Cycle-based TCP-Friendly algorithm

Omar Ait-Hellal*, Lidia Yamamoto, Guy Leduc
Research Unit in Networking
Institut Montefiore, B28
Université de Liège, B-4000, Liège 1, Belgium.
E-mail: {oaithel, yamamoto, leduc}@run.montefiore.ulg.ac.be

Topic: Global Internet

Abstract

Several TCP-Friendly algorithms have been recently proposed to support multimedia applications. These algorithms try to mimic the congestion control behavior of TCP. However, the oscillatory (bursty) nature of TCP traffic is widely known to be unsuitable for many real time applications. This behavior often results in annoying QoS oscillations for the users. In the present paper we describe a new TCP-Friendly algorithm based on the TCP cycle estimation. We show through simulations that the proposed algorithm is able to smooth the oscillations while keeping fairness towards TCP. We also present a refinement for the parameters of Mathis' formula, which takes the round trip time variation into account.

keywords

TCP, TCP-Friendly, real-time, congestion control, Internet.

*Corresponding author. Tel.: +32 4 366 2699. Fax: +32 4 366 2989

1 Introduction

Adaptive mechanisms for congestion control have a central role in the efficient sharing of the network resources among a large number of users. These mechanisms also have the role of preventing congestion in the network [2, 4]. The fact, however, that the control is performed by the sources (that are not policed by the network), and not by the network, makes it hard to protect the network from applications that might not use such mechanisms (e.g. from video conferences that use UDP with no rate adaptation). Hence, not only applications that adapt their rates suffer from congestion, but also those that do not adapt their rates, since they experience high loss rate, and therefore a poor QoS.

With recently proposed mechanisms such as RED [3], multimedia applications have to control their sending rates in order to maintain a reasonable loss rate. Taking layered video for instance, it is often preferable to receive part of the video stream (a limited number of layers) with a small loss rate, than a complete stream with a high loss rate.

Considerable research is being carried out in order to add rate control to real-time multimedia applications, in such a way to make them cooperative with TCP. This kind of rate control schemes are referred to as TCP-Friendly [11].

In this paper we propose an algorithm that dynamically adapts its rate based on an estimation of the TCP cycle duration. Indeed, by observing the loss rate and the round-trip time, it is possible for a real-time application to estimate the duration of a cycle, and therefore the loss ratio that an equivalent TCP source would experience under the same network conditions. The cycle estimation, the computed loss, and the observed loss can then be used to calculate a smoothed TCP-Friendly rate.

The paper is structured as follows: in section 2 an overview on the related work to TCP-Friendly algorithm is given. A refinement of the $\frac{1}{\sqrt{p}}$ formula [11, 13], for the rate estimation is described and proved. In section 3, we propose a new TCP-Friendly algorithm based on the cycle estimation. We present several simulations to show the convergence and the fairness of the proposed algorithm in section 4. Finally, we conclude by some remarks and future work in section 5.

2 Related work

A lot of research effort is currently being dedicated to TCP-Friendly schemes for real-time traffic [11, 13, 14]. They consist in algorithms that mimic the congestion control behavior of TCP for real-time traffic. In this section we describe some of them. Since many of the schemes proposed, including our own, make use of the Real-Time Protocol (RTP) and its control companion RTCP [17], we start with a brief description of these protocols.

2.1 Feedback information

Internet real-time multimedia applications generally use the Real-Time Transport Protocol (RTP) to transport their data. RTP [17] (data) packets contain header information that helps these applications to perform a number of typically needed tasks, such as payload type identification, packet sequence reordering, media synchronization, etc. RTP is accompanied by RTCP, a control protocol (RTCP) designed to monitor RTP sessions in a light-weight and scalable manner. RTCP (control) packets carry, among other information, receiver reports (RR) that include QoS-related feedback information such as the experienced packet loss, delay and jitter. These reports are sent to all session members, and can then be used for a variety of tasks such as QoS adaptation, reception quality statistics, network management.

In order to scale to large multicast sessions it is recommended that the total amount of RTCP traffic in a session be kept as a fraction of the to-

tal session traffic (less than 5 %). Therefore the amount of RTCP information that each session member is allowed to produce must be inversely proportional to the number of members in the session. To achieve this, each session member listens to the reports from other members in order to estimate the group size, and adjusts its report sending rate accordingly. As a result, RTCP reports from a given participant are generated less often as the total number of session participants grows.

TCP-Friendly schemes can use the feedback information in RTCP for different purposes such as: to adapt the sending rate [18]; to coordinate among different receivers in order to choose how many video layers to join [21, 22]; to take transcoding decisions in a distributed way [8]. The authors of these schemes have pointed out difficulties in the use of RTCP. Reports may be lost, the frequency of RTCP reports may vary during the session, not all the necessary or convenient fields are present. Some authors prefer to rely on other types of feedback, not based on RTCP [14, 16, 20].

