TCP Vegas-like algorithm for layered multicast transmission

Omar Ait-Hellal
Department of Computer Science,
Long Island University
1 University Plaza, Brooklyn,
New York 11201-8423.

Guy Leduc
Research Unit in Networking,
Institut Montefiore,Bat B28,
Université de Liège,
B4000 Liège 1, Belgium.

Abstract

Layered multicast is probably the most elegant solution to tackle the heterogene ity problem in multicast delivery of real-time multimedia streams. However, the multiple join experiments carried out by different receivers in order to detect the available bandwidth make it hard to achieve fairness. In the present paper, we present a simple protocol, inspired from TCP-Vegas, that reduces considerably the unnecessary join experiments while achieving intra-session and inter-session fairness as well as being TCP-Friendly.

1 Introduction

Layered transmission is seen as the most elegant way to transmit real-time multimedia streams over the Internet. The technique consists of dividing the stream into multiple layers and multicasting each layer over a separate multicast group. Then, it is up to the receivers to choose the appropriate number of layers they can receive with a reasonable loss rate.

Several algorithms have been proposed for the rate adaptation at the receivers [13, 16, 4, 10, 7] to cite just few. In all these algorithms the persisting problem of *one layer oscillation* is not completely solved. Indeed, when the estimated loss rate at a given receiver is small enough, the receiver makes join experiment in order to check whether the available bandwidth could support a new layer. If it cannot, then the layer is dropped and the experiment is scheduled for sometime later. Unfortunately, if the available bandwidth remains, for a long period of time, insufficient to support a new layer, the receivers will keep joining and dropping an upper layer unnecessarily. This may alter the fairness among different receivers (intra-session fairness), and specially among different sessions (inter-session fairness). The problem may be worse if several sessions compete for an available bandwidth that is not sufficient for supporting any session's new layer [4].

In [13], a back-off mechanism, similar to TCP's timeout [8], is introduced in order to reduce these unnecessary joins. When a receiver joins a new layer and the join fails within a certain period of time "D", then the join timer for that layer is backed-off (multiplied by a constant larger than 1), so that the next join will be delayed enough. The timeout "D" is computed dynamically by monitoring the failed joins, and estimating the *detection time* which is the time after which a join is considered to be a success. This solution, unfortunately, even if it reduces the frequency of unnecessary joins, does not solve them completely. On the other hand, it may penalize the convergence of the algorithm for long sessions.

In the present paper we present a simple scheme based on the queuing delay estimation, and inspired from TCP Vegas [6]. For optimality reasons, our algorithm needs to estimate the round trip time only once over the session, and thus introduces a negligible overhead. However, it can work well without any round trip time estimation, and therefore scales very well to large multicast groups. Due to space limitations, we'll focuss more on the description of our algorithm rather than its evaluation; therefore many simulations where we evaluate our algorithm and comparing it to others (mainly [13]), in terms of covergence, stability and intra as well as inter-session fairness, will be subject of a future communication.

2 Background

2.1 Feedback information

Internet real-time multimedia applications generally use the Real-Time Transport Protocol (RTP) to transport their data. RTP [15] (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 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 our algorithm RTCP reports are used only at the beginning of the session to estimate the round trip time. In case the estimation of the one way delay is hard to achieve, we propose to use a virtual round trip time derived from TCP performance.

2.2 TCP Friendly and bounded fairness

TCP-Friendly protocols are protocols for real-time applications (which typically use UDP) that try to mimic the behavior of an ideal TCP connection.

TCP-Friendly algorithms are more fair, in the sense that the bandwidth a connection will get is close to that of an ideal TCP connection [11, 12, 3]. An ideal TCP connection would use a bandwidth given by the following formula [11]:

$$B = \frac{C \cdot MTU}{\overline{RTT} \cdot \sqrt{P}}, \tag{1}$$

where, C is a constant (ranging from 1.22 to 1.3 [12, 11, 3]), MTU is the mean packet size, $\overline{RTT}$ is the average round trip time and P is the loss ratio observed by the destination.

From this formula we can see that if a given source sends data with the rate B and the observed loss rate at the destination is close to $P = \left(\frac{C \cdot MTU}{\overline{RTT} \cdot B}\right)^2$, then the source is TCP friendly[1]. Therefore, if, in addition, the $\overline{RTT}$ is known, the source (or the destination as will be shown later) can adapt its rate such that the above equation can be met.

