer of the protocol stack get segmented and carried by shorter packets of a lower layer, the so called early packet discard (EPD) mechanism can be efficient in saving bandwidth [1], [2], [4], [5], [6]. In particular, when ATM is used as a transport mechanism to carry IP packets, EPD exploits the fact that once an ATM cell belonging to an IP packet is discarded, all other ATM cells which belong to the same IP packet will be discarded at the destination node, because the IP packet cannot be reassembled.

In this paper we propose a simple algorithm for the implementation of the EPD scheme, which takes into account *fairness* between the IP streams which arrive at an output port of an output-queued ATM switch. The algorithm aims at fairness and high bandwidth utilization.

Section II gives an overview of related work. Sections III-V describe the details of the WFEPD algorithm. Finally, simulation results are presented in Section VI.

II. Related work

In the context of IP over ATM in general and IP over the Available Bit Rate (ABR) and Unspecified Bit Rate (UBR) traffic classes in particular, the issue of fairness and high bandwidth utilization has recently gained much attention. However, much fewer work provides simultaneously a predefined share for the individual IP streams (irrespectively whether the layer above is TCP or UDP), maintains high bandwidth utilization and suggests a sliding window implementation (as opposed to the STOP-and-GO type of implementation). Our goal here is to devise and implement an algorithm which possesses these three features.

The basic EPD scheme was first studied by [1], [2], [3], and it has been found that EPD leads to significant performance improvement of TCP connections. This improvement is mainly due to the fact that EPD discards cells selectively, thereby avoiding the transmission of corrupted IP packets. Since in EPD the packet discards are essentially random, several papers suggested improvements to EPD in order to provide fairness among different traffic types. The FRED mechanism proposed by [5] categorizes sources as non-adaptive, fragile and robust according to their responses to congestion. The motivation for doing so is the key observation that EPD tends to punish cooperative sources (i.e. ones reacting to network congestion) but not non-cooperative ones. This has also been pointed out by e.g. [6] and [12], and significant improvements have been made adopting the EPD to the UBR and ABR service categories, however no explicit quantification of the fair share can be made in these algorithms, which is then kept irrespectively of the layer above IP. Selective packet discard (ESPD) and its further refinement have been proposed by [13] and [14], but also here the investigation focuses on TCP. The ESPD is based on the observation that the random packet discard in EPD "spreads" packet losses over many sessions, which, in turn triggers time-outs across most TCP sessions and causes TCP synchronization. ESPD combats this problem by forcing selected sources to shrink their window sizes, however it seems uncertain how this algorithm behaves in a mixed TCP/UDP environment. The Weighted Fair Early Packet Discard (WFEPD) algorithm we propose here attempts to provide pre-defined, explicit shares for all cooperative (TCP) and non-cooperative (UDP) sources while maintaining high bandwidth utilization.

III. The Weighted Fair Early Packet Discard Mechanism

Consider a FIFO queue with capacity R [byte/s], where ATM cells of N IP streams arrive and there is a *fair share* $\alpha_i > 0$ associated with each stream for which : $\sum_{i=1}^{N} \alpha_i \leq 1$. It is the objective of the WFEPD algorithm to provide on the long

term average for each IP stream at least $\alpha_i * R$ server capacity irrespectively whether the protocol layer over the IP layer is TCP or UDP. To ensure high server utilization even in the case when some of the sources underutilize their fair shares, the algorithm proportionally redistributes the server capacity left over by underutilized connections among those that are temporarily sending more than their respective fair shares.

The WFEPD algorithm relies on a per-flow state information in that it keeps a record of the bandwidth usage of each individual IP stream. The WFEPD uses this information to compare the actual bandwidth usage (and current server load) with that determined by the fair share of that IP stream. In case of congestion, the ATM cells of *violating IP streams* will be discarded, thus protecting conforming IP streams. Accordingly, three distinct parts of the proposed algorithms can be distinguished:

1. the procedure for the continuous measurement of the rate at which cells are accepted from the individual flows
2. the algorithm that calculates the available bandwidth for each of the flows based on the α_i share of the flow, the current rate of the flow and the current server load. It also determines which of the flows are violating.
3. the mechanism that identifies critical situations when violating sources are getting to overload the server and some interaction is necessary (cell discard) in order to protect conforming streams

IV. Rate measurement

In WFEPD, the switch maintains a record of the accepted traffic from each stream. In the simplest case this can be the accepted number of cells, n_i or bytes b_i, during some period T. Thus, during period T the total number of accepted bytes from the N sources is $\sum_{i=1}^{N} b_i$. The average rate of the individual flows can be easily calculated as $r_i^{av} = \frac{b_i}{T}\ [byte/sec]$ and when the sum of the average rates exceeds the server capacity $\sum_{i=1}^{N} r_i^{av} > R$ we can be sure that some of the sources violated their shares at least during the period T. If this situation is persistent and the buffer management does not intervene, the buffer will overflow and violating sources can grab an unfair proportion of the server capacity. Of course, other types of average rate estimation algorithms than the one above are also applicable. In the following we propose and compare two methods from applicability point of view. The difference between the two measurement algorithms is how the average is estimated, or in other words what the characteristic of the averaging filter is.

A. Sliding window

The first method applies a rectangle window of length T which is shifted forward with equal δ intervals (Figure 2), $T = n * \delta$. The algorithm maintains n counters per each flow $c_{i,j}$ which stores the accepted number of bytes in the i^{th} δ interval form the j^{th} connection. We store only the last n of the $c_{i,j}$'s for each j, that is from $(c_{i,j},\ c_{i-1,j}, \ldots, c_{i-(n-1),j})$. The algorithm maintains an additional counter $(C_j(t = k * \delta))$ per each flow that stores the total number of accepted bytes in the past $[t - T, t]$ interval from the j^{th} connection. The $C_j(t)$ counters are then updated according to the following recursive expression:

$$C_j((k+1)\delta) = C_j(k\delta) - c_{k-n,j} + c_{k+1,j} \qquad (1)$$

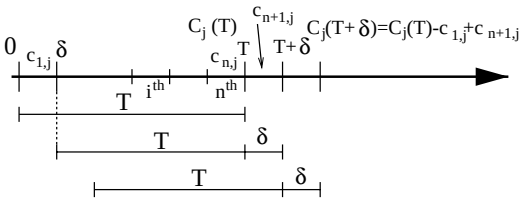

Fig. 2. Sliding window with fixed δ shifting

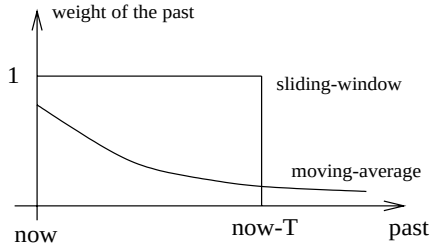

Fig. 3. Filter characteristics

We note that an alternative solution that results in a more sophisticated resolution of the measurement intervals is to shift the window forward at each cell arrival and departure. However, this version of the algorithm would require a timer update at each cell arrival and departure and is therefore less realistic. The moving average rate measurement method, which is discussed in the next subsection, simplifies the implementation of the sliding window mechanism.

B. Moving average

This measurement method requires only two variables per each flow. One of them stores the measured average rate (r_i^{av}) and the other counts the number of accepted bytes since the last update (c_i^{δ}). The average is updated at regular time intervals (δ) with the following expression. (Of course the c_i^{δ} counters are reset at the and of each δ period.)

$$r_i^{av} = q * r_i^{av} + (1-q) * \frac{c_i^{\delta}}{\delta}, \qquad 0 < q \leq 1 \qquad (2)$$

Thus the formula gives the weighted sum of the measured average rate up to now and the rate of the last δ period. The weight of the present in the sum decays with an exponent of q. In contrast to that, the sliding window mechanism has memory about the past with equal weight in the last T period and it forgets everything that happened before T. The difference between the sliding window and the moving average algorithms is the knowledge what they have about the past, see Figure 3.

Note that in case of both rate measurement methods only those cells are counted that have been admitted into the buffer. The discarded cells are not included in the measured rate of the flow. This guarantees that a flow sending large packets will not suffer unfairness against another flow that sends smaller packets, as the algorithm measures the actual rate at which the flow got served which is independent of the packet size.

V. IDENTIFYING VIOLATING AND PENDING SOURCES

A. The basic algorithm

The second part of WFEPD is concerned with distinguishing *violating* and *non-violating* sources. The WFEPD algorithm identifies non-violating and violating sources such that the non-violating ones will get the whatever bandwidth (i.e. number of cells) they had offered for the queue, whereas the violating sources will not get the demanded bandwidth, since the server must drop the excess cells. Thus we consider a source to be *violating* if its offered number of cells is more than what the fairness criteria, to be defined later, allows for it. It might happen that a source exceeds its equal share ($\alpha_i * R * T$) but it is still not *violating* according to the algorithm. This can happen only if there are other sources who have not utilized their shares completely. This insures high server utilization.

The fairness criteria is defined as follows. Let the set of all connections be denoted by Φ and we define another set (Θ) which is initially zero. The Θ set will include those connections whose offered traffic is entirely accepted i.e. they are not violating. At the end of the algorithm the ($\Phi - \Theta$) set will consist of only violating flows. We denote with R_Θ the sum of the average rates of the flows in the Θ set ($R_\Theta = \sum_{i \in \Theta} r_i^{(av)}$). At each step we distribute the capacity ($R - R_\Theta$) among the flows in the ($\Phi - \Theta$) set proportionally to their fair shares (α_i). Those flows that we have been assigned more bandwidth than their average rate will be moved to the Θ set. This is repeated until we can not move any more connections from ($\Phi - \Theta$) to Θ.

Thus the algorithm ensures that sources acquire bandwidth proportionally to their original shares. If some of the flows underutilize their respective fair shares, the remaining bandwidth is again distributed to those flows that require more bandwidth.

Then the WFEPD algorithm works according to the description below:

- An ordered list of the IP streams is determined such that $Stream_1 > Stream_2 > ... > Stream_N$, if and only if: $r_1^{av}/\alpha_1 \geq r_2^{av}/\alpha_2 \geq ... \geq r_N^{av}/\alpha_N$. That is, the sources are ordered according to their respective (relative) measured average rates.
- The *overflow* rate, ψ_T is defined as (used even in [14]):

$$\psi_T := \sum_{i=1}^{N} r_i^{av} - R \tag{3}$$

That is, if $\psi_T \geq 0$ then it gives the difference between the sum of the measured average incoming rate of the flows and the total server capacity. This means that on the average the total incoming rate is higher than the rate at which the server can give service.

- Suppose that the first $\omega - 1$ sources are violating, (and sources from $(\omega), ..., N$ are non-violating) where ω is to be determined later. Then the total bandwidth for the violating sources is given by:

$$\begin{aligned} R^{(v)} &= R - \sum_{i=\omega}^{N} r_i^{(av)} = \\ &= \sum_{i=1}^{\omega-1} r_i^{(av)} + \sum_{i=\omega}^{N} r_i^{(av)} - \psi_T - \sum_{i=\omega}^{N} r_i = \\ &= \sum_{i=1}^{\omega-1} r_i^{(av)} - \psi_T \end{aligned} \tag{4}$$

- Next, the WFEPD algorithm has it as an objective that the violating sources obtain the same share (between themselves) of the bandwidth which is available for them (see above) as their original share was :

$$r_i^{(sch)} = R^{(v)} * \frac{\alpha_i}{\sum_{j=1}^{\omega-1} \alpha_j} < r_i^{av} \qquad i = 1, ..., \omega - 1 \tag{5}$$

where $r_i^{(sch)}$ denotes the average rate at which the WFEPD buffer will actually accept cells from (the violating) source i.

