# A stable and flexible TCP-friendly congestion control protocol for layered multicast transmission

Ibtissam El Khayat and Guy Leduc

Research Unit in Networking
University of Liège
Institut Montefiore - B28 - Sart Tilman
Liège 4000 - Belgique

**Abstract.** We propose an improvement of our RLS (Receiver-driven Layered multicast with Synchronization points) protocol, called CIFL for "Coding-Independent Fair Layered mulaticast", along two axes. In CIFL, each receiver of a layered multicast transmission will try and find the adequate number of layers to subscribe to, so that the associated throughput is fair towards TCP and stable in steady-state. The first improvement is that CIFL is not specific to any coding scheme. It can work as well with an exponentially distributed set of layers (where the throughput of each layer $i$ equals the sum of the throughputs of all layers below $i$), or with layers of equal throughputs, or any other scheme. The second improvement is the excellent stability of the protocol which avoids useless join attempts by learning from its unsuccessful previous attempts in the same (or better) network conditions. Moreover, the protocol tries and reaches its ideal TCP-friendly as soon as possible by computing its target throughput in a clever way when an incipient congestion is confirmed.

## 1 Introduction

Contrary to the current compression standards (e.g. JPEG, MPEG-x, H.26x), wavelet-based compression techniques (e.g. JPEG 2000) allow for flexible and highly scalable (in resolution, time and quality) formats. Although inter-frame wavelet video coding is still an open research area, it will enable very scalable video transmission where the data stream can be split into several hierarchical layers whose bit contents (and thus throughputs) can be defined in a very flexible manner. Therefore, we believe that any congestion control protocol dedicated to video transmission to an heterogenous set of receivers should be independent from the relative and absolute throughputs of each layer. It should behave as well with an exponentially distributed set of layers (where the throughput of each layer $i$ equals the sum of the throughputs of all layers below $i$), or with layers of equal throughputs, or any other scheme.
A multicast congestion control protocol has to allow all receivers to reach their optimal level as quickly as possible. By optimal, we mean a fair

share of the available bandwidth. We consider intra-session fairness (i.e. among receivers of the same session) and inter-session fairness (i.e. towards other sessions of the protocol or towards TCP connections).
A receiver-driven layered multicast (RLM) approach to solve the heterogeneity problem was first proposed by Mccanne in [8]. In RLM, every layer represents an IP multicast group and subscription to a layer implies subscription to all the lower layers. The receiver adds and drops layers according to the network state. This receiver-driven approach is probably the most elegant way to solve the multicast problem. It was later used in RLC [13], MLDA [12] and PLM [4]. The main concern of RLM was the intra-session fairness. To achieve it, a coordination mechanism between receivers has been designed. RLM was not designed to be TCP-friendly (i.e. fair towards TCP), nor to guarantee inter-session fairness. RLC was designed to be fair towards TCP connections whose round trip time (RTT) was close to one second, but not in general. RLC and MLDA support some form of inter-session fairness, in the sense that two competing RLC ( MLDA) sessions will get the same number of layers in steady-state, which means that both sessions get the same throughput only in cases where the two sessions have partitioned their layers so that they have the same throughputs in all layers. This cannot be the case in general.
In an earlier work, we have proposed a protocol, called RLS [3], that provides intra-session and inter-session fairness guarantees. For example, for a large range of RTTs, the ratio of throughputs between RLS and TCP remains in the interval $[\frac{1}{3}, 3]$, which is excellent compared to RLM and RLC. However, we noted that RLS, though stable, still performed too many unsuccessful join experiments. Moreover, RLS was designed to work with exponentially distributed layers only. In this paper, we propose a better protocol, called CIFL, which improves RLS along the following lines:

- We make no hypothesis on the throughputs of the layers, they can have any value.
- The receivers reach the optimal level quickly.
- The stability is better, because the receivers learn from their past failures to join some layers under some conditions. This makes the received throughput very smooth, and improves fairness too.