2.2 TCP-Friendly: Mathis' formula and its refinement

An equation for computing the rate of an equivalent TCP connection is given in [11, 13]:

$$B = \frac{C \cdot MTU}{\bar{R} \cdot \sqrt{p}} \qquad (1)$$

where B is the computed rate, p is the loss rate, C is a constant given as $C = 1.30$ in [13] and $C = 1.22$ in [11], MTU is the mean segment size and $\bar{R}$ is the average round trip time. Hereafter, using previous results on the average round trip time of an ideal TCP Reno [1], a constant $C = 1.27$ is computed.

Indeed, for TCP it is well known that the average round trip time over a given window size, is approximately $\overline{R}_W = \frac{W}{B}$, where B is the available bandwidth for the TCP connection. Without loss of generality, we consider that B is given in segment per unit of time, and the MTU size is 1 segment. A cycle in TCP begins by a window size of $\frac{W_{max}}{2}$ and ends when the window reaches its maximum size namely W_{max}. Hence, given that the total number of segments sent during a cycle is $N = \frac{3}{8}W_{max}(W_{max} + 2)$, the average round trip time over a cycle ($\overline{R}$) can be then approximated by [1]

$$\overline{R} = \sum_{i=\frac{W_{max}}{2}}^{W_{max}} \frac{i}{N} \cdot \overline{R}_i = \sum_{i=\frac{W_{max}}{2}}^{W_{max}} \frac{i}{N}\frac{i}{B} \approx \frac{7}{9}\frac{W_{max}}{B}$$

Hence the average window [1] size ($\overline{W}$) is given by

$$\overline{W} = \overline{R} \cdot B \approx \frac{7}{9}W_{max}$$

Since in an ideal behavior of TCP Reno, only one loss occurs during a cycle, then the loss ratio is given by

$$p = \frac{1}{N} \approx \frac{1}{\frac{3}{8}W_{max}^2} = \frac{8}{3}\left(\frac{7}{9}\right)^2 \frac{1}{\overline{W}^2}$$

$$\Longrightarrow \qquad B \approx \frac{1.2701}{\overline{R} \cdot \sqrt{p}}. \qquad (2)$$

Our algorithm (see section 3) uses this formula in a different way, in order to estimate the (smoothed) rate that an equivalent TCP connection would use.

Equation (1) taken as it is, does not apply for loss rates larger than 0.16 [11], nor does it apply for connections with a large round trip time (see figure 1), since TCP in this case cannot fully use the available bandwidth and equation (1) is based on the assumption that the full utilization is achieved. Therefore, using this equation in this context would result in underestimating the available bandwidth.

Figure 1 shows an example where the TCP-friendly algorithm based on equation (1) does

[1] Average window should be interpreted here as the size of the window of an **equivalent** connection that sends with a fixed window size.

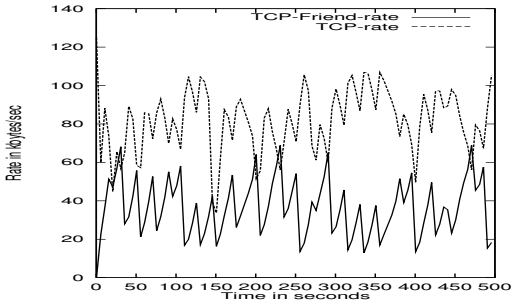

Figure 1: Rate for TCP and TCP-Friendly, for large delay.

not get a fair share with TCP. The simulated TCP-Friendly algorithm works as follows: if the loss reported in the RTCP RR is zero, then the rate is increased in a TCP-like manner, by $TRCTP/(2 \cdot \overline{R}^2)$, where $TRTCP$ is the time separating two RTCP receiver reports, otherwise the rate is computed using equation (2). In this example the TCP-Friendly algorithm shares a bottleneck link of 1 Mbps with TCP Reno, the buffer size is of 40 packets and the round trip time for both connections is 400 ms, the time between two RTCP receiver reports is around 5 seconds. In figure 1 as well as in all the figures of the present paper, the rate is plotted every 5 seconds.

An algorithm derived from equation (1) is presented in [19], where the authors reported a good fairness between the algorithm and a TCP connection, for MPEG-1 video transfer. However, they did not address the oscillation problems of the scheme, neither did they show simulations or experiments in a complex environment.