- The WFEPD algorithm identifies *violating* sources according to the following. It looks for the smallest such ω which fulfills (6). This smallest such integer is denoted by ω_{min} and ω_{min} is the first non-violating source. That is the sources $((\omega_{min}), ..., N)$ are non-violating and the first $(\omega_{min} - 1)$ sources are violating.

$$r_\omega^{(av)} \leq (\sum_{i=1}^{\omega} r_i - \psi_T) * \frac{\alpha_\omega}{\sum_{i=1}^{\omega} \alpha_i} \tag{6}$$

- Thus, the first $\omega_{min} - 1$ sources will get

$$r_i^{(sch)} = \frac{\alpha_i}{\sum_{i=j}^{\omega_{min}-1} \alpha_j} * (\sum_{j=1}^{\omega_{min}-1} r_j^{(av)} - \psi_T) \tag{7}$$

rates assigned, while the rest of the sources $(\omega_{min}, ..., N)$ will get what they offered ($r_i^{(sch)} = r_i^{(av)}$).

The WFEPD algorithm assures that the violating IP streams (among themselves) get their original share, and that the non-violating IP streams do not receive higher (weighted) rate than any violating IP stream. Furthermore, the WFEPD algorithm utilizes the entire server capacity and it takes into account the original weighted shares assigned to the IP stream.

B. Identifying pending sources

While it is sufficient to identify violating and non-violating sources in the above fashion, [2] suggests that a soft transition between the "no-dropping" state and the "full-dropping" state is advantageous. Therefore, also the WFEPD algorithm applies probabilistic dropping for those flows that are in a predefined interval around their calculated shares. The motivation is similar to [2] where the RED algorithm also proposes probabilistic dropping between the minimum and the maximum queue thresholds. This extension requires the definition of a *bottom threshold* (th_b) and a *top threshold* (th_t). We define *pending* sources as those flows whose measured average rate is between the *bottom* and *top* thresholds around their calculated shares ($r_i^{(sch)} * th_b \leq r_i^{(av)} \leq r_i^{(sch)} * th_t$, $0 < th_b < 1$, $th_t \geq 1$). Note that in WFEPD, the same algorithm can be applied to identify pending sources as the one for identifying *violating* sources. That is we have the following steps:

- First we have to identify the violating sources with the algorithm discussed above.
- Next the same algorithm is applied again with the exception of equation (6): The algorithm looks for the smallest such ν which

fulfills (8). This smallest integer is denoted by ν_{min} and ν_{min} is the first pending flow:

$$r_\nu^{(av)} > (\sum_{i=1}^{\nu} r_i^{(av)} - \psi_T) * \frac{\alpha_\nu}{\sum_{i=1}^{\nu} \alpha_i} * th_b \quad (8)$$

- Next we look for the largest integer (ν_{max}) which satisfies (9):

$$r_\nu^{(av)} \leq th_t * r_\nu^{(sch)} \quad (9)$$

and ν_{max} is the last pending source.

Thus the sources from $\nu_{min} ... \nu_{max}$ will be marked as pending sources. As we will see in the next subsection, probabilistic dropping is applied for pending sources.

C. Probabilistic cell dropping

Once the violating, pending and non-violating streams have been identified, the WFEPD accepts/rejects cells into the buffer depending on the buffer state. Specifically, the WFEPD applies the following acceptance/rejection rules to cells belonging to the different streams:

- Cells of non-violating sources are always accepted.
- Cells of pending sources are probabilistically accepted. This means that there is a parameter of the algorithm which tells the maximum probability (p_p) of discarding packets of pending sources. We get the probability of discarding the packet of a pending source with the following expression:

$$p = \frac{r_i^{(av)} - th_b * r_i^{(calc)}}{th_t * r_i^{(sch)} - th_b * r_i^{(calc)}} * p_p \quad (10)$$

where $r_i^{(calc)}$ is the bandwidth that is theoretically calculated for a pending source with the algorithm in Section V.

$$r_i^{(calc)} = (\sum_{j=1}^{\nu_{min}-1} r_j^{(av)} - \psi_T) * \frac{\alpha_i}{\sum_{j=1}^{\nu_{min}-1} \alpha_j} \quad (11)$$

Formula 10 says that the drop probability linearly increases up to p_p depending on the difference between the calculated rate and measured average rate.

- Cells of *violating* sources are discarded with probability 1.

Recall that the third part of the WFEPD is concerned with identifying buffer states in order to execute these acceptance/rejection rules. Similarly to other variants of the EPD, the WFEPD simply uses a predefined threshold in the buffer to identify congestion situations when the cell discard mechanism (depending on the classification of the source) must be executed. In the simulation examples we applied a buffer threshold equal to zero which means that as soon as a source becomes *violating* or if it is just pending but it has been decided to be dropped (with probability p), the discard mechanism is executed regardless of buffer occupancy.

D. A Numerical example

For illustration purposes, let the α_i shares be the following:

I. $\alpha_1 = 0.25$
II. $\alpha_2 = 0.20$
III. $\alpha_3 = 0.15$
IV. $\alpha_4 = 0.10$
V. $\alpha_5 = 0.20$
VI. $\alpha_6 = 0.10$

Let the server capacity be $R = 100\ byte/sec$. Further assume that the measured average rates and the calculated r_i/α_i values are the following:

$r_1 = 30 \quad r_1/\alpha_1 = 120$
$r_2 = 30 \quad r_2/\alpha_2 = 150$
$r_3 = 30 \quad r_3/\alpha_3 = 200$
$r_4 = 20 \quad r_4/\alpha_4 = 200$
$r_5 = 15 \quad r_5/\alpha_5 = 75$
$r_6 = 5 \quad r_6/\alpha_6 = 50$

Now the WFEPD rearranges the sources according to their r_i/α_i values. Below we list the new order of connections with their respective r_i/α_i values:

$i = 1 \quad III. \quad 200$
$i = 2 \quad IV. \quad 200$
$i = 3 \quad II. \quad 150$
$i = 4 \quad I. \quad 120$
$i = 5 \quad V. \quad 75$
$i = 6 \quad VI. \quad 50$

According to the WFEPD algorithm we are now looking for the smallest ω such that (6) is satisfied. In this example this is $\omega_{min} = 5$, because:

$$\omega = 5: \quad r_\omega = 15 \leq (\sum_{i=1}^{5} r_i - \psi_T) * \frac{\alpha_5}{\sum_{i=1}^{5} \alpha_i} = 21.11 \quad (12)$$

Thus, the accepted rates in this case for the first four connections is given by $r_1^{(sch)} = 17.14, n_2^{(sch)} = 11.422, n_3^{(sch)} = 22.85, n_4^{(sch)} = 28.57$. That is, $\sum_{i=1}^{4} r_i^{(sch)} = 80$ and the last two connections get their offered 20 byte/sec rates.

