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

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 most typical real-time applications. Adopting such behavior would result in annoying QoS oscillations for the users of such real-time applications. 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.

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 [3]. 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.
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 [8].
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 average rate and round-trip time, it is possible for a real-time application to estimate the duration of an ideal cycle, and therefore the loss ratio that an equivalent TCP source would experience under the same network conditions. The cycle estimation, the computed loss, the observed cycle, and the observed loss can then be used to calculate a smoothed TCP-Friendly rate.
The paper is structured as follows: In section 2 some theoretical background is shown, including a refinement of the $\frac{1}{\sqrt{p}}$ formula [8, 10]. In section 3, the cycle-based TCP-Friendly algorithm is described. Simulation results can be found in section 4. We conclude by some remarks and future work in section 5.

2 TCP-Friendly: Mathis's formula and its refinement

An equation for computing the rate of an equivalent TCP connection is given in [8, 10]:

$$B = \frac{C \cdot MTU}{\bar{R} \cdot \sqrt{p}} \tag{1}$$

where B is the computed rate, p is the loss rate, C is a constant given as $C = 1.30$ in [10] and $C = 1.22$ in [8], 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, 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 one 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]

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

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

$$\begin{aligned} p = \frac{1}{N} &\approx \frac{1}{\frac{3}{8}W_{max}^2} = \frac{8}{3}\left(\frac{7}{9}\right)^2 \frac{1}{\overline{W}^2} \\ \Longrightarrow &\quad B \approx \frac{1.2701}{\overline{R} \cdot \sqrt{p}} \end{aligned} \qquad (2)$$

Our algorithm (see section 3) uses Equation (1) with $C = 1.27$, with the loss rate calculated over the duration of a TCP cycle, 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 [8], nor does it apply for connections with a large round trip time, since TCP in the latter 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.

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 works on TCP-Friendly schemes, the duration of a TCP cycle seems to be taken into account. Indeed, it is well known [1, 7] that the ideal window behavior in TCP is cyclic. The cycle is delimited by two consecutive losses, and its duration is proportional to the bandwidth 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 the duration of a cycle (*Cycle*). The algorithm we propose below is based on this observation. We first compute the duration of an ideal cycle, and measure the experienced loss over the last cycle. If the source has experienced a loss rate not closer to the ideal loss rate (one loss per cycle), then a target sending rate is computed using the ideal loss rate, such that we get closer to the ideal situation of one loss per cycle.
The loss of one segment over a cycle can be written as:

$$\begin{aligned} p = \frac{1}{N} &= \frac{1}{\frac{3}{8}W_{max}(W_{max}+2)} = \frac{1}{B{\cdot}Cycle} \\ \Longrightarrow Cycle &= \frac{\frac{3}{8}W_{max}(W_{max}+2)}{B} \\ &\approx 0.62 \cdot \overline{R}^2{\cdot}B + 0.96 \cdot \overline{R}. \end{aligned} \qquad (3)$$

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

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

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. The loss ratio is computed using the feedback indications sent by the destination to the source via RTCP (receiver) reports [12]. These reports are also used to estimate the round trip time. To estimate the loss over the observed cycle, a sliding window is used (in case many RTCP reports are received during a cycle).

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*.
Figure 1 gives a pseudo-code of the core 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