The paper is organized as follows. We first remind some basic concepts in section 2. We explain the principles of CIFL in section 3, and show its simulated performance results in section 4.

## 2 Basic concepts

### 2.1 TCP-Friendly

TCP is the most widespread traffic in the internet and any new congestion controlled protocol has to be designed to be TCP-friendly, which means that it gets an average share of the bandwidth (approximately) equal to the average share TCP would get in the same conditions. As TCP is unicast and we are considering multicast protocols, the definition should be refined as follows. A multicast protocol is TCP-friendly if each receiver gets an average share of the bandwidth equal to the average

share a TCP connection, between that source and that receiver, would get.
In Best-effort networks, there is no reason to favour video transmission over TCP given the importance of the latter. In Integrated Services networks where receivers can reserve some (minimum) bandwidth for the video stream, one could let receivers get more bandwidth provided that this extra share is fairly allocated. In Differentiated Services networks where video stream can be aggregated with others and may, not be in the same class as TCP flows, inter-sessions fairness will be achieved if all video flows adopt the same definition of fairness (and TCP-friendliness may be a good candidate for that). So in all cases, TCP friendliness seems a good requirement to fulfill.
The throughput of TCP (in bps) in steady-state, when the loss ratio is below 16%, is roughly given by the following formula [6]:

$$B_{tcp} \approx \frac{C.s}{\sqrt{p}\overline{RTT}} \text{ with } C = \sqrt{\frac{3}{2}} = 1,22 \quad (1)$$

where $s$ is the packet size (in bits), $\overline{RTT}$ is the mean round trip time (in sec) and $p$ the packet loss ratio. A more precise formula that takes TCP timers into account can be found in [9].
The TCP cycle is the average delay between two packet losses in steady-state. So we have one packet loss per cycle, which can be formulated as:

$$p = \frac{s}{B_{tcp}.Cycle}, \quad (2)$$

where $s$ is the packet size in bits and $Cycle$ the duration of the TCP cycle as described above. From (1) and (2), we derive:

$$Cycle = \frac{B_{tcp}.\overline{RTT}^2}{C^2 s} \quad (3)$$

## 2.2 Coordination of receivers

It was pointed out in [8], that a multicast congestion control protocol cannot be effective if the subset of receivers behind the same router act without coordination. Indeed, if a receiver creates congestion on a link by requesting a new layer, another receiver (receiving less layers) might interpret its resulting losses as a consequence of its (too high) level of subscription and may end up dropping its highest layer unnecessarily (because this layer will continue to be received by other receivers). So coordination is necessary, RLM has proposed to use announcement messages, and RLC to use synchronization points (SPs). SPs are special packets in the data stream. Receivers can only join a new layer just after receiving an SP. In RLC, each layer has its own SPs, and the receiver can only join layer $i+1$, when it receives an SP in layer $i$. [10] shows that the presence of SPs leads to a low redundancy and gives better fairness. That is the reason why RLS and CIFL build their coordination of receivers on the existence of SPs. The SPs will also contain information about the number of layers and their respective throughputs.

# 3 The CIFL protocol

Our goal is to create a layered multicast congestion control protocol which is:

1. TCP-friendly.
2. Stable: as few unsuccessful join experiments as possible.
3. Generic: independent from the throughput of each layer. To achieve that CIFL will estimate the ideal throughput, and will join, or leave, one or several layers at once to reach a throughput which is close to the computed target, based on estimations of the RTT and the loss ratio.
4. Careful before adding layers at SPs, but quick at removing layers when an incipient congestion is confirmed. This is to be compared with the Additive Increase Multiplicative Decrease (AIMD) scheme of TCP.

