Media-friendly and TCP-friendly Rate Control Protocols for Multimedia Streaming

Adrian Sterca, Hermann Hellwagner, *Senior Member, IEEE,* Florian Boian, Alexandru Vancea

***Abstract*—This paper describes a design framework for TCP-friendly and media-friendly rate control algorithms for multimedia streaming applications. The idea of this framework is to start from TFRC's (*TCP-Friendly Rate Control*) transmission rate and then alter this transmission rate so that it tracks the media characteristics of the stream (e.g., bitrate) or other application characteristics like the client buffer fill level. In this way, the media-friendly property of the algorithm is achieved. We give three rules that guide how the TFRC throughput should track the evolution of the stream's media characteristics and remain TCP-friendly in the long term. We also present, as proof of concept, four simple media-friendly and TCP-friendly congestion control algorithms built using the aforementioned framework. These congestion control algorithms are better suited for multimedia streaming applications than traditional TCP congestion control or smooth congestion control algorithms like TFRC. We have performed evaluations of two of the four proposed media-friendly and TCP-friendly congestion control algorithms under various network conditions and validated that they represent viable transport solutions, better than TFRC, for variable bitrate video streams. More specifically, our two media-friendly and TCP-friendly congestion control algorithms maintained a TCP-friendly throughput in the long term in all experiments and avoided an empty buffer at the client side in situations when TFRC could not achieve this.**

***Index Terms*—TCP-friendly congestion control, media-friendly, multimedia streaming**

I. Introduction

Streaming multimedia data has very strict network demands. It requires a stable large bandwidth and almost isochronous communication. These requirements are not met by best-effort networks like the Internet, where network characteristics like available bandwidth and delay are constantly fluctuating. Forced to exist in such unhealthy environments, multimedia streaming applications need to continuously adapt their demands to these varying network conditions. This usually implies some form of controlled degradation of the multimedia stream's quality.

End-to-end congestion control algorithms like TCP's AIMD (*Additive Increase Multiplicative Decrease*) adapt the transmission rate of applications to the available bandwidth in the network. However, TCP's AIMD produces high fluctuations in the transmission rate which does not go well with modern audio-video codecs which expect predictive, stable bandwidth. With the help of buffers at the client side which prefetch in advance multimedia data from the server, multimedia streaming applications can, in general, cope with transient fluctuations in the transmission rate (i.e., avoid player freezes at the client). However, when using TCP, transmission rate fluctuations tend to exist for the whole duration of the streaming session. In addition, an initial data prefetch time of more than a few seconds before the client's player starts playing the stream is not easily tolerated by the end user. Also, even if the available bandwidth in the network allows it, some multimedia streaming applications do not support large enough prefetch buffers (e.g., live broadcasting, video conferencing, etc.). Roughly speaking, the QoS conditions provided by TCP are acceptable for multimedia streaming (i.e., avoid player freezes) if the average transmission rate is twice as much as the average bitrate (i.e., bandwidth demand) of the stream [2].

Targeting more stable transmission rates, smooth TCP-friendly congestion control algorithms were developed, perhaps the most well known being TFRC (*TCP-Friendly Rate Control*) [3]. These smooth TCP-friendly congestion control algorithms react less drastically to congestion, so that they achieve a smoother transmission rate than TCP, but on average, their long-term transmission rate stays equal to TCP's rate under the same network conditions. Smooth TCP-friendly congestion control is more suitable than TCP's AIMD for multimedia streaming because it gives a predictable transmission rate and even if this transmission rate is less than the bitrate demands of the stream, the stream can be more easily adapted to a predictive transmission rate.

However, smooth TCP-friendly congestion control like TFRC is not the best solution for multimedia streaming. A multimedia streaming application following blindly the transmission rate given by TFRC is good from a network perspective and *good but not optimal* from the application's perspective. In other words, TFRC is too much "network-friendly", but not sufficiently "media-friendly." Several papers [4], [5], [22] outline some of the problems multimedia streaming applications face when using TFRC. One problem of TFRC is that in some situations its throughput can overshoot or undershoot the throughput of TCP under the same network conditions. This can be caused by approximations/imperfections of the TCP Reno equation [20] or by wrong estimations of the RTO (*Retransmission Timeout*) or the loss-event rate [4]. Also, the authors of [5] claim that TFRC is not smooth enough to be termed "media-friendly."

Another problem emerges when the average available bandwidth is close to the average bitrate of the stream. VBR

A. Sterca, F. Boian and A. Vancea are with the Department of Computer Science, Babes-Bolyai University, Romania; email: {forest,florin,vancea}@cs.ubbcluj.ro.

H. Hellwagner is with the Institute of Information Technology, Alpen-Adria-Universität Klagenfurt, Austria; email: hellwagn@itec.uni-klu.ac.at.

(*Variable Bit Rate*) codecs compliant to MPEG video coding standards can vary a lot the output bitrate of a video stream between scene changes in order to preserve a relative constant quality throughout the video (e.g., the bitrate can vary 20 times from one stream second to the next one) [6]. In other words, the bitrate of such a stream is certainly not smooth. For these kinds of streams and also for real-time encoding applications which do not afford a large buffer, it is better if the transmission rate is not necessarily smooth, but rather that it tracks the evolution of media characteristics like bitrate across the stream. Besides the bitrate of the stream, other media or application characteristics could be relevant for (i.e., could influence) the evolution of the transmission rate: bitrate averaged over a scene, burstiness, PSNR values or other quality indicators, client buffer fill levels, etc. For example, if the client prefetch buffer is well filled, the transmission rate could be decreased, but when the client prefetch buffer fill level gets low, the transmission rate should get higher than average. Of course, the transmission rate must also obey network-related characteristics (i.e., must have a TCP-friendly behavior). Such a congestion control algorithm that has a transmission rate which tracks the media characteristics of the stream is termed media-friendly.

Generally speaking, if we refer to the average available bandwidth in the network throughout the streaming session by *AAB* and to the average bitrate of the stream by *ABS*, we have the following scenarios:

- If *AAB* is much higher than *ABS* and the available bandwidth in the network does not fluctuate consistently, typically *no congestion control is required* during streaming.
- If *AAB* is at least $2\times$*ABS*, typically TCP's AIMD congestion control *is acceptable* for obtaining good quality at the receiver [2].
- If *AAB* is only a little higher than *ABS*, TFRC *is good* for multimedia streaming [7].
- If *AAB* is around (more or less) *ABS*, Media-friendly and TCP-friendly Rate Control *is optimal* for multimedia streaming [22].

Our goal in this paper is to build TCP-friendly and media-friendly congestion control algorithms that are better suited for multimedia streaming than traditional congestion control. As congestion control is done by controlling the transmission rate of the sender, we will use the terms "rate control" and "congestion control" interchangeably throughout the paper.

The remainder of the paper is organized as follows. In Section II we review related work. Section III presents the main results of the paper which are three theorems that help us make TFRC media-friendly for multimedia streaming applications. Following this, we outline in Section IV four simple, proof-of-concept, media-friendly and TCP-friendly congestion control algorithms built using the theorems from Section III. Section V describes some practical considerations which are important for a full implementation of the congestion control algorithms proposed in Section IV and Section VI presents simulation results using two of the four algorithms presented in Section IV. The paper ends with conclusions in Section VII.

II. Related Work

TCP-Friendly Rate Control [3], [7] is an equation-based congestion control that has two main components: the throughput function and the WALI (*Weighted Average Loss Intervals*) mechanism for computing the loss rate. The throughput function is the throughput equation of a TCP-Reno source [8]:

$$X(p) = \frac{1}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \qquad (1)$$

where X is the sending rate in packets/sec, R is the round-trip time (RTT), p is the steady-state loss event rate and $t_{RTO} = 4 * R$ is the TCP retransmit timeout value. This throughput function is the basis of TCP-friendliness of TFRC. WALI, the mechanism for computing the loss rate as a weighted average of the last 8 loss intervals, is responsible for the smoothness of throughput. Studies reveal that indeed TFRC's throughput is smoother than the throughput of TCP [9]–[11], but it also has some limitations [4], [5].

Binomial congestion control [12] is a window-based congestion control that offers smoother throughput than TCP. It does that by increasing the congestion window less aggressively upon a packet acknowledgment and decreasing it less drastically upon a loss event. There are also several other proposals for smooth TCP-friendly congestion control [13]–[15].

The work presented in [16] is the closest to our work. The authors develop a media-friendly and TCP-friendly congestion control based on TFRC using a two-timescale approach: they compute the long-term average of the throughput according to TFRC, but they modify this throughput on a smaller timescale according to the rate of increase/decrease of a utility function obtained from the rate-distortion characteristics of the stream. There are several differences between their approach and ours. First, the utility function used in [16] was developed for MPEG FGS (*Fine Granular Scalable*) video streams, while our approach targets general layered scalable video streams. Our framework can very well be used for MPEG FGS streams, just that the application needs to decide on a proper streaming bitrate (and our framework does not help the application in this manner), but once this is done, our media-friendly and TCP-friendly congestion control framework does its best to successfully deliver that bitrate. Second, since in [16] the rate-distortion utility function is not scaled with the TFRC throughput function (i.e., they can have different value ranges), we are not sure that this utility function (actually the derivative of the rate-distortion utility function is used in the text) will have significant influence on TFRC's throughput (even on small timescales) in all network scenarios. In other words, we expect the influence of the rate-distortion utility function on TFRC's throughput to be rather small in many network setups.

A totally different approach in congestion control derived from optimization theory is taken in [17], [18]. The authors of these papers formulate the problem of sharing bandwidth in a best-effort network as an optimization problem and derive gradient-like algorithms for solving this problem. These congestion control algorithms obtain optimal bandwidth allocation among competing sources; the allocation is optimal related to the utility functions of all participants. However, these papers use general utility functions and they do not consider the specific characteristics of multimedia streaming applications. Also, there is no experimental evidence that these primal and dual congestion control algorithms are fair to real TCP implementations. Another possible solution to making congestion control more media-friendly is MulTFRC [1] and *n-TCP-friendly congestion control* which try to emulate the throughput of n concurrent TCP flows. But these solutions depart from a pure TCP-friendliness and generally do not support a dynamic value for the n parameter.

III. Making TFRC Media-friendly

Ideally, multimedia streaming applications should never lack bandwidth. However, in best-effort networks this is not always possible as the available network bandwidth changes frequently during the lifetime of the streaming session. TCP and smooth congestion control algorithms like TFRC alleviate to some degree the problem of ever changing network conditions. However, when the average available bandwidth is close to the average bitrate of the stream, a congestion control algorithm with a transmission rate that follows the media characteristics of the stream, e.g., the instant (one-second) bitrate, should be more beneficial for multimedia streaming. Ideally, the transmission rate of a media-friendly congestion control algorithm should track the bitrate of the stream strictly, but this will most probably lead to a TCP-unfriendly behavior. This is why our strategy is to start with a TCP-friendly transmission rate like the one given by TFRC and add media-friendly properties to it. But care must be taken when adding these media-friendly properties so that the final transmission rate remains TCP-friendly in the long term.