To guarantee suitable conditions for TCP, it would be sufficient to guarantee a bounded fairness [10]. *By bounded fairness we mean that if two (equidistant) connections i and j are competing for a bandwidth B, and are not restricted somewhere (both can send at the maximum bandwidth B), then the shares would satisfy $\frac{B_i}{B_j} \leq K$, where K is a constant.*

TCP Vegas is an example of algorithms that satisfy the bounded fairness [2, 6, 14]. The algorithm we propose below ensures a bounded fairness between the receivers of the same session as well as between different sessions.

3 Rate adaptation using RTT variations

The scheme we propose in the present paper is an enhanced scheme of the one presented in [4]. In [4] a TCP-friendly like adaptation algorithm is proposed, in which the rate is updated as follows: The receiver computes the rate $B(l_i)$ at which RTP packets arrive when it has subscribed to i layers as well as the loss ratio $P(l_i)$ by observing the sequence number of the received packets. It joins an upper layer only if [12, 4]

$$P(l_i) < L_{min_i} \triangleq \frac{1.613 \cdot \alpha(l_i)}{RTT^2 \cdot B(l_i)^2} \tag{2}$$

and it leaves the upper layer only if

$$P(l_i) > L_{max_i} \triangleq \frac{1.613 \cdot \beta(l_i)}{RTT^2 \cdot B(l_i)^2} \tag{3}$$

[1]In the rest of the paper we will express B in mean packet size per second, so that MTU disappears

where $\alpha(l_i)$ and $\beta(l_i)$ are constants set respectively to 0.5 and 1.5, and RTT is the round trip delay (without the queuing delay) measured at the beginning of a session.

3.1 Mean round trip time and TCP Vegas

The congestion window in TCP represents the number of packets (amount of data) sent but not yet acknowledged (packets in transit). Thus, by varying the size of this window, TCP controls its sending rate [8]. The congestion window is increased if no losses are observed otherwise it is decreased. In Vegas version of TCP, the mean round trip time RTT is estimated every time an acknowledgment is received, and the window is updated once every round trip time, more precisely upon reception of the acknowledgment of the last packet of the previous window of data (see [11] for more details on the window of data). Vegas keeps also the minimum over all measured round trip times (T). It increases the window only if $\frac{Win}{T} - \frac{Win}{RTT} < \frac{\alpha}{T}$ and decreases it if $\frac{Win}{T} - \frac{Win}{RTT} > \frac{\beta}{T}$, where α and β are constants typically set to 2 and 4 packet size respectively. In case the above difference is between the two thresholds, the window is kept unchanged (stable).

From the above formulas, we deduce that TCP Vegas increases, *resp.* decreases, its window only if $RTT-T < \alpha * \frac{RTT}{Win}$, *resp.* $RTT-T > \beta * \frac{RTT}{Win}$. On the other hand $\frac{Win}{RTT}$ is the sending rate of TCP (available bandwidth), thus TCP Vegas increases, *resp.* decreases, its window only if $\epsilon \triangleq RTT - T < \frac{\alpha}{B}$, *resp.* $\epsilon \triangleq RTT - T > \frac{\beta}{B}$. ϵ is simply the mean queuing delay and can be estimated as discussed below (see section 3.2), even without having an RTT estimation, and without the synchronization of the clocks, just assuming no drift.

3.2 Round Trip time and queuing delay estimation

To estimate the round trip time, one solution is the use of RTCP reports [15], so that the source can estimate its round trip time to a given receiver as follows: each Receiver Report (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: $RTT = t_{rr} - t_{lsr} - d_{lsr}$. In a receiver driven algorithm, all the decisions on joining or leaving a group are made at the destination, hence the destination should have an estimation of its RTT, if necessary. This can be done either by receiving the RTT directly from the source or computing it by the destination itself, in the same way as above (by swapping the sender and the receivers). The drawback of the two solutions is that the source has to manage several RTCP reports from different destinations (compute the d_{lsr} and remember the time t_{lsr} for each receiver, or to send the RTTs once computed to all the destinations). This leads, unfortunately to feedback implosion in case of a large group size. Nevertheless, one can reduce that by estimating the RTT only at the join time, by doing something similar to *ping*. In our algorithm, the last solution is adopted, thus the estimated round trip time is constant during the whole session and corresponds to the round trip delay (round trip time without the queuing delay).