Now it remains to identify the pending sources. We assume a threshold of $th_b = 0.7$, which means that those connections whose measured average rates are more than the 70% of their calculated shares are qualified as pending sources. Now we are looking for the smallest ν that satisfies 8. This is $\nu_{min} = 6$, because it is not satisfied for $\nu = 5$:

$$\nu = 5: \quad r_\nu = 15 > 0.7 * (\sum_{i=1}^{5} r_i - \psi_T) * \frac{\alpha_5}{\sum_{i=1}^{5} \alpha_i} = 14.7 \quad (13)$$

However $\nu_{min} = 6$ satisfies the inequality:

$$\nu = 6: \quad r_\nu = 5 \leq 0.7 * (\sum_{i=1}^{6} r_i - \psi_T) * \frac{\alpha_6}{\sum_{i=1}^{6} \alpha_i} = 7 \quad (14)$$

We can see that the source $V.$ is not violating but it has been qualified as a flow that is close to be violating. How close it is really to be violating depends on the threshold (th_b).

Note that in the above example all connections which have exceeded their respective shares ($\alpha_i * R$) have been penalized. This is however, not necessarily the case. Let

I. $\alpha_1 = 0.01$
II. $\alpha_2 = 0.4$
III. $\alpha_3 = 0.4$
IV. $\alpha_4 = 0.1$
V. $\alpha_5 = 0.05$
VI. $\alpha_6 = 0.04$

It can easily be seen that in this case the I. and the V. sources will be punished (which will get 2 instead of 30 and 13 instead of 20 byte/s accepted rate). The IV. and VI. source will not be punished, i.e. it will get all its 25 byte/s offered traffic, despite that they both have sent at a higher rate than their absolute fair shares ($\alpha_i * R$).

VI. Simulation results

We have implemented the WFEPD algorithm with both the moving average and the sliding window rate measurement methods. Their performance is equivalent, however the moving average method is less complex to implement. Thus in the following we present simulation results for the WFEPD algorithm applying the moving average rate measurement method. In this section we consider the following cases, where we are interested in the performance of it.

First we demonstrate that the algorithm works well for TCP sources which require the same share of bandwidth from the total server capacity. We compare the performance of the WFEPD with a conventional FIFO buffer and point out the unfair bandwidth distribution in case of FIFO. We also consider a case when different TCP sources have different shares. Next, simulation results are presented for the case when TCP sources are mixed with non-adaptive UDP sources. As TCP backs off during congestion periods, misbehaving sources can acquire as much bandwidth as they want, provided that TCP and UDP share a common FIFO buffer. This results in even more severe unfairness than in the first case. As we will see, with appropriately set parameters, the WFEPD can restore fairness in this case as well. We then address the fairness issue in a heterogeneous round-trip time (RTT) environment, where it is a well known phenomenon that TCP connections with low RTT can grab an unfair proportion of the bandwidth from connections with higher RTT. We show that the WFEPD algorithm provides fairness for connections with different RTTs. We proceed with showing that the WFEPD algorithm scales well to high capacity (155 Mbps) ATM back-bone switches. Finally, we discuss the speed of convergence of the algorithm and show that it takes the algorithm approximately 10 sec to stabilize. It is also discussed that this time depends on the parameter setting of the moving average rate measuring filter.

The simulation setup is shown in Figure 4. We have simulated 10 sources that can be both "ftp-like" TCP sources and UDP sources depending on the particular configuration. The ftp source is modeled as a bulk data transfer while the greedy UDP source is realized by a bursty ON-OFF source. Each of these sources has its own UBR, AAL5/ATM VCC. The flow-control mechanism of the TCP protocol is NewReno [16]. The

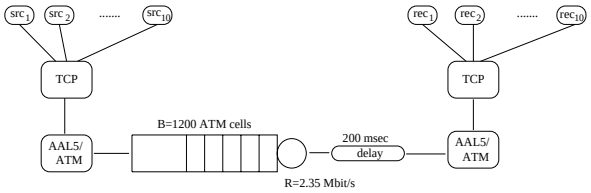

Fig. 4. Simulation setup

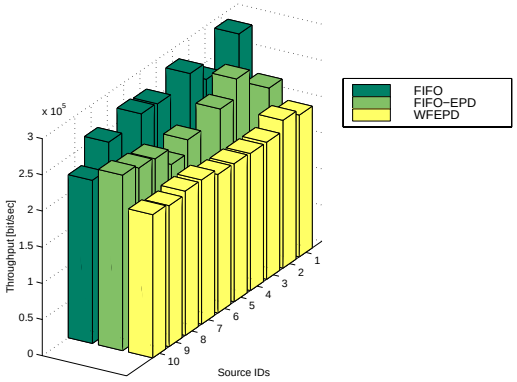

Fig. 5. Throughput of TCP sources with FIFO and FIFO-EPD and WFEPD (View 1)

output buffer of the ATM switch can store 1200 ATM cells and has a server capacity of 2.35 Mbps. The reason why we set the buffer size relatively large compared to the server rate was to avoid packet losses due to buffer overflow, thus let the WFEPD algorithm to execute cell dropping. Later we illustrate that the algorithm also works well in a buffer of size 500 ATM cells. We have inserted a constant 200 msec delay in the path that insures together with the queuing delay a realistic end-to-end delay in the order of 300-400 msec. The parameter setting of the moving average filter is the following: $q = 0.95$, $\delta = 0.01\ sec$ (equation 2). In each case we have run 5 minutes long simulations and measured the total throughput of the sources during this interval.