An intuitive example of what our framework for building TCP-friendly and media-friendly rate control algorithms is based on, is depicted in Figures 1 and 2. Fig. 1 shows the bitrate evolution of a typical video stream (*Elephants Dream*[1]). This video stream is also used in the simulations section, later in the paper. In Fig. 2, the TFRC line represents a typical evolution of the transmission rate (throughput) of a TFRC flow and the UTFRC line depicts the same throughput modified to track the evolution of the bitrate of the stream from Fig. 1. For readability reasons, only the bitrate of the first 300 stream seconds is used in Fig. 2. We call this modified TFRC behavior *UTFRC* (*Utility-driven TCP-Friendly Rate Control*). We can see that UTFRC increases TFRC's throughput between seconds 100 and 150 because the video stream bitrates in this time interval are higher than average in Fig. 1, but the throughput is decreased in seconds 170 through 300, because the bitrates in this interval are low in Fig. 1.

Mathematically formulated, we seek a TCP-friendly and media-friendly congestion control algorithm with a transmission rate of the following form:

$$X(t) = M(q(t)) * X_{TFRC}(t) \tag{2}$$

where t is time, $X_{TFRC}(t)$ is the transmission rate computed by TFRC at time t using Eq. 1, $M(q(t))$ is a media-friendly function and $q(t)$ is an n-dimensional function giving the values of various media or application characteristics over time:

$$q(t) = (m_1, m_2, .., m_n)(t)$$

where each of $m_1(t), m_2(t), .., m_n(t)$ is a function that measures one media/application characteristic like bitrate, PSNR value, client buffer fill level, etc. The function $M(q(t))$ embodies the usefulness of increasing TFRC's throughput (Eq. 1) according to the requirements of the streaming application. Typically, function $M(q(t))$ is increasing with respect to $q(t)$, as an increase in a media characteristic like bitrate means higher bandwidth requirements of the streaming application.

Before giving the rules which should be followed when choosing the media-friendly function, $M(q)$, in order for the transmission rate to remain TCP-friendly, we give two formal definitions.

Definition 1: A flow is termed *TCP-friendly* if its long-term average transmission rate does not exceed the transmission rate of a TCP flow under the same network conditions [19], [20]:

$$\lim_{t \to \infty} \frac{1}{t} \int_0^t X(s)ds \le X_{TCP}$$

where X_{TCP} is given by Eq. 1 and in this equation R and p are taken as average values across the interval $[0, t]$.

Definition 2: A flow with a transmission rate $X(t) = X(t, q(t))$ where $q(t)$ represents media/application parameters

[1]http://www.elephantsdream.org, resolution 1280x720, 24 fps, average bitrate 1923.6 kbps, encoded using VideoLAN's libx264 H.264 video encoder

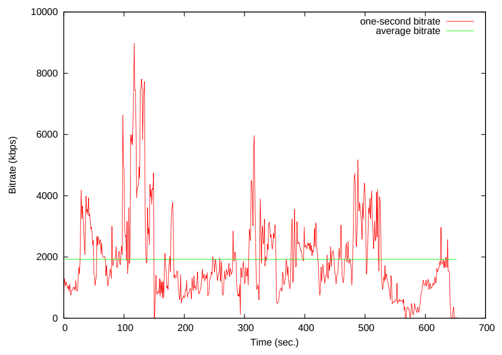


Fig. 1. The bitrate per second of the *Elephants Dream* video stream

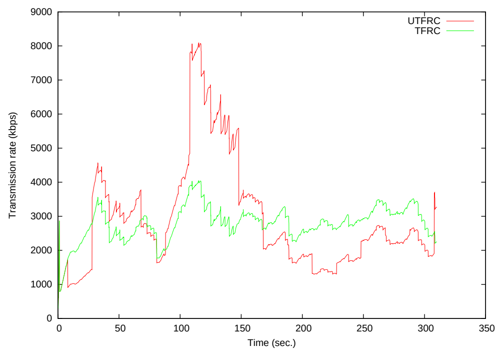


Fig. 2. TFRC throughput modified according to the stream bitrate (only the first 300 seconds of the stream are used). The TFRC line depicts the transmission rate computed by TFRC at each update step (i.e., when a feedback packet is received). The UTFRC line is the transmission rate computed by TFRC multiplied with $b(t)/b_{avg}$ where $b(t)$ is the bitrate for second t of the stream from Fig. 1 and b_{avg} is the average bitrate over the whole stream.

(as defined above) is termed *media-friendly* if the increase/decrease rate of its transmission rate with respect to $q(t)$ follows the shape of a media/application characteristics function:

$$\frac{\partial X}{\partial q(t)}(t, q(t)) = k(t)\frac{\partial M}{\partial q(t)}(q(t)) + o(t)$$

where $k(t) > 0$ is a scaling factor, $M(q(t))$ is a function of media/application characteristics and $o(t)$ is a residual term much smaller than $k(t)\frac{\partial M(q(t))}{\partial q(t)}$. We consider the function $q(t)$ to be very general and it can include one or several media/application characteristics like the evolution of the bitrate of the stream, of its PSNR values, of the client buffer fill level, etc. The simplest and media-friendliest transmission rate would be $X(t) = X(t, q(t)) = M(q(t)) = b(t)$ where t has a 1 second granularity and $b(t)$ is the bitrate of the stream in second t. That transmission rate would indeed be media-friendly, yet not TCP-friendly.

We will now state three theorems that provide guidelines for choosing the media-friendly function, $M(q(t))$, so that the resulting congestion control remains TCP-friendly in the long term.

Theorem 1: If the values of $M(q(t))$ and $X_{TFRC}(t)$ are stochastically independent and $E[M(q(t))] = 1$[2], then the control given in Eq. 2 is TCP-friendly and media-friendly.

Proof: Proving media-friendliness is straight forward if we consider the definition of media-friendliness. In order to prove the TCP-friendliness, we just have to take the expectation of both factors of Eq. 2:

$$\begin{aligned}&E[X(t)] = E[M(q(t)) * X_{TFRC}(t)]\\ &= E[M(q(t))] * E[X_{TFRC}(t)] = E[X_{TFRC}(t)]\end{aligned}$$

where the second equality comes from the fact that values of $M(q(t))$ and $X_{TFRC}(t)$ are stochastically independent and the third equality comes from $E[M(q(t))] = 1$. ■

Basically, the theorem states that if the values of the media-friendly function fluctuate around 1, having the average value of 1, then the long term average transmission rate of the media-friendly congestion control will equal the average transmission rate of TFRC.

Theorem 2: If the values of $M(q(t))$ and $X_{TFRC}(t)$ are positively or negatively correlated and $E[M(q(t))] = 1$, then the following hold:
a) If $M(q(t))$ and $X_{TFRC}(t)$ are negatively correlated, then the control given in Eq. 2 is TCP-friendly.
b) If $M(q(t))$ and $X_{TFRC}(t)$ are positively correlated, then the control given in Eq. 2 is not TCP-friendly (i.e., its throughput is larger than the throughput of a TCP-friendly protocol).

Proof: The proof is similar to the proof of Theorem 1 and is based on the fact that $cov[M(q(t)), X_{TFRC}(t)]$[3] is positive when variables are positively correlated and negative when variables are negatively correlated. ■

Before we state the third theorem of TCP- and media-friendliness, we need to present some notations. The notations are taken from [20]. Let us first consider the following function:

$$g(\theta) = RTT\sqrt{\frac{2}{3\theta}} + 4RTT(3\sqrt{\frac{3}{8\theta}})\frac{1}{\theta}(1 + \frac{32}{\theta^2})$$

where θ is the length (in packets) of a loss interval and RTT is the round-trip time. Note that $g(\theta)$ is just the inverse of the TFRC throughput equation (Eq. 1) considering $p = \frac{1}{\theta}$. In the following lines, we consider RTT to be relatively constant across a connection; this is consistent with the Internet experience where RTT is relatively constant across an Internet path and a connection's path changes rarely. The loss event rate, p, is considered the fraction of loss events observed in the number of packets sent over a long time interval [3]:

$$p = \frac{1}{E[\theta]}$$

where θ is the length of the current loss interval (in packets). $\hat{\theta}$ is an estimator of $1/p$ used by TFRC:

$$\hat{\theta} = \sum_{l=1}^{8} w_l \theta_{-l}$$

θ_{-l} is the length of the l-th most recent loss interval (in packets), $w_l = 1$ for $l \in [1, 4]$ and $w_l = 1 - \frac{l-4}{4+1}$ for $l \in [5, 8]$.

Using the above notation it can be noted that:

$$X_{TFRC}(\hat{\theta}) = \frac{1}{g(\hat{\theta})}$$

where X_{TFRC} is the transmission rate of TFRC and $\hat{\theta}$ is the loss interval estimator used by TFRC. Now the expected long-term transmission rate of X from Eq. 2 can be written as:

$$E[X] = \frac{E[\theta]}{E\left[\frac{\theta\, g(\hat{\theta})}{M(q)}\right]} \qquad (3)$$

The numerator from that equation denotes the expected length in packets of a loss event interval, while the denominator is the expected duration (in time) of a loss event interval.

Theorem 3: Assuming that $cov[\theta, \hat{\theta}] \leq 0$, for any media-friendly function $M(q) : \mathbb{R}^n \to \mathbb{R}$, where functions q measure media, application and network characteristics, satisfying the following:
a) $E[M(q)] = 1$
b) $cov\left[\theta\, g(\hat{\theta}), \frac{1}{M(q)}\right] \geq 0$ (i.e., $\theta\, g(\hat{\theta})$, the time between two loss events, and $\frac{1}{M(q)}$ are stochastically independent or positively correlated),
the control $M(q(t)) * X_{TFRC}(t)$ is TCP-friendly, where $X_{TFRC}(t)$ is the basic TFRC control [20].

Proof: The proof is based on the fact that function g is convex on $[0, \infty)$, just like the proof of Theorem 1 in [20]. All we need to show is that:

$$\frac{E[\theta]}{E\left[\frac{\theta\, g(\hat{\theta})}{M(q(t))}\right]} \leq \frac{1}{g(E[\hat{\theta}])} \qquad (4)$$

We start by observing that function g is convex on $[0, \infty)$. Based on this, we can write:

$$g(\hat{\theta}) \geq (\hat{\theta} - m)g^{'}(m) + g(m) \qquad \text{for all m} \geq 0$$

[2]$E[X]$ is the expectation of random/continuous variable X.
[3]$cov[X, Y]$ is the covariance between variables X and Y.