Having an estimation of the round trip delay, the round trip time can be approached at the receivers without using any other information from the source except RTP headers (timestamps). Indeed, the round trip time can be measured by estimating the queuing delay separately each time an RTP packet is received. More specifically, assume that the round trip delay is T, and assume two RTP packets respectively with timestamps[2] t_s^1 and t_s^2. Assume that their arrival time at the receiver are respectively t_r^1 and t_r^2. If the clock of the sender is synchronized with that of the receiver then the one way delay is easy to estimate and therefore one can have an estimation of the RTT since the delay from the source to the receiver can be assumed to be constant. Otherwise, just assuming no drift between the clocks, we have to compute $Dl \triangleq D_1 - D_2 \triangleq (t_r^1 - t_s^1) - (t_r^2 - t_s^2)$ which repre-

[2]The timestamp represents the instant when the packet was transmitted by the source.

sents the queuing delay if one of the packets has a zero queuing delay (thus $RTT = T + Dl$). Since we are not able to determine whether one of the packets has zero queuing delay then we have to keep the minimum of all estimated D_i's, let it be $D_{min} = min(t_r^n - t_s^n)$, and compute the queuing delay as if D_{min} is the smallest one way delay, and use the above formula to have an estimation of the queuing delay (i.e. $t_r^n - t_s^n - D_{min}$ is the queuing delay of the nth packet).

3.3 Adapting Vegas to multicast layered transmission

The adaptation of TCP-Vegas to multicast layered transmission can be done as follows: each receiver estimates the aggregate rate of the layers it is receiving, and uses it together with the estimated queuing delay to control the received rate as TCP Vegas does, i.e. adds a new layer if the queuing delay ϵ is smaller than $\frac{\alpha}{B(l_i)}$ and drops a layer whenever the queuing delay is larger than $\frac{\beta}{B(l_i)}$. This adaptation raises two main problems:

-Firstly, as discussed above, TCP Vegas allows every connection to have at least α extra buffers in the network [2]. So, in case of multiple sessions, the number of allowed extra buffers will be proportional to the number of connections, and thus might be too large (exceeding the buffering capacity of the network) to avoid losses. If the layers are dropped only if the queuing delay is larger than $\frac{\beta}{B(l_i)}$, a receiver may never see a queuing delay larger than this value (hence layers are not dropped) even if the loss rate is large enough.

The solution to the above problem, is to keep also an estimation of the loss rate, and drop the highest layer (i) whenever the queuing delay is larger than $\frac{\beta}{B(l_i)}$ or the loss rate exceeds a certain threshold L_{max_i}. This threshold can be set to a fixed reasonable loss rate, or dynamically depending on the received rate (TCP Friendly, see below).

-Secondly, the fairness will be independent of the round trip time and thus non TCP Friendly. Indeed, it can be shown that two TCP Vegas connections i, j sharing a same bottleneck link, will get shares satisfying $\frac{\beta}{\alpha} > \frac{\mu_i}{\mu_j} > \frac{\alpha}{\beta}$, whatever the round trip time of the connections are. Thus TCP Vegas achieves a proportional fairness [9] based on the queuing delay (or equivalently on the number of crossed congested routers), which is in our view better than TCP's fairness.

To achieve a TCP-like fairness, one could implement the following solution: we know that if the actual bandwidth achieved by TCP (or a TCP friendly algorithm) is B_{act} with a loss rate of L_{act}, then following equation 3, and assuming that the RTT does not change enough when the number of layers changes (i.e the number of layers changes from i to j and $(RTT_i/RTT_j)^2 \approx 1$), when the achieved bandwidth is B_{new}, the new loss rate L_{new} should satisfy $L_{new} \leq L_{act} \cdot \left(\frac{B_{act}}{B_{new}}\right)^2$. In practice, we can set an upper loss rate L_{up} for the first layer (or for a minimum required bandwidth B_{min}). So, if the estimated loss rate, when receiving less than B_{min} exceeds the threshold, then it means that there are not enough available resources in the network and the receiver has to leave the session. Otherwise, the upper loss rate threshold L_{max_i} is set dynamically, as the layers are added or dropped, as follows:

$$L_{max_i} = L_{up} \cdot \left(\frac{B_{min}}{B_{l_i}}\right)^2 \tag{4}$$

where B_{min} is the bandwidth consumed by the base layer or a minimum required bandwidth, and B_{l_i} is the bandwidth when the receiver receives i layers. Having such a threshold ensures a bounded fairness with TCP connections, as observed in our simulations.

