Performance Evaluation of Profiler Mechanisms for the Internet Assured Service

Maria-Dolores Cano, Fernando Cerdan, Joan Garcia-Haro, Josemaria Malgosa-Sanahuja
Polytechnic University of Cartagena
Department of Information Technologies and Communications
Campus Muralla del Mar s/n (Ed. Hospital de Marina)
30202 Cartagena, Spain
Ph. +34 +968 32 5368
Fax +34 +968 32 5338
{mdolores.cano, fernando.cerdan}@upct.es

ABSTRACT

As Internet is rapidly growing and receiving traffic from multimedia applications that are sensitive to available bandwidth and delay experienced in the network, there is a strong need for quality of service (QoS) support. The Integrated and Differentiated Service models are two approaches for adding QoS to Internet. The Assured Service is an end-to-end service based on the Differentiated Service architecture. In this paper, we study and compare the performance of three profiler mechanisms to provide the guaranties of an Internet Assured Service. Two of them, TSW and Leaky Bucket are the most commonly used, and the third is a new Counter Based profiler, which is proposed in this paper. The study is done by simulation employing TCP RENO sources.

Keywords: Assured Service, Profilers, Queue management, RIO algorithm

1. Introduction

The Internet is rapidly growing and receiving traffic from different multimedia applications that are sensitive to available bandwidth and delay experienced in the network. Therefore, there is a strong need for quality of service (QoS) provision [1]. For this reason, new network protocols, traffic management functions, and technologies came up for supporting such variety of multimedia applications.

The Differentiated Services approach defines a group of mechanisms to treat packets of aggregated flows with different priorities according to the information carried in the Diff. Serv. field of the IP packet header. Packets are classified and marked to receive a particular per-hop forwarding behavior (PHB) on nodes along their path. Complex classification and conditioning functions (metering, marking, shaping) need only to be implemented at boundary nodes. Interior nodes perform a set of forwarding behaviors (PHBs) to aggregates of traffic which have been appropriately marked.

A set of PHBs are being standardised at the IETF to develop end-to-end differentiated services. Two PHBs are currently in deep discussion: the Expedited Forwarding PHB (EF-PHB) and the Assured Forwarding PHB (AF-PHB). The Assured Forwarding Per Hop Behavior (AF-PHB) [2] currently allows a domain server (DS) provider to offer for general use, until four different levels of assurance (AF classes), to deliver IP packets. IP packets must belong to one of the AF classes. Within each AF class, either the user or the DS domain may mark an IP packet with three levels of loss precedence. The AF-PHB is selected to implement the end-to-end Assured Service (AS).

The user of an Assured Service [3] expects a low dropping probability as long as it sends traffic within the expected capacity profile. Packets of individual flows are marked as IN (in profile) or OUT (exceeding the profile). Routers inside the network do not distinguish packets of individual flows but they implement a preferential drop mechanism giving preference to IN packets. The Assured Service is intended to give the customer the assurance of a minimum average throughput, the target rate, even during periods of congestion. The remaining bandwidth should be shared in a fairly manner among all competing sources.

Each TCP source of an AS capable host, uses a profiler to distinguish assured packets (IN) from excess packets (OUT). The leaky bucket and the average rate estimator TSW (Time Sliding Window) are the two most commonly accepted profilers, which interact with a router based on the RIO algorithm [3]. In [4] it is stated that a Leaky bucket allows a better performance due to the bursty nature of TCP sources. Nevertheless, in [5] a TSW estimator was used with a stable configuration for the mapping of an assured service on an ATM network.

In this paper, we use TCP input traffic sources to present a comparative study by simulation of different types of profilers providing the assured service guarantees. These profiler mechanisms involve such amount of parameters (instantaneous TCP sending rate, RTT, thresholds, target rate, etc.) that the number of IN and OUT marked packets depends on them critically. In addition, the profiler interacts with the buffer mechanisms in the router (RIO), since according to the congestion state, many OUT dropped packets may provoke a drastic TCP window reduction then, affecting the instantaneous sending rate along the expected target rate. We also present a new profiler mechanism based on counters. Unlike TSW or Leaky Bucket, it provides a good performance marking IN packets whose rate is quite close to the target rate in most of the cases. We explore the right configuration of the main parameters associated to the three mechanisms, TSW, Leaky Bucket and Counter Based, in order to achieve the expected target rate as well as a fair distribution of the excess bandwidth. Simulation results presented in this work are relative to link bandwidth and buffer capacity set, and they have a confidence interval of 90%.

The rest of this paper is organized as follows. In section 2 we describe the assured service architecture as well as the model used for simulations. Section 3 is devoted to describe, evaluate and compare the performance of the two most used profiler mechanisms (TSW and Leaky Bucket) as well as the new Counter Based profiler. Finally, section 4 concludes de paper.

2. Assured Service architecture

In the current assured service architecture there are two main components as shown in Figure 1. One (T) is the profiler usually located at the network edge access point. A profiler may include a meter and a marker. Meters measure temporal properties (e.g. rate) of a traffic stream that are used by markers to separate assured packets (IN) from excess packets (OUT) of the traffic stream. The other component is the router. A router located inside the network, buffers and forwards the aggregated traffic. The queue management most commonly implemented in the routers employs twin RED algorithms to preferentially drop OUT packets. The mechanism is known as RIO (RED with IN and OUT) and was first proposed by Clark and Fang [3]. RIO calculates an average queue size for IN packets and another for (IN+OUT) packets in the queue. The probability of discarding an IN packet depends on the IN packet queue size, whereas the probability of discarding OUT packets depends on the total queue size. Each RED algorithm has three parameters (Pmax, Min and Max) which define the normal operation phase [0, Min), the congestion avoidance phase [Min, Max) and the congestion control phase [Max, ∞). The RIO parameters are 40/70/0.02 for IN packets and 10/40/0.2 for OUT packets. Weight_in and Weight_out RED parameters used to calculate the average queue size have been chosen equal to 0.002 as recommended in [6].

