Layer thickness in congestion-controlled scalable video

Jean-Paul Wagner and Pascal Frossard

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Signal Processing Laboratory - LTS4
CH-1015 Lausanne, Switzerland

ABSTRACT

We address the problem of the proper choice of the thickness of pre-encoded video layers in congestion-controlled streaming applications. While congestion control permits to distribute the network resources in a fair manner among the different video sessions, it generally imposes an adaptation of the streaming rate when the playback delay is constrained. This can be achieved by adding or dropping layers in scalable video along with efficient smoothing of the video streams. The size of the video layers directly drives the convergence of the congestion control to the stable state. In this paper, we derive bounds on both the encoding rates of the video layers that depend on the prefetch delay that can be used for stream smoothing. We then discuss the practical scheduling aspects related to the transmission of layered video when delays are constrained. We finally describe an implementation of the proposed scheduler and we analyze its performance in NS-2 simulations. We show that it is possible to derive a media-friendly rate allocation for layered video in different transmission scenarios, and that the proper choice of the layer thickness improves the average video quality when the prefetch delay is constrained.

Keywords: Congestion control, scalable video, packet scheduling, layered video

1. INTRODUCTION

Congestion control certainly plays a crucial role in streaming systems, where it permits to distribute the network resources in a fair manner between the different users. Congestion control can be implemented using window-based algorithms (e.g. TCP) or rate-based algorithms. Contrarily to data bulk transmission, media streaming impose important timing constraints, which necessitates to adapt the source rate to the constraints imposed by the congestion control in order to avoid large playback delays. As media streams can also have very rate-distortion different characteristics, a share of bandwidth may present a different benefit or utility for the streams sharing a bottleneck link. It is thus generally interesting to adapt the behavior of the congestion control to the specificities of data. We have for example shown in[1] that a media-friendly Rate and Congestion Control (RCC) Algorithm can be implemented straightforwardly using the framework of Network Utility Maximization.[2] A key condition for such a system to be efficient is the capacity to match the sending rates of each source in the network to their respectively available channel bandwidth, or equivalently, to the rate control decisions taken by the RCC Algorithm. Fine rate adaptation for streaming of stored video streams can be achieved by transcoding, or by the Fine Granularity Scalability (FGS) property in scalable encoding. However, these solutions present important limitations in terms of computational complexity, or respectively low encoding efficiency that often prevent their wide use in practical scenarios. In this paper, we consider the streaming of hierarchically layered video streams that do not have the FGS property. Hence the video can only be adapted by adding or removing a complete video layer to or from the stream. This *coarse granularity scalability* feature is actually the only scalability feature that has been adopted into any encoding standard, hence its practical importance. In this paper, we analyze the influence of the thickness of coarse layers for the stability of the congestion control, and the quality of the decoded video streams.

Congestion control decisions are generally influenced by the rates at which the sources actually transmit video data. This dependence is in general not taken into account in the design of media streaming systems. Most of the time the assumption that the channel bandwidth varies independently of the sending rates prevails in the media

Further author information: Pascal Frossard, E-mail: pascal.frossard@epfl.ch

communication community. Such a situation would allow a source to adapt its source rate by adding/dropping video layers, once it observes that a corresponding channel bandwidth is available. We argue in this paper that sending rate and channel rate are not independent, and that a sender needs to carefully adapt both its source and sending rates in order to contend for available transmission resources. Indeed, if a source sends at a lower rate than the one coputed by the RCC Algorithm, other sources in the network will tend to contend for the share of bandwidth that has been left unused by this behavior. It is thus crucial to implement a sending rate that is as close as possible to the rate decided by the RCC Algorithm, in order to correctly implement the sequence of controls that is proven to drive the system into a stable state. Senders that transmit at higher rates tend to create congestions and behave unfairly, while senders that transmit at lower rates are unable to probe for potentially available increased transmission resources.

In layered video, only a discrete set of *video source rates* is available for transmission, given by $v = \sum_{i=1}^{l} \lambda^i$, for $1 \leq l \leq L$. Here each operand λ^i of the sum corresponds to the rate of one of the L available layers, which are all given by the set $\mathcal{L} = \left\{\lambda^1, \ldots, \lambda^L\right\}$. In order to efficiently use the available network resources and to keep the system stable, the RCC Algorithm may however decide to use a transmission rate that can not be matched by any of the L available source rates. In this paper, we analyze how we can temporarily generate these intermediate sending rates from a buffer of available video data by using smoothing techniques.[3–6] This will provide some flexibility in adapting the source rates to the channel bandwidth. Following this approach, we will derive conditions on both the encoding rates in $\mathcal{L}$ and the incurred prefetch delays. These conditions typically depend on the characteristics of the underlying RCC Algorithm, which computes the available transmission rates.

The remainder of this paper is organized as follows: in Section 2 we provide some details on Rate and Congestion Control as well as smoothing techniques, which will be useful in our developments. The adaptation of the sending rate to the congestion control constraints is discussed in Section 3. We then show in Section 4 how to compute the thickness of pre-encoded layers, based on the properties of the congestion control algorithm and the prefetch delay bounds. In Section 5 we address practical scheduling issues that arise using our solution, and propose a practical algorithm which efficiently addresses all of these considerations. Finally we provide simulations stemming from our NS-2 implementation of the proposed system in Section 6.

