

# Optimal Delivery of Multi-Media Content over Networks

Arthur D. Allen  
Burst.com Inc.  
613 4<sup>th</sup> Street  
Santa Clara CA 95404  
+01 707-541-3870

Arthur\_allen@hotmail.com

## ABSTRACT

In this paper, we describe scalable optimal methods for delivering archived and live multi-media content from servers to multi-media client players endowed with substantial RAM or disk-based buffers. These methods result from the application of Linear Optimization Theory (Linear Programming) to the problem of how best to modulate the flow rate of constant-bit-rate (CBR) content for all sessions linking a server to its clients, in which session flow rates are subject to upper and lower bound constraints, and aggregate flow cannot exceed a specified maximum. An efficient  $O(n)$  algorithm to maximize aggregate flow is described. We propose a tunable minimum constraint on session flows that is shown to result in a rapid and sustained accumulation of reserve content within a player's buffer. An associated Call Admission Control (CAC) algorithm is also described. The benefits of the methods described include improved server efficiency, enhanced end-user experience (QOS), cost effective end-to-end content delivery, directly from origin servers to clients without need of intervening edge-caching technology.

## General Terms

Algorithms, Performance, Design, Economics, Experimentation, Theory, Optimization.

## Keywords

Video-on-Demand, Live Events, Ad Insertion, QOS, Linear Programming, Optimization, CAC, CBR, VBR bandwidth smoothing, Edge-caching.

## 1. INTRODUCTION

### 1.1 Overview

Few today would challenge the proposition that streaming technology has a very bright future ahead. Yet predicting how soon this promise will be realized, and in what manner, is a venture fraught with uncertainty and risk. True, Internet radio is now in common use, with a growing community of users worldwide. By contrast, video streaming, which requires considerably more bandwidth and expense, is still by and large confined to the viewing of brief clips of music videos, movie trailers, interviews, press announcements, and the like.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Multimedia '01, September 2001, Ottawa, Ontario, Canada.

Copyright 2001 ACM 1-58113-000-0/00/0000...\$5.00.

A number of independent factors have contributed to this outcome: an immature broadband Internet infrastructure that results in frequent congestion storms; high backbone transmission costs; lack of available content; a multiplicity of competing standards for media encoding and delivery; last but not least, streaming server technology that is arguably not all that it could be. The net effect of all this has been that users are frequently disappointed by long delays, unwelcome pauses and minuscule screen images, causing many to turn away, there to await better days in the fullness of time.

While the planet waits, the software development community has been doing yeoman's work improving player and server technology. Much of this effort has been directed at overcoming the shortcomings of the delivery infrastructure *by software means*[5]. Thus, in an attempt to minimize pauses, increasingly efficient and resilient transport protocols now deliver *adaptive streams* at