As regard of the simulations we used the topology of Figure 1 where a number of TCP sources send traffic at link rate to their profilers. These profilers will be either the TSW, Leaky Bucket or the new Counter Based mechanism. After profiling, IN and OUT packets are aggregated in a single flow through a RIO router.

The sliding window protocol of TCP sources has been simulated using a model where the following figures and assumptions are applied:

- Processing delay between layers as well as interaction between TCP and the application layer by the socket layer have not been considered.
- TCP sources are greedy sources. TCP layer always has a segment to send as long as the allowed window permits it.
- Destination sources only send acknowledgements, which are never delayed. Hence, segments are acknowledged as soon as they are received.
- Acknowledgements always advertise the maximum window size and never get lost.
- All packets generated by the source have the maximum segment size (MSS), which has been set to 10,000 bytes.
- The maximum window size equals the bandwidth delay product for WAN environments.
- The granularity of the retransmission timer has been set to 0.2 seconds, unless other value is specified.

The simulation tool has a clock system that controls the events during simulation time with a granularity that depends on the link rate. For our simulations, the link rate is 33 Mbps, although because of that granularity of the model, it can remain a little bit below.

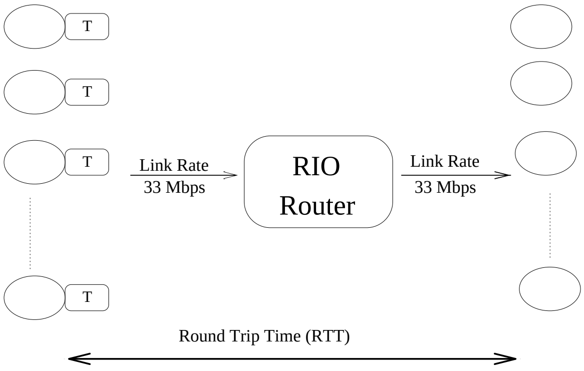

Figure 1. Network topology for simulations

3. Profiler mechanisms

In this section, we describe and study the three profiler mechanisms, the TSW rate estimator, the Leaky Bucket algorithm, and the new proposal presented in this paper, the Counters based mechanism. We evaluate and compare the performance of these three algorithms in terms of dependency of the achieved profiles with RTT variations, and distribution of the excess link bandwidth.

3.1. Leaky Bucket algorithm

According to the use of this mechanism, it might be interesting marking more or less IN packets depending of how permissive we want our mechanism. Leaky Bucket defines *Increment* as the number of units of time that pass between the arrival of a packet and the arrival of next packet if the stream flows exactly at the target rate. Besides, it defines *Tolerance* as the number of units of time that added to the *Increment* will set the tolerance of the system. For simplicity, we express *Tolerance* as *TAU* times the *Increment*, where *TAU* is an integer or fractional value.

The implementation of the algorithm is explained as follows. Given the Theoretical Arrival Time (TAT) of packets, we compare the Packet Arrival Time (PAT) with TAT. If TAT is less or equal than PAT the packet is marked as IN. If the packet arrived before the theoretical time but inside a fixed margin (*Tolerance),* then the packet is marked as IN as well, otherwise it will be an OUT packet. See the pseudo-code of the algorithm and its flowchart in Figure 2.

```
Initially:
    X=0;
    LCT=0;
    Aux=0;
    Ta indicates current instant of time;
For each packet arrival:
    Aux=X-(Ta-LCT);
    if Aux<0
        { Aux=0;
          X=Increment;
          LCT=Ta;
          Marked packet as IN;
        }
    else if Aux<Tolerance
            { X=Increment+Aux;
              LCT=Ta;
              Marked packet as IN;
            }
         else
          Marked packet as OUT
```

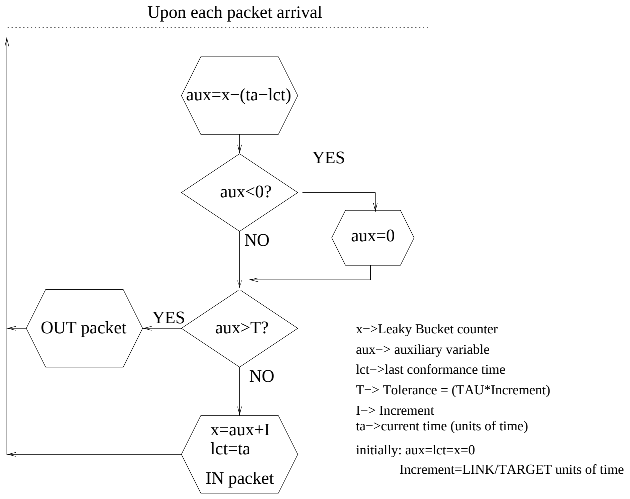

Figure 2. Leaky Bucket implementation and flowchart

3.1.1. Achieved rates for IN packets and excess bandwidth distribution

The performance of the three profilers under study depend on many parameters including those related to the TCP traffic (RTT (Round Trip Time), MSS (Maximum Segment Size), timer granularity, …). We will see that the performance of a profiler is quite dependent on the RTT of the TCP connection. In this case, we evaluate the Leaky Bucket varying the main parameter *TAU*. We tried to find the range of *TAU* values for assuring target rates independently on RTT. According to simulations, *TAU* values from 0 to 20 guarantee a good performance in the sense that the achieved IN packets rate of each source reaches or exceeds slightly its target rate.