## 3.1 Estimation of the Round Trip Time

Each receiver has to estimate its RTT to the source. The classical scheme is to ping the sender from time to time, e.g. each time the receiver joins or leave a layer, or more frequently. However, for large sessions, the sender can be flooded by ping requests. If routers are active, a solution based on [1] can be used, but we are looking for a solution that does not involve routers. If the sender knows the number $r$ of receivers and the number $p$ of ping requests it can process between two SPs, it can provide these numbers in the SPs.
Knowing these values, receivers can ping the sender with probability $\frac{p}{r}$. We do not require that ping requests be immediately followed by a ping response from the sender. To achieve that, we implement a scheme similar to RTCP [11]. Suppose a receiver sends a ping request at time $R_s$ which is received by the sender at time $S_r$. The sender stamps the ping request at its arrival, and when it is able to send a ping response, say at $S_s$, it stamps the response with that time value. If the sender is quick, $S_s$ will be (almost) equal to $S_r$, but in any case the time spent at the sender can be computed as $S_s - S_r$. At $R_r$ the receiver will get the ping response and perform the following operations:

$$Rec\_Send = (1-g)Rec\_Send + g(S_r - R_s)$$
$$Send\_Rec = (1-g)Send\_Rec + g(R_r - S_s)$$
$$RTT = Rec\_Send + Send\_Rec$$

If all the data packets are timestamped, the receiver can continously estimate the $Send\_Rec$ value by using all the packets it receives. Between two pings, the $Rec\_Send$ can change without being noticed though, which requires that pings are not too distant from each other. This is also useful to compensate clock drift.

### 3.2 The join

**Synchronization points.** As said before, we use the SPs to coordinate the receivers. Contrary to RLC, SPs are only present in the first (base) layer and not in all of them. When a SP is received and if the decision to join is not taken, the receiver remains deaf to congestion during a deaf period $T_d$. This is necessary because this congestion can be induced by another receiver that has used that SP to get more layers. In practice the distance between SPs is at least 4 seconds. This distance is enough to be greater than any common deaf period (see next section). However, to avoid all kinds of synchronizations, the distance between SPs is randomized. It will vary between 4 and 16 seconds.

**Increase of the throughput.** The receiver tries and estimates the bandwidth TCP would get in a similar situation. To do so, it will use formula (1) which requires to know its loss ratio. But the latter has a meaning only when it is computed over a duration close to a TCP cycle. Indeed, remember that formula (1) is only valid in steady-state. So, the receiver will refrain from using an SP to get more layers if it did not stay at least one TCP cycle at the current level. When it is the case, the receiver computes the bandwidth it can get as follows:

$$B_{next} = \frac{Cs}{\overline{RTT}\sqrt{P_{cycle}}}.$$

with $P_{cycle}$ the loss ratio computed over the last TCP cycle (see formula (3)). If there were no loss, the throughput can be doubled. That is similar to TCP which would have doubled its window after a cycle. When the receiver has computed its optimal bandwidth, it joins the suitable number of layers to get the closest possible to the computed throughput. To do so, it is necessary that the SPs contain information about the throughputs of all the layers.

**Stabilization.** When the receiver has no good estimation of its RTT, e.g. because there is a large number of receivers and the pings are done less frequently, the estimated bandwidth can be overestimated. In this case, the receiver would join layers that it would leave soon after. These unsuccessful join experiments can be avoided if the receiver can learn something from past failures. To this end, the CIFL receiver will record the network state[1] as it was just before any unsuccessful join experiment. To do this, every receiver maintains a square matrix $QD$ with one row (and one column) per layer. Each element $QD_{i,j}$ of the matrix represents the minimum queuing delay the receiver has ever monitored before any unsuccessful join experiment from level $i$ to level $j$.
When a receiver at level *current* wants to join level *target*, it checks its matrix to see if it has already failed to join any layer below *target*

[1] The network state is measured by the mean queuing delay computed over an equivalent TCP cycle

with a queuing delay that was below the current queuing delay. It can be computed as follows:

```
tested = current + 1
while (tested <= target &&
         current_queuing_delay < QD(current,tested))
       { join
         incr tested }
```

When the receiver has reached a stable level and is subjected to very few (or no) losses, it basically spends its time computing estimations of the RTT, the queuing delay and the loss rate, refraining from joining at SPs.