In Figures 5 and 6 only TCP sources are present, all with equal share. We see that in the FIFO case some connections experience very low bandwidth while others acquire bandwidth above their required share ($\alpha_i * R$). By applying the pure EPD mechanism the sum of the throughputs is increased but the unfair bandwidth distribution still remains. However with the WFEPD buffer management scheme all the 10 TCP connections get around 200 kbps which is approximately equal to the $0.1 * R = 235\ kbps$ share minus the TCP/IP and ATM overheads.

Next we set the buffer size to 3000 ATM cells and measure the queue length distribution in order to identify the most reasonable buffer size that is still enough for the WFEPD algorithm to provide fairness. This is shown in Figure 7. This Figure shows that in most of the cases the queue length is under 500 ATM cells. Thus we set the queue length to 500 ATM cells and measure the throughput of the ftp sources in a 5 minutes long simulation. The result is shown in Figure 8, which indicates that the WFEPD algorithm can provide fairness even with a smaller buffer as well.

In Figure 9 we demonstrate that the WFEPD algorithm re-

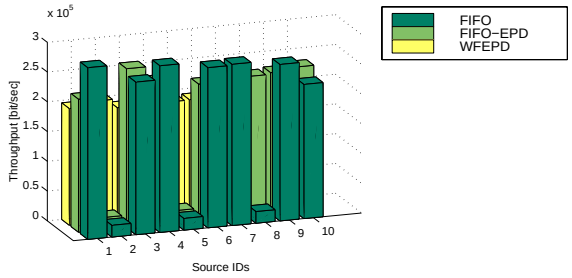

Fig. 6. Throughput of TCP sources with FIFO and FIFO-EPD and WFEPD (View 2)

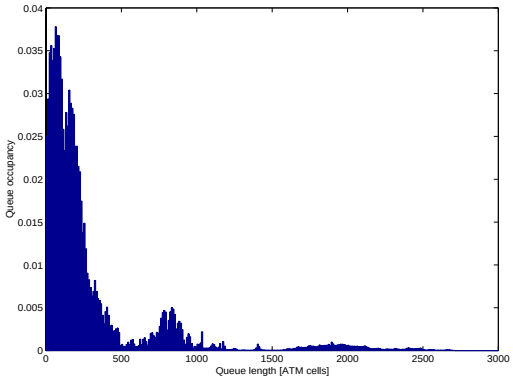

Fig. 7. Queue length distribution, delay=200 msec

quires proper parameter settings in order to be really fair. The same configuration setup has been simulated as the previous one with the exception that the q parameter of the moving-average filter has been changed. We measured the largest deviation between the throughput of two TCP connections in the function of this q parameter. The deviation in throughputs can be interpreted as a measure of fairness. We see that increasing the the q parameter improves the fairness of the algorithm. Larger q means that the tail of the averaging filter degrades more slowly; in other words the system has more memory.

The parameter setting of the moving-average filter determines how the rate changes of the sources are percepted by the WFEPD buffer and it is in close relation with the achieved fairness. In Figure 10 we selected one connection and plotted the rate measured by the WFEPD algorithm in the function of time.

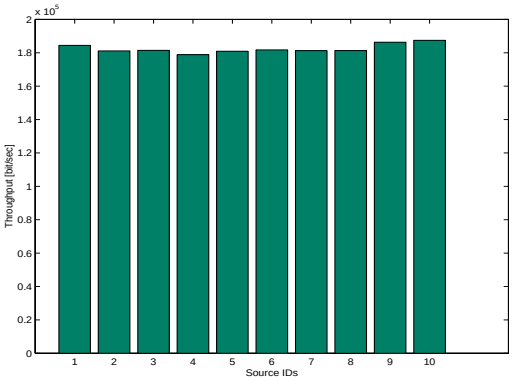

Fig. 8. Throughput of TCP sources with WFEPD, buffer size=500, delay=200 msec

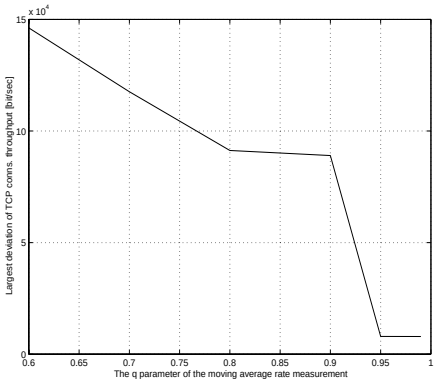

Fig. 9. Throughput deviation between TCP sources depending on the averaging filter q parameter

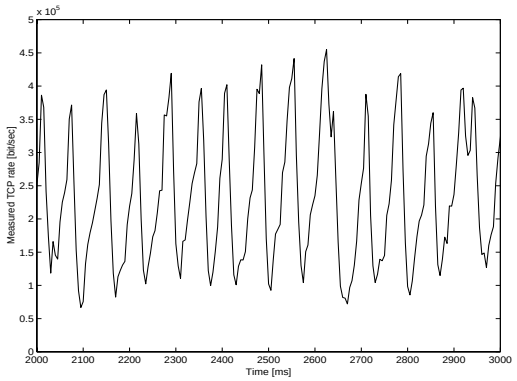

Fig. 10. The rate of a TCP source measured by WFEPD, delay=200 msec

The measured rate fluctuates around the mean; the amplitude of the fluctuation depends on the parameters of the moving-average filter. If we use a more slowly degrading moving-average characteristics the fluctuations are better smoothed out and we get smaller rate amplitudes and better fairness. The trade-off here is that by applying larger time scales improves fairness but the the algorithm gets less dynamic and the the queues get longer. The correct parameter setting will have even more importance in the heterogeneous round-trip time environment which will be discussed later.

In Figure 11 the results for the non-equal share case are shown. The first 5 TCP connections have a share of $\alpha_i = 1/15$ and the second half of the sources have $\alpha_i = 2/15$. We can see that the WFEPD algorithm provides for each stream exactly the required share.