2. PRELIMINARIES

2.1 Rate and Congestion Control

We provide a brief high-level description on the principles of Rate- and Congestion Control Algorithms (RCC). In this paper, the rate at which the video is encoded or decoded is called the source rate. The sending rate $s(t)$ represents the rate at which video data are sent to the network. Finally, we denote by $c(t)$ the available rate at which an application can transmit (i.e., the available channel bandwidth computed by the RCC Algorithm). In general, the channel rate that is available for use by any networked application is computed by an RCC algorithm. This rate is periodically recomputed and the resulting sequence of rates forms a control sequence that drives the network into a stable state. RCC algorithms can be implemented in lower layers of the network stack, as it is the case in TCP for example,[7] which runs on the transport layer, or in the application layer itself, such as for example in TCP-frienldy Rate Control (TFRC).[8] These are distributed algorithms that take local decisions in order to converge to a globally stable network state. In TCP, a stable state is characterized by a *fair* sharing of the available network resources among all competing flows. Based on previous local decisions and feedback on the current network state, the RCC at each sender decides iteratively at which rate it should transmit data to its client. If each sender in the network applies their local sequence of controls (i.e., sending rates) that are generated in this way, the distributed algorithm will converge to a *stable* network state, which is characterized by high resource utilization and bounded per-flow losses.

If senders do not comply to the controls taken by their RCC instance, there is no guarantee for the network to reach a stable state. Hence non-complying sources tend to drive the network into an unstable state in the worst case and to bad resource utilization in the best case. In traditional data transfers over the Internet, these observations are of minor relevance: data is transferred without delay constraints and the Transport Layer takes care of both implementing the RCC and adjusting the sending rate accordingly. In media-streaming applications however, the RCC is preferentially implemented in the Application layer. This stems from the fact that, on the one hand, the application needs to have some control on the rate that is allocated. On the other hand, the application needs to know the rate decided by the RCC in order to adapt the bitstream that is to be transmitted accordingly due to timing constraints. Hence the sending rate needs to be matched to the channel rate at the Application layer, and this is typically achieved by selecting a proper source rate and efficient smoothing.

2.2 Scheduling using prefetching

A key component in any rate adaptation mechanism is the scheduler. It builds a transmission schedule by deciding which packet is to be transmitted at which time during the transmission process. This decision typically depends on previously taken decisions, as well as on information about the currently available channel bitrate. In streaming scenarios involving layered video, the scheduler decides when to transmit which layer of which frame.

While streaming video, there are timing constraints that need to be satisfied at any time t. Let k denote the index of the frame with the most stringent timing constraint. If frame k, and any frame leading up to k, has not been scheduled for transmission by time t, it will not reach the decoder by the time its decoding timestamp expires. In that case, the client will experience a buffer underflow.

Let D denote the constant *playback delay* at the receiver. For the sake of simplicity, we express delays in a number of frames throughout this paper, i.e., if $D = 1$, then its duration is equal to the inverse of the stream's framerate. A receiver waits until it has received D frames of video before it starts decoding, or playing out the video stream that it receives. Hence D corresponds to a buffer on the receiver side, which is used to compensate for sudden bandwidth variations. Let then D_p denote the *prefetch delay.* It corresponds to a buffer at the sender side, in which the latter stores D_p frames that are available for transmission, but whose transmission deadline has not yet expired. The data in this buffer can be exploited to temporarily increase the sending rate above the actual source rate. This results in a partial transfer of the buffer to the receiver end. Obviously, the dimensioning of D_p is important for any smoothing algorithm. In addition, the size of the the prefetch delay also influences the choice of the receiving buffer, as data sent in advance have to be temporarily stored at the decoder. Note finally that the total delay experienced between encoding and decoding is the summation of these two delay components, in addition to a transmission delay that we suppose to be bounded by a constant without loss of generality.

In the following section, we will analyze more thoroughly which are the conditions on the layer rates and on the prefetch delay that need to be satisfied in order to adapt the source rates to the available channel rates.

3. ADAPTATION TO CONGESTION CONTROL CONSTRAINTS

In general, the rate profiles that are allocated by a Rate and Congestion Control algorithm can be partitioned into two disjoint sets of periods. Periods of *stable state* are characterized by a steady, relatively constant and smooth rate. In contrast, during periods of *convergence*, the RCC takes controls that allow the system to go from one stable state to the next. This happens whenever the availability of the network resources changes, due to different flows joining or leaving the network, or to topology changes for example.

If we can make sure that during both of these characteristic periods we will be able to generate the sending rates that match the controls dictated by the RCC, we can make sure that the network remains stable. A stable network state is characterized by an efficient utilization of the network resources and bounded per flow losses.

We suppose, without loss of generality, that the stable state rate c which is reached by the RCC is a random variable that can take on values in a rate range given by $[c_{min}, c_{max}]$, and which is characterized by a probability distribution $p_c(x)$ defined on the same range. Using a set of layers $\mathcal{L}$, we can however only transmit at L rates for a sustained period of time, corresponding to the L achievable source rates. If the RCC Algorithm converges to a rate c that can not be matched by any of the source rates, $v = \sum_{i=1}^{l} \lambda^i$, for $1 \leq l \leq L$, the sender should select to transmit at the source rate that is closest to, but smaller than c. Obviously, the maximal efficiency is reached when the number of layer is large, or equivalently when the layers are very small (i.e., FGS encoding). As most systems in practice only offer a limited layer granularity, we study in the next section how we can partially overcome the drawback that is represented by thicker layers.