### 3.3 The leave

If the decision to join layers can be done at (not so frequent) SPs and after a cycle has elapsed at the current level, the decision to abandon layers when congestion appears should be taken more quickly. So, when the receiver detects a potential incipient congestion by a packet loss, it will start monitoring the loss ratio $P_{T_m}$ over a short interval (denoted $T_m$), and then the receiver will compute the number of layers it decides to abandon. We will discuss the value of $T_m$ later, but we know it has to be short, say very few RTTs to fix ideas. The problem is that $T_m$ is in general short compared to a TCP cycle, which makes it impossible to use equation [6] to compute a new (lower) target throughput. In order to propose another formula to compute that throughput, we will require that, when the suitable number of layers are abandoned, the receiver will not join any layer before a minimum amount of time (denoted $T_c$) has elapsed. Clearly, $T_c$ should be larger than an equivalent TCP cycle, as before any join experiment, and should end at an SP. However, the distance between SPs being random, the future occurrences of SPs are unknown. In the calculation however, we will consider that all SPs are equally spaced out of 10 sec, which is the average spacing between SPs.

To derive our formula, we define

- $T_m$ is the monitoring period starting at a probable incipient congestion detected by a packet loss in steady-state (or induced by a join experiment of the receiver).
- $T_c$ ($c$ for compensation) is the minimal period during which the receiver will have to stay at its new level before joining any layer. It is computed as described above.
- $B_{current}$ is the current throughput, which will remain so during $T_m$,
- $B_{target}$ is the unknown throughput the receiver will request after leaving some layers, and will keep during at least $T_c$.

To compute $B_{target}$, we require that CIFL should get a TCP-friendly throughput over the $T_m + T_c$ interval.
Figure 1 shows the parameters we use, and illustrates also that $T_c$ finishes at an SP arriving after the expiration of the cycle.
Let $\alpha = \frac{B_{current}}{B_{target}}$, the mean throughput of the receiver is:

$$\overline{B} = \frac{T_m B_{current} + T_c B_{target}}{T_m + T_c} = \frac{T_m \alpha + T_c}{\alpha(T_m + T_c)} B_{current}$$

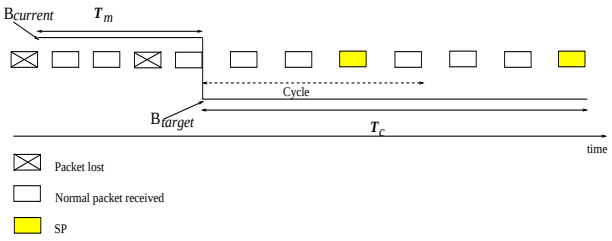



**Fig. 1.** A monitoring period followed by its compensation period

We suppose that there is no loss during $T_c$, which means that the loss ratio $p$ over $T_m + T_c$ is:

$$p = \frac{packets\ lost\ during\ (T_m + T_c)}{packets\ sent\ during\ (T_m + T_c)} = \frac{packets\ lost\ during\ T_m}{packets\ sent\ during\ (T_m + T_c)}$$

Whereas: $packets\ sent\ during\ (T_m + T_c) = (T_m + T_c)\overline{B}$
and: $packets\ lost\ during\ T_m = P_{T_m} T_m B_{current} = P_{T_m} T_m . \frac{\alpha(T_m+T_c)}{\alpha T_m + T_c}\overline{B}$
A simple replacement gives: $p = P_{T_m} \frac{\alpha T_m}{\alpha T_m + T_c}$,
TCP, which has a loss ratio equal to $p$, receives in average:

$$B_{tcp} = \frac{sC}{\overline{RTT}\sqrt{P_{T_m} \frac{\alpha T_m}{\alpha T_m + T_c}}}$$

If we equate both throughputs, i.e. $\overline{B} = B_{tcp}$, we derive that:

$$\alpha = \frac{P_{T_m} T_m T_c}{(T_m + T_c)^2 \frac{C^2 s^2}{\overline{RTT}^2 B_{current}^2} - {T_m}^2 P_{T_m}}$$

**Recapitulation.** When the receiver detects an incipient congestion, or just after joining a layer, it monitors the loss ratio during $T_m$ and then computes $\alpha$.