In [14] Towsley et al. propose a TCP-Friendly scheme, that is based on previous analytical work [15] in which the timeout is taken into account. They proposed an algorithm in which the rate is multiplied by two when there are no losses (slow-start); otherwise it is calculated as function of RTT, a timeout associated with RTT, and the maximum window size of an equivalent TCP connection (W_{max}). The difficulty of this algorithm resides on the dynamic estimation of W_{max}.

The main drawback of the method in [14] as well as equation (1) for computing the rate, is the resulted oscillatory behavior. They mimic TCP too closely, and as a consequence, are not well adapted to typical real-time transmissions (e.g. audio, video). To solve the oscillations problem, many algorithms have been proposed. Below, we describe some of them.

2.3 TCP-Friendly and smoothness

In the Loss-Delay Based Adjustment Algorithm (LDA) [18], the sender estimates the TCP-friendly bandwidth given the packet loss rate, round-trip time (RTT) and the bottleneck bandwidth calculated from feedback information reported by each receiver. It then adapts its rate to the minimum estimated, by performing an additive increase and multiplicative decrease of the rate.

The LDA authors show that the algorithm is sufficiently efficient and fair. However, only the case of equidistant sources (similar RTT) was taken into account. Figure 2 illustrates an example where LDA gets almost the hole available bandwidth for large RTTs. The parameters of the related simulation are the same as in the previous example (figure 1).

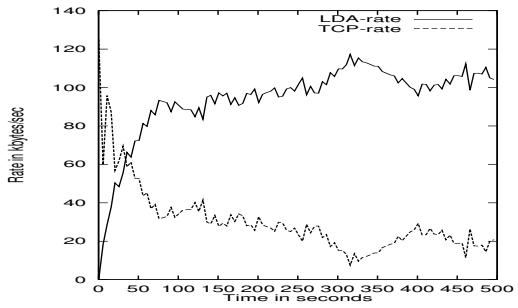

Figure 2: Rate for TCP and LDA algorithm, for large delay.

The second drawback of the LDA algorithm is the bottleneck bandwidth estimation. To this end, LDA uses the "packet pair" method by [21], which could be inaccurate especially for wide-

area networks. In LDA, the choice of the reduction factor (for rate decrease) is very important, since it determines the trade-off between fast adaptation to losses and rate stability; this makes the algorithm difficult to use.

The TCP-Friendly Rate Adaptation Protocol (RAP) [16] is another algorithm based on linear increase and exponential decrease. The algorithm is designed to be typically used by busy media servers which need to deliver a large number of streams to different clients. Such servers cannot afford calculating a TCP-Friendly rate for every source. Therefore, an aggregation scheme is proposed. The strong point of this algorithm is its simplicity of calculation, thus saving CPU load, which makes it especially suited for heavily loaded media servers.

We consider RAP a bit similar to LDA. The difference between them is the estimation of different parameters. In the presented algorithm, the inter-departure time of packets is exponentially decreased or additively increased, depending respectively on whether a positive fraction of loss is reported or not. The simulations presented in the paper do not show enough fairness between RAP and TCP, where at the same time the oscillations are not investigated. Moreover, the algorithm uses a feedback mechanism based on acknowledgments like TCP, which constitute a non negligible overhead, and makes its adaptation to multicast very difficult.

A unicast TCP-friendly algorithm based on RTT thresholds is proposed in [20]. When the RTT observed by a receiver at a given time is above a certain threshold, the receiver calculates a predicted rate by decrementing the current average received rate by a constant value K. When the observed RTT is below the threshold, the predicted rate is given by the average rate plus K. This prediction is then sent back to the data source. There, it is combined with the rate predicted by the TCP throughput model (equation 1), in order to compute a new mixed RTT/TCP-friendly rate which allows an earlier reaction to congestion.

The results from [20] are promising for continuous rate applications, since the proposed algorithm prevents losses and oscillations. However, the simplistic nature of the RTT-based rate prediction needs to be further investigated, mainly for determining the RTT thresholds as well as the increment K. Furthermore, its fairness with respect to TCP is still not completely clear.

3 Proposed algorithm

3.1 Cycle estimation for an ideal TCP Reno

The main motivation of our work comes from the fact that in none of the previous work on TCP-Friendly schemes, the duration of a TCP cycle seems to be taken into account. Indeed, it is well known [1, 9] that the ideal window behavior in TCP is cyclic. The cycle is separated by two consecutive losses, and its duration is proportional to the bandwidth delay (including queuing delay) product. Hence, in the case of large cycles (e.g. due to large buffers), algorithms which do not take this duration into account, will see a zero loss in many reports, before a report carrying a high loss ratio arrives (since losses occur at the end of a cycle). This ratio, in fact does not reflect an equivalent ideal TCP loss ratio, namely one packet every cycle.