The duration of a convergence period depends in general on the RCC algorithm, and is called the *convergence time.* It is a random variable that we denote by δ and which has a probability distribution $p_\delta(x)$. This probability

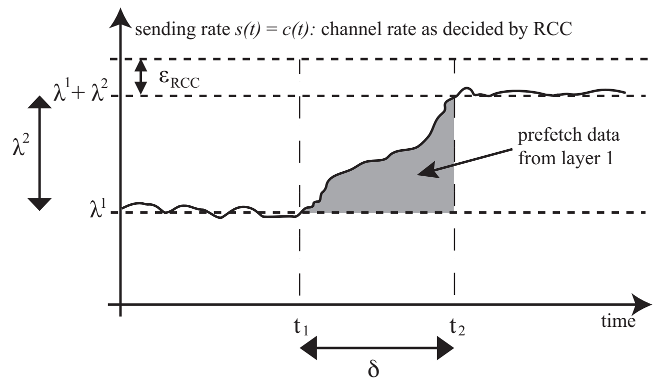

sending rate s(t) = c(t): channel rate as decided by RCC
ε_RCC
λ¹+ λ²
λ²
λ¹
prefetch data from layer 1
t₁
t₂
time
δ

Figure 1. Illustration of the rate profile during a convergence phase.

distribution can in general not be expressed analytically, but it can be inferred from statistics.[9,10] From above, we know that in the case of transmitting a layered stream, the steady states that are achieved correspond in practice to the rates that can be achieved using the layers that are available. In a convergence period, the RCC will take controls $c(t)$ that typically do not correspond to any of these rates, but which need to be taken to move the transmission rate from one steady state to the next. Moreover, the controls taken during the convergence periods may exceed the rate of the new steady state by an overshoot rate of ϵ_{RCC} bps for a limited amount of time. Figure 1 sketches the evolution of the available channel rate $c(t)$ from an initial state in which layer 1 of average rate λ^1 can be transmitted, to a new stable state which allows for two layers of combined rate $\left(\lambda^1 + \lambda^2\right)$ to be delivered. In this example $\delta = (t_2 - t_1)$. Obviously we can straightforwardly transmit layer 1 up to time t_1, and both layers 1 and 2 starting from time t_2. However, in order to drive the channel up to rate $c(t_2)$, we need to send additional data in between the time instants t_1 and t_2, as indicated by the grey-shaded area. Failing to do so will leave the sending rate at λ^1. This in turn will not allow the sender to contend for the available transmission resources. As a result of which, these resources would get allocated to other contending flows. The additional available sending rate, as given by the grey-shaded area in Figure 1, can be filled by either one of the three following approaches:

- sending dummy data in order to probe the channel. Although this solution is always feasible it will have a negative impact on the goodput of the application, meaning that network resources are used without yielding any application-level utility gain.
- sending redundant stream data such as Forward Error Correction (FEC). This is also feasible, but it reverts to the same situation as above if all data are in fact received.
- sending a part of the bitstream ahead of time. Using this scheme, all data that is transmitted will result eventually in an application-level utility gain, and it thus enhances the goodput of the application.

We opt for the latter solution, which consists in prefetching part of the video bitstream into a buffer at the client, whose decoder will only consume this part at a later time. This buffer undergoes thus a *filling phase* during the convergence time δ. As the stream we are considering is hierarchically encoded, layer 1 must be available at the decoder anyway in order for the corresponding layer 2 data to be decodable. Hence by prefetching layer 1 data ahead of time, the benefits are threefold:

- we can achieve the sending rate imposed by the RCC algorithm and increase the network stability,
- we make the decoding process more robust as layer 2 will be transmitted only when layer 1 is already available,
- we maximize the goodput of the streaming application by avoiding to send dummy or redundant data.

This general idea is similar to work presented in[11] in the context of TCP congestion control. However the authors therein do not consider a bounded prefetch delay. They imply that the client has unlimited buffering capacity, which is not always the case in practice.

4. COMPUTING THE NUMBER OF LAYERS AND THEIR RESPECTIVE RATES

In this section, we compute the appropriate thickness of pre-encoded layers, based on the delay bounds and the characteristics of the congestion control algorithm. We will use the concept of *smoothing* that we have outlined earlier in Section 2.2. Typically the exact shape of $c(t)$ in between t_1 and t_2 in Figure 1 is not known analytically, hence we apply the following integral upper bound on the possible traces of $c(t)$:

$$\int_{t_1}^{t_2} \left(c(\tau) - \lambda^1\right) d\tau \leq \left(\lambda^2 + \epsilon_{RCC}\right) \delta. \tag{1}$$

Conceptually this means that if, during the timespan δ we have $\left(\lambda^2 + \epsilon_{RCC}\right) \delta$ bits available to be transmitted in addition to the $\lambda^1 \delta$ bits needed for layer 1 itself, then we can generate all sending traces $c(t)$ during the timespan δ that are bounded by Equation (1). This includes the traces generated by the RCC in order to converge to the next stable state.

The prefetch delay gives us a bound on the time horizon, and hence on the amount of data, which we can use to prefetch video data. Once the streaming session has started, the scheduler is not able to send data at any given time instant t that is further than D_p frames, respectively time units, into the future. From Figure 1 it can easily be seen that the maximum amount of data that is available for prefetching is given by $\lambda^1 D_p$. If this amount of data allows the sender to increase the transmission rate to $\left(\lambda^1 + \lambda^2\right)$ during the timespan δ, then layer 2 can be transmitted later on. Otherwise the sender continues to transmit only layer 1. Combining all of the above we get:

$$\lambda^2 \leq \lambda^1 \frac{D_p}{\delta} - \epsilon_{RCC}. \tag{2}$$

This is a condition on the layer thickness of layer 2, which depends on the allowed prefetch delay D_p as well as on the used RCC algorithm. Indeed, the latter can be characterized by its convergence time δ and the potential overshoot during convergence periods, ϵ_{RCC}. We can generalize this result straightforwardly to higher layers by recursion, i.e., for layer $(i + 1)$ we have:

$$\lambda^{i+1} \leq \lambda^i \frac{D_p}{\delta} - \epsilon_{RCC}. \tag{3}$$

Using relation (3) with equality, i.e.,:

$$\lambda^{i+1} = \lambda^i \frac{D_p}{\delta} - \epsilon_{RCC}, \tag{4}$$

we can thus iteratively compute the minimum number of layers that are needed in order to make sure that we can converge from one stable state to the next, as well as their specific rates λ^l, once the statistical properties of the RCC Algorithm are known. If the channel rate can take on values in $[c_{min}, c_{max}]$, we set $\lambda^1 = c_{min}$ and apply Equation (4) iteratively until at the n^{th} iteration we have $\lambda^n > c_{max}$. The minimum number of layers that need to be encoded is thus given by $L = n - 1$ and their respective average source rates are given by $\mathcal{L} = \left\{\lambda^l\right\}_{l=1}^{L}$.