- If it is greater than 1 [2], the receiver leaves the suitable number of layers to get a throughput close to $\frac{B_{current}}{\alpha}$. Then the receiver ignores losses during the deaf period $T_d$, which is necessary to let the network reach its new state and monitor it. Initially, the deaf period is equal to $1\overline{RTT}$, but it is updated each time a layer is added or removed as follows. Knowing $time_{drop}$, the time at which layers were abandoned, and $time_{last}$, the reception time of the last packet belonging to one of the dropped layers, the receiver makes an exponential smoothing of $T_d$ with the new value "$time_{last} - time_{drop}$".
- Else, it does nothing as it treats the losses as resulting from a small transient congestion.

Such a scheme would not be easily adapted to TCP itself because TCP may not receive enough segments during an RTT (when its window is small) to compute $\alpha$ accurately.
Note also that if a receiver is in a leave evaluation when an SP is received, and a deaf period is started, the leave evaluation is cancelled. Otherwise, the receiver may be falsely confused by a transient congestion due to a join experiment by another receiver.

[2] The development we have made is meaningless if $\alpha$ is less than 1.

**The $T_m$ value.** In this section, we briefly discuss the choice of $T_m$. We know that:

$$B_{tcp} = \frac{T_m B_{current} + T_c B_{target}}{T_m + T_c},$$

So, when $T_m$ increases, $B_{target}$ decreases and $\frac{B_{target}}{B_{tcp}}$ decreases too. If $B_{target} \ll B_{tcp}$ the receiver at the end of $T_c$ will normally increase its bandwidth to reach $B_{tcp}$. To avoid this oscillation, we need $B_{target} \simeq B_{tcp}$. As $T_c$ and $\alpha$ are fixed,

$$B_{target} \rightarrow B_{tcp} \text{ implies } T_m \rightarrow 0$$

So $T_m$ has to be short. However, as TCP takes decisions at every $\overline{RTT}$, if the CIFL receiver evaluates its loss ratio over a duration shorter than $\overline{RTT}$, it will get a bad estimation. For this reason, $T_m$ has been fixed to $1\overline{RTT}$.

### 3.4 Start-up phase

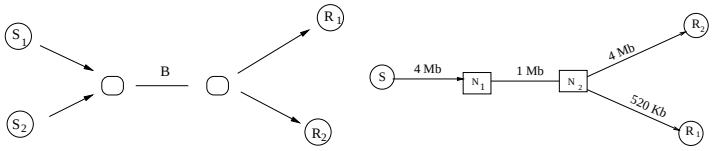
We have explained how the CIFL receiver behaves in steady-state. However, this behaviour is unsuitable at the very beginning, because it tends to mimic TCP in congestion avoidance, instead of a TCP in the slow-start phase. Therefore, when TCP and CIFL start together, CIFL would not get its fair share, or only after a much longer period.
In this section, we describe the start-up phase of CIFL. In this phase, the receiver uses all the SPs to join new layers, so that it doubles its throughput at every SP. For a set of exponentially distributed layers, this would mean adding a layer (but only) at every SP. In other schemes, the receiver may join several layers at once. This mimics the exponential takeoff of TCP, which continues until the throughput of subscribed layers is greater than the peak throughput actually received. Once this state is reached, the receiver drops all layers above the maximum received throughput and exits the start-up phase.
Moreover, this more aggressive phase is used to estimate the bottleneck capacity by measuring the smallest delay between two received packets. Knowing this bottleneck capacity, the receiver will not attempt to join layers that would lead to a throughput above this value. This will reduce the number of unsuccessful joins, compared to other protocols like [13], [8], [12].
If, later during the session, packets happen to transit through another path with more bandwidth, or if the network is simply less congested, the receiver will discover it, because it will continue to estimate the bottleneck capacity as follows:

$$estimate\_bw_i = max(estimate\_bw_{i-1}, \frac{pktsize}{t\_recv_i - t\_recv_{i-1}})$$

On the other hand, if the traffic is routed to less provisioned or more congested links, it is not a real problem, because this estimated bottleneck capacity will just become overestimated, and thus a bit less useful to avoid unsuccessful join experiments.

### 3.5 Scalability

When the number of receivers is large, receivers will ping the source less frequently, which means that the RTT estimation may be less accurate.




(a) 2 sources 2 receivers (b) 2 receivers and one source

**Fig. 2.** Topologies

Note however, that the RTTs are still adjusted at every received packet by using the timestamping approach, but regular resynchronizations are also useful to compensate clock drift and delay variations on the return path as explained in section 3.1.
Note also that a less accurate estimation of the RTT will simply lead to a bandwidth target which is less TCP-Friendly (see formula (1)). Moreover, some experiences performed over internet have shown us that during 1 hour the ratio between the longest RTT and the shortest one is rarely greater than 2.

## 4 Simulations

For our simulations we use the network simulator NS ([7]). We will start by showing that the receiver, once at its optimal level, does not make unsuccessful join experiments. Then we will show that intra-session and inter-session fairness are fulfilled, both towards TCP and towards other CIFL sessions.

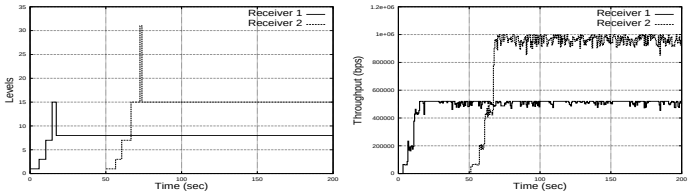
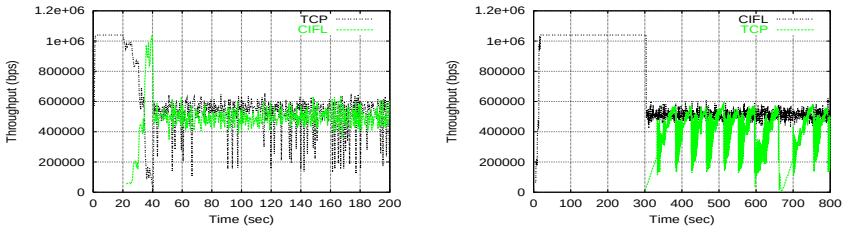



**Fig. 3.** Two different CIFL receivers beginning at different times

### 4.1 Several receivers

In this section, we will show the importance of the deaf periods. To this end, we use the topology of Figure 2(b). We see on Figure 3 that when receiver $R_2$ starts, receiver $R_1$ has already reached its optimal throughput. The newcomer will create congestion on the $N_1 - N_2$ link and $R_1$ will be subjected to this congestion. Without deaf periods, receiver $R_1$ would react by leaving some layers. This decrease would be useless because the layers it would leave would continue to transit through the bottleneck. In section 4.4 of [3], we showed that without deaf periods and when the delay to $R_1$ is larger than the delay to $R_2$, $R_1$ loses its fair share. With deaf periods, receiver $R_1$ does not react to losses caused by $R_2$. We can see by the same occasion that the receivers reach quickly their optimal level thanks to the quick start-up phase. They also find that they cannot go over the bottleneck capacity. They leave some layers and maintain the optimal throughput until the end of the simulation.