Logically, as *TAU* increases (from 0 to 20), higher the throughput is for the IN packets. This is obvious since the more we increment the TAU value, the more packets are marked as IN. This property of the Leaky Bucket algorithm allows making up for deficiencies in achieving the target rate due to bad network performance by incrementing *TAU*. However, this option should be considered carefully to avoid using bandwidth out of the target rate contract. Therefore, if there are no deficiencies to compensate, IN packets rates are guaranteed with *TAU* =0 and *TAU*=10. Figures 3 and 4 plot the IN packets throughput (bps) as a function of the RTT (ms) for different values of *TAU*, assuming all sources with same RTT. Simulations for these two figures have been performed with eight sources, whose contracted rates are 1, 1, 2, 2, 3, 3, 4 and 4 Mbps from source 0 to source 7 respectively, but only sources 1 and 6 have been depicted.

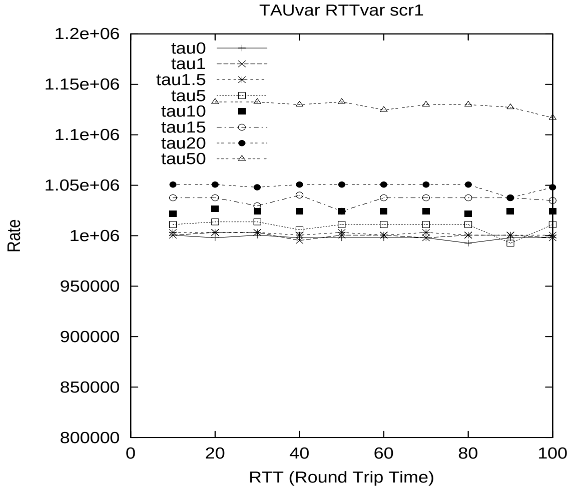

Figure 3. Achieved IN Throughput with TR of 1Mbps

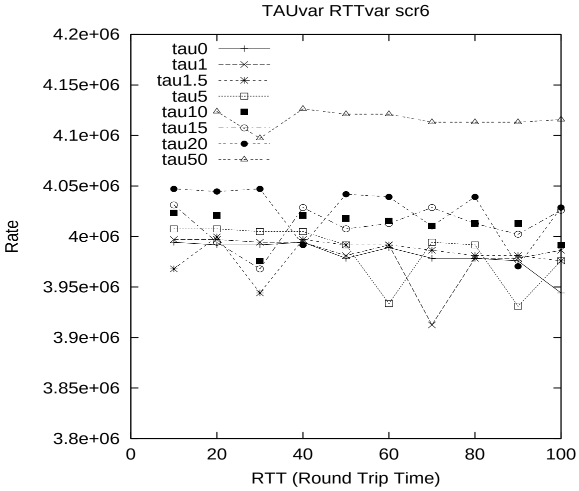

Figure 4. Achieved IN Throughput with TR of 4 Mbps

In the case that all sources have the same RTT, IN throughputs achieve the target rates for many different values of RTT. However, the higher the target rates the bigger the variations, not exceeding 2.5% for *TAU* less than 20. When we have different RTT for each TCP source, results for achieved IN throughputs are also good. In Tables 1 and 2 we can see the achieved IN packets rate for simulations with both RTT cases.

In terms of sharing the excess bandwidth, the distribution is done fairly if all sources have the same RTT. We agree that sources with lower targets obtain a little bit more bandwidth that those with higher ones as shown in Table 2. Notice that shares do not vary more than 12% approximately in the worst case, and in the best case the variation is around 4%. Also, from Figures 5 and 6 we can observe the dependency of the bandwidth sharing for different values of RTT (from 10 ms to 100 ms), when all sources have the same RTT. The ideal graph would be a unique line, hence, the thinner the graph the better result. The distribution is more impartial for *TAU*=10 than for the *TAU*=0. For the case when each source have a different RTT, bandwidth sharing becomes more irregular, being sources with lower RTT and lower target rates favored. Comparing the *Excess BW* columns from Table 1 and 2, we see that the distribution is less equitable.

For *TAU*=10 all sources are slightly above their profiles, and the sharing of the excess link bandwidth is quite regular. Notice, that for extreme situations, where multiplexed connections have very different RTT and these RTT differ among them more than 60ms approximately, sources with greater targets and higher RTT do not achieve their profiles.

This drawback may be due to the TCP characteristics, because some connections have to wait much more time than others for their acknowledgements in case of packet losses and retransmissions.

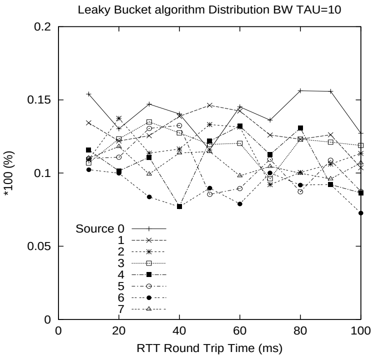

Figure 5. %BW vs. RTT with TAU=0

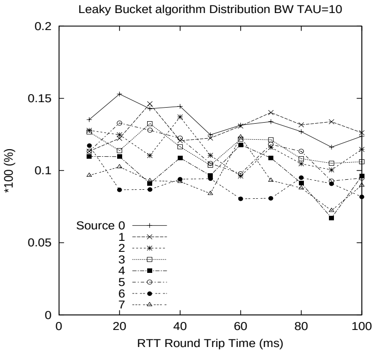

Figure 6. %BW vs. RTT with TAU=10

Table 1. LB simulations with distinct RTT each source (R_T≡Target Rate)