3.3.1 Avoiding unnecessary join experiments

Our solution is to back-off the lower threshold on the queuing delay each time a join to a new layer fails within a certain timeout D (problematic layer), hopping to stabilize the receiver at the previous layer. When the available bandwidth becomes sufficient to add a new layer, the queuing delay decreases, and is likely to go below the lower threshold, allowing then the receiver to carry out a new join experiment.

To avoid a potential underutilization (starvation), which can be caused by the back-off of the lower threshold, we maintain a limit to the latter. This limit should be set adequately so that if ressources become available to add safely a new layer, then the mean estimated queuing delay should go below that value indicating that a new layer can be added. We set the limit to the mean queueing delay estimated just before the current layer was added (see section 3.3.3), hopping that the same conditions (queueing delay) will lead to the same results (i.e. the current layer is safely added).

3.3.2 Protocol state machine

Here we give a brief description of how layers are added and dropped, see [4] for more details.

The receiver uses a timeout in order to avoid successive joins and drops, thus drops or joins are separated at least by the timeout D (see figure 1). This timeout (D) is estimated as in RLM (Receiver-driven Layered Multicast) [13]. We first estimate the *detection time* (TD) each time a layer is dropped. We used an estimator for each layer; when layer i is dropped, $TD[i]$ is updated to the time T elapsed between the last join, and the drop, i.e. at the nth drop, TD is updated as: $TD_n[i] = (1 - a) \cdot TD_{n-1}[i] + aT$. The deviation of the *detection time* is also estimated using a first order filter, i.e $Dev_n[i] := (1 - b) \cdot Dev_{n-1}[i] + b \cdot Diff$, where $Diff = |T - TD_{n-1}[i]|$, a and b are constants smaller than 1 (typically set to 0.25). The timeout is then set to $D = g_1 TD[i] + g_2 Dev[i]$, where i is the dropped or added layer. The estimation of the *detection time* above is similar to that of RLM, however RLM uses a single estimator for TD, and the time T above is considered as the time elapsed between the last join to the layer we are dropping and the drop. This may lead to large timeouts and therefore to slow convergence.

A measurement interval of T_M seconds (0.5) is used to estimate the loss rate by using a sliding window (see [4]). The decisions are made at the timeout D expiration and each T_M interval.

When a new layer is added (transition (S) to (W) in figure 1, where the receiver is assumed to have layer i as its upper layer), and the loss rate exceeds L_{max_i} or the queuing delay exceeds its upper threshold, before the expiration of the timeout D, then the layer is dropped and considered problematic, and thus the threshold α for that layer is divided by 2 (transition (W) to (D)). If the layer is not dropped within the timeout D then the join is considered successful (transition (W) to (S)).

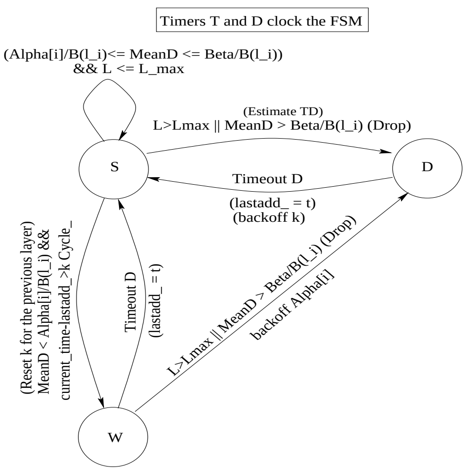
A join to a new layer cannot be carried out if the estimated time_to_join is larger than the current time (transition (S) to (W)). Firstly this helps to introduce some randomness in joins between several receivers, avoiding synchronized joins. Secondly it helps also to space in time the joins to the same layer, if the latter is problematic. More specifically, the time_to_join is reset after each expiration of the timeout D, to the current time plus the duration of a cycle times a constant back-off k. The cycle represents the estimation of a cycle in an ideal TCP (the time separating two consecutive decreases of the congestion window, see [4] for more details). We chose this value to give more chances to receivers receiving a small bandwidth to carry out joins, since the duration of a cycle is proportional to the received bandwidth and is given approximately by: $Cycle \approx 0.61 \cdot B(l_i) \cdot RTT^2$, where $B(l_i)$ is the aggregate received rate, and RTT is the estimated round trip delay (see section 3.2). A constant k is kept for each layer, and backed-off (multiplied by 2) every time the layer is dropped (transition (D) to (S)), and divided by 2 every time a join to the corresponding layer has succeeded. Thus, this will reduce the frequency of joining problematic layers.