In order to have a valid estimation of the loss ratio, the latter should be estimated over a duration of a cycle ($Cycle$). The algorithm we propose below is based on this observation. We first compute the duration of a cycle, then using a weighted moving average, our algorithm computes a mean of the observed loss over this cycle, and adjusts its sending rate accordingly. The loss of one segment over a cycle can be written as:

$$p = \frac{1}{N} = \frac{1}{\frac{3}{8}W_{max}(W_{max}+2)} = \frac{1}{B \cdot Cycle}$$

$$\Longrightarrow Cycle = \frac{\frac{3}{8}W_{max}(W_{max}+2)}{B}$$

$$\approx \quad 0.62 \cdot \overline{R}^2 \cdot B + 0.96 \cdot \overline{R}. \qquad (3)$$

Hence, if the sending rate in the present cycle is B, then the loss ratio for an ideal TCP Reno would be

$$Loss = \frac{1}{B \cdot Cycle} \approx \frac{1.61}{B^2 \cdot \overline{R}^2 + 1.54 \cdot B \cdot \overline{R}}. \quad (4)$$

Using this formula, the Cycle-Based Rate Adaptation Algorithm (CBRAA) that we propose estimates the loss ratio that an ideal TCP Reno connection would experience, and then determines the available bandwidth depending on the observed loss ratio (carried out in RR reports).

3.2 CBRAA: Cycle-Based Rate Adaptation Algorithm

CBRAA is based on the observation that if a given source behaves as Reno TCP, then the observed loss rate should be given by equation (4), where B is the sending rate in packets per second, and referred to in the sequel as *Rate*.

As observed in many simulations, in the case where the rate of RTCP reports is large (time between two reports is smaller than the TCP cycle), increasing the rate only when the instantaneous loss ratio is zero results either in large oscillations and thereby underestimating the available bandwidth as is the case for equation (1) (see figure 1), or overestimating it as is the case for LDA (see figure 2). Thus, the use of the mean loss ratio instead of the instantaneous loss ratio is preferable for both keeping a smoothed rate and getting a good fairness. In our algorithm a weighted moving average (WMA_factor) is used for this purpose (see figure 3). This factor takes the duration of a cycle into account, so that the observed loss ratio is a mean taken over a cycle, which is more accurate than the instantaneous loss rate.

The estimated RTT that we used is the weighted moving average of the RTTs calculated at the source every time it receives an RTCP RR, and the factor of the moving average is the same as the one used for the loss (*Loss*). These RTTs are calculated using the method suggested in the RTP/RTCP specification [17] (LDA [18] also uses this method). It works as follows: each RR contains, for each RTP source, the timestamp (t_{lsr}) carried in the last RTCP Sender Report (SR), and the elapsed time (d_{lsr}) between receiving the last SR and sending the RR; for a source that receives a RR at time t_{rr}, the RTT can be calculated as: $RTT = t_{rr} - t_{lsr} - d_{lsr}$.

For every Receiver Report (RR) do:

```
TRTCP += (new_TRTCP - TRTCP)/8.0;
WMA_factor = min(TRTCP/Cycle, 0.95);
srtt += (rtt - srtt)*WMA_factor;
Loss += (RR->loss - Loss)*WMA_factor;
Cycle = srtt*(0.62*Rate*srtt + 0.96);
Loss_th = 1.0/(Rate*Cycle);

Rate_th = Rate;
if (Loss < 0.5*Loss_th)
  Rate_th = 1.27/(rtt*sqrt(
   alpha* Loss_th + (1 - alpha)*loss));
else if (Loss > 1.5*Loss_th)
  Rate_th = 1.27/(rtt*sqrt(
   beta*Loss_th + (1 - beta)*loss));

Rate = gamma*Rate + (1 - gamma)*Rate_th;
```

Figure 3: CBRAA: Cycle-Based Rate Adaptation Algorithm

Figure 3 gives a pseudo-code of CBRAA. To explain how the CBRAA algorithm works, assume that at a given time a CBRAA source sends its data (frames) at the rate *Rate*. If *Rate* is the adequate sending rate i.e. the same bandwidth of an equivalent TCP connection, then the observed loss ratio will be close to the theoretical one, namely $Loss\_th = \frac{1.61}{Rate^2 \cdot srtt^2 + 1.54 \cdot Rate \cdot srtt}$, where *srtt* is the average round trip time, and therefore the rate should be kept unchanged in this case. We considered that the loss ratio is close to the "ideal" loss ratio, if the former is between half and one and half the ideal loss ($0.5 \leq \frac{Loss}{Loss\_th} \leq 1.5$).