We then multiply both sides of the above inequation by θ:

$$\theta\, g(\hat{\theta}) \geq (\theta\hat{\theta} - \theta\, m)g^{'}(m) + \theta\, g(m)$$

Taking expectation of both sides yields:

$$E[\theta\, g(\hat{\theta})] \geq E[\theta\hat{\theta} - \theta\, m]g^{'}(m) + E[\theta]g(m)$$

Let $m = E[\theta] = E[\hat{\theta}]$. By multiplying both sides by $\frac{1}{E[M(q(t))]} = 1$, we get:

$$E[\theta\, g(\hat{\theta})]\frac{1}{E[M(q(t))]} \geq cov[\theta, \hat{\theta}]g^{'}(m) + mg(m)$$

By observing that $E\left[\frac{1}{M(q(t))}\right] \geq \frac{1}{E[M(q(t))]}$ since function $\frac{1}{x}$ is convex for $x > 0$, applying Jensen's inequality, and using hypothesis b), we obtain:

$$E\left[\frac{\theta\, g(\hat{\theta})}{M(q(t))}\right] \geq E[\theta\, g(\hat{\theta})]E\left[\frac{1}{M(q(t))}\right] \geq E[\theta\, g(\hat{\theta})]\frac{1}{E[M(q(t))]} \geq cov[\theta, \hat{\theta}]g^{'}(m) + mg(m)$$

Then we divide m by the left-hand and the right-hand sides of the above inequation and we obtain:

$$\frac{m}{E\left[\frac{\theta\, g(\hat{\theta})}{M(q(t))}\right]} \leq \frac{m}{mg(m) + cov[\theta, \hat{\theta}]g'(m)}$$

Written differently, the above inequation looks like:

$$\frac{E[\theta]}{E\left[\frac{\theta\, g(\hat{\theta})}{M(q(t))}\right]} \leq \frac{1}{g(m)}\frac{1}{1 + cov[\theta, \hat{\theta}]\frac{g'(m)}{mg(m)}}$$

Knowing that $m = E[\theta] = E[\hat{\theta}]$, $g(m)$ is a decreasing function of m, $cov[\theta, \hat{\theta}] \leq 0$ and $X_{TFRC}(E[\hat{\theta}]) = \frac{1}{g(E[\hat{\theta}])}$, we can conclude that Eq. 4 is satisfied which is just what we need to prove. ■

There is practical evidence in [20] that $cov[\theta, \hat{\theta}] \leq 0$ (i.e., loss events are stochastically independent).

IV. UTFRC ALGORITHMS

In this section we present four variants of media-friendly and TCP-friendly algorithms constructed using the above framework. All four algorithms have the form given in Eq. 2, the only difference being the form of the media-friendly function, $M(q)$. The algorithms are presented here as examples and proof of concept for our UTFRC framework; the first and the third algorithm are thoroughly considered and evaluated in Section VI.

The first congestion control algorithm includes only the bitrate of the stream as media/application characteristic and has the following media-friendly function:

$$M(q(t)): \qquad M_1(q(t)) = \frac{b(t)}{b_{avg}} \tag{5}$$

where $b(t)$ is the bitrate measured in bits/s (bps) for second t of the stream and b_{avg} is the average bitrate over the whole stream (in bits/s). Based on the definition of media-friendliness, it can be easily shown using Theorem 1 that the following proposition holds:

Proposition 1: The long-term average transmission rate of the simple UTFRC control given by Equations 2 and 5 is TCP-friendly and media-friendly.

The media-friendliness is straightforward to prove and the TCP-friendliness can be proved once we observe that $E[M(q(t))] = 1$.

Among the advantages of this simple UTFRC we mention: it is very simple to implement and not time consuming during streaming, especially if the bitrate for each second is computed off-line (the average bitrate has to be computed off-line). However it has some disadvantages: it does not consider other media or application characteristics besides bitrate, like client buffer fill level, or PSNR values.

The second congestion control algorithm includes also the client buffer fill level (which is stochastically dependent on network parameters, i.e., X_{TFRC}), besides the bitrate of the stream. Its media-friendly function has the following form:

$$M(q(t)): \qquad M_2(q(t)) = \frac{b(t)}{b_{avg}} + \frac{1}{c}(E[\Delta] - \Delta) \tag{6}$$

where $b(t)$ and b_{avg} have the same meaning as in the previous congestion control algorithm, Δ is the current fill level (in seconds) of the client prefetch buffer and $E[\Delta]$ is an estimator of the average fill level of the client prefetch buffer across the whole streaming session. For this media-friendly function, its value is decreased when the buffer is well filled (i.e., $E[\Delta] < \Delta$) and is increased when the buffer is sparsely filled (i.e., $E[\Delta] > \Delta$). The term $\frac{1}{c}, c \geq 1$ is a scaling factor for the term $E[\Delta]-\Delta$ so that the value of the media-friendly function, $M(q(t))$, always stays positive. c must be a number such that $\frac{b_{min}}{b_{avg}} + \frac{1}{c}(E[\Delta]-\Delta_{max}) > 0$ (i.e., $c \geq \frac{b_{avg}}{b_{min}} \cdot (\Delta_{max}-E[\Delta])$) where b_{min} is the minimum one second bitrate value of the stream and Δ_{max} is the maximum buffer fill level across the streaming session. In addition, the value of $M(q(t))$ must be upper bounded by a maximum value, so that it does not generate a transmission rate significantly higher than TFRC's transmission rate. Using Theorem 3, it can be proved that the second congestion control algorithm is media-friendly and TCP-friendly in the long term:

Proposition 2: The long-term average transmission rate of the simple UTFRC control given by Equations 2 and 6 is TCP-friendly and media-friendly.

The media-friendliness is straightforward to prove and the TCP-friendliness can be proved using Theorem 3, once we observe that $E[M(q(t))] = 1$ if $E[\Delta]$ is a good estimator of the average fill level of the client prefetch buffer across the whole streaming session and we note that the time passed between two loss events is stochastically independent with the inverse of the media-friendly function from Equation 6 (i.e., condition b) of Theorem 3).

We can build a more advanced media-friendly TFRC by considering the following guidelines:

- If the client prefetch buffer is sparsely filled (e.g., the fill level is smaller than a threshold), the utility (i.e., media-friendly value) should be high. The rationale is that if there were not a large enough throughput, the buffer at the client might run empty and stream playout might stall.
- If the buffer is well filled, then the utility (i.e., media-friendly value) should be small, but still it should follow the slope of the bitrate.

The goal is to keep the buffer fill level at a reasonable safe value. The media-friendly function of this congestion control algorithm is:

$$M(q(t)): \qquad M_3(q(t)) = M(b,\Delta) = U_b(b) + U_\Delta(\Delta) \quad (7)$$

where b is the bitrate for the current stream second and Δ is the number of stream seconds saved in the client's buffer. Similar to all the other media-friendly functions presented in this section, the values of $M(b,\Delta)$ should oscillate around 1. Let $minM$ be the minimum value of $M(b,\Delta)$ and $maxM$ be the maximum value of $M(b,\Delta)$ so that $M(b,\Delta) \in [minM, maxM]$. Also, let W_1 be a number in the interval $(minM, maxM)$. Function $U_b(b)$ is a linear mapping of the current second's bitrate b from the interval $[b_{min}, b_{max}]$ of all possible one-second bitrates of the stream to the interval $[minM, W_1]$ (b_{min} refers to the minimum and b_{max} to the maximum one-second bitrate of the entire streaming process):

$$U_b(b) = minM + (W_1 - minM) \cdot \frac{b - b_{min}}{b_{max} - b_{min}} \quad (8)$$

The higher the bitrate of the current stream second, b, the higher the utility value of $U_b(b)$ will be. Because function $U_b(b)$ is a linear mapping on the interval $[minM, W_1]$ and, as we will see, function $U_\Delta(\Delta)$ is a linear mapping on the interval $[0, maxM - W_1]$, we can say that W_1 defines the relative weights of the $U_b(b)$ and $U_\Delta(\Delta)$ terms in the end value of the media-friendly function $M(b,\Delta)$. The higher the value of W_1 is, the higher the contribution of the $U_b(b)$ term will be to the value of $M(b,\Delta)$ and the smaller the contribution of $U_\Delta(\Delta)$ will be. For example, if $minM = 0.5$, $maxM = 2.1$ and $W_1 = 1.7$, then function $U_b(b)$ would be responsible for approximately $3/4$ of the $M(b,\Delta)$ value and function $U_\Delta(\Delta)$ would be responsible for approximately $1/4$ of its value.

Function $U_\Delta(\Delta)$ takes values in the interval $[W_2, maxM - W_1]$ when Δ is smaller than a threshold value and in the interval $[0, W_2]$ when Δ is greater than the threshold value; W_2 is a number such that $0 < W_2 < maxM - W_1$. The expression of the function is as follows:

$$U_\Delta(\Delta) = \begin{cases} (maxM - W_1) \cdot \Big(1 - \frac{(maxM - W_1) - W_2}{maxM - W_1} \cdot \frac{\Delta}{thresh} \\ \quad - max\left(0, \frac{thresh - E[\Delta]}{2thresh}\right)\Big) & , \Delta \le thresh \\ W_2 \cdot \left(1 - \frac{\Delta - thresh}{\Delta_{max} - thresh}\right) & , \Delta > thresh \end{cases} \quad (9)$$

where $E[\Delta]$ is the average buffer value from the beginning of the streaming session up to now and Δ_{max} is a reasonable maximum buffer value. The term $max\left(0, \frac{thresh - E[\Delta]}{2thresh}\right)$ is a penalty term and was introduced as an incentive for applications which maintain a lower buffer in order to get a higher utility and, thus, a larger throughput; for those applications, the value $E[\Delta]$ is small and because of this term, $U_\Delta(\Delta)$ gets smaller over time. The first branch of the function $U_\Delta(\Delta)$ is, ignoring the penalty term, an inverse linear mapping of the current buffer value, Δ, from the interval $[0, thresh]$ to the interval $[W_2, maxM - W_1]$ and the second branch is also an inverse linear mapping of Δ from the interval $[thresh, \Delta_{max}]$ to the interval $[0, W_2]$. The parameter W_2 is, like the parameter W_1, a weight parameter that defines the relative weight between the two branches of function $U_\Delta(\Delta)$. The smaller the value of Δ is, the higher the value of $U_\Delta(\Delta)$ will be. The threshold value $thresh$ in the above equation has to be an estimator of the average buffer fill level across the entire streaming session. More information on computing the value of $thresh$ is given in Section V-D.