(a) Short RTT (b) Long RTT (bw)

**Fig. 4.** Bottleneck sharing between TCP and CIFL

### 4.2 TCP versus CIFL

**Reminder.** Previous studies ([5] and [2]) have shown that the most severe situations to get fairness towards TCP are the following:

- Either TCP begins before the multicast protocol, when the RTT is short,
- Or TCP begins after the multicast protocol, for long RTTs.

In the first case, TCP is so aggressive that it prevents the multicast protocol (RLM, RLC) to get a reasonable share of the capacity. In the second case, TCP is so slow and so vulnerable that it cannot get a reasonable throughput. This happens because TCP's clock is its RTT, which is not the case for the other multicast protocols. We will see that CIFL performs much better in the extreme cases.

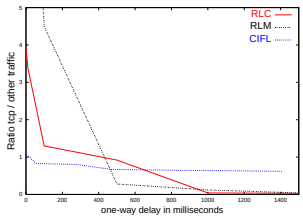



**Fig. 5.** RLM, RLC and CIFL according to the one way delay

**Simulations.** We use the topology of Figure 2(a) with one TCP source and one CIFL receiver to test the behaviour of both traffics. The parameter values are: $B_0 = 64$ kbps and $B_{i+1} = B_i + B_0$ i.e. $\forall i L_i = L_0 = B_0$. Figure 4(a) shows the sharing in the first extreme case, i.e. a short RTT (1 ms) and TCP begins before CIFL. We see that CIFL is so aggressive in the beginning that it can take more bandwidth than TCP. After this, it decreases its throughput to get exactly what TCP gets. In the medium term, the sharing ratio, $\frac{rate_{TCP}}{rate_{CIFL}}$, is 1. For the second extreme case, the RTT is approximatively equal to 2 sec. Figure 4(b) illustrates this case and shows that when TCP begins, CIFL halves its throughput by leaving 8 layers, and does not perform any future attempt. TCP pays for its greediness. In fact, every time TCP wants to have more bandwidth than the optimum, it creates congestion and has to decrease its throughput. And since the RTT is large, TCP needs a long time to reach again its fair share of the bandwidth. That is why the ratio is around 0.6 for long RTTs, while it remains slightly below 1 for short RTTs.
To better illustrate the dependence of the sharing ratio according to the RTT, we carried out several experiments where we changed only the

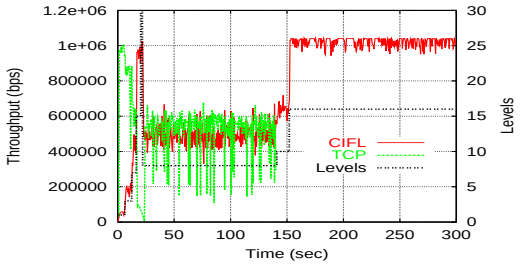



**Fig. 6.** The behaviour of the receiver when TCP disappears

propagation delay. We have done it for RLM and RLC too (for more results concerning these two protocols, refer to [5]). The results are illustrated on Figure 5. The CIFL curve remains close to 1, contrary to RLM and RLC whose ratios converge to zero for long RTTs and are far above 1 for short RTTs.

Now we show that a CIFL receiver can find a better optimal level, when a competing TCP connection stops. We consider topology 2(a) with TCP and CIFL beginning simultaneously. After both protocols have reached their fair bandwidth, TCP stops. There are two possibilities:

1. If the next SP arrives more that a cycle after TCP has disappeared, the CIFL receiver is unlikely to have suffered loss during the last cycle. If so, it will double its bandwidth (limited to the bottleneck capacity it had computed).
2. If the next SP arrives less than a cycle after TCP has disappeared, the CIFL receiver can still have noticed some losses during the last cycle. This case is illustrated in Figure 6. The CIFL receiver computes its new estimated fair share and decides to join a certain number of layers (in our case, 2 layers). When the next SP arrives, the CIFL receiver behaves as in the previous case.