Now if *Rate* is less than what a TCP Reno

connection would use, then the CBRAA source will experience a loss rate smaller than $Loss\_th$, and thus its rate should be increased. Increasing the rate in a TCP-like manner would result in huge oscillations if the frequency of RTCP RRs is large, and therefore a poor utilization of the link. Indeed, during a time separating two RTCP RRs ($TRTCP$ in figure 3), TCP Reno would increase its rate by approximately $\frac{TRTCP}{srtt^2}$, since the window size is increased by one segment every RTT, which corresponds to the increase of the rate by $\frac{1.0}{srtt}$ every round trip time. TRTCP is recommended to be quite large [17], since RTCP traffic should not exceed 5% of the session bandwidth, and it increases with the size of the group in the multicast case. Hence, $\frac{TRTCP}{srtt^2}$ can go beyond the rate itself, and can result in an aggressive algorithm. In fact this is a common drawback, of the linear increase exponential decrease algorithms in general, since using a small increase factor would result in a slow convergence, and a high factor would result in high loss rate, while the full utilization of the network capacity might not be guaranteed.

CBRAA uses equation (1) differently for the increase and the decrease as well, so that the problem of zero loss is avoided and the increase as well as the decrease in the rate, is inversely proportional to the observed loss. Hence, when the observed loss tends to the theoretical loss ratio, the increase and the decrease in the rate tend to zero. To compute the rate, the following formula is used when the loss is small enough ($Loss \leq \frac{0.5}{Loss\_th}$)

$$Rate\_th \triangleq \frac{1.27}{srtt\sqrt{\alpha\ Loss\_th + (1-\alpha)\ Loss}} \quad (5)$$

and the same formula with another smoothing factor β, instead of α is used when the loss is large $Loss > \frac{1.5}{Loss\_th}$. This allows us to give more importance for the decrease than the increase in the rate.

It is easy to check that $Rate\_th$ is increasing while $Loss$ is smaller than $Loss\_th$ and decreasing otherwise. The use of a moving average for the rate i.e. $Rate = \gamma \cdot Rate + (1-\gamma) \cdot Rate\_th$ allows us to get a smoothed rate. In order to avoid affecting the convergence severly, γ should be kept small enough (less than 0.5). The optimal value of γ in fact depends not only on α and β but also on the tradeoff we want to get between the smoothness (stability) and the convergence. Our work is still in progress to find optimum values for the coefficients α, β and γ.

The choice of different parameters (α, β see figure 3) for the increase and the decrease, may help to get a rapid convergence of the algorithm and to avoid oscillations. Large values for α result in a slow convergence and a smoothed rate, while small values result in a fast convergence but an oscillatory rate.

In all cases the increase $f(\alpha)$ (or the decrease $f(\beta)$) in the rate can be approximated by (since $Rate \approx \frac{1.27}{srtt\sqrt{Loss\_th}}$)

$$\begin{aligned} f(\alpha) &\approx |\gamma \cdot Rate + (1-\gamma) \cdot Rate\_th - Rate| \\ &\approx \left| \frac{1-\gamma}{\sqrt{\frac{\alpha \cdot Loss\_th + (1-\alpha) \cdot Loss}{Loss\_th}}} - 1 + \gamma \right| Rate. \end{aligned}$$

Taking $\gamma = 0.3$ and $\alpha = \beta = 0.5$ as an example, we can easily check that the increase rate varies between $0.30 \times Rate$ when there are no losses, and $0.1 \times Rate$ when the loss rate equals $0.5 \times Loss\_th$. This is not a very aggressive algorithm, since if there are no losses at all (for a long period, remember a moving average on the loss) then increasing the rate by 30% its value is not so aggressive, since the rate is likely to be small. On the other hand, when the loss varies between $1.5 \times Loss\_th$ and, say, $3 \times Loss\_th$, the decrease in the rate varies between $0.07 \times Rate$ and $0.21 \times Rate$ respectively. This is a desired behavior since if there are losses then it means that the rate of CBRAA is high and reducing it by 7% (for a loss lightly larger than $Loss\_th$) is sufficient, since the loss ratio is close enough to the ideal loss.

In the next section, we present several simulations to show how CBRAA behaves comparatively to TCP Reno and to a TCP Friendly that uses directly equation (2). From now on, we call TCP-Friend the algorithm based on equation (2) which works as follows: no smoothing in the loss ratio is done, and if the loss reported by RTCP RRs is zero then the rate shall be increased in the TCP-like manner, by $\frac{TRTCP}{2 \cdot srtt^2}$ which corresponds to half what would be the increase in TCP Reno. If the reported loss is larger than zero then TCP-Friend, use equation (2) to compute its rate.