Next, the simulation results for the mixed TCP and UDP case are discussed. Now there are 5 TCP and 5 UDP connections sharing the common buffer of Figure 4. In the first case we assume non-violating UDP sources; that is they are not transmitting more than their announced average rate. In Figure 12 we see that in that case even with simple FIFO buffering the equal bandwidth distribution is guaranteed. However, the situation drastically changes when the UDP sources start to misbehave and send more packets than what was announced. They can grab as much bandwidth as they want and they always get what they offer as in Figure 13. This is due to the flow control mechanism of TCP. The sending TCP endpoint presumes congestion at every packet loss and decreases sending rate, thus it

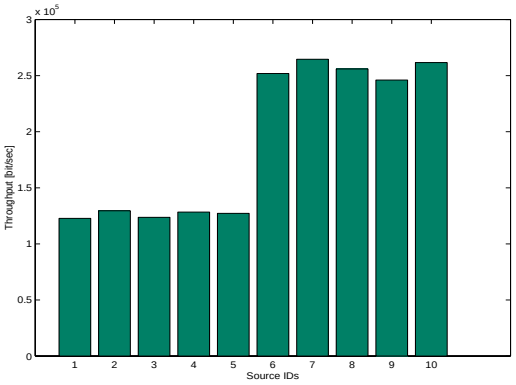

Fig. 11. Throughput of TCP sources with different α_i shares (WFEPD)

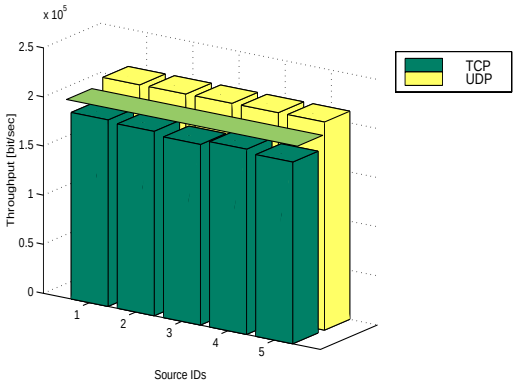

Fig. 14. Throughput of TCP and violating UDP sources with WFEPD buffer

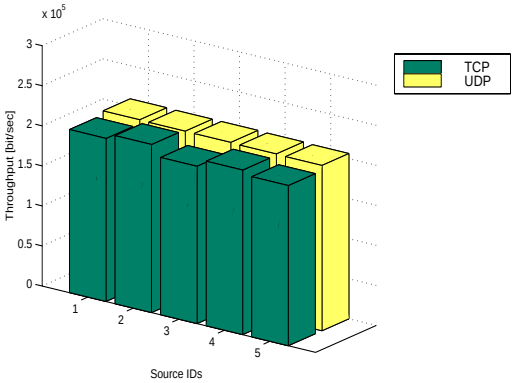

Fig. 12. Throughput of TCP and non-violating UDP sources sharing a common FIFO buffer

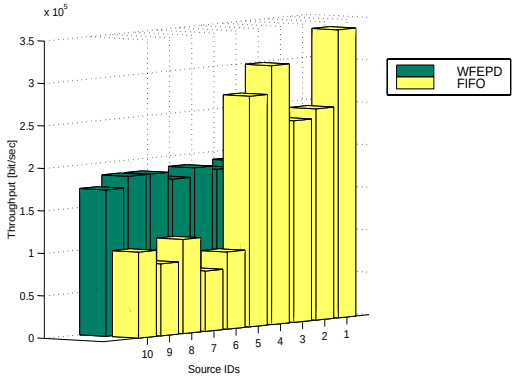

Fig. 15. Throughput of TCP sources with different RTTs, in case of FIFO and WFEPD (View 1)

makes more bandwidth available for misbehaving sources.

The unfair treatment of TCP sources can be greatly improved if we replace the FIFO buffer with the WFEPD buffer. The result is shown in Figure 14. The vertical surface indicates the theoretically achievable maximum throughput of TCP sources in the case if they got exactly their required shares. The actually measured throughputs are just 3-4% below the theoretical maximum.

We now investigate the performance of the algorithm in a heterogeneous RTT environment. We modified the simulation setup in Figure 4 such that for one half of the connections (sources 1-5) the delay object introduced 100 msec delay while for the other half of the connections (sources 6-10) it was 300 msec. In Figure 15 and 16 the throughput of the sources can be seen in case of FIFO and WFEPD from different viewpoints. The well known phenomenon of squeezing out long RTT connections in a FIFO buffer is clearly visible in the Figures. In contrast to FIFO the WFEPD algorithm can provide total fairness among the sources with different RTTs. Here we would like to point out the importance of correct parameter settings of the moving-average rate measuring filter. In order to find out the correct settings first we simulated homogeneous RTT environment with 100 msec delay. The best parameters turned out to be $q = 0.95$, $\delta = 0.005\ sec$. In the heterogeneous

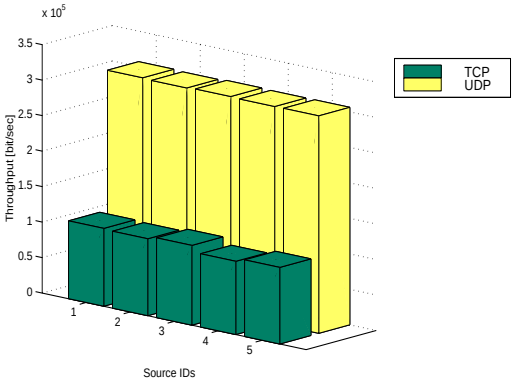

Fig. 13. Throughput of TCP and violating UDP sources sharing a common FIFO buffer

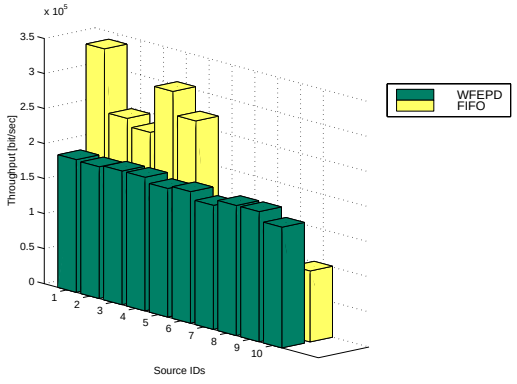

Fig. 16. Throughput of TCP sources with different RTTs, in case of FIFO and WFEPD (View 2)