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

For every Receiver Report (RR) do:

```
Cycle = srtt*(0.62*Rate*srtt + 0.96)
Loss_th = 1.0/(Rate*Cycle)
Loss_obs = nlost_obs_cycle / total_sent_obs_cycle

if (Loss_obs > a · Loss_th)
  Rate_th = decrease(Rate)
else if (Loss_obs < b · Loss_th)
  Rate_th = increase(Rate)

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

Figure 1: CBRAA: Cycle-Based Rate Adaptation Algorithm

theoretical one, namely *Loss_th*, and therefore the rate should be kept unchanged in this case.
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 average rate could in theory be increased. However, due to the sparse feedback provided by RTCP, our experiments show that increasing the rate in this situation can lead to long-term larger loss rates and therefore long-term TCP-unfriendliness. Therefore, we opted to probe for available bandwidth only when a loss rate over a cycle is smaller than $b \cdot Loss\_th$ ($b < 1$). We are currently working to find out the optimal values for a and b, which are set in our simulations respectively to 1.5 and 0.5.
The use of a weighted moving average for the rate allows us to get a smoothed rate. In order to avoid affecting the convergence severely, the smoothing factor α should be kept small enough (less than 0.5). The choice of α in fact determines the tradeoff between smoothness (stability) and convergence. 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. We have also observed that the smoothness may affect the fairness, in the sense that a too smooth rate may stay too long under or over the rate of an equivalent TCP connection. The functions "increase(Rate)" and "decrease(Rate)", we use in our simulations introduce also smoothing factors, in order to moderate between the theoretical loss rate (Loss_th) and the observed loss (loss_obs). Thus the rate is computed following equation (2), with p set to $\beta Loss\_th + (1 - \beta) Loss\_obs$. We are currently working on other functions for the increase and decrease that will allow better convergence and stability for a wide range of parameters, since the previous functions have limitations in some cases.
In the next section, we present several simulations to show how CBRAA behaves comparatively to TCP Reno and to a TCP Friendly that uses equation (2) directly. From now on, we call "TCP-Friend" a simple 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, where $TRTCP$ is the average time between two consecutive RTCP receiver reports. If the reported loss is larger than zero then TCP-Friend, uses 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. 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).

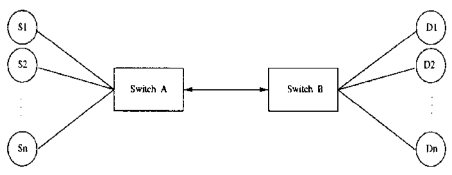

Figure 2: Simulation setup

We used the REAL network simulator [6] to simulate the topology described in figure 2. 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, and the packet discarding discipline is a simple drop tail. 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 $\alpha = 0.5$, and the minimum rate for each TCP-Friendly or CBRAA source is set to $1.0/srtt \times packet\_size$ bps (the initial rate is set to 5 $pkts/s$).

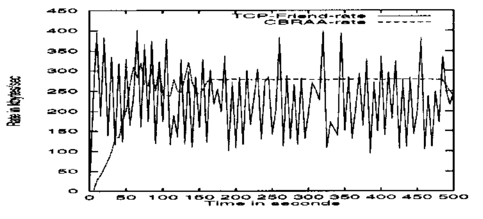

Figure 3: Rate for Mathis ("TCP-Friend") and CBRAA sources, the average RTT is 120 ms.

4.1 CBRAA and equation (1)

Consider a TCP-Friendly algorithm based on Mathis formula as described in Section 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 2 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 5 packets per second (40kbps).

In figure 3, 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 slightly less than the fair 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 convergence speed in CBRAA depends on the effect of the parameter α, and other smoothing parameters, we introduce for "decrease (Rate) and increase(Rate)" (see figure 1).

4.2 Reaction to arriving and leaving sources

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

Consider the model shown in Figure 2, 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 4 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.

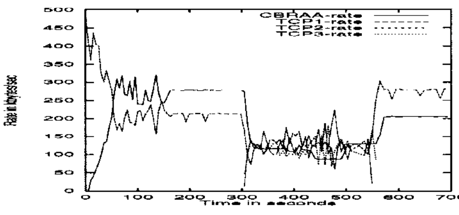

Figure 4: Behavior of CBRAA, when sources arrive and depart, the average RTT (for all sources) is ~120 ms.

4.3 Fairness

Since thresholds are used in computing the rate, we expect that the fairness of CBRAA may vary. The previous example (figure 4) 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.

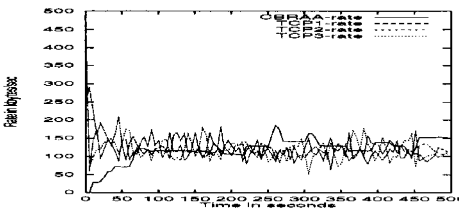

Figure 5: Fairness for equidistant sources. The average RTT is ~120 ms.

Figure 5 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 sessions have different round trip time. We considered the same example of figure 2 with one TCP source and two CBRAA sources. The TCP session has a propagation delay of 25ms, and the two CBRAA sessions have respectively 25ms and 100 ms as propagation delay.

Figure 6 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

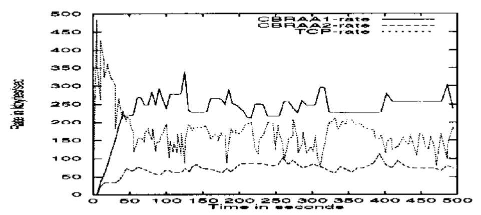

Figure 6: Effect of RTT on the average rate

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 [9] together with mechanisms such RED [8] would reduce this unfairness, and would make the proposed scheme proportionally fair [5].

5 Conclusions

We proposed a TCP-friendly algorithm which seeks to smooth the source sending rate while at the same time allowing TCP to get a reasonable fair share of the available bandwidth by using thresholds. The algorithm is based on the TCP cycle estimation, which is used to calculate a mean rate that TCP would have had over the same cycle duration.