4 Simulations

In order to study the behavior of CBRAA, we compared it with TCP Reno and with the original TCP-Friendly by Mathis as described in Section 2.2. We performed a set of simulations, in order to observe the basic, individual behavior of the algorithm in terms of convergence and fairness, and to show how CBRAA behaves for non-equidistant sources (i.e. sources that observe different RTTs).

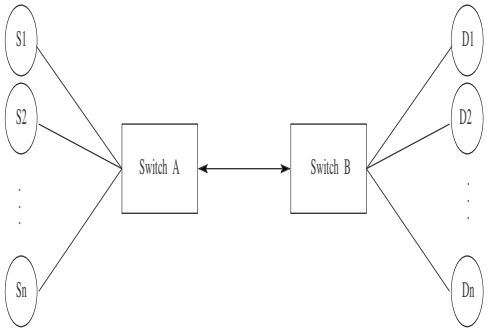

Figure 4: Simulation setup

We used the REAL network simulator [7] to simulate the model described in figure 4. In this model, we have n sources (S_i) sending to n destinations (D_i), and sharing the same bottleneck link of capacity 4 Mbps. All the other links have a capacity of 10 Mbps. All links are bidirectional and symmetric, and the end-to-end propagation delay is 50ms in each direction. The buffer size in the bottleneck access router (router A in the figure) is 20000 bytes. Each source is either TCP Reno, Mathis TCP-Friendly or CBRAA. All sources send packets of constant size 1000 bytes. For all the simulations using this scenario we considered $\alpha = \beta = 0.5$, and the minimum rate for each TCP-Friendly or CBRAA source is set to $10 \times packet\_size$ bps.

4.1 CBRAA and equation (1)

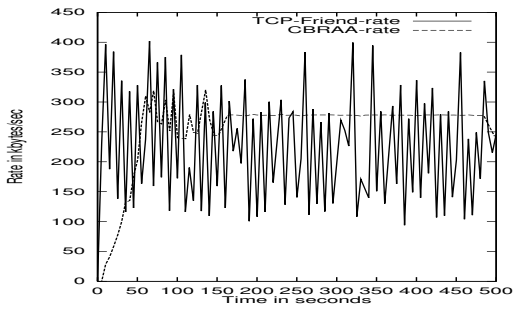

Figure 5: Rate for Mathis ("TCP-Friend") and CBRAA sources, the average RTT is 120 ms.

Consider a TCP-Friendly algorithm based on Mathis formula as described in Section 2.2. Our aim in this section is to show how CBRAA behaves comparatively to the direct use of equation (1). We used the scenario of Figure 4 with one TCP Reno and one Mathis TCP-Friendly source. The simulation time is 500 seconds. After that, we replaced the Mathis source with a CBRAA source, and ran the simulation for another 500 seconds. The initial rate for the CBRAA source is set to 10 packets per second (80kbps).

In figure 5, we plot the goodput for the Mathis TCP-Friendly source and the CBRAA source. We can see that after 60 seconds the rate of the CBRAA source is very smooth, while that of the Mathis source remains oscillatory. CBRAA gets approximately 2Mbps while TCP-Friend source gets slighly less than the faire share.

With respect to the convergence speed, in the example, after 60 seconds of simulation CBRAA reaches its fair share of 2Mbps. In fact, the con-

vergence speed in CBRAA depends on the combined effect of the parameters α, β and γ. The larger α is, the slower the increase in the rate is, and similarly, the larger β is, the slower the decrease in the rate is. Finally, for large values of γ the convergence is very slow.

4.2 Reaction to arriving and leaving sources

In the present section we consider the reaction of CBRAA to the leaving and arriving sources.

Consider the model shown in Figure 4, where at the beginning only one TCP source and one CBRAA source are competing for 4Mbps. At time 300 seconds, two TCP connections arrive. Figure 6 shows how CBRAA reacts to the arriving sources. In approximately 10 seconds (two RTCP RRs) each connection gets almost the same fair share (1 Mbps). The two arriving TCP sources leave the network after transmitting 30Mbytes each, respectively at time 550 and 560. CBRAA regains a rate of 1.6 Mbps (which is close to its fair share) in less than 20 seconds after the two TCP connections leave the network.

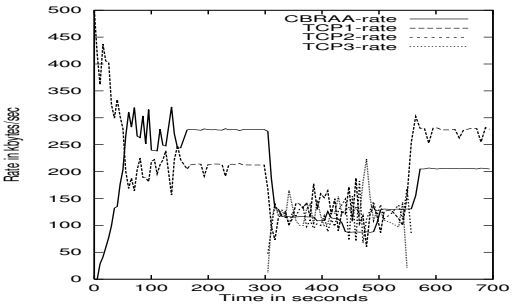

Figure 6: Behavior of CBRAA, when sources arrive and depart, the average RTT (for all sources) is ∼120 ms.