Simulation with TAU=10					
H	RTT	R_T	IN packets rate	Total throughput	Excess BW
0	10	1	1.003200	3.725132	2.721932 (20.9%)
1	20	1	0.987360	2.940960	1.953600 (15.1%)
2	30	2	2.001120	3.867600	1.866500 (14.4%)
3	40	2	1.987920	3.468960	1.481040 (11.4%)
4	50	3	2.996400	4.501200	1.504800 (11.6%)
5	60	3	2.983200	3.883440	0.900240 (6.9%)
6	70	4	3.991680	4.836480	0.844800 (6.5%)
7	80	4	3.989040	4.852320	0.863280 (6.7%)
	Total	20	19.95	32.07609	12.13619 (93.4%)
///					
Simulation with TAU=0					
H	RTT	R_T	IN packets rate	Total throughput	Excess BW
0	10	1	0.997920	3.780480	2.782560 (21.5%)
1	20	1	0.995280	2.996400	2.001120 (15.4%)
2	30	2	1.990560	3.925680	1.935120 (14.9%)
3	40	2	1.995840	3.473474	1.477634 (11.4%)
4	50	3	2.991120	4.348080	1.356960 (10.5%)
5	60	3	2.988480	3.925680	0.937200 (7.2%)
6	70	4	3.986400	4.918320	0.931920 (7.2%)
7	80	4	3.944160	4.717680	0.773520 (5.9%)
	total	20	19.88976	32.08579	12.19603 (94%)

Table 2. LB simulations with same RTT for all sources (R_T≡Target Rate)

Simulation with TAU=10					
H	RTT	R_T	IN packets rate	Total throughput	Excess BW
0	50	1	1.024320	2.821520	1.797200 (13.8%)
1	50	1	1.024320	2.792480	1.768160 (13.6%)
2	50	2	1.998480	3.636924	1.638444 (12.6%)
3	50	2	2.019600	3.447520	1.447920 (11.1%)
4	50	3	3.017520	4.355192	1.337672 (10.3%)
5	50	3	3.022800	4.463920	1.441120 (11.1%)
6	50	4	4.018080	5.327200	1.309120 (10.0%)
7	50	4	4.015440	5.192560	1.177120 (9.1%)
	Total	20	20.14056	32.037316	11.916756 (91.6%)
///					
Simulation with TAU=0					
H	RTT	R_T	IN packets rate	Total throughput	Excess BW
0	50	1	0.997920	2.504938	1.507018 (11.6%)
1	50	1	0.997920	2.901360	1.903440 (14.7%)
2	50	2	1.980000	3.730320	1.750320 (13.5%)
3	50	2	1.993200	3.554641	1.561441 (12.0%)
4	50	3	2.993760	4.583040	1.589280 (12.3%)
5	50	3	2.991120	4.110480	1.119360 (8.6%)
6	50	4	3.978480	5.166480	1.188000 (9.1%)
7	50	4	3.999030	5.491200	1,492170 (11.5%)
	total	20	19.92144	32.042459	12.121019 (93.3%)

3.2. Time Sliding Window Algorithm

Here we study the performance of the rate estimator algorithm introduced by Fang in [3], the Time Sliding Window (TSW). This system consists of two different parts, a rate estimator and a tagging algorithm. The TSW rate estimator estimates the average rate upon each packet arrival and decays the past history over a *Winlength* period of time. *Winlength* is a constant parameter pre-configured initially. See the rate estimator implementation in Figure 7. Ideally the profiler should keep a TCP connection oscillating between 0.66 and 1.33 of its contracted rate, so that on average the connection can achieve the target rate

This profiler mechanism marks packets as OUT with probability *p* (Equation 1) when the estimated average rate exceeds a certain threshold. Two approaches are followed. The first remembers a long past history and packets are marked as OUT with probability p when the average exceeds the target rate. The second approach remembers a short period of time, and packets are marked as OUT with probability p when the average rate exceeds a threshold of βeta (β) times the target rate (β > 1). In general, TSW is quite sensitive to this threshold due to the TCP congestion control mechanism which determines the instantaneous TCP sending rate. In our simulations we follow the second approach. Although TSW mechanism is not recommended in some papers [4], and posterior improvements are not scaleable solutions [7], we will try to provide a better understanding of this mechanism.

```
Initially:
    Winlength=a constant;
    Avg_rate=connection target rate;
    T_front=0;
Upon each packet arrival:
    Bytes_in_TSW=Avg_rate*Winlength;
    New_Bytes=Bytes_in_TSW+pkt_size;
    Avg_rate=New_bytes/(now-T_front+Winlength);
    T_front=now;
```

Figure 7. TSW implementation

$$p = \frac{\text{Average\_rate} - \text{Target\_rate}}{\text{Average\_rate}} \qquad (1)$$

3.2.1. βeta dependency with window length

The first question to face up in TSW is the correct election of βeta and window length (*Winlength)* to see their impact on the achieved IN throughputs. In our simulations we use the same example with eight TCP sources described in the earlier section. We found some common characteristics from these simulations:

a) The larger the window length the higher the number of packets marked as IN.
b) The greater the βeta value the higher the number of packets marked as IN.
c) There is a relative stability in the achieved profiles for window lengths ranging from 160 to 200 ms, and βeta between 1 and 2.

Points a) and b) are too generic, and the irregular deviations around the contracted target due to the RTT variations and the burstiness of the TCP flows make very difficult to figure out the best parameter selection. Figures 8 and 9 depict the achieved IN rates in bps for sources 0 and 6 respectively, as a function of the window length in ms and for different values of βeta. Apparently it is hard to know what values to choose, since a good selection for a specific target rate is not for others.