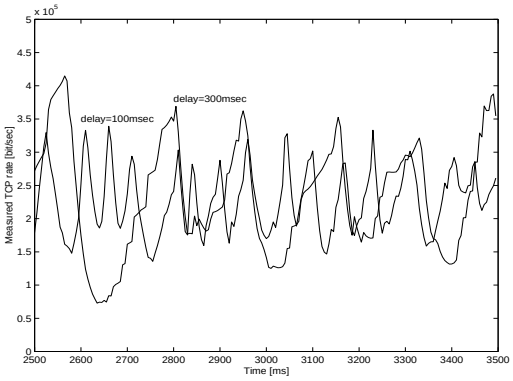

Fig. 17. The rate of a TCP source measured by WFEPD, different RTTs

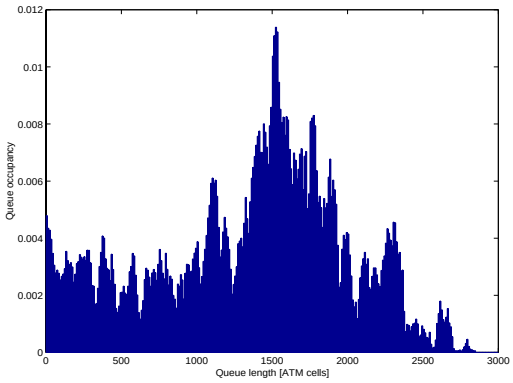

Fig. 18. Queue length distribution, different RTTs

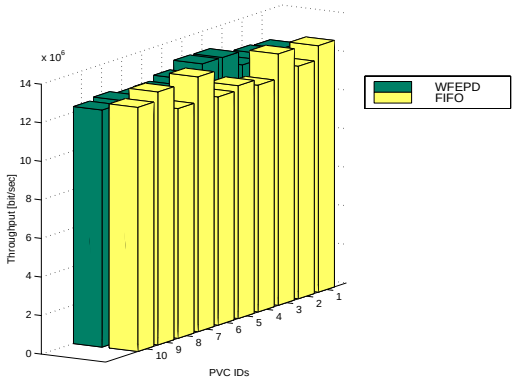

Fig. 19. Throughput of PVC flows on a 155 Mbps back-bone in case of FIFO and WFEPD

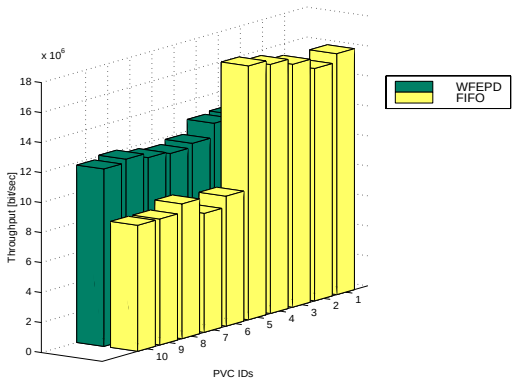

Fig. 20. Throughput of PVC flows on a 155 Mbps back-bone with different number of TCP connections per PVC

RTT case (delay=100 and 300 msec) we set the parameters such that the moving average characteristics decays three time slower ($q = 0.966$, $\delta = 0.005\ sec$) than in the homogeneous, 100 msec delay case. In other words it smoothes out the average on a longer time scale. The reason is that now we have connections in the system with three times higher time constants. To illustrate the above reasoning we plotted the rate measured by the WFEPD algorithm for two TCP connections, one with 100 msec delay in its path the other with 300 msec delay (Figure 17). The rates of the two TCP connections fluctuates around the same mean but the longer RTT connection fluctuates with a higher period and higher amplitude. A TCP connection with higher RTT uses larger congestion windows and behaves less dynamically than a connection with lower RTT. This is the reason of the increased period time. The increased amplitude is also caused by the larger congestion windows. The traffic of a TCP connection with a larger congestion window results in larger fluctuations in the rate of its traffic, that is the traffic will be more bursty. This explains the higher amplitudes in the rate of the longer RTT connection. This is the reason why we have to increase the time scale of the moving-average rate measuring filter by about three times compared to the homogeneous 100 msec delay case. The trade-off here is that we had to increase the buffer size to 2000 ATM cells in order for the WFEPD algorithm to actually be fair. The queue length distribution is shown in Figure 18. Note that the queue length distribution would have increased anyway as we increased the RTTs which results in more bursty behavior of TCP sources.

Next we discuss the scalability of the algorithm. We simulated a 155 Mbps ATM switch output port where we have 10 PVC (Permanent Virtual Circuit) flows for which we want to guarantee fairness. In each PVC flow we have 60 TCP connections. The acquired throughput of the PVC flows is shown in Figure 19. We conclude that in the case of aggregated TCP connections the FIFO mechanism is almost as fair as the WFEPD mechanism, provided that each PVC carries the same number of TCP connections. This is because the aggregated traffic of many TCP connections does not exhibit responsiveness to packet losses and it tends to behave like a non-responsive, greedy UDP source. Note that the WFEPD algorithm is still superior over FIFO in this case as well as it can provide an arbitrarily set α_i shares which is not possible with FIFO. Furthermore FIFO becomes unfair if the number of TCP connections in the ATM PVCs are not equal. We simulated the same configuration with the difference that now in the first five PVCs (PVC IDs: 1-5) there are 60 TCP connections, while in the rest of the PVCs (PVC IDs: 6-10) there are only 30 TCP connections in each. The measured throughput of the PVC flows are shown in Figure 20. It can be observed that the PVC flow which carries more TCP connections can grab an unfair proportion of the server capacity. This is because as we put more and more TCP connections into one PVC flow the less responsive and the more aggressive their aggregate rate will be.