4.3 Fairness

Since thresholds are used in computing the rate, we expect that the fairness of CBRAA may vary. The previous example (figure 6) illustrates this; the average rate of CBRAA is close to 270 kbyte/s during the time 70-300, and only 200 kbyte/s during the time 570-700.

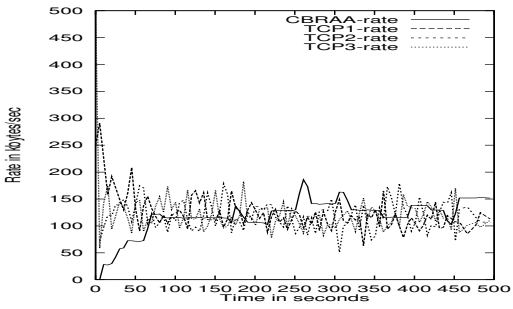

Figure 7: Fairness for equidistant sources. The average RTT is ∼120 ms.

Figure 7 illustrates the previous example with three TCP connections and one CBRAA source. All the sources in this case have a propagation delay of 50 ms. We can see that all the connections get almost the same fair share. CBRAA gets a bit less than the TCP sources (116 kbyte/s average rate for CBRAA, and for the TCP sources 119, 123, and 124 kbyte/s, respectively).

Let us now consider the case where the sources have different round trip time. We considered the same example of figure 4 with one TCP source and two CBRAA sources. The TCP source has a propagation delay of 25ms, and the two CBRAA sources have respectively 25ms and 100 ms as a propagation delay.

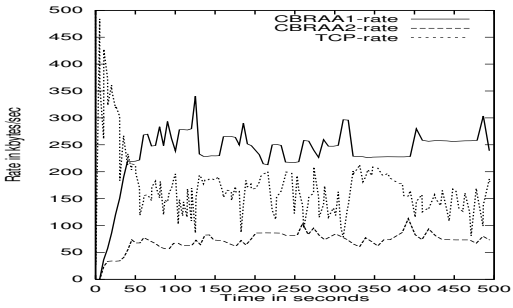

Figure 8: Effect of RTT on the average rate

Figure 8 plots the rate of each source. We ran the simulation for 500 seconds, and get the following results: the average round trip time of the

TCP source and one CBRAA source is ∼70 ms, and that of the other CBRAA source is ∼220 ms. Hence, since the rate is inversely proportional to the RTT (and if all the sources experience the same loss rate), ideally TCP and one CBRAA source (the one with smallest RTT) should get about 215 kbyte/s each, and the other CBRAA source should get about 69 kbyte/s.

The obtained rate for the TCP source is about 178 kbyte/s, and the average rates of the two CBRAA sources are respectively 233 and 69 kbyte/s, which are very close to the ideal values. Due to limited buffer sizes, when multiple connections are running, TCP is likely to get less than its fair share, due to its retransmission mechanism and timeouts. We believe that the use of TCP SACK [12] together with mechanisms such RED [11] would reduce this unfairness, and will make the proposed scheme proportionally fair [6].

5 Conclusions

In the present work, we proposed a TCP-friendly algorithm which smooths the source sending rate and allows TCP to get a reasonable fair share of the available bandwidth by using thresholds. The algorithm is based on the TCP cycle estimation, and a dynamic estimation of the loss rate as well as the sending rate.

Our first simulation results show that the fairness of the proposed scheme is kept within reasonable bounds, although not strictly guaranteed due to the use of thresholds. The use of closer thresholds would help improving the fairness, but would also result in deep oscillations which are not desirable for real-time multimedia applications in general. Actually, since real-time and TCP applications have very different characteristics and requirements, we do not seek the perfect guaranteed fairness, we rather seek a level of fairness which allows TCP and TCP-Friendly sources to maintain reasonable rates. Some examples of characteristics of real-time applications which differ from those of TCP applications are: a relatively smooth rate is required, feedback from receivers can be sparse, there are minimum and maximum sending rates. Therefore, to meet the requirements of the two kinds of applications simultaneously is a very hard task, and keeping them closer enough would be sufficient.

In the presence of multiple connections and small buffer sizes, TCP Reno is likely to lose multiple packets successively. To recover from these situations, TCP Reno is generally forced to wait for the timeout, resulting in a drastic decrease in the rate. TCP SACK solves this problem by enabling to recover from multiple successive losses without waiting for the timeout. Hence, the deployment of TCP SACK in the Internet will help improving the fairness between CBRAA and TCP, among other benefits. We plan to verify the behavior of our algorithm in comparison with TCP SACK as a next step.