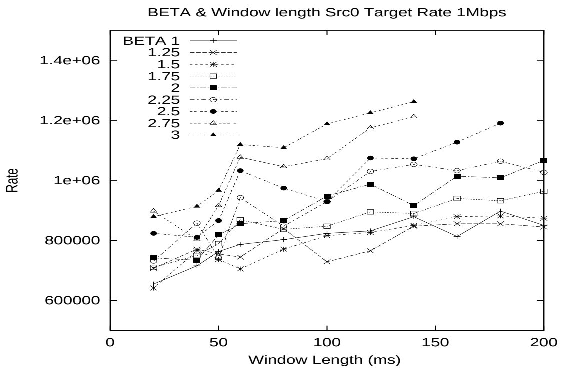

Figure 8. βeta vs. Window Length

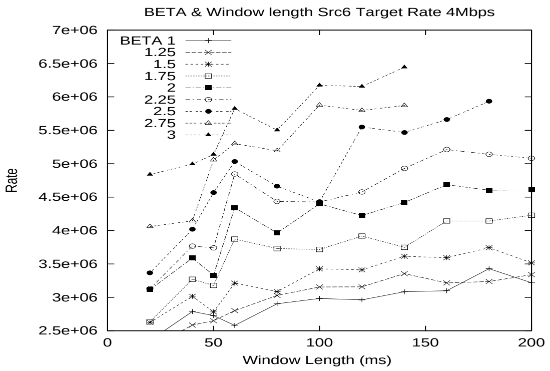

Figure 9. βeta vs. Window Length

On the other hand, c) leads us to perform new simulations with the topology shown in Figure 1, varying the number of sources and their contracted rates. We selected a window length of 200 ms (stability zone) and different RTT. With these new results, we found an easy way to select the value of βeta depending on the target rate of the source. In Figure 10 we show the recommended values for βeta when RTT is lower than 50 ms. Table 3 presents an example with four sources, where we observe the achieved rates for IN packets if βeta and window length are selected as indicated above.

To study what happens when these parameter are not properly established, we conducted two simulations with eight sources (0 to 7) and targets of 1, 1, 2, 2, 3, 3, 4 and 4 Mbps respectively. In the first simulation, we depict values for the source number 7 in Figure 11,where *Winlength* is 30 ms and βeta is 2.5 for all sources. In the second one, we show also values for the source number 7 in Figure 12, where *Winlength* is 200 ms and βeta values are given depending on the targets as shown in Figure 10. Note the substantial discrepancy between both Figures.

From our simulations, we also found that as the RTT growths (RTT > 40ms), achieved IN packets throughput decreases strongly for sources with high target rates (target > 12Mpbs). The contrary effect occurs for sources with low target rates, but these increments are much smaller. Consequently, in the former situation βeta values should be modified (+0.25 approximately), while in the latter βeta can remain in its original value. Table 4 shows an example when we multiplex two sources with target rates of 1Mbps and 16Mbps. We appreciate that the achieved IN throughput improves if we increase βeta in 0.25.

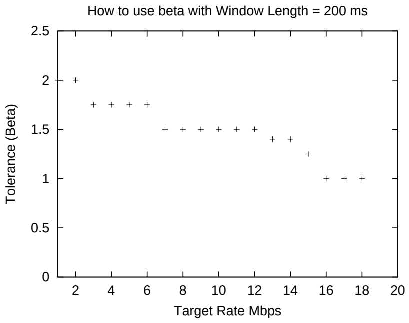

Figure 10. Approximate estimation of how to select βeta values

Table 3. TSW with proper βeta values and window length

First Case (distinct RTTs;low RTT for low targets)				
H	**RTT**	**β**	**R_T**	**IN Throughput**
0	10	2	1	1.00056
1	20	1.75	4	4.35864
2	30	1.5	8	8.38728
3	40	1.4	14	13.7385
///				
Second Case (distinct RTTs;high RTT for low targets)				
0	40	2	1	1.02432
1	30	1.75	4	4.49592
2	20	1.5	8	8.44800
3	10	1.4	14	14.5648

Table 4. Variation of βeta due to the RTT effect

First Case (RTT >60ms for source with target>8Mbps)				
H	**RTT**	**β**	**R_T**	**IN Throughput)**
0	10	2	1	1.10088
1	**50**	**1**	**16**	**13.11552**
///				
Second Case (RTT <60ms for source with target>8Mbps)				
0	50	2	1	1.04544
1	**10**	**1**	**16**	**15.92712**
///				
First Case with β modified				
0	10	2	1	1.12992
1	**50**	**1.25**	**16**	**16.42344**

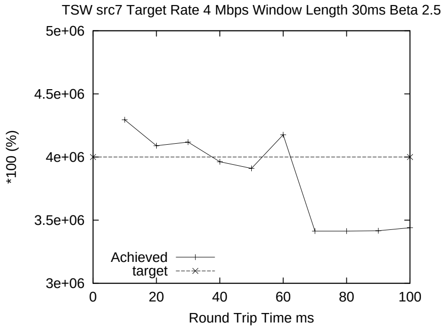

Figure 11. Example of bad performance with not proper βeta

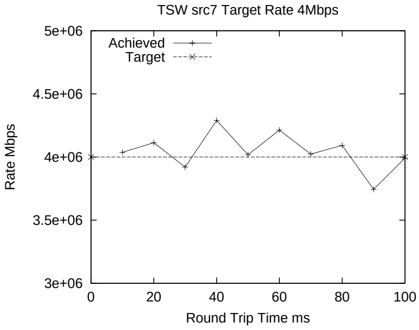

Figure12. Example of better performance with proper βeta

3.2.2. Achieved rates for IN packets and excess bandwidth distribution

Given the same example simulation of eight connections with contracted rates of 1, 1, 2, 3, 3, 4, and 4 Mbps each, we set a window length of 200 ms for all sources, and βeta equal to 2 for sources 0 to 3, and 1.75 for sources from 4 to 7. Hence we compare the performance of this mechanism with the Leaky Bucket.

In Figures 13 and 14 for sources 1 and 6 respectively, we plot the achieved IN throughput as a function of the RTT (from 10 to 100 ms). We can observe that IN packets rate achieved is more irregular than in Leaky Bucket. These figures correspond to the example where all sources have the same RTT. The performance is also good for connections with different RTT each (see Table 5 and 6).