The receiver remains in state (S), while the mean queuing delay is between the two thresholds and the loss rate is smaller than L_{max_i}. It transitions to state (D) (from (S)) and drops a layer if the loss rate exceeds the threshold L_{max_i} or the mean queuing delay is larger than the upper threshold, then after D it comes back to state(S).

3.3.3 Pseudo code

In the present section we give a simplified pseudo code for the above algorithm to show how α is updated. Consider that a given receiver has layer i as its upper layer.

```
MeanD is estimated as described in sec 3.1
 if ((MeanD > Beta/B(l_i))||(Loss > L_max)) {
  if (layer i is newly added) {
 // timeout D is pending
```



Timers T and D clock the FSM
(Alpha[i]/B(l_i)<= MeanD <= Beta/B(l_i)) && L <= L_max
(Estimate TD)
L>Lmax || MeanD > Beta/B(l_i) (Drop)
S
D
Timeout D
(lastadd_ = t)
(backoff k)
(Reset k for the previous layer)
MeanD < Alpha[i]/B(l_i) &&
current_time-lastadd_>k Cycle_
Timeout D
(lastadd_ = t)
L>Lmax || MeanD > Beta/B(l_i) (Drop)
backoff Alpha[i]
W

Figure 1: Protocol state machine

```
 // layer i is thus problematic
    Drop layer i;
    Alpha[i-1] = max (Alpha[i-1] / 2,
           Mean[i-2]*B(l_(i-1)));
  } else { // layer i is not problematic
     Drop i;  }
 } else { if (MeanD < (Alpha[i] / B(l_i))) {
      if (Now > Time_to_join) {
         Mean[i] = MeanD; Join layer i+1;
         wait D;  }  } }
if the join to i+1 has succeeded
 Alpha[i] = min (2, Alpha[i]*2);
```

4 Simulations

Due to space limitations we'll show only few simulations obtained using the NS-2 [1] simulator. Extensive simulations have been done to investigate the effects of RTT, packet size, and the number of connections (multicast and TCP) and the results are very promising and will be subject of a future communication. Here we'll present two small experiments to show the convergence (and fairness) of VALM (TCP Vegas Algorithm for Layered Multicast) and the smoothness in the throughput. In these experiments, five (5) different unicast VALM sessions (each with one receiver) are competing for a bottleneck node of 2.5 Mbps, and 100 packets buffering capacity, with five (5) TCP (Reno) connections. The round trip delay (round trip time without queuing delay) of the sessions, TCP included, is about 500ms. We consider an upper loss rate of 25% when receiving the first layer, and set dynamically the loss rate threshold following equation (4). The measurement interval for the loss estimation is set to 0.5 seconds, and the duration of all the simulations is 2000 seconds. The stream sent by the multicast session is composed by 10 CBR-like [13] equivalent layers of 50 kbps each.

4.1 VALM vs RLM

As seen in many simulations, RLM [13] is indeed fair when the connections start almost at the same time. However, when a connection (a TCP connection or RLM session) starts while another session is in its steady state, the starting connection is very likely to get much less bandwidth than the running one. In order to compare VALM to RLM, and to show an example where RLM fails in achieving fairness, we consider the scenario described above, with one RLM (vs VALM) session and one TCP connection (instead of 5). The bottleneck link is set to 0.5 Mbps (instead of 2.5Mbp). The TCP connection starts 400 seconds after the multicast session.

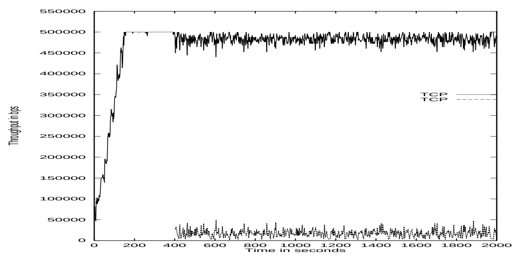


Figure 2: A TCP connection starting while an RLM session is in its steady state

Figure 2, plots the throughput of RLM and TCP. We can see that the TCP connection remains at the very low stages and does not get its fair share. The opposite scenario, where RLM session starts 400 seconds after the TCP connection, also leads to comparative situation, even though a little improvement in the fairness is seen.