Note that if the probability distribution of δ is known, either analytically or as an approximation through statistics, we can dimension the above layers by setting δ large enough so that is covers a high percentage of $p_\delta(x)$. Indeed, if $P(x < \delta)$ goes to 1, which represents a conservative approach, encoding the layers as outlined in this section will make sure that we can generate the sending rates dictated by the RCC algorithm with high probability.

Finally, we can also solve the inverse problem, which consists in finding the minimum prefetch delay D_p, given a set of layers $\mathcal{L}$ and a RCC Algorithm that is characterized by its convergence time δ and its potential overshoot rate ϵ_{RCC}. From above we immediately get:

$$D_p = \delta \cdot \left(\sup\left\{\frac{\lambda^2 + \epsilon_{RCC}}{\lambda^1}, \frac{\lambda^3 + \epsilon_{RCC}}{\lambda^2}, \ldots, \frac{\lambda^L + \epsilon_{RCC}}{\lambda^{L-1}}\right\}\right). \tag{5}$$

It is important to note that, although multiple finer layers tend to increase the overall stability of the RCC Algorithm and allow for smaller prefetch delays, they also typically exhibit a worse compression efficiency as compared to coarser layers. This will be illustrated in Section 6 where we present our simulation results. Moreover the complexity of scheduling algorithms is typically increased when more layers are available for transmission. It is therefore important to choose properly the thickness of the layers as a trade-off between network stability, compression efficiency and scheduling complexity.

5. PRACTICAL SCHEDULING ASPECTS

In this section we will discuss some important practical aspects of the smoothing approach that we have outlined in Section 4. Let us consider once more the simple scenario that we have used earlier in order to outline some practical issues that may arise. We suppose that we have reached a steady state at rate λ^1 and that the RCC algorithm converges to a new steady state at rate $(\lambda^1 + \lambda^2)$. From the previous section, we know that if the layers are encoded so as to satisfy relation (3), then we can generate with high probability the sending rates that match the control decisions taken in the convergence period. For the sake of clarity, we will set the playback delay D to 0 in what follows, without loss of generality.

5.1 Repeatable prefetching

It is important to notice that, in the worst case, if we succeeded to attain the transmission rate $(\lambda^1 + \lambda^2)$, we have exhausted the complete prefetch delay D_p for layer 1. In that case, the prefetch buffer for layer 1 has been completely filled during the timespan δ, corresponding to a *filling phase*. By prefetching at most D_p frames, we have thus been able to ramp up the transmission rate to $(\lambda^1 + \lambda^2)$, and from there on we can transmit both layers if the network status does not change for some time. Clearly, if the same operation is to be repeated at a later time, each *filling phase* needs to be followed as soon as possible by a corresponding *draining phase*, which liberates the previously filled prefetch buffer.

If after the *filling phase* of layer 1, the rate $(\lambda^1 + \lambda^2)$ has been reached and should be sustained, the *draining phase* of the prefetch buffer for layer 1 data needs to be accompanied by another *filling phase* of layer 2 data. During this phase, as much data as is drained from the prefetch buffer of layer 1 will have to be prefetched into the buffer of layer 2 – keeping the overall prefetch buffer for both layers 1 and 2 at a constant filling level.

So, once the transmission rate of $(\lambda^1 + \lambda^2)$ is reached, the scheduler should in a first step make sure that for both layers all frames up to k are scheduled. Then it should preferentially select frames from the higher layer(s) in order to fill the rate surplus.

Another case that needs to be considered in this light is the settling into a stable state that corresponds to a rate that can not be achieved by a source rate constructed from the set $\mathcal{L}$. Suppose that the highest layer that can be transmitted at a given time is given by the index l^*, and that the RCC Algorithm converges to a steady state c, $\lambda^{l^*} < c < \lambda^{l^*+1}$. Using the mechanism outlined above, the scheduler will try to achieve any rate decided by the RCC Algorithm by exploiting the prefetch buffers for each of the layers up to l^*. The above loop will end up by first exploiting all of the available prefetch buffers. At this point, the actual transmission rate has not increased to the point where a further layer could be transmitted, but has settled at c. The scheduler will be unable to sustain the channel rate decided by the RCC Algorithm. It needs to drop its actual sending rate and settle into a steady rate that equals to the average video source rate that is closest to the channel rate, as discussed in Section 3. The actual sending rate for the l^* selected layers will read as:

$$s(t) = v = \sum_{l=1}^{l^*} \lambda^l. \tag{6}$$

However, as all the prefetch buffers are completely filled, the scheduler would no longer be able to increase its sending rate at any later time instant. Therefore the scheduler needs to settle into a steady state which is slightly smaller than that in Equation (6), in order to allow for a *draining phase* of the prefetch buffers, hence:

$$s(t) = \sum_{l=1}^{l^*} \lambda^l - \epsilon_\lambda \tag{7}$$

Note that as soon as an amount of data equal to $\lambda^* D_p$ has been drained from the aggregate prefetch buffer consisting of the prefetch buffers for layers 1 up to l^*, the scheduler can try to follow the decisions of the RCC Algorithm once more. According to Equation (3), this free prefetch buffer will allow the sending rate to be potentially ramped up to the average video rate given by

$$v = \sum_{l=1}^{l^*+1} \lambda^l, \tag{8}$$

which allows the next layer to be transmitted. In what follows we call this condition the *draining condition* for the prefetch buffer of layer l^*. In absence of other changes on the used channel, this condition will be satisfied $(\lambda^* D_p) \,/\, (\lambda^* - \epsilon_\lambda)$ time units after the rate was set to (7). This timespan corresponds to the time it takes to drain the previously filled prefetch buffer in the worst case where it was completely filled. In practical scenarios, the factor ϵ_λ should be kept small, so that the sending rate is still close to the steady source rate of the transmitted layers. While this means that the prefetch buffer takes a longer time to be completely drained, it also makes sure that the sender does not allow a large rate ϵ_λ to be available for contention by other flows in the network.

The considerations that we have outlined in this section do not apply if D_p is very large as the prefetching process will always be repeatable.

5.2 Implementation

The RCC algorithm decides to send data at rate $c(t)$ during the next control interval of length Δ_c based on channel feedback and previously taken rate controls. Given this new control, and knowing the set of available rates $\mathcal{L}$, the sender computes the index l^* of the highest layer that can be sustained using the sending rate $c(t)$. This is computed as:

$$l^* = \max_{1 \le l \le L} \quad \texttt{s.t.} \sum_{i=1}^{l} \lambda^i \le c(t). \tag{9}$$

This index is provided as input to the scheduler, along with l^*_{prev}, which is the index of the highest layer that was selected in the previous control interval. The scheduler also knows the frame index k which indicates up to which frame all of the selected layers need to be scheduled by the end of the current control interval, in order to meet all timing constraints. For control interval j, k reads as:

$$k = j \cdot \Delta_c \cdot f, \tag{10}$$

where f denotes the stream's framerate in frames per second. Finally, the control decision $c(t)$ is communicated to the scheduler in the shape of an available bit budget b for the schedule that is to be computed.

Figure 2 provides a more detailed view of the scheduler's inner life. Here, b_s denotes the number of bits that has already been added to the schedule. Note that we allow $b_s > b$, as we choose to add only *complete* frames of each layer to the transmission schedule. The scheduler keeps a state variable that indicates whether it needs to wait for the prefetch buffers to be emptied or whether it can try to ramp up the rate as dictated by the RCC algorithm. This variable is denoted as *steady* in Figure 2 and it is initialized to *false* for all streams that start transmission. This flag allows the scheduler to verify the *draining condition* that was discussed in the previous Section. The work flow of the scheduler is quite straightforward:

- In Step 1, all the frames up to k are added to the schedule, for all of the selected layers from layer 1 to layer l^*.
- In Step 2, we allocate the surplus of available rate by prefetching data from higher layers first. As mentioned earlier, this makes sure that the prefetching operation used to increase the rate is repeatable throughout streaming session.
- There is an intermediate Step 1.5 which is performed if the number of layers chosen for transmission in the current control interval is smaller than the number of layers l^*_{prev} that has been selected in the previous control interval. In that case, the scheduler checks if it can still allocate frames for previously sent layers, up to the frame indexed by k. Note that this might be possible if a prefetch buffer has been built up for layer l^*_{prev} earlier on. In case this step fails, it is rolled back.

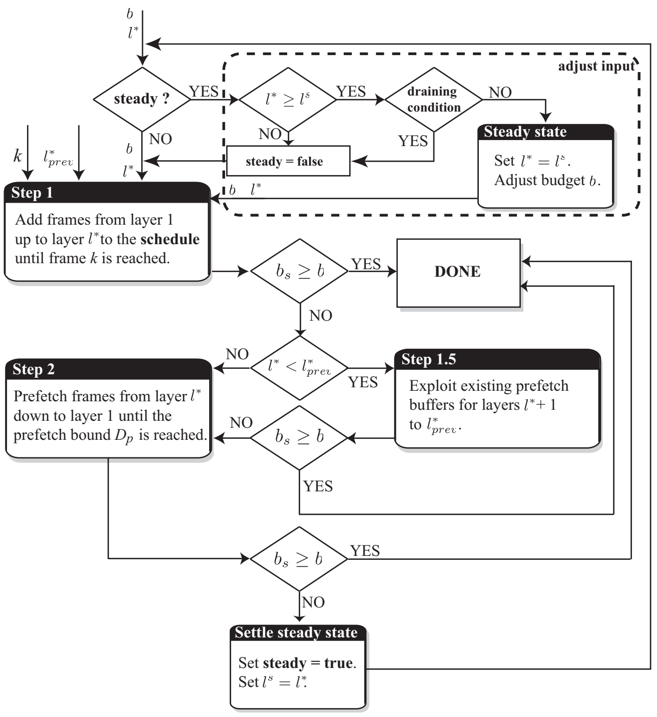


Figure 2. Workflow of the adaptive scheduler that implements our algorithm.

- If at the end of Step 2, the bit budget has not been exhausted, then the computed rate $c(t)$ cannot be matched with a sending rate. This happens if the RCC computes a stable state for which there is no matching source rate, or if the prefetching fails due to an unexpectedly long convergence time of the RCC. In that case, the sender settles into a steady state and sets its transmission rate to (7). The complete scheduling loop is rolled back and run again with an adjusted bit budget (i.e., an adjusted rate $c(t)$) that accounts only for the layers that have been selected for transmission and further allows for the prefetch buffers to be drained if necessary.