Equations 7–9 describe a family of media-friendly functions characterized by parameters: $minM$, $maxM$, W_1, W_2 and $thresh$. We used in our simulations specific values for these parameters which are detailed in Section V-D. In order to prove that the congestion control algorithm described in Equations 2 and 7–9 is TCP-friendly in the long term, we cannot directly apply Theorem 3 because $E[M(b,\Delta)] \neq 1$. But we can use a transformation for the bitrate values, b, so that we get $E[M(b,\Delta)] \approx 1$. Due to the way function $U_\Delta(\Delta)$ is defined, we can expect the average value for Δ to be around $thresh$ and the expectation $E[U_\Delta(\Delta)]$ to be around the value W_2. This lets the expectation of $M(b,\Delta)$ be:

$$E[M(b,\Delta)] \approx E[U_b(b)] + W_2$$

All we need to do is to get $E[U_b(b)]$ to be $1 - W_2$. We do this by modifying the bitrate distribution of our video stream while keeping as much as possible the same relative differences between bitrates. This is presented in Section V-D.

The fourth media-friendly and TCP-friendly rate control algorithm we present in the following includes the "signal energy" or "signal power" of each frame from the video stream in the media-friendly function expression. By the "signal energy" of a frame we understand the variability (i.e., difference) of that frame with respect to the previous one, that is the non-redundant, original information that this frame adds to the video stream. In order to quantify the signal energy contained in each video frame, we compute for each frame the distortion induced in the perceived stream by not delivering that specific frame (and thus playing the previous frame once more). For measuring this distortion we use a simple MSE (*Mean Squared Error*) metric. The MSE metric is always computed between the current frame and the previous one. Note that also PSNR (*Peak Signal to Noise Ratio*) could have been used for measuring the distortion caused by a missing frame. After having computed the signal energy contained in each video frame (i.e., the distortion induced by not delivering that frame), we compute using these values an average signal energy across the whole video stream. All these computations are typically done off-line, but can be done on-line also at computational costs.

During streaming, whenever UTFRC updates its transmission rate (i.e., once per RTT or when a loss event is detected, whichever comes first), it uses for the media-friendly function a value greater than 1 if the signal energy of the current streaming second is above average (i.e., the distortion in case of frame loss is above average) or a value smaller than 1 if the signal energy of the current streaming second is below average (i.e., the distortion in case of frame loss is below average). The signal energy of a stream second is the sum of the signal energy of each frame from that specific stream second and the signal energy of a frame is, as specified above, the MSE between that frame and the previous frame in the stream. In this manner, the transmission rate will also track the signal energy distribution of the video stream (i.e., the rate control algorithm is thus, media-friendly). Thus, the media-friendly function will have high values for video stream seconds which contain a lot of movement or scene changes and will have low values for more static video stream seconds.

More specifically, the distortion-based media-friendly congestion control has the following media-friendly function:

$$M(q(t)): \qquad M_4(q(t)) = \frac{SE(t)}{SE_{avg}} \tag{10}$$

where the time t has a one-second granularity, $SE(t)$ is the signal energy for the t-th second of the video stream and SE_{avg} is the average of $SE(t)$ computed across all the seconds of the video stream.

Once we observe that $E[\frac{SE(t)}{SE_{avg}}] = 1$ and the fact that the signal energy distribution across the stream is stochastically independent of any network parameter, we can easily show using Theorem 1 that the following proposition holds:

Proposition 3: The long-term average transmission rate of the simple UTFRC control given by Equations 2 and 10 is TCP-friendly and media-friendly.

V. Implementation Considerations

Before we present the simulations, it is necessary to properly define UTFRC as a congestion control protocol, because Equation 2 just presents the formula for computing the transmission rate and is very laconic, omitting the time-scales of the protocol, feedback type, feedback times, etc. Such practical considerations will be worked out in the following subsections.

Since the four variants of UTFRC protocols given in the previous section are just examples of possible media-friendly functions suited for the UTFRC protocol and no claim is made that these functions are the best media-friendly functions that can be used in UTFRC, we only consider in the remainder of the paper the first and the third media-friendly function. However, the next two subsections, V-A and V-B, still apply to any general UTFRC protocol defined as in Equation 2.

Since UTFRC is based on TFRC, it shares the same feedback and transmission semantics, but it adds the media-friendly factor to the transmission rate formula. UTFRC uses the same type of client feedback as TFRC, i.e., loss-event based feedback, and has two time-scales (i.e., the time intervals after which the protocol updates its state): the time-scale of the media-friendly factor (i.e., $M(q(t))$ from Equation 2) and the time-scale of the TFRC factor (i.e., $X_{TFRC}(t)$ from Equation 2). The time-scale of the TFRC factor is given by the feedback rate of TFRC, i.e., once per round-trip time interval or immediately after a loss event is detected, whichever comes first. UTFRC reevaluates its transmission rate at this time-scale. The media-factor time-scale is discussed in the following subsection.

A. Time-scale of the Media-friendly Factor

Normally, there should be only one time-scale for the UTFRC protocol presented in Equation 2, which should be given by the feedback interval of TFRC (i.e., the time-scale of the TFRC factor, $X_{TFRC}(t)$). However, as the feedback interval of TFRC is one RTT or less, and a variable value that is only known during the streaming session, it cannot be used before the beginning of the streaming session when the static parts of the media-friendly factor (e.g., the bitrate values) are being constructed. Also, a time value so small for the media-factor update interval is likely to produce highly variable values that are not very good at capturing the global utility trajectory of the video stream. Having highly variable values for the utility function, especially within small time intervals, will incur large fluctuations in the overall transmission rate (computed by Equation 2) which is not desired for a congestion control protocol; such large fluctuations often lead to increased congestion, unfair utilization of network bandwidth, and ultimately to under-utilization of the network capacity. All these reasons suggest that the time-scale of the media-friendly function, $M(q(t))$, (i.e., the update time interval of this function) should be at least one second long. Generally, an update interval of several seconds is a good time-scale for the media-friendly function, especially for video streams that have highly variable media/application characteristics (e.g., highly variable bitrate).

The media-friendly function can have two types of parameters: *static* parameters whose values can be determined before the streaming session starts (e.g., bitrate values, signal power) and *dynamic* parameters that have values which can only be computed during the streaming session (e.g., client buffer fill levels, packet delay). For example, the first media-friendly function described in Equation 5 has only a static part, because it uses only values that can be evaluated before streaming starts (e.g., bitrate values). In contrast, the third media-friendly function described by Equation 7 has a static part, $U_b(b)$, and a dynamic part, $U_\Delta(\Delta)$.

We have chosen a time-scale of at least one second for the media-friendly functions used in UTFRC protocols because of the reasons discussed above, so the media-friendly function changes its values at the beginning of each streaming second. In addition, in case of video streams with highly fluctuating media-friendly function values, these utility values should be further averaged over several seconds or the function should be smoothed out for a more media-friendly behavior of the protocol. This averaging/smoothing should also be done in order to keep the values of the media-friendly function approximately between 0.5 and 2.0, because we have seen in our experiments that if the media-friendly factor is higher than 2.0 for a long period of time (e.g., one to several tens of seconds), the network-friendly protocol part (i.e., the TFRC factor) will see a smaller loss-event rate than the loss-event rate seen by another normal TFRC flow under the same network conditions; thus, it would see a TCP-friendly transmission rate larger than the transmission rate computed by a normal TFRC flow. This behavior is also documented in several papers, for example [4] and [1]. On the other hand, if the media-friendly factor is below 0.5, the $X_{TFRC}(t)$ computed by UTFRC is smaller than the transmission rate of a TFRC flow under the same network conditions; this is due to intrinsic restrictions of TFRC that have the purpose to smooth out the throughput (more specifically, the rule $X_{TFRC} \leq 2 \times X_{recv}$ where X_{recv} is the bandwidth received since the last feedback).

As each value of the media-friendly function, at least its static part, characterizes a distinct second of the video stream, the cardinality of the set of media-friendly function values equals the length of the video stream in seconds. For the rest of the section, it is useful to consider the values of the media-friendly function (either the whole function, or just the static component of the function) as a vector with n utility values, where n is the length of the video stream in seconds. But the t parameter in Equation 2 tracks the streaming time, not the media time. If the streaming session ends early and takes less than n seconds (for example, if the available bandwidth in the network is much larger than the average bitrate of the stream), then not all the media-friendly values from the utility vector would be used during the streaming session. Conversely, if the available bandwidth in the network is less than the average bitrate of the stream, playing might freeze at the client, additional buffering might be required, and the duration of the entire streaming session might be larger than n seconds. This is problematic, since the values of the static part of the media-friendly function were chosen in such a way that the average of those values across the n seconds of the stream has a specific value (i.e., $E[\frac{b(t)}{b_{avg}}] = 1$ for the first media-friendly function and $E[U_b(b(t))] = 1 - W_2$ for the third media-friendly function). This contributes to the TCP-friendliness of the protocol.

Thus, whenever the streaming time is ahead of playout time by a significant time interval (i.e., when there is a significant time distance between the video second that is currently streamed by the server and the video second that is currently played out by the client), the utility array (i.e., media-friendly values) must be adapted, meaning that a number of media-friendly values from the end of this utility array must be eliminated and all the media-friendly values that were not used up to this moment must be proportionally reduced or increased so that when the streaming is finished, the overall average utility of the used utility values must be the desired one (i.e., 1 for the first media-friendly function and $1 - W_2$ for the third function). Conversely, when the streaming session takes longer than n seconds (i.e., the length of the video stream) due to buffering and stalled playing, the utility vector is supplemented with enough utility values, each of them being equal to the average utility value across the initial n seconds of the video (e.g., 1 for the first media-friendly function and $1-W_2$ for the static part of the third media-friendly function).

B. Enforcing TCP-friendly Behavior

Theorems 1, 2 and 3 prove that under special conditions, the expected long-term transmission rate of a UTFRC flow is not more than the expected long-term transmission rate computed using the TCP throughput equation. The Law of Large Numbers assures that if the streaming session lasts for a long time interval (possibly infinite), the expected transmission rate will equal the long-term average transmission rate. But this does not happen in the case of an actual real streaming session which has a finite time duration. Although the expected transmission rate is close to the long-term average transmission rate, they can be different. To enforce the equality of the long-term average transmission rate of UTFRC to the long-term average transmission rate computed using the TCP throughput equation, we use a mechanism of *virtual bandwidth* that is either borrowed from or to other flows in order to satisfy media constraints. This *virtual bandwidth* is the total amount of bandwidth that the UTFRC flow uses above its fair share rate computed by TFRC. But it is also the bandwidth that is returned back to the network, when the utility value (i.e., media-friendly value) of UTFRC is smaller than 1; in this case the *virtual bandwidth* is negative. This *virtual bandwidth* is maintained at the sender using a variable called $virtual\_bandwidth\_bytes$ (for the sake of brevity abbreviated as vbb in the following) which does not store a bandwidth value per se, but the data volume (transmitted bytes) component of a bandwidth value (without the time component). This variable is enough to approximate the *virtual bandwidth* that is borrowed from the network (above the TCP-fair rate) and then returned back to the network in a later time interval. vbb is initialized as 0 in the beginning of the streaming session and is updated using the following formula whenever a new feedback is received from the client:

$$vbb = vbb + (B - R_{TFRC} * D)$$

where D is the duration of the last feedback interval, the interval ended by the receipt of the current feedback packet, B is the number of bytes sent in the last feedback interval and R_{TFRC} is the transmission rate computed by TFRC (i.e., without the media-friendly factor) for the last feedback interval. If the media-friendly factor was less than 1 in the last feedback interval, *virtual bandwidth* is decreased, and if the factor was more than 1, then *virtual bandwidth* increases.