In figure 3, the same scenario as above is simulated using VALM instead of RLM. The figure shows that TCP converges to a relatively fair share even though the convergence is relatively slow. The other scenarios, where VALM starts

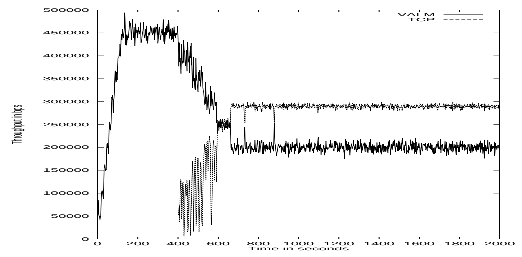


Figure 3: A TCP connection starting while a VALM session is in its steady state

at the same time or later than TCP give almost the same results (VALM converges to the same fair share).

4.2 Rate stabilization

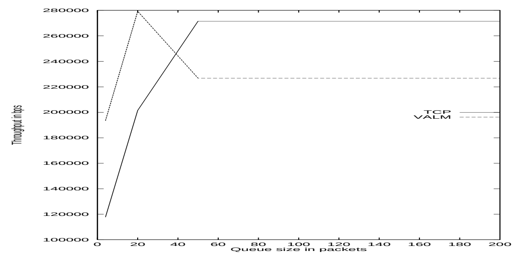


Figure 4: Mean throughput for TCP and VALM, as the queue size varies

The aim of the proposed algorithm is to keep the receiver in a stable state, where unnecessary join experiments are avoided. However, it is known that when the queue length is not large enough, Vegas cannot stabilize its rate [2]; and therefore so is our algorithm. To show the effects of the queue size on the performance of VALM, we run the scenario described in 4 with five VALM (unicast) session and five TCP connections.

Figure 4 shows the mean achieved throughput for VALM together with TCP when the queue size varies from 4 packets to 200 packets. We observe that as the queue size increases, the utilization of the link increases. When the queue size is larger than 50 packets the rate of each VALM session converges to a stable value. Another observation, as expected, is that the rate gets stable (no oscillation [5]) when the queue size is larger than $\beta \cdot Number\_connections$; in our case the number of connections is 10, then a queue size larger than 40 packets is needed to get a stable rate for VALM.

5 Conclusion

In this paper we presented a new scheme for layered multicast delivery over the Internet. Our scheme is inspired from TCP Vegas, and thus behaves like it. It is fair, convergent and more importantly TCP-Friendly. In extensive simulations, we have found that VALM achieves intra and inter-session fairness, while reaching near full utilization when the buffers are large enough. Unfortunately, when the buffers in the intermediate switches are not sufficiently large, the utilization and fairness tend to be poor because of Vegas' conservatism. Currently, we are investigating ways to tackle this problem by making Vegas more aggressive in similar situations.

References

[1] Network Simulator (NS-2), http://www.isi.edu/nsnam/ns/

[2] O. Ait-Hellal, E. Altman, "Analysis of TCP Vegas and TCP Reno", In *Telecommunication systems*, 2000.

[3] O. Ait-Hellal, L. Yamamoto, G. Leduc, "Cycle-based TCP-Friendly algorithm", *In Proc. Of Globcomm'99, Rio de Janeiro*, Dec. 1999.

[4] O. Ait-Hellal, L. Yamamoto, L. Kuty, G. Leduc, "Layered multicast using a TCP friendly algorithm", Research report, University of Liege, Belgium, 1999.

[5] O. Ait-Hellal, E. Altman, "Stability of ABR congestion control using the theory of delayed differential equations", *International Journal of Systems Science*, Vol. 34, Numbers 10-11, 15 August-15 September 2003, pp. 575-584.

[6] 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.

[7] I. El Khayat and G. Leduc, "A stable and flexible TCP-friendly congestion control protocol for layered multicast transmission", *Proc. of 8th Int. Workshop on Interactive Distributed Multimedia Systems (IDMS'2001)*, 4-7 Sep. 2001, Lancaster, UK.

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

[9] 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.

[10] X. Li, M.H. Ammar, S. Paul,"Video Multicast over the Internet", *IEEE Network Magazine*, April 1999

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

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

[13] S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven layered multicast", in *Proceedings of ACM Sigcomm*, pages 117-130, Palo Alto, California, August 1996.

[14] Samios, C. (Babis) and M. K. Vernon, "Modeling the Throughput of TCP Vegas", *Proc. ACM SIGMETRICS 2003 Int'l. Conf. on Measurement and Modeling of Computer Systems (Sigmetrics 2003)*, San Diego, June 2003.

[15] 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).

[16] T. Turletti, S. F. Parisis, J. Bolot, "Experiments with a Layered Transmission Scheme over the Internet", *INRIA Research Report* No 3296, Nov. 97.