Comparing the results for IN packets throughput in the TSW and Leaky Bucket, we conclude that the Leaky Bucket performance is better than TSW.

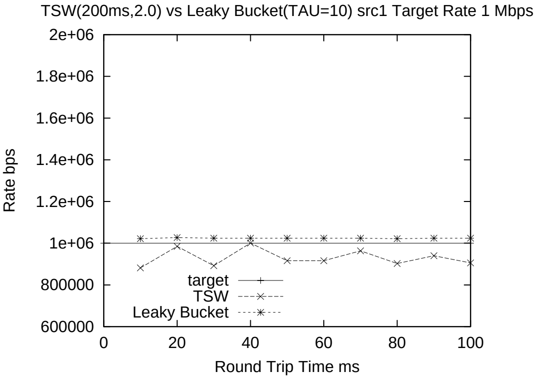

Figure 13. Achieved throughput for IN packets with target of 1Mbps

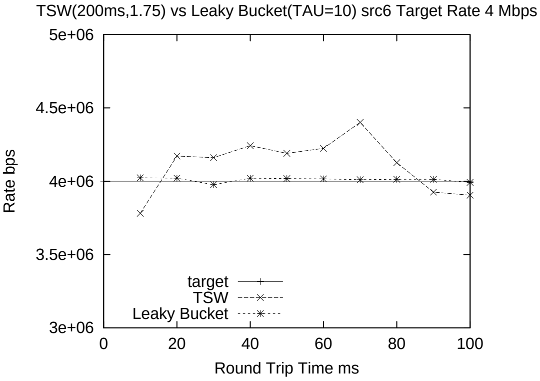

Figure 14. Achieved throughput for IN packets with target of 4Mbps

Regarding the bias of the distribution of the excess bandwidth, we distinguish two cases, same RTT for all sources and different RTT for each source. In the former case, results are worse than those obtained for the Leaky Bucket algorithm. There is more disparity between sources. In the latter case, the distribution tends to be slightly fairer than results got with the Leaky Bucket algorithm. Therefore, there is a compromise when the RTT differs greatly among sources between getting strict IN rates or obtaining a fairer bandwidth sharing.

From our results, we advocate for using Leaky Bucket algorithm if the RTT among sources does not differ severely. Furthermore, TSW is difficult to implement because of the bursty nature of TCP flows and the great amount of parameters and dependencies of this algorithm.

Table 5. TSW with distinct RTT among sources

H	RTT	β	R_T	IN packet rate	Total rate	Excess BW
0	10	2	1	0.992649	2.724480	1.731831 (13.3%)
1	20	2	1	0.817600	2.587200	1.769600 (13.7%)
2	30	2	2	2.053920	3.764640	1.710720 (13.1)
3	40	2	2	2.059200	3.438376	1.379176 (10.7)
4	50	1.75	3	2.876040	4.466800	1.590760 (12.2)
5	60	1.75	3	2.914560	4.276800	1.362240 (10.5%)
6	70	1.75	4	3.780480	4.913040	1.132560 (8.7)
7	80	1.75	4	4.036560	5.385600	1.349040 (10.4)
		Total	**20**	**19.5310099**	**31.556936**	**12.025927 (92.6%)**
//						
H	**RTT**	**β**	**R_T**	**IN packet rate**	**Total rate**	**Excess BW**
0	80	2	1	1.005840	1.950960	0.945120 (7.3%)
1	70	2	1	0.979440	2.101440	1,122000 (8.6%)
2	60	2	2	2.098800	3.252480	1,153680 (8.9%)
3	50	2	2	2.173600	3.226080	1,052480 (8.1%)
4	40	1.75	3	3.173280	4.510889	1,337609 (10.3%)
5	30	1.75	3	3.165360	4.408800	1,243440 (9.5%)
6	20	1.75	4	4.538160	6.016560	1,478400 (11.4%)
7	10	1.75	4	4.271520	6.206640	1,93512 0(14.9%)
		Total	**20**	**21.406000**	**31,673849**	**10,267849 (79%)**

Table 6. TSW with same RTT for all sources

H	RTT	β	R_T	IN packet rate	Total rate	Excess BW
0	50	2	1	1.000560	2.647920	1,64736 (12.7%)
1	50	2	1	0.916080	2.766720	1,85064 (14.2%)
2	50	2	2	2.043360	3.461040	1,41768 (10.9%)
3	50	2	2	2.027520	3.260400	1,23288 (9.5%)
4	50	1.75	3	2.962080	4.131600	1,16952 (9.0%)
5	50	1.75	3	2.938320	4.031280	1,09296 (8.4%)
6	50	1.75	4	4.189680	5.375911	1,186231 (9.1%)
7	50	1.75	4	4.018080	4.994880	0,9768 (7.5%)
		total	**20**	**20,09568**	**30,669751**	**10,574071(81.5%)**

3.3. Counters Based Algorithm

The third approach discussed in this paper is the Counters Based algorithm. The advantage of using this new proposal lies on the possibility of *reusing* the bandwidth for IN packets that has not been used before, which is not possible with the Leaky Bucket. In addition, observe that results depend only on the round trip time which clearly make the mechanism easier to configure.