### 4.3 Fairness towards another CIFL session

We will show that CIFL fulfills this requirement. This is so because all CIFL sessions try to be TCP-friendly. We use two different sessions, one with $B_0^1$ equal to 64 Kbps and the other one with $B_0^2$ equal to 124 Kbps. To be in a difficult case, the session with the smallest base layer begins when the other one has already reached its optimal level. Figure 7 illustrates the result. We see that both sessions get the same bandwidth, although these bandwidths correspond to different number of layers in the two sessions.

## 5 Conclusion

We have proposed a congestion control protocol for layered multicast transmission, called CIFL, that ensures:

- intra-session fairness,
- fairness towards TCP, the ratio $\frac{TCP}{CIFL}$ is close to 1
- fairness between sessions in terms of throughput (instead of levels),
- stability, the receiver uses its past failures to performs a sort of reinforcement learning.

We have simulated our protocol in different situations, and the obtained results show that intra- and inter-session fairness is fulfilled even when there are several TCPs and several CIFL in competition.

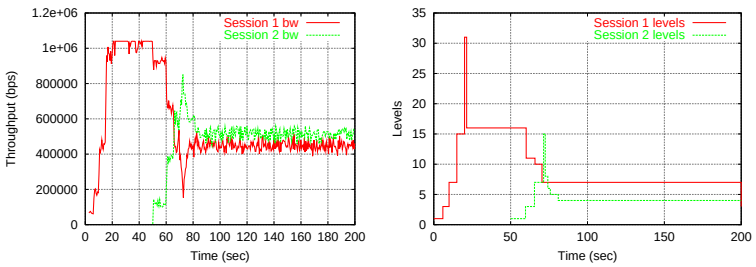



**Fig. 7.** Fairness between 2 different CIFL sessions

## References


1. A Basu and J. Golestani. Estimation of receiver round trip times in multicast communications. Technical report, Bell Laboratories, http://www.bell-labs.com/user/golestani/rtt.ps, 1999.
2. Ibtissam El Khayat. Comparaison d'algorithmes de contrôle de congestion pour la vidéo multipoints en couches. Master's thesis, University of Liège, Belgium, June 2000.
3. Ibtissam El Khayat and Guy Leduc. Congestion control for layered multicast transmission. *To appear in Networking and Information Systems*, 2000.
4. A Legout and E. W. Biersack. PLM: Fast convergence for cumulative layered multicast transmission schemes. In *Proceedings of ACM SIGMETRICS'2000*, Santa Clara, CA, USA, June 2000.
5. A Legout and W Biersack. Pathological behaviors for RLM and RLC. In *Proceedings of NOSSDAV'2000*, Chapel Hill, North Carolina, USA, June 2000.
6. M. Mathis, J. Semke, Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.
7. S McCanne and S Floyd. *The LBNL Network Simulator*. Lawrence Berkeley Laboratory, 1997.
8. S McCanne, V Jacobson, and M Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM'95*, pages 117–130, Palo Alto, California, 1995.
9. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP reno performance: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM'2000*, August 2000.
10. Dan Rubenstein, Jim Kurose, and Don Towsley. The impact of multicast layering on network fairness. In *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, September 1999.
11. Schulzrinne, Casner, Frederick, and Jacobson. RTP: A transport protocol for real-time applications. *Internet-Draft ietf-avt-rtp-new-01.txt (work inprogress)*, 1998.
12. D Sisalem and A Wolisz. MLDA: A TCP-friendly congestion control framework for heterogenous multicast environments. In *Eighth International Workshop on Quality of Service (IWQoS 2000)*, Pittsburgh, June 2000.
13. Lorenzo Vicisano, Jon Crowcroft, and Luigi Rizzo. TCP-like congestion control for layered multicast data transfer. In *Proceedings of IEEE INFOCOM'98*, San Francisco, CA, March 1998.