This first version of CBRAA assumes a saturated source, i.e. a source that always has data to send. Nevertheless, a real-time source can not adapt its rate continuously, but rather in steps. The size of the increment generally depends on the amount and type of codecs available. Our algorithm can be used in such situations by differentiating between the effective sending rate (i.e. rate at which packets are actually sent to the network) and the computed sending rate (i.e. rate computed by CBRAA based on the observed network conditions), in such a way that the effective sending rate is always below the computed rate.

The simulation results presented here are peer-to-peer oriented to facilitate the study of the algorithm. Work is in progress to study its behavior in a multicast environment where each receiver may experience very different network conditions. The heterogeneity problem is generally dealt with by using layered encoding [21]: each layer is transmitted over a different multicast group, and each receiver chooses how many groups to join based on its observed network conditions. CBRAA could be adapted to this

context for a smoother rate estimation at the receivers, avoiding unnecessary group joins and leaves. On the other hand, for some applications layered encoding is not always possible or efficient. For these cases the source could adapt to the receiver supporting the lowest bandwidth, but that would decrease the quality for all the other receivers.

We believe that the receiver heterogeneity problem can only be efficiently tackled with the help of more intelligent network elements, such that the right adaptation can be performed just where it is needed, and can be customized for those receivers that need it. A few active network [23] nodes carefully placed (e.g. at the border between a high bandwidth fixed network and a low bandwidth mobile network) could perform application specific adaptation transparently to the end systems.

References

[1] O. Ait-Hellal, E. Altman, "Analysis of TCP Vegas and TCP Reno", *ICC'97*, Montreal, June 1997. In submission process to Telecommunication systems.

[2] L.S. Brakmo, L.L. Peterson, "TCP Vegas : End to End Congestion Avoidance on a Global Internet", *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465-1480, October 1995.

[3] S. Floyd, V. Jacobson, "Random Early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, August 1993.

[4] V. Jacobson, "Congestion avoidance and control". *ACM SIGCOMM'88*, pp. 273-288, 1988.

[5] V. Jacobson, "Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno", *Proceedings of the Eighteenth Internet Engineering Task Force*, pp. 365, University of British Columbia, Vancouver, B.C, September 1990.

[6] F. P. Kelly, A.K. Maulloo, D.K.H. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability",*Journal of the Operational Research Society*, 49, pp. 237-252, 1998.

[7] S. Keshav, "REAL: A network simulator", *Department of Computer Science, UC Berkeley, Technical Report* 88/472, 1988.

[8] I. Kouvelas, V. Hardman, J. Crowcroft, "Network Adaptive Continuous-Media Applications Through Self Organized Transcoding", NOSSDAV 98, Cambridge, UK, July 1998.

[9] T. V. Lakshman, U. Madhow, "Performance analysis of window-based flow control using TCP/IP: the effect of high bandwidth-delay products and random loss", *IFIP Transactions C-26, High Performance Networking*, pp. 135-150, North-Holland, 1994.

[10] T. V. Lakshman, U. Madhow, B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", submitted for publication.

[11] J. Mahdavi, S. Floyd, "TCP-Friendly", Technical note sent to the *end2end-interest* mailing list, January 8, 1997.

[12] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options". RFC 2018, April 1996.

[13] M. Mathis, J. Semke, J. Mahdavi, T. ott, "The macroscopic behavior of the TCP congestion avoidance algorithm", *Computer Communication Review*, 27(3), July 1997.

[14] J. Padhye, J. Kurose, D. Towsley, "A TCP-Friendly Rate Adjustment Protocol for Continuous Media Flows over Best Effort Networks", CMPSCI Tech. Report 98-047, October 1998.

[15] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Throughput: a Simple Model and its Empirical Validation", UMASS CMPSCI Tech Report TR98-008, February 1998.

[16] R. Rejaie, M. Handley, D. Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Real-time Streams in the Internet." To appear in IEEE INFOCOMM'99, March 1999.

[17] H. Schulzrinne, S. L. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet RFC 1889, January 1996 (update in progress).

[18] D. Sisalem, H. Schulzrinne, "The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme", NOSSDAV 98, Cambridge, UK, July 1998.

[19] W. Tan, A. Zakhor, "Error Resilient Packet Video for the Internet" (postscript), submitted to ICIP 98.

[20] F. Toutain, "TCP-friendly Point-to-Point Video-like Source Rate Control", submitted to Packet Video'99.

[21] T. Turletti, S. F. Parisis, J. Bolot, "Experiments with a Layered Transmission Scheme over the Internet."

[22] L. Vicisano, J. Crowcroft, L. Rizzo, "TCP-like congestion control for layered multicast data transfer", IEEE INFOCOM'98, San Francisco, March/April 1998.

[23] D. L. Tennenhouse et al., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.