We want to make sure that, when the streaming session ends, the *virtual bandwidth* will ideally be 0 or close to 0 which roughly means that the extra bandwidth (i.e., above the rate computed by TFRC) that was borrowed from the network by the media-friendly factor was returned back to the network. In order to enforce this, we must identify a point in streaming time after which media-friendly function values will be ignored and the transmission rate is either decreased below the transmission rate computed by TFRC (if *virtual bandwidth* is positive) or the transmission rate is increased above the transmission rate computed by TFRC (if *virtual bandwidth* is negative) until the end of the streaming session. Also we must choose the increase/decrease factor for every transmission rate update. Let this be α; we use a constant increase/decrease factor because it is simple to implement. Let also $left\_to\_stream$ denote the remaining bytes from the video stream left to be streamed and X the current TFRC computed transmission rate.

If vbb is positive, at the current decreased transmission rate $X - X \cdot \alpha$, the streaming will end in $\frac{left\_to\_stream}{X*(1-\alpha)}$ seconds. Due to α, the vbb will also be decreased in each transmission round (because we give away in each transmission round $X \cdot \alpha$ from the bandwidth that is rightfully ours), and in $\frac{vbb}{\alpha * X}$ seconds vbb will drop to 0. If we equal these two time values, we get the time point after which we must decrease the transmission rate by a factor of α, so that when streaming ends, vbb will be 0:

$$\frac{left\_to\_stream}{X*(1-\alpha)} = \frac{vbb}{\alpha * X}$$

We use a constant decrease factor of $\alpha = 0.5$ (so the transmission rate is decreased by a factor of $1-\alpha = 0.5$), so we start the decrease process when $vbb \geq left\_to\_stream * \frac{\alpha}{1-\alpha} = left\_to\_stream$.

Conversely, when vbb is negative and the transmission rate is increased by $X \cdot \alpha$, the streaming will end in $\frac{left\_to\_stream}{X*(1+\alpha)}$ seconds and vbb will become 0 after $\frac{-vbb}{\alpha * X}$ seconds. The corresponding time equality will be:

$$\frac{left\_to\_stream}{X*(1+\alpha)} = \frac{-vbb}{\alpha * X}$$

We also use $\alpha = 0.5$ for the increase factor (so the transmission rate is increased by a factor of $1+\alpha = 1.5$), so the increase process starts when $-vbb \geq left\_to\_stream * \frac{\alpha}{1+\alpha} = left\_to\_stream * \frac{1}{3}$.

C. Specific Settings for the First UTFRC Variant

This subsection presents specific settings for the first UTFRC variant used in the simulations. For the first UTFRC variant we precompute the media-friendly function values for each second of the video using Equation 5 and store them in the *utility vector*. Because these utility values are highly fluctuating for our test video stream used in the simulations (*Elephants Dream*), we further aggregated the utility values by computing the average utility across 40 consecutive seconds and used this average value for each of the 40 consecutive seconds. Averaging utilities over 40 seconds was enough to have all utilities between 0.5 and 2.0 for our test video. During the streaming session, the transmission rate is updated every time feedback is received from the client using Equation 2 where the media-friendly factor is taken from the precomputed utility vector for the current streaming second. During streaming, if the streaming time is ahead of playout time by 10 seconds and if under the current transmission rate this difference is estimated to last until the end of streaming, the utility vector is adjusted by eliminating the last 10 values from it. Assuming the new length of the utility vector (after removal of the last 10 utilities) is n and the current streaming second is denoted by $current\_second$, we must scale the unused utilities (i.e., utilities from the vector that are after the $current\_second$ index) so that the average of all used utilities by the time the streaming is finished is 1. The following scale factor must be added to every component $utility[i]$ where $i \geq current\_second$:

$$scale\_factor = \frac{n - used\_utilities - unused\_utilities}{n - current\_second}$$

where $used\_utilities$ are utilities up to the $current\_second$ and $unused\_utilities$ are utilities from the $current\_second$ to the new end of the utility vector, n. If, on the other hand, streaming the n seconds of the video stream takes longer than n seconds due to playing being stalled caused by an empty client buffer, the utility vector will be supplemented with enough utility values of 1.

D. Specific Settings for the Third UTFRC Variant

For the third UTFRC variant we used the following parameter settings in our simulations: $minM = 0.5$, $maxM = 2.1$, $W_1 = 1.7$ and $W_2 = 0.15$. $minM$ and $maxM$ were set following the argumentation presented in Section V-A. W_1 was chosen so that it gives a larger weight to the evolution of the stream's bitrate (i.e., to function $U_b(b)$) and a smaller weight to the evolution of the client buffer (i.e., to function $U_\Delta(\Delta)$). W_2 gives a slightly larger interval of values to the first branch of function $U_\Delta(\Delta)$ (i.e., when Δ is small). The values for W_1 and W_2 worked best for our experiments and although there is a large set of values for these two parameters that should work, finding the best values for the parameters of the third media-friendly function is outside the scope of this paper. We precomputed the values of the static part of the media-friendly function, $U_b(b)$, for each second of the video using Equation 8. But before we used Equation 8, we transformed the distribution of one-second bitrate values of the video so that $E[U_b(b_i)] = U_b(b_{avg}) = 0.85$ (i.e., $1 - W_2$), where b_{avg} is the average bitrate over all seconds of the video. This is required by condition a) of Theorem 3 (assuming we can ensure that $E[U_\Delta(\Delta_i)] = W_2 = 0.15$, which is discussed later in this subsection). One way to do this transformation while keeping as much as possible the relative differences between the bitrate values is to keep b_{min} and b_{max} unchanged and to add or subtract a constant from all other (one-second) bitrate values so that the value of the function $U_b()$ for the new average will be 0.85. This constant is $(new\_b_{avg} - b_{avg}) * \frac{n}{n-2}$ where n is the length in seconds of the video stream and new_b_{avg} is the new average bitrate after applying the transformation. After applying this transformation to the one-second bitrate distribution, we may further need to average the bitrate values over groups of consecutive seconds and use this average value for each of the seconds from that group for the same reasons as mentioned for the first UTFRC variant (i.e., avoid high fluctuations of utility values). In the simulations in Section VI we average the bitrate values for groups of 40 consecutive seconds. Then we can compute the values of the static part of the media-friendly function, $U_b()$, for all the seconds in the stream. The dynamic part, $U_\Delta()$, will be computed in real time during the streaming session.

During the streaming session, just like for the first UTFRC variant, the transmission rate is updated every time feedback is received from the client using Equation 2. The media-friendly factor is computed using the precomputed $U_b()$ values and the real-time computed $U_\Delta()$ values. The static utility values, $U_b(b_i)$, are adapted using a mechanism identical to the one presented for the first UTFRC variant whenever the streaming time is ahead of playout time by 10 seconds, with the notable difference that the average of $U_b(b_i)$ is now 0.85, not 1. The dynamic utility values, $U_\Delta(\Delta_i)$, are computed using Equation 9 at the beginning of each second. In order for condition a) of Theorem 3 to be valid, we need the threshold variable, $thresh$, from Equation 9 to be an estimator of the average buffer fill level across the entire streaming session. If the instant buffer fill level (i.e., Δ) is around $thresh$ most of the time, then, although we can not warrant this, there are high chances that the average of $U_\Delta(\Delta_i)$ is 0.15. The *virtual bandwidth* mechanism described in a previous subsection will anyway enforce TCP-friendly behavior of the protocol. The estimator of the average buffer fill level, $thresh$, is computed in each second of streaming using the following algorithm:

```
streaming_duration = (n*b_avg) / tfrc_rate;
thresh = 0;
current_buffer = 0;
old_rate=(used_utilities/(current_second+1))*tfrc_rate;
transmission_rate = old_rate;
for (i=0; i<=current_second; i++) {
        current_buffer +=transmission_rate - b[i];
        if (current_buffer>0) thresh +=current_buffer;
}
future_rate=(unused_utilities/(n-current_second))*tfrc_rate;
transmission_rate = future_rate;
for (i=current_second+1; i<streaming_duration; i++) {
        current_buffer +=transmission_rate - b[i];
        if (current_buffer>0) thresh +=current_buffer;
}
thresh = thresh / (b_avg*streaming_duration);
```

In this algorithm we use the following notations: *streaming_duration* is the estimated duration of the streaming session at the current constant transmission rate (*tfrc_rate*), *n* is the length in seconds of the video stream, *current_second* is the current transmission second, *b[i]* is the bitrate of the *i*-th second of the video, *tfrc_rate* is the average transmission rate computed by TFRC (up to *current_second*), *used_utilities* is the sum of all static utilities (i.e., values of $U_b()$) used up to now, *unused_utilities* is the sum of unused static utilities (i.e., the values $U_b(b[i])$ for $i > current\_second$). We have used the static utilities when computing $transmission\_rate$ because we want to estimate the average client buffer as produced by the TFRC transmission rate and the static part of the media-factor (which is normally higher than the average client buffer produced just by using the TFRC-computed transmission rate).

The Δ_{max} threshold from the $U_\Delta()$ formula (Equation 9) is set in our simulations to 10 times the initial buffer (i.e., the buffer built up in the first 8 seconds of the streaming session). Also, because the term $max\left(0, \frac{thresh-E[\Delta]}{2thresh}\right)$ from the $U_\Delta()$ formula is intended to decrease the long-term utility value for flows that maintain a small buffer fill level at the client for a long time, it should not be applied in the beginning of the streaming when the buffer fill level is small; the same holds for $E[\Delta]$ which is computed from the instant client buffer fill levels measured each second up to the present time. In our simulations, we apply this penalty term only after a time interval of at least one third of the video stream's length in seconds has passed from the beginning of streaming.

VI. Simulations

In this section we discuss the results of the simulations we have performed using the ns-2 simulator in order to assess the value of the Utility-driven TCP-Friendly Rate Control (UTFRC) approach. As said in the previous section, we only consider in these simulations the first and the third media-friendly function. We show that they are suitable functions for UTFRC and that UTFRC is beneficial for video streaming applications.