To implement this algorithm we only need two counters, C1 and C2. C1 is initialized to one, so that first packet is always marked as IN, and C2 to the inverse of the contracted rate in units of time. At each unit of time, C2 decrements by one unit. If C2 is zero, C1 increments one unit and C2 takes its initial value. If there is a packet arrival, we check the value of C1. If it is greater than zero then the packet is marked as conforming (IN) and C1 decrements by one unit, otherwise is marked as non conforming (OUT). The pseudocode of this new profiler and its flowchart is shown in Figure 15.

```
Initially:
    Counter1=1;
    Counter2=Link/Target;
For each unit of time:
    Counter2--;
    if Counter2<=0
        {
        Counter1++;
        Counter2=Link/Target;
        }
    if packet arrival
        if Counter1>0
            Mark packet as IN
        else
            Mark packet as OUT
```

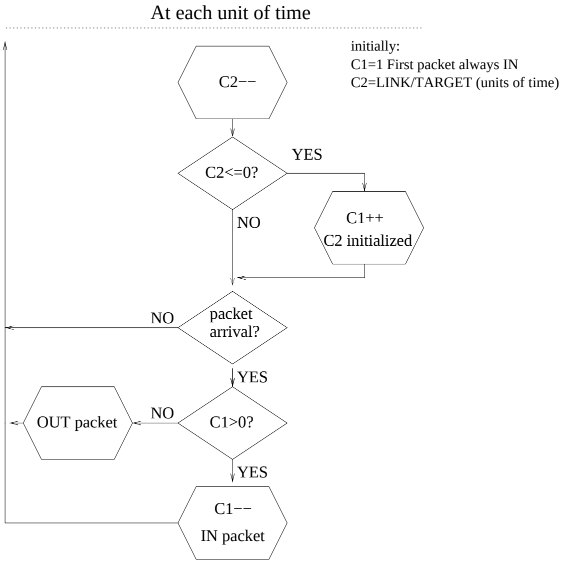

Figure 15. Counters Based implementation and flowchart

3.3.1. Achieved rates for IN packets and distribution of excess bandwidth

With this approach simulation results show that the IN throughputs achieved by all sources are exactly the established target rate with hardly noticeable variations (less than 1%). Moreover, those tiny variations in the IN packets are not comparable with variations produced in the Leaky Bucket algorithm, being much smaller in our Counter Based profiler. Since TSW got worse results in terms on dependency of IN packets rates with RTT than Leaky Bucket, it is obvious that the Counter Based mechanism is also better than TSW. See in figures 16 to 18 the comparison of achieved IN throughputs with the target rate for sources 1, 4, and 6 in the same example of eight TCP sources, for different values of RTT having all sources the same RTT. Simulations with different RTT for each source have shown that achieved IN packets throughputs keep on strictly the contracted target rates irrespective of the value of RTT.

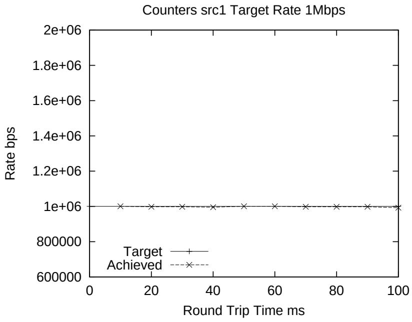

Figure 16. Achieved rate for IN packets

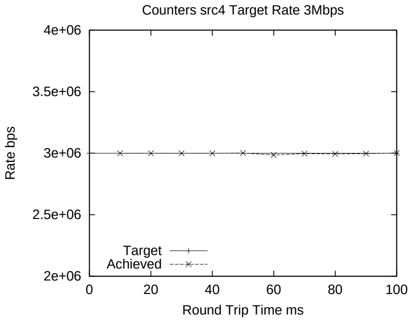

Figure 17. Achieved rate for IN packets

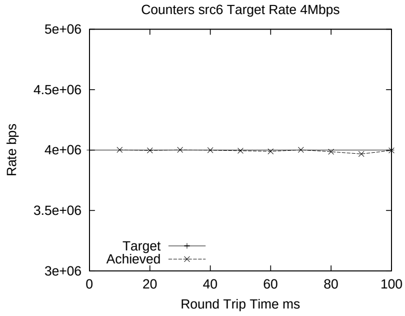

Figure 18. Achieved rate for IN packets

As regard of the excess bandwidth distribution there is a lack of fairness, (see Tables 7 and 8). As commented earlier, this is due to the RTT effect in situations where each source has a different RTT. Those streams with lower RTTs are favored, receiving more bandwidth sources with smaller targets. This fact also occurred in the Leaky Bucket and TSW algorithms, but TSW responds better under these circumstances if we set properly the parameters.

Nevertheless, when all sources have the same RTT, the excess bandwidth is distributed equitably among the sources, being sources with lower RTT and small targets propitiated. Results are even better than the obtained for the Leaky Bucket (for RTT < 50ms) and TSW. The difference in the sharing of the remaining link bandwidth does not exceed 12% in the worst case, and decreases to 4% in the best situation. Comparing Figure 19, where we show the sharing of the remaining bandwidth, with Figures 5 and 6 the Counters Based profiler has a fairer distribution.

Therefore, there is again a compromise to chose between an even distribution of excess bandwidth or achieving successfully the contracted profiles. Considering that the sharing of the excess bandwidth is never fulfilled completely (even in the best case there will always be a difference in the distribution of the bandwidth), we result that the use of the Counter Based mechanism is the best option among the three studied in this work as long as we exclude scenarios where there are big differences among RTT of different connections (difference > 40 ms). Regarding the bandwidth sharing, the TSW scheme behaves better than the other mechanisms when difference > 40ms, however the contracted target rate is difficult to achieve.