Finally we address the speed of convergence of the algorithm. We point out that the speed of convergence depends on the parameter settings of the moving-average rate measuring filter. If

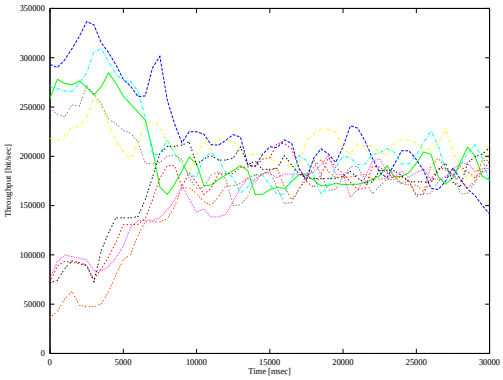

Fig. 21. Throughput of TCP sources in time, delay=100 msec

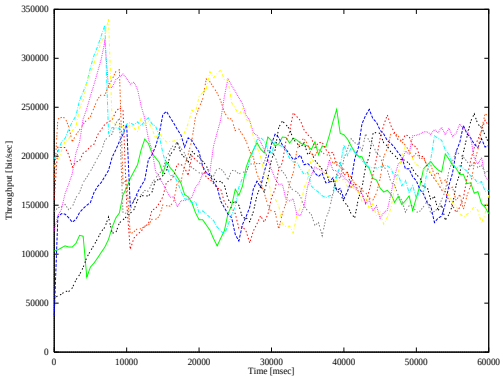

Fig. 22. Throughput of TCP sources in time delay=300 msec

we have a fast decaying filter characteristics then the algorithm stabilizes more quickly. However if we have connections with large RTTs in the buffer we have to increase the time factor of the moving-average filter to guarantee fairness as shown above, but this means at the same time that the algorithm will stabilize more slowly. We illustrate this with two simulation results. First we simulated the configuration in Figure 4 with 100 msec delay in the path, while in the second case we increased the delay to 300 msec. In Figure 21 and 22 the throughput of the 10 sources are plotted against time for the 100 msec and 300 msec delay cases respectively. In each point in time we plotted the throughput of the connections in the forthcoming 5 sec. Thus we can read from the figure the point in time where the throughputs of the sources start to behave similarly. For the 100 msec delay case it is around 10 seconds, while for the 300 msec delay case it turns out to be approximately three times as much (30 seconds). The connection with longest RTT determines the time parameters of the moving-average filter which in turn defines the speed of convergence of the algorithm.

VII. Concluding Remarks

In this paper we have described a simple Weighted Fair Early Packet Discard algorithm which is applicable for determining the allocated bandwidth for IP streams carried over ATM VCC's. The algorithm guarantees long term fairness independently of the upper layer protocol (i.e. TCP or UDP). We have proposed to apply this algorithm together with the well known EPD mechanism to provide high bandwidth utilization. The WFEPD relies on a per-session state, which contains a moving averaged information about the resources (i.e. bandwidth) used during the past T time by each flow. Furthermore, the WFEPD relies on the concept of violating, pending and non-violating sources, which are dependent on the server rate R, sliding window size T and the maximum length of the IP packets, L_{max}. There is a clear trade-off between the actually achieved fairness (i.e. the throughput difference between the sources) and the T window size and in order for the WFEPD to be fair a minimum value must be set for $R * T$. Thus, the $R * T$ value must be greater than the $N * L_{max}$ product.

Acknowledgments

We would like to thank Juan Noguera at Ericsson Japan for his advice on the sliding window mechanism and for the inspiring discussions. We also thank the unknown reviewers for their many insightful comments and suggestions.

References

[1] Allyn Romanov and Sally Floyd, "Dynamics of TCP Traffic over ATM Networks", *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, May, 1995.

[2] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, August, 1993.

[3] Chien Fang, Helen Chen and Jim Hutchins, "A Simulation Study of TCP Performance in ATM Networks", *Proc. IEEE GLOBECOM '94*, pp. 1217-1223, Nov, 1994.

[4] Lampros Kalampoukas and Anujan Varma, "Performance of TCP over Multi-Hop ATM Networks: A Comparative Study of ATM-Layer Congestion Control Schemes"

[5] Dong Lin and Robert Morris, "Dynamics of Random Early Detection", *ACM SIGCOMM '97*, 1997.

[6] Omar Elloumi and Hossam Afifi, "RED Algorithm in ATM Networks"

[7] H. Heffes and D. M. Lucantoni, "A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 6, September, 1986.

[8] K. Sriram and W. Whitt, "Characterizing Superposition Arrival Processes in Packet Multiplexers for Voice and Data", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 6, September, 1986.

[9] J. Roberts and J. T. Virtamo, "The Superposition of Periodic Cell Arrival Streams in an ATM Multiplexer", *IEEE Transactions on Communications*, Vol. 39, No. 2, February, 1991.

[10] Raif Onvural, "Asynchronous Transfer Mode Networks, Performance Issues", *Artech House*, ISBN 0-89006-804-6, 1995.

[11] Bernhard Suter, T. V. Lakshman, Dimitrios Stiliadis, Abhijit Choudhury "Efficient Active Queue Management for Internet Routers", *Bell Laboratories, Lucent Technologies*, November, 1997.

[12] Sammy Chan, Moshe Zukerman, Eric W. M. Wong, K. T. Ko and Edmund Yeung, "Achieving Fair and High Packet-level Throughput in ABR Service", *IEEE International Conference on Communications, ICC '98*, pp. 635-639.

[13] K. Cheon and S. S. Panwar, "Early Selective Packet Discard for Alternating Resource Access of TCP over ATM-UBR", *Proc. of the 22nd IEEE Annual Conference on Computer Networks, LCN '97*, pp. 306-316, Minneapolis, Nov., 1997.

[14] Kangsik Cheon and Shivendra S. Panwar, "On the Performance of ATM-UBR with Early Selective Packet Discard", *IEEE International Conference on Communications, ICC '98*, pp. 221-227.

[15] M. Ajmone Marsan, K. Begain, R. Lo Cigno, M. Munafo, "Performance of TCP File Transfers over the Explicit Rate ABR ATM Service Category", *IFIP 5th International Conference on Telecommunications - Modeling and Analysis*, pp. 248-258, March, 1997.

[16] "The NewReno Modification to TCP's Fast Recovery Algorithm" *draft-ietf-tcpimpl-newreno-00.txt; Work in progress*