In order to be of use for transporting video streams over the Internet, UTFRC must have two properties: it must be TCP-friendly to other flows and be media-friendly to the streaming application. So in our tests, we first check whether the average transmission rate of a UTFRC flow is similar to the average transmission rate of a TCP-friendly flow (e.g., a TFRC flow) under the same network conditions; second, we validate that streaming video data over UTFRC is actually beneficial for the streaming application, i.e., streaming video data over UTFRC is better than streaming video data over existing TCP-friendly congestion control protocols like TFRC. We assess this improvement by considering the client buffer(s): a smaller number of empty client buffer events/periods (thus, fewer/shorter playout stalls) and a smaller variation in client buffer fill levels for UTFRC are expected to show the superiority of UTFRC as compared to TFRC.

As we will see later in this section it is very difficult for a UTFRC flow to obtain exactly the same average throughput as a TFRC flow under the same network conditions. TCP does not guarantee this equality, nor does TFRC. This is shown for example in [7], Figure 9, where the difference between average transmission rates of two TCP or TFRC flows competing in the same network is between 20% and 40%.

We use in the simulations the bitrate distribution of the *Elephants Dream* movie which has an average bitrate of 1923.6 kbps and is 653 seconds long. The bitrate per second of this movie was depicted in Fig. 1. We use two ns-2 network setups for our simulations, a simplistic one and a complex one. In the simplistic setup, the network is shared by 16 TCP flows sending FTP data and 2 TFRC flows, denoted here by TFRC1 and TFRC2, which were streaming the *Elephants Dream* movie. The topology of the network is a typical dumbbell topology. Each flow has its own independent source and destination nodes which are linked to the bottleneck link by independent access links having a one-way delay between 1 and 5 ms. The one-way delay of the bottleneck link is 50 ms, the queue policy is RED with default parameters and the queue size is twice the bandwidth-delay product. The capacity of the bottleneck link is varied between experiments from 60 Mbps to 36 Mbps, with 4 Mbps decrements. So, the network capacity has the following 7 values in the simple network setup: 60 Mbps, 56 Mbps, 52 Mbps, 48 Mbps, 44 Mbps, 40 Mbps, and 36 Mbps. We have chosen these network capacity values so that a regular TFRC flow will be able to stream the test video with no empty buffer times (in our network setup) when the 60 Mbps value is used, and then starts experiencing an increasing number of empty buffer times, as the network capacity gradually decreases to 36 Mbps. We want to see how UTFRC behaves in situations in which TFRC is not able to avoid empty client buffers. For each of these 7 network capacities, we run 3 simulations:

- **No media-friendly** - a simulation in which TFRC1 and TFRC2 are regular TFRC flows;
- **1st media-friendly function** - a simulation in which TFRC1 is modified to be a UTFRC flow using the first media-friendly function (called UTFRC1(M_1) in the following text) and TFRC2 is a regular TFRC flow;
- **3rd media-friendly function** - a simulation in which TFRC1 is modified to be a UTFRC flow using the third media-friendly function (called UTFRC1(M_3) in the

following text) and TFRC2 is a regular TFRC flow.

The TCP versions we use are TCP Reno and NewReno. The TCP flows start at random times distributed between seconds 1 and 2 of the simulation. The TFRC/UTFRC flows always start at second 0 of the simulation. We implemented the player buffer in ns-2 at the server-side, not the client-side, so that its values are readily available to the congestion control algorithm. The buffer is filled by packets as soon as they are sent by the server and it is drained at the end of each playout second by the amount of bytes of that specific video second. If the buffer does not have enough video data at the end of a playout second, then playing is stalled until the buffer is filled with enough video data so that the playout can move to the next second. In this manner, the buffer does not take into account the propagation delay from the server to the client. Although it had been more realistic for the buffer to be implemented at the client side, because RTT is relatively constant in our simulations and because, as we have seen in the simulations, empty buffer times tend to be clustered in consecutive seconds, we do not expect significant changes if the buffer had been simulated at the client side. Also, this way of simulating the playout buffer affects in the same manner the TFRC flows and the UTFRC flows we are comparing.

A final observation related to the media-friendly factor is that in our simulations, at the beginning of the streaming process, there is an initial buffer ramp up period of 8 seconds in which playing has not yet started and the transmission rate used by the flow is the one computed by TFRC. Playing of the video and consequently, the draining of the client buffer, is only started after these 8 seconds initial buffering period. We do not apply the media-friendly factor in these 8 seconds period because the TFRC-computed transmission rate is just reaching a steady-state level and fluctuates a lot.

The simulation lasts for 657 seconds, which is a little more than the length of our video stream. In all cases, we measure the playout buffer fill level, the average throughput, the transmission rates, and the loss event rate. Due to space limitation we only summarize in Table I our findings. In Table I, the columns represent the network capacity for the bottleneck link used in the simulations (60 Mbps, 56 Mbps, etc.) and the rows present measurements for flows TFRC1 and TFRC2 in each of the three types of simulations: the no media-friendly simulation, the 1st media-friendly function simulation, and the 3rd media-friendly function simulation. For each of the two flows, the number (count) of independent seconds when the playout buffer was empty, the average throughput at the end of the streaming session, and the time the streaming session ended are recorded. For example, in the no media-friendly simulation scenario with a network capacity of 60 Mbps, flow TFRC1 had 0 times an empty playout buffer, had an average throughput of 3.139 Mbps at the end of the streaming session and finished streaming the whole video to the client at time 400.18 seconds of the simulation. In each simulation, the average throughput attained by TCP flows is also displayed in order to outline the TCP-friendliness of UTFRC. We highlight in bold font the situations when the number of empty buffer seconds is greater than zero. We can see here that UTFRC gives fewer empty buffer events for flow TFRC1, either when the first or the third media-friendly function is being used, while maintaining TCP-friendliness.

Because we cannot present in detail the measurements of each cell of this table (due to the large number of figures this would yield), we take a typical simulation, for example the one when the network capacity is 44 Mbps, and present for this capacity the detailed evolution of flows UTFRC1 (using the M_1 and M_3 media-friendly functions) and TFRC2 in Figures 3 to 9. Figure 3 presents the average throughput of flows UTFRC1(M_1) and TFRC2 and we can see here that at the end of the streaming, the average throughput for flow UTFRC1(M_1) is very close to the average throughput of TFRC2. The same can be seen in Figure 4 for the case when the third media-friendly function is used for the UTFRC flow. There are two lines drawn for the UTFRC flow, in each of Figures 3 and 4. The red line plots the average throughput obtained by the UTFRC flow and the green line plots the average throughput of the UTFRC flow but considering only the TFRC component from Equation 2 without the media-friendly factor. Figures 5 and 6 present the buffer fill levels of flows UTFRC1 and TFRC2 when the first media-friendly function is used (in Figure 5) and when the third media-friendly function is used (in Figure 6). Although both flows stream the same video and, as we have seen in Figures 3 and 4, they have the same average throughput at the end, we can see that the buffer fill level is much higher for the UTFRC flow. Figure 7 presents the transmission rate evolution for flows UTFRC1(M_1) and TFRC2. The transmission rate obtained when the third media-friendly function is used is similar to the transmission rate obtained by flow UTFRC1(M_1) and we omit this figure. Finally, Figures 8 and 9 show that UTFRC does not modify significantly the loss-event rate seen by the flow in comparison to the loss-event rate seen by a TFRC flow. The first media-friendly function (M_1) is being used in Figure 8, while the third media-friendly function (M_3) is being used in Figure 9 for the UTFRC flow.

In the complex network setup, the network topology is identical to the one used in the simple setup, but the number of flows sharing the network has been increased. This time, the network is shared by 16 TCP flows sending FTP data and 8 TFRC flows (denoted here by TFRC1, TFRC2, TFRC3, TFRC4, TFRC5, TFRC6, TFRC7, TFRC8) which are streaming versions of the *Elephants Dream* movie. These versions of the *Elephants Dream* movie were obtained from the original *Elephants Dream* movie (whose bitrate distribution is depicted in Figure 1) by shifting around groups of consecutive seconds from the video (i.e., changing the order in the stream

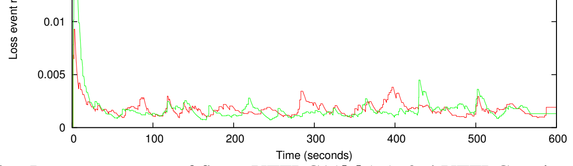


Fig. 9. Loss-event rates of flows UTFRC1(M_3) (a 3rd UTFRC variant flow) and TFRC2 (regular TFRC flow)

		Network capacity						
		60 Mbps	56 Mbps	52 Mbps	48 Mbps	44 Mbps	40 Mbps	36 Mbps
No media-friendly	TFRC1	0 (3.139) 400.18	**37** (2.614) 480.56	**65** (2.503) 501.84	**77** (2.322) 541.01	**211** (2.145) 585.72	**526** (1.973) 636.69	**609** (1.754) -
	TFRC2	0 (3.059) 410.57	**41** (2.653) 473.47	**75** (2.480) 506.55	**83** (2.431) 516.78	**124** (2.280) 550.94	**536** (2.005) 626.50	**565** (1.780) -
	Avg. TCP	2.786	2.682	2.485	2.316	2.129	1.966	1.817
1st media-friendly function	TFRC1 (UTFRC1(M_1))	0 (2.736) 459.12	0 (2.719) 462.00	0 (2.917) 430.59	0 (2.432) 516.58	0 (2.241) 560.57	**129** (1.926) 652.20	**152** (1.910) -
	TFRC2	0 (3.017) 416.28	**4** (2.787) 450.72	**82** (2.529) 496.69	**94** (2.294) 547.56	**254** (2.188) 574.10	**546** (1.978) 634.96	**607** (1.745) -
	Avg. TCP	2.810	2.663	2.462	2.332	2.137	1.980	1.816
3rd media-friendly function	TFRC1 (UTFRC1(M_3))	0 (2.845) 441.59	0 (2.587) 485.46	0 (2.565) 489.68	0 (2.318) 541.88	0 (2.136) 588.10	**53** (1.954) 642.89	**243** (1.927) 651.90
	TFRC2	0 (3.180) 395.02	**24** (2.831) 443.74	**58** (2.595) 484.05	**103** (2.409) 521.52	**123** (2.209) 568.52	**477** (2.015) 623.30	**586** (1.838) -
	Avg. TCP	2.794	2.647	2.481	2.306	2.152	1.983	1.806

TABLE I
RESULTS FOR THE SIMPLE NETWORK SETUP SIMULATION. A CELL CONTAINS: THE NUMBER OF SECONDS WHEN THE PLAYOUT BUFFER WAS EMPTY, THE AVERAGE THROUGHPUT OF THE FLOW (IN MBPS; INSIDE BRACKETS) AT THE END OF THE STREAMING SESSION AND THE TIME WHEN THIS STREAMING SESSION ENDED. THE AVERAGE TCP THROUGHPUT (IN MBPS) IS ALSO SHOWN FOR EACH SIMULATION.