Table 7. Counters with distinct RTT each source

H	RTT	R_T	IN packets rate	Total rate	Excess BW
0	10	1	1.000560	3.833280	2,832720 (21.8%)
1	20	1	1.000560	3.210240	2,209680 (16.99%)
2	30	2	1.998480	3.887743	1,889263 (14.53%)
3	40	2	1.998480	3.524400	1,525920 (11.74%)
4	50	3	3.001680	4.258320	1,256640 (9.67%)
5	60	3	2.999040	3.994320	0,995280 (7.65%)
6	70	4	3.996960	4.802160	0,805200 (6.19%)
7	80	4	3.965280	4.678080	0,712800 (5.48%)
	total	20	**19,961040**	**32,188543**	**12,227503 (94.06%)**
//					
H	**RTT**	**R_T**	**IN packets rate**	**Total rate**	**Excess BW**
0	80	1	1.000560	2.143680	1,143120 (8.79%)
1	70	1	0.997920	2.637360	1,639440 (12.61%)
2	60	2	1.998480	3.188975	1,190495 (9.16%)
3	50	2	1.977360	3.223440	1,246080 (9.58%)
4	40	3	3.001680	4.279440	1,277760 (9.83%)
5	30	3	2.999040	4.804800	1,805760 (13.89%)
6	20	4	4.002240	5.710320	1,708080 (13.14%)
7	10	4	3.999600	6.388800	2,389200 (18.38%)
	total	20	**19,976880**	**32,376815**	**12,399935 (95.39%)**

Table 8. Counters with same RTT all sources

H	RTT	R_T	IN packets rate	Total rate	Excess BW
0	50	1	1.000560	2.879897	1,879337 (14.45%)
1	50	1	1.000560	2.529120	1,528560 (11.76%)
2	50	2	1.998480	3.495360	1,496880 (11.51%)
3	50	2	1.998480	3.603600	1,605120 (12.35%)
4	50	3	3.001680	4.250400	1,248720 (9.60%)
5	50	3	3.001680	4.530240	1,528560 (11.76%)
6	50	4	3.994320	5.381218	1,386898 (10.67%)
7	50	4	3.994320	5.306400	1,312080 (10.09%)
	total	20	**19,990080**	**31,976235**	**11,986155 (92.20%)**

.

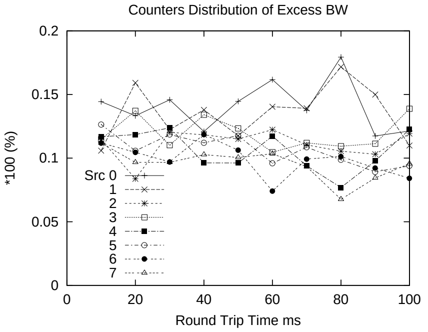

Figure 19. Distribution of excess bandwitdth in Counters based system for all sources with same RTT

4. Conclusions

In this paper, we have performed a comparative simulation study of different types of profilers to provide the Assured Service guarantees. TCP input traffic sources have been used. In this scenario, we have explored the right configuration of the main parameters associated to both the TSW and the Leaky Bucket algorithms. Likewise, we have presented a new profiler mechanism based on Counters.

The Leaky Bucket accomplishes good performance in terms of achieved contracted rates for *TAU* in the [0, 10] interval. Depending on how permissive we want to establish the IN packets rate, we select 0 (more strict) or 10(more tolerant). In the last case, not only all sources would be slightly above their profiles, but the sharing of the excess link bandwidth is fairer than with a *TAU* value of 0. Distribution of remaining bandwidth is quite equitable when the RTT of the sources are similar. However, for extreme situations where connections have very different RTT (they differ more than 60 ms approximately), sources with greater targets and higher RTT will not achieve their profiles, and there will be an odd sharing of the excess bandwidth. This drawback may be due to the TCP characteristics. A possible solution would be using a TCP implementation with a better congestion mechanism like selective acknowledgement.

For the TSW profiler, we have found an easy way to select the parameter βeta for each TCP source depending on the target, so that TSW fulfill an acceptable performance in terms of achieved contracted profiles. Nevertheless, IN packets rates in TSW present higher variations around the target than in Leaky Bucket or Counters Based mechanisms for both situations, identical RTT and different RTT. However, the distribution of the excess bandwidth is fairer in TSW than in Leaky Bucket or Counters when RTTs differ among sources. TSW complex implementation is an important factor to take into consideration when choosing what policing mechanism is better.

The Counters Based mechanism, is a new proposal presented in this paper. The implementation is based on two counters that keep track of the target rate of the sources. It achieves a very good performance in terms of contracted rates even better than the Leaky Bucket profiler. Regarding the bias of the distribution of the excess bandwidth, Counters Based scheme presents a fair sharing except for extreme situations. Due to the simplicity of the Counters Based algorithm and its excellent performance, we consider it as a new mechanism to take into account in future works about differentiated services.

ACKNOWLEDGEMENTS

This work was partly supported by the Spanish Research Council under grant TIC2000-1734-C03-03.

REFERENCES

[1] F. Cerdan, J. Malgosa, J. Garcia-Haro, F. Monzo, F. Burrull, “Providing QoS to TCP/IP traffic: an Overview” PROMS 2000, pp.91-99 ISBN 83-88309-05-6, October 2000.

[2] J. Heinanen, F. Baker, W. Weiss and J. Wroclawski, “Assured Forwarding PHB Group” RFC 2597.

[3] D. Clark and W. Fang, “Explicit Allocation of Best-Effort Packet Delivery Service” IEEE/ACM Transactions on Networking, VOL 6 No. 4 August 1998.

[4] J. Ibanez, K. Nichols, “Preliminary Simulation Evaluation of an Assured Service” draft-ibanez-diffserv-assured-eval-00.txt, 1998.

[5] F.Cerdan, O. Casals,”Mapping an Internet Assured Service on the GFR ATM Service” Lecture Notes in Computer Science 1815 (Networking 2000), pp.398-409 ISBN/ISSN: 3-540-67506-X/0302-9743. Ed Springer-Verlag, May 2000.

[6] S. Floyd and V. Jacobson “Random Early Detection Gateways for congestion Avoidance” IEEE/ACM Transactions on Networking, August 1993.

[7] Lin W, Zheng R and Hou J, “How to Make Assured Services More Assured” Proceedings of ICNP, Toronto, Canada, October 1999.