Our first simulation results show that the fairness of the proposed scheme is kept within reasonable bounds, although not strictly guaranteed. Using smaller smoothing factors 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 might be sparse, there are minimum and maximum sending rates. Therefore, trying to meet the requirements of the two kinds of applications simultaneously is a very hard task, and keeping them closer seems sufficient.
This first version of CBRAA assumes a saturated source, i.e. a source that always has data to send. Nevertheless, a real-time source generally is not able to adapt its rate continuously, but rather in steps. The size of the increment 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 multicast where each receiver may experience very different network conditions.

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

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

[4] 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.

[5] 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.

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

[7] 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.

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

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

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

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

[12] 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).

IMPROVING TCP/IP OVER GEOSTATIONARY SATELLITE LINKS

Chadi Barakat, Nesrine Chaher, Walid Dabbous and Eitan Altman
INRIA
2004, route des Lucioles, 06902 Sophia Antipolis, France

Abstract

We focus in this paper on the undesirable phenomenon of early buffer overflow during Slow Start (SS) when TCP operates in large Bandwidth-Delay Product networks such as those including Geostationary satellite links. This phenomenon, already identified in [1, 6], is caused by the bursty type of TCP traffic during SS. It results in an underestimation of the available bandwidth and a degradation in TCP throughput. Given the high cost and the scarcity of satellite links, it is of importance to find solutions to this problem. We propose two simple modifications to TCP algorithms and illustrate their effectiveness via mathematical analysis and simulations. First, we reduce the SS threshold in order to get in Congestion Avoidance before buffer overflow. Second, we space the transmission of packets during SS.

Introduction

TCP [4, 10] uses two algorithms *Slow Start (SS)* and *Congestion Avoidance (CA)* to control the flow of packets in the Internet. With SS, the Congestion Window (W) is set to one segment and it is incremented by one segment for every non-duplicate ACK received. If we suppose that the window advertised by the receiver doesn't limit the source throughput, this process continues until the SS threshold (W_{th}) is reached or losses occur. In the first case, the source moves to CA where W is increased slower by one segment for every window's worth of ACKs. In the second case, W and W_{th} are reduced and a recovery phase is called. TCP supposes that the new W_{th} is a more accurate estimate of the network capacity.

In this paper, we investigate a problem that occurs when TCP operates in a network having small buffers compared to its Bandwidth-Delay Product (*BDP*). It is the problem of early buffer overflow and packet losses during SS before fully utilizing the available bandwidth. These losses are due to the high rate at which TCP sends packets during SS (every ACK triggers the transmission of a burst of two packets). If the network buffers are not large enough to absorb this high rate, they will overflow early. The window size when this overflow is detected is a wrong estimation of the network capacity. But TCP considers it as the maximum reachable window and reduces its W_{th} which results in a throughput degradation. This problem has been studied in [1, 6]. In these works, the authors show that when the Tahoe version of TCP [4] is used in a network with small buffers, two consecutive SS phases are required to get in CA. A second SS is called when the buffer overflow in the first SS is detected.

Due to their high BDP and the limitations on buffer size on satellite board, this problem is very likely to appear in Geostationary satellite links. Given the high cost and the scarcity of these links, a solution to early losses during SS is required. We propose in this paper two possible changes to TCP in order to solve this problem. The impact of these changes on TCP performance is mathematically analyzed. The results are then validated by a set of simulations using `ns`, the Network Simulator [7].

Our first proposition consists in reducing the SS threshold W_{th} so that to get in CA before the overflow of buffers. A similar idea is proposed in [3] to set W_{th} to the BDP of the path crossed by the connection. In our analysis, we find the explicit expression for the required W_{th} to get rid of these early losses. A study of the throughput as a function of W_{th} is also performed.

Second, we propose to reduce the rate at which TCP transmits packets during SS. Instead of sending immediately a burst of two packets in response to an ACK, the source inserts a certain delay before the transmission of the second packet. This proposal is similar to the one proposed in [9] for spacing the ACKs on the return path. The solution in [9] requires intelligence in routers whereas our solution requires only change at the sender. Similar propositions can be found in [8, 11]. The difference from our work is that these works aim to accelerate the SS phase. They propose to bypass SS by transmitting directly at a large window which may overload the network. In our work, we keep the SS phase but we space the packets transmitted in a burst.