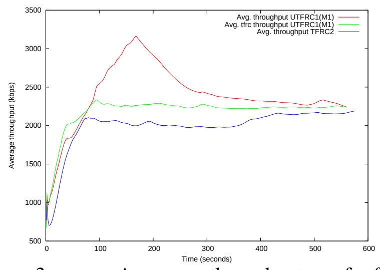


Fig. 3. Average throughputs of flows UTFRC1(M_1) (a 1st UTFRC variant flow) and TFRC2 (regular TFRC flow)

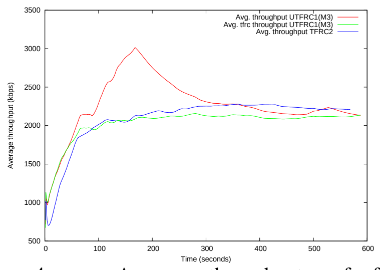


Fig. 4. Average throughputs of flows UTFRC1(M_3) (a 3rd UTFRC variant flow) and TFRC2 (regular TFRC flow)

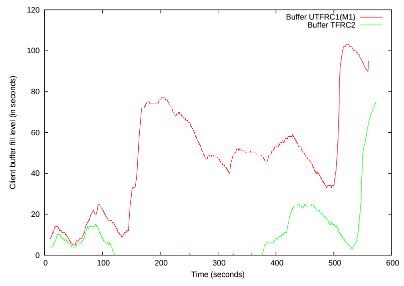


Fig. 5. Buffer fill levels of flows UTFRC1(M_1) (a 1st UTFRC variant flow) and TFRC2 (regular TFRC flow)

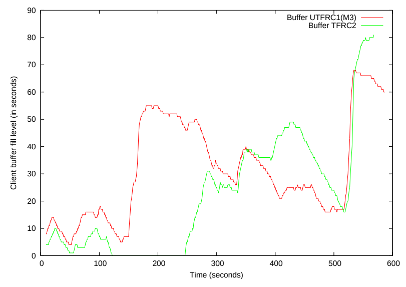


Fig. 6. Buffer fill levels of flows UTFRC1(M_3) (a 3rd UTFRC variant flow) and TFRC2 (regular TFRC flow)

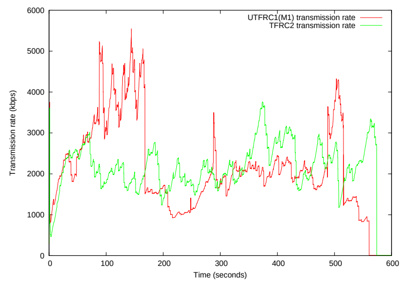


Fig. 7. Transmission rates of flows UTFRC1(M_1) (a 1st UTFRC variant flow) and TFRC2 (regular TFRC flow)

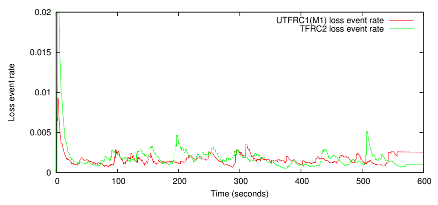


Fig. 8. Loss-event rates of flows UTFRC1(M_1) (a 1st UTFRC variant flow) and TFRC2 (regular TFRC flow)

of these 40 consecutive video seconds groups) so that the average bitrate of the new stream does not change, but the bitrate distribution changes significantly. The bitrates of these four artificial video streams are depicted in Figure 10, but for readability of this figure, the bitrates are averaged over groups of 40 consecutive seconds for each video stream. In the complex network setup, flows TFRC1 and TFRC5 were streaming the video stream #1, flows TFRC2 and TFRC6 were streaming the video stream #2, flows TFRC3 and TFRC7 were streaming the video stream #3 and, finally, flows TFRC4 and TFRC8 were streaming the video stream #4.

In addition, 6 other TCP flows start at random times between seconds 100 and 200, and seconds 300 and 400, respectively, and they all last random times between 1 and 100 seconds. These flows add load to the network and give

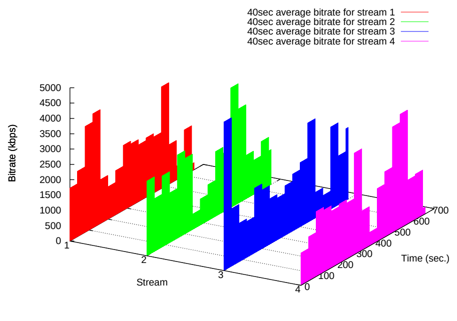


Fig. 10. 40 seconds average bitrates of the 4 artificial video streams used

a more fluctuating bandwidth. The access links of the flows' destination and source nodes are still having a one-way delay between 1 and 5 ms. The one-way delay of the bottleneck link is still 50 ms, the queue policy is RED with default parameters and the queue size is twice the bandwidth-delay product. The network capacity of the bottleneck link is this time: 64 Mbps, 60 Mbps, 56 Mbps, 52 Mbps, 48 Mbps, 44 Mbps. We start from a value larger than the one used in the simple setup (i.e., 64 Mbps), because this time the network is shared by more flows and we use fewer values in order to better summarize the results in the table below. For each of these 6 network bandwidths, we run 3 simulations:

- **No media-friendly** - a simulation in which all TFRC flows are regular TFRC flows;
- **1st media-friendly function** - a simulation in which TFRC1, TFRC2, TFRC3, TFRC4 are modified to be UTFRC flows using the 1st media-friendly function (and called UTFRC1(M_1), UTFRC2(M_1), UTFRC3(M_1), UTFRC4(M_1) in the following text) and the remaining TFRC flows are not media-friendly;
- **3rd media-friendly function** - a simulation in which TFRC1, TFRC2, TFRC3, TFRC4 are modified to be UTFRC flows using the 3rd media-friendly function (and called UTFRC1(M_3), UTFRC2(M_3), UTFRC3(M_3), UTFRC4(M_3) in the following text) and the remaining TFRC flows are not media-friendly.

The results are summarized in Table II. This table has approximately the same format as Table I, but in this case we have slightly different and fewer network capacities (in order to avoid an overly large Table II). We present the measurements for all 8 TFRC/UTFRC flows and we omit the streaming end time of each flow. So, for each flow the number of empty buffer seconds is shown and, in parentheses, the average throughput of the flow at the end of the streaming. We can see in this table that all flows experience increasing empty buffer counts as the network capacity drops gradually from 64 Mbps to 44 Mbps. But the UTFRC flows (flows UTFRC1, UTFRC2, UTFRC3 and UTFRC4) manage to avoid empty client buffers much better than the regular TFRC flows (flows TFRC5, TFRC6, TFRC7 and TFRC8). In this table we can see that both media-friendly functions, M_1 and M_3, managed to completely avoid empty buffers when the network capacity was 64 Mbps and in the 60 Mbps case, only 2 UTFRC flows had empty buffer levels for the 1st media-friendly function and 1 UTFRC flow had empty buffer levels for the 3rd media-friendly function, while regular TFRC flows had empty buffer counts larger than 100 (except flow TFRC8). In addition, when the network capacity was 56 Mbps or 52 Mbps, the empty buffer counts were significantly lower for all UTFRC flows than the empty buffer counts obtained for regular TFRC flows. For the 48 Mbps and 44 Mbps network capacities, most UTFRC flows still achieved empty buffer counts smaller than their corresponding regular TFRC flows.

Because the results obtained in the experiments performed in the complex network setup are summarized in Table II and because they are similar to the results obtained for the simple network setup in the sense that they show the superiority of the UTFRC framework (with the 1st and 3rd media-friendly function) over TFRC for video streaming, we will only plot two figures with typical results, one plotting average throughput and the other plotting client buffer fill levels for the 3rd media-friendly function UTFRC. In Figure 11 we can see the average throughput for all flows (4 TFRC and 4 UTFRC flows) are approximately equal. This figure is obtained from a simulation where the network capacity was 56 Mbps, the 3rd media-friendly function was used for the UTFRC flows and flows TFRC5, TFRC6, TFRC7, TFRC8 were regular TFRC flows. Flows UTFRC1(M_3) and TFRC5 were streaming video stream #1 from Figure 10, flows UTFRC2(M_3) and TFRC6 were streaming video stream #2, flows UTFRC3(M_3) and TFRC7 were streaming video stream #3 and, finally, flows UTFRC4(M_3) and TFRC8 were streaming video stream #4. Since the throughputs of all flows are approximately equal we can conclude that all UTFRC flows were TCP-friendly. The second figure for the complex network setup is Figure 12 which depicts the client buffer fill levels for the same experiment as the one used for Figure 11. We can see in this figure that all UTFRC flows using the 3rd media-friendly function (flows UTFRC1(M_3), UTFRC2(M_3), UTFRC3(M_3) and UTFRC4(M_3)) managed to sustain higher, more stable buffer fill levels than the other regular TFRC flows (flows TFRC5, TFRC6, TFRC7 and TFRC8) which led to fewer empty buffer counts for these former flows. Equivalent results were obtained for all network capacity configurations and also for the 1st media-friendly function, but we omit these plots due to space constraints.

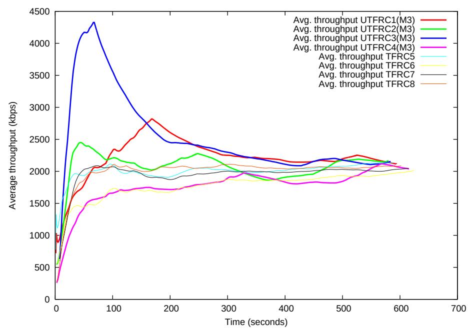


Fig. 11. Average throughputs for 8 flows in the complex network setup (the UTFRC flows use the 3rd media-friendly function and TFRC5, TFRC6, TFRC7, TFRC8 are regular TFRC flows)

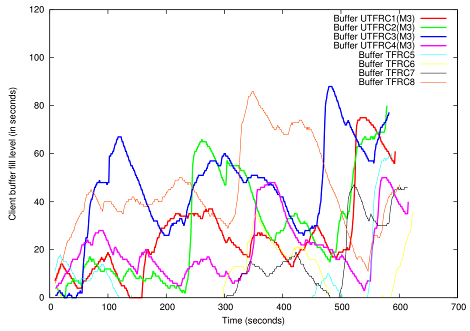


Fig. 12. Buffer fill levels for 8 flows in the complex network setup (the UTFRC flows use the 3rd media-friendly function and TFRC5, TFRC6, TFRC7, TFRC8 are regular TFRC flows)

VII. Conclusions

We have presented in this paper a congestion control framework more suitable for multimedia streaming applications