  The scheduler stores the highest selected layer as $l^s = l^*$ and will adjust the sending rate down to the same rate (7) during the subsequent control intervals, until the draining condition is fulfilled. If that is the case, the scheduler is again able to ramp up the sending rate high enough to potentially deliver an additional higher layer.

6. SIMULATIONS

6.1 Setup

So far we have presented the results of this paper in a generic form. By choosing δ and ϵ_{RCC} appropriately, the rates in the layer set $\mathcal{L}$ can be computed to match a wide variety of Rate and Congestion Control Algorithms. In our simulations, we focus our attention on applying the findings from the current paper to the Media Friendly Rate and Congestion Control Algorithm that we presented and validated in.[1] This algorithm provides the capability to inherently distinguish the transported streams and their rate needs, while allocating smooth rate profiles that allow for high and steady quality video delivery and TCP-friendly rate allocation.[1]

We have encoded the SOCCER (CIF, 30Hz) and CREW (4CIF, 30Hz) test sequences into a set of layers that provide SNR scalability using the H.264 SVC reference codec. Decoding additional layers will thus decrease the distortion in the received signal. We have considered 3 encodings in particular:

- SOCCER (coarse) encodes the SOCCER CIF@30Hz sequence into 5 layers. Each layer having an average bitrate of 400kbps.

- SOCCER (fine) encodes the same sequence into 8 layers. The base layer having an average bitrate of 400kbps, while all of the enhancement layers provide an average rate of 200kbps.
- CREW encodes the CREW 4CIF@30Hz sequence into 8 layers. The base layer having an average bitrate of 1Mbps, while all of the enhancement layers provide an average rate of 300kbps.

In order to generate concave and continuous Utility functions for these streams, we interpolate the utility values in between the layer boundaries by linear segments. According to our scheduling algorithm, if a layer l^* is being transmitted, higher rates are generated by prefetching information from l^* until $l^* + 1$ can be delivered. We hence motivate our choice of Utility functions by the fact that due to the hierarchical relationship between layers l^* and $l^* + 1$, the Utility of layer l^* increases linearly with the amount of prefetched layer l^* data.

One interesting feature of the RCC Algorithm[1] is that due to the smooth rate profiles it generates, the overshoot rate ϵ_{RCC} is in practice negligible. This leaves the convergence time δ to be computed. Our findings indicate that for our RCC Algorithm, δ will depend heavily on the scenarios that are considered and mostly on the Utility functions that are used. We have computed the maximum expected value of δ for each of the considered simulation scenarios. In order to do so, we have run each scenario first using the appropriate Utility functions, but by sending traffic that is always able to match the computed channel bandwidth. We have then measured the durations it has taken for the RCC to bridge the rate gap between two layers. The maximum of these values is then selected as the convergence time δ for the considered scenario. The such learned value provides a good starting point in order to validate our findings. Although this learning phase is not achievable in practice, a conservative estimate on the convergence time can always be extracted from experiments.

We have implemented the scheduler depicted in Figure 2 in NS-2. The average loss rate in the stable state π is set to 5% for all simulations and κ equals 1. We have simulated a network topology where a bottleneck link is shared by several video sessions. Each sender delivers a given video stream to its client, as given in Table 1. Finally, the transmission delay is set to zero in our simulations.

6.2 Layer thickness and fair bandwidth allocation

In the first set of simulations we consider some very simple topologies in order to illustrate some key characteristics that stem directly from our findings. Throughout this set of simulations, we have set $D_p = 90$ frames, unless otherwise stated. This is in line with equation (5). Indeed the convergence time we expect for these simulations was computed to be 3 seconds (90 frames), and the maximum ratio between the rates of 2 subsequent layers is 1.

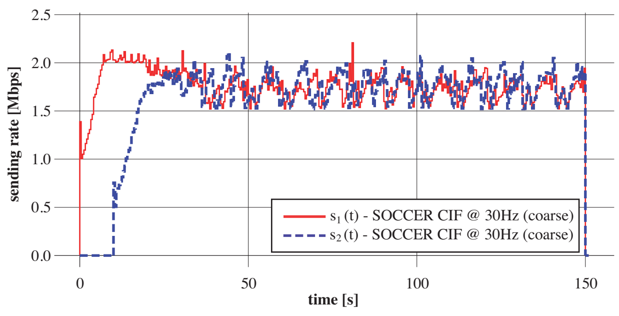
In Scenario 1, we use the sender-receiver pairs 1 and 2, which transmit the coarse layered version of the SOCCER test sequence. We set the bottleneck capacity to 3.5Mbps. The resulting sending rates and the received rates are plotted in Figures 3 a) and b) respectively. As expected, the two streams end up sharing the available network resources adequately, and each flow gets an equal average rate which allows for the transmission of all layers up to layer 4 once the steady state is reached.

In Scenario 2, where the bottleneck capacity is again set to 3.7Mbps, but this time we let the sender-receiver pairs 3 and 4, transmitting the fine layered version of the SOCCER sequence, share the bottleneck. The resulting transmission rates are shown in Figure 4 and indicate that using finer layers, we can indeed increase the stability of the algorithm. Both streams end up being treated in a similar way and fairly share the resource, as the optimal rate controls can now be matched by a corresponding video rate.