		Network capacity					
		64 Mbps	60 Mbps	56 Mbps	52 Mbps	48 Mbps	44 Mbps
No media-friendly	TFRC1	**80** (2.351)	**92** (2.282)	**148** (2.153)	**542** (1.924)	**608** (1.794)	**605** (1.670)
	TFRC2	0 (2.329)	**208** (2.209)	**114** (2.106)	**301** (2.018)	**599** (1.799)	**637** (1.660)
	TFRC3	**132** (2.438)	**230** (2.202)	**269** (2.092)	**571** (1.940)	**638** (1.840)	**647** (1.649)
	TFRC4	0 (2.374)	0 (2.293)	0 (2.066)	0 (2.045)	**220** (1.870)	**444** (1.670)
	TFRC5	**66** (2.465)	**149** (2.230)	**496** (2.009)	**510** (1.946)	**551** (1.828)	**582** (1.691)
	TFRC6	0 (2.291)	**53** (2.271)	**156** (2.089)	**287** (2.022)	**574** (1.796)	**635** (1.687)
	TFRC7	**137** (2.321)	**276** (2.181)	**359** (2.058)	**647** (1.871)	**648** (1.794)	**646** (1.612)
	TFRC8	0 (2.316)	0 (2.176)	**128** (1.982)	**130** (1.965)	**278** (1.765)	**465** (1.704)
1st Media-friendly function	UTFRC1(M_1)	0 (2.293)	0 (2.218)	0 (2.071)	**292** (1.979)	**572** (1.733)	**612** (1.641)
	UTFRC2(M_1)	0 (2.332)	**4** (2.282)	**6** (2.106)	**58** (2.037)	**463** (1.881)	**633** (1.724)
	UTFRC3(M_1)	0 (2.257)	**7** (2.269)	**30** (2.093)	**47** (2.017)	**344** (1.854)	**631** (1.720)
	UTFRC4(M_1)	0 (2.347)	0 (2.264)	**21** (2.137)	**271** (1.954)	**488** (1.883)	**580** (1.691)
	TFRC5	**88** (2.455)	**127** (2.223)	**428** (2.091)	**477** (1.981)	**543** (1.778)	**608** (1.606)
	TFRC6	0 (2.353)	**112** (2.176)	**325** (2.062)	**457** (1.903)	**622** (1.845)	**636** (1.659)
	TFRC7	**174** (2.294)	**267** (2.223)	**401** (2.025)	**630** (1.897)	**648** (1.731)	**645** (1.621)
	TFRC8	0 (2.368)	0 (2.241)	0 (2.087)	**172** (1.922)	**266** (1.827)	**515** (1.563)
3rd Media-friendly function	UTFRC1(M_3)	0 (2.370)	0 (2.134)	**20** (2.116)	**93** (1.944)	**442** (1.861)	**529** (1.812)
	UTFRC2(M_3)	0 (2.276)	**3** (2.271)	0 (2.166)	**27** (1.949)	**237** (1.887)	**502** (1.770)
	UTFRC3(M_3)	0 (2.288)	0 (2.304)	**5** (2.152)	**40** (1.963)	**78** (1.875)	**636** (1.761)
	UTFRC4(M_3)	0 (2.421)	0 (2.305)	0 (2.040)	**94** (1.957)	**158** (1.880)	**489** (1.805)
	TFRC5	**73** (2.369)	**191** (2.164)	**371** (2.133)	**550** (1.903)	**558** (1.843)	**609** (1.623)
	TFRC6	0 (2.387)	**122** (2.209)	**381** (2.012)	**388** (1.983)	**637** (1.775)	**635** (1.646)
	TFRC7	**252** (2.235)	**269** (2.115)	**306** (2.043)	**463** (1.964)	**648** (1.787)	**645** (1.674)
	TFRC8	0 (2.404)	0 (2.182)	0 (2.086)	**257** (1.858)	**292** (1.779)	**507** (1.665)

TABLE II
RESULTS FOR THE COMPLEX NETWORK SETUP SIMULATION. A CELL CONTAINS: THE NUMBER OF SECONDS WHEN THE PLAYOUT BUFFER WAS EMPTY AND THE AVERAGE THROUGHPUT (IN MBPS) OF THAT FLOW AT THE END OF ITS STREAMING SESSION (IN BRACKETS)

than classical TCP-friendly congestion control algorithms. Our congestion control framework termed UTFRC (*Utility-driven TCP-Friendly Rate Control*) relies on TFRC for achieving TCP-friendliness and tries to make it more media-friendly or application-friendly. Although our framework uses TFRC, it is important to emphasize that in theory other smooth TCP-friendly congestion control algorithms could be used by our framework instead of TFRC, but this has to be validated in practice. Besides the framework for building TCP-friendly and media-friendly algorithms for multimedia streaming we have also outlined in the paper four proof-of-concept congestion control algorithms developed with the UTFRC framework. From these four proof-of-concept congestion control algorithms, two were presented in detail and extensively tested through network simulations. From a theoretical point of view, we stated and proved in the paper three theorems and some additional propositions that give us guidelines on how to combine the media-friendly factor and the TCP-friendly factor in UTFRC so that the resulting congestion control protocol is a valid one and beneficial for multimedia streaming applications. A significant number of tests were performed for two of the UTFRC variants presented in order to prove they achieve TCP-friendliness and media-friendliness in the long term. These tests also showed that UTFRC is beneficial for video streaming applications. UTFRC is well suited for various video streaming services like video on demand, video broadcasting, live streaming, etc. As future work, we plan to develop other media-friendly functions for UTFRC using other media and application characteristics and to validate that other TCP-friendly congestion control algorithms, besides TFRC, can be used in UTFRC for achieving TCP-friendliness.

REFERENCES

[1] D. Damjanovic, M. Welzl, *An Extension of the TCP Steady-State Throughput Equation for Parallel Flows and Its Application in MulTFRC*, IEEE/ACM Trans. on Networking, vol. 19, no. 6, pp. 1676–1689, Dec. 2011.

[2] B. Wang, J. Kurose, P. Shenoy, D. Towsley, *Multimedia Streaming via TCP: An Analytic Performance Study*, ACM Trans. on Multimedia Computing, Communications and Applications, vol. 4, no. 2, pp. 16:1–16:22, May 2008.

[3] S. Floyd, M. Handley, J. Padhye, J. Widmer, *TCP Friendly Rate Control*, RFC 5348, September 2008.

[4] I. Rhee, L. Xu, *Limitations of Equation-based Congestion Control*, IEEE/ACM Trans. on Networking, vol. 15, no. 4, pp. 852–865, Aug. 2007.

[5] Z. Wang, S. Banerjee, S. Jamin, *Media-Friendliness of A Slowly-Responsive Congestion Control Protocol*, In Proc. of NOSSDAV 2004, pp. 82–87.

[6] F. Pereira, T. Ebrahimi, *The MPEG-4 Book*, Prentice Hall PTR, 2002.

[7] S. Floyd, M. Handley, J. Padhye, J. Widmer, *Equation-Based Congestion Control for Unicast Applications*, In Proc. of ACM SIGCOMM 2000, pp. 43–56.

[8] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, *Modeling TCP Throughput: A Simple Model and its Empirical Validation*, In Proc. of ACM SIGCOMM 1998, pp. 303–314.

[9] D. Bansal, H. Balakrishnan, S. Floyd, S. Shenker, *Dynamic Behavior of Slowly-responsive Congestion Control Algorithms*, In Proc. of ACM SIGCOMM 2001, pp. 263–274.

[10] Y. R. Yang, Min Sik Kim, S. S. Lam, *Transient Behaviors of TCP-friendly Congestion Control Protocols*, In Proc. of IEEE Infocom 2001, vol. 3, pp. 1716–1725.

[11] S. Floyd, M. Handley, J. Padhye, *A Comparison of Equation-Based and AIMD Congestion Control*, ACIRI, February 2000, http://www.aciri.org/tfrc/.

[12] D. Bansal, H. Balakrishnan, *Binomial Congestion Control Algorithms*, In Proc. of IEEE Infocom 2001, vol. 2, pp. 631–640.

[13] R. Rejaie, M. Handley, D. Estrin, *RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet*, In Proc. of IEEE Infocom 1999, vol. 3, pp. 1337–1345.

[14] D. Sisalem, H. Schulzrinne, *The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme*, In Proc. of NOSSDAV 1998.

[15] I. Rhee, V. Ozdemir, Y. Yi, *TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming*, April 2000. NCSU Technical Report.

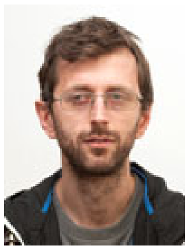
[16] J. Yan, K. Katrinis, M. May, B. Plattner, *Media- and TCP-Friendly Congestion Control for Scalable Video Streams*, IEEE Trans. on Multimedia, vol. 8, no. 2, pp. 196–206, April 2006.

[17] F. P. Kelly, A. K. Maulloo, D. K. H. Tan, *Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability*, J. Oper. Res. Soc., vol. 49, no. 3, pp. 237–252, Mar. 1998.

[18] S. H. Low, D. E. Lapsley, *Optimization Flow Control—I: Basic Algorithm and Convergence*, IEEE/ACM Trans. on Networking, vol. 7, no. 6, pp. 861–874, Dec. 1999.

[19] S. Floyd, K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, IEEE/ACM Trans. on Networking, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[20] M. Vojnovic, J.-Y. Le Boudec, *On the Long-Run Behavior of Equation-Based Rate Control*, IEEE/ACM Trans. on Networking, vol. 13, no. 3, pp. 568–581, 2005.
[21] The ns-2 Network Simulator, http://www.isi.edu/nsnam/ns/.
[22] A. Sterca, *UTFRC - Utility-driven TCP-Friendly Rate Control for Multimedia Streams*, In Proc. of Euromicro PDP 2009, pp. 167–172.



Adrian Sterca is a lecturer at the Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania. He received the Ph.D. degree from Babes-Bolyai University in 2009. He received 2 research grants from the Romanian funding agency and was a visiting researcher at the Institute of Information Technology (ITEC), Klagenfurt University, Austria in 2003, 2004 and 2006. He is a member of the ACM. His current research interests are networking, multimedia streaming and image processing.

Hermann Hellwagner is a full professor of Informatics in the Institute of Information Technology (ITEC), Klagenfurt University, Austria, leading the Multimedia Communications group. His current research areas are distributed multimedia systems, multimedia communications, and quality of service. He has received many research grants from national (Austria, Germany) and European funding agencies as well as from industry, is the editor of several books, and has published more than 200 scientific papers on parallel computer architecture, parallel programming, and multimedia communications and adaptation. He is a senior member of the IEEE, member of the ACM, GI (German Informatics Society) and OCG (Austrian Computer Society), and Vice President of the Austrian Science Fund (FWF).

Florian Boian is a full professor of Computer Science at the Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania. He has authored more than 100 scientific papers and published several books on Java-based frameworks, operating systems and distributed systems. He won several research grants from the Romanian funding agency. His current research interests are web services and distributed systems.

Alexandru Vancea is a lecturer at the Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania. He received the Ph.D. degree in Computer Science from Babes-Bolyai University in 2000. His current research interests are parallel programming and multimedia streaming. He has close collaborations with the industry on medical software.