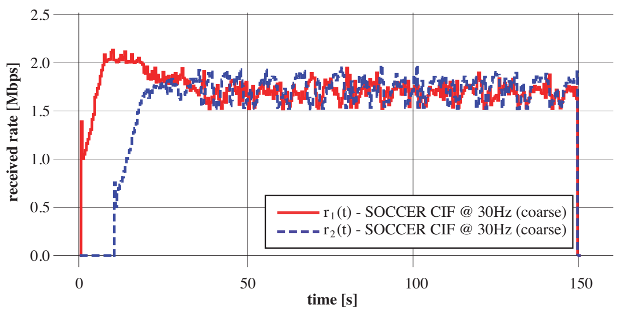
Note that the mean Y-PSNR of the transmitted streams and the mean loss in Y-PSNR due to congestion losses are given for reference in Table 2. The rate and congestion control algorithm[1] is characterized by steady state losses which are tightly bounded by the factor $\pi = 5\%$. We have implemented an *active node* at the

$S_{1,2} \rightarrow R_{1,2}$	SOCCER, CIF @ 30Hz (coarse)
$S_{3,4} \rightarrow R_{3,4}$	SOCCER, CIF @ 30Hz (fine)
$S_{5,6} \rightarrow R_{5,6}$	CREW, 4CIF @ 30Hz

Table 1. Distribution of the 3 test video sequences among the 6 sender-receiver pairs.



(a)



(b)

Figure 3. (a) Allocated transmission rates for Scenario 1. (b) Received rates for Scenario 1.

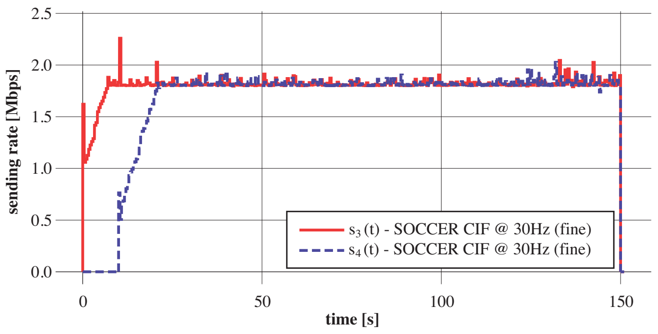


Figure 4. Allocated transmission rates for Scenario 2.

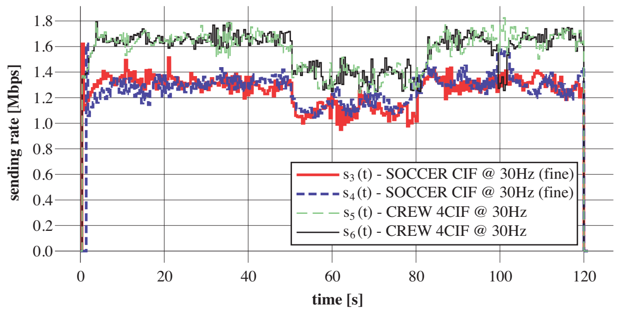
bottleneck router that discards first the packets of least importance whenever it has reached its forwarding capacity. Note that the differential treatment of packets at the bottleneck router does not influence the behavior of the congestion control. It simply permits to avoid dramatic quality degradations due to packet losses. It offers similar benefits in all the scenarios tested here and does not influence the relative performance of the different algorithms.

6.3 Adaptation to network dynamics

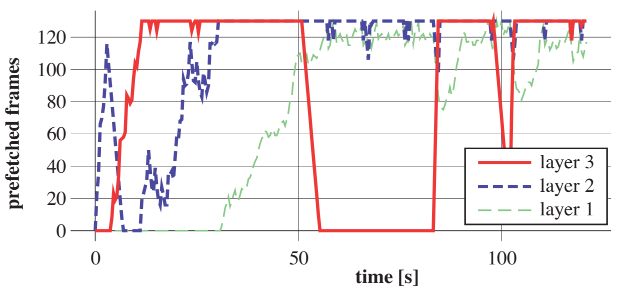
In the second set of simulations, we provide results on the performance of our proposed system in a more dynamic network setting. We consider a bottleneck capacity of 6Mbps and set the prefetch bound first to $D_p = 130$, which allows to satisfy relation (5). The bottleneck is shared by 4 sender-receiver pairs: senders 3 and 4 transmit the fine layered version of the SOCCER test sequence, while senders 5 and 6 transmit the CREW sequence. The simulation run lasts for 120 seconds. During 30 seconds, and starting 50 seconds into the simulation, we inject some constant bitrate (CBR) background traffic into the bottleneck, thus reducing its forwarding capacity for the media streams. The CBR rate is set to 1Mbps. We refer to this simulation run as Scenario 3.

Figure 5 shows the resulting sending rates for all of the 4 streams. Most importantly, it can be noticed that our system is still able to distinguish the 2 types of traffic on the bottleneck through the use of their Utility functions. It allocates the rates accordingly in an effort to maximize the overall Network Utility. On average the similar streams get the same number of layers throughout the simulation run, which is confirmed by the resulting Y-PSNR values listed in Table 2. Moreover the rates converge quickly to a new stable state once the bottleneck capacity drops, and recover to the initial steady state as soon as the background traffic is switched off.

In Figure 5 we illustrate the filling level of the prefetching buffers for the 3 layers that are delivered by sender 6. This shows how higher layers are preferentially used for prefetching in an effort to keep the prefetching process repeatable. It can also be seen that around the 100th second of the simulation run, no data for layer 3 is sent, but as the corresponding information has been prefetched earlier, all of layer 3 is still transmitted for all of the frames after the bottleneck capacity is restored.



(a)



(b)

Figure 5. (a) Allocated transmission rates for Scenario 3. (b) Evolution of the prefetching buffer of sender 6 in Scenario 3.

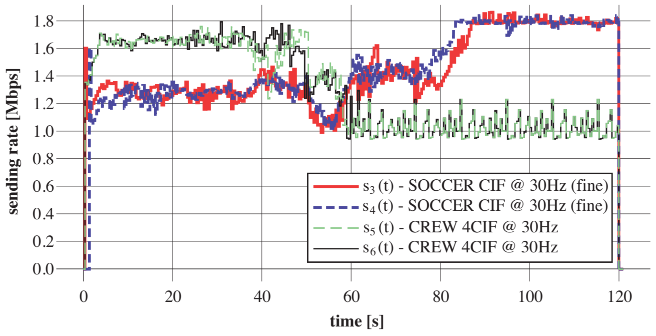


Figure 6. Allocated transmission rates for Scenario 4.

Finally, in Scenario 4, we illustrate the behavior of the system if the layer rates and the chosen prefetch delay do not comply to the condition given in Equation (5). This is achieved by setting a smaller prefetch delay $D_p = 100$ for senders 5 and 6, which are streaming the CREW sequence. The resulting sending rates for each of the 4 streams are depicted in Figure 6. Both senders 5 and 6 start to transmit at relatively high sending rates. Once the background traffic sets in, it becomes apparent that the senders do not have enough data in their prefetch buffers in order reach the previously achieved stable state. This results from the fact that D_p is too small, or conversely that given D_p, the rate of the encoded layers is too large. Once the background traffic is switched off, senders 3 and 4 are able to contend for the available bottleneck bandwidth and transmit the complete set of available layers. Meanwhile, senders 5 and 6 are stuck at transmitting layer 1. Although the bottleneck is not fully used, they alternate filling and draining phases of their prefetch buffers, but are never able to reach a sending rate that would allow for the transmission of further layers. On average the similar streams still get the same number of layers throughout the simulation run, which is again confirmed by the resulting Y-PSNR values listed in Table 2.

7. CONCLUSIONS

In this paper we have considered the problem of adapting the source rates that are generated by layered video streams to the available channel bandwidth. When the sending rates are adapted by prefetching and smoothing, we give conditions on the thickness of pre-encoded video layers and prefetch delays. These conditions depend on the characteristics of the underlying Rate and Congestion Control Algorithm. We have further discussed practical scheduling aspects related to the transmission of these video layers. Furthermore, we have given a practical implementation of a scheduler that meets all of our design criteria. Finally, we have illustrated that using layers that are properly encoded according to precomputed rates, it is possible to extend the media friendly rate allocation algorithm from[1] to the case of layered streams. Our results indicate that in general, having multiple layers of smaller average bitrate is highly beneficial in terms of both network stability and incurring delay, as expected. However, the coding performance of the video sequences is degraded when more layers of smaller average rate are encoded. The proper choice of the encoding rates therefore represents an important trade-off between network stability and video distortion characteristics.

REFERENCES

[1] Wagner, J.-P. and Frossard, P., "Media-friendly distributed rate allocation," *Signal Processing Laboratory Technical Report - TR-ITS-2008.003* (Mar 2008). Available at http:lts4www.epfl.ch.

[2] Kelly, F. P., Maulloo, A. K., and Tan, D. K. H., "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society* **49**, 237–252(16) (Mar 1998).

[3] Feng, W., "Rate-constrained bandwidth smoothing for the delivery of stored video," *IS&T/SPIE Conference on Multimedia Networking and Computing* , 316–327 (Feb 1997).

[4] Boudec, J. L. and Verscheure, O., "Optimal smoothing for guaranteed service," *IEEE/ACM Transactions on Networking (TON)* (Jan 2000).

[5] Thiran, P., Boudec, J.-Y. L., and Worm, F., "Network calculus applied to optimal multimedia smoothing," *IEEE INFOCOM* **3**, 1474–1483 (Apr 2001).

[6] Verscheure, O., Frossard, P., and Boudec, J. L., "Joint smoothing and source rate selection for guaranteed service networks," *Proceedings of IEEE INFOCOM* (Jul 2001).

[7] Jacobson, V., "Congestion avoidance protocol," *ACM SIGCOMM* , 314–329 (1988).

[8] Handley, M., Floyd, S., Padhye, J., and Widmer, J., "Rfc3448: Tcp friendly rate control (tfrc): Protocol specification," *Internet RFCs* (Jan 2003).

[9] Bakthavachalu, S. and Labrador, M., "Tfrc friendliness and the case of ecn," *International Journal of Communication Systems* (Jan 2004).

[10] Cho, H. S. and Lee, J., "Atfrc: Adaptive tcp friendly rate control protocol," *Information Networking: Networking Technologies for Enhanced Internet Services* (Jan 2003).

[11] Rejaie, R., Handley, M., and Estrin, D., "Layered quality adaptation for internet video streaming," *IEEE Journal on Selected Areas in Communications* **18**, 2530–2543 (Jan 2000).

Scenario 1	Mean Y-PSNR at sender	Mean loss at receiver
$S_1 \rightarrow R_1$	38.08 dB	0.96 dB
$S_2 \rightarrow R_2$	37.85 dB	0.98 dB
Scenario 2		
$S_3 \rightarrow R_3$	38.79 dB	0.90 dB
$S_4 \rightarrow R_4$	38.74 dB	0.82 dB
Scenario 3		
$S_3 \rightarrow R_3$	37.59 dB	1.15 dB
$S_4 \rightarrow R_4$	37.72 dB	1.18 dB
$S_5 \rightarrow R_5$	35.12 dB	0.40 dB
$S_6 \rightarrow R_6$	35.11 dB	0.51 dB
Scenario 4		
$S_3 \rightarrow R_3$	38.18 dB	0.21 dB
$S_4 \rightarrow R_4$	38.21 dB	0.19 dB
$S_5 \rightarrow R_5$	34.72 dB	0.19 dB
$S_6 \rightarrow R_6$	34.74 dB	0.18 dB

Table 2. PSNR quality for the different streaming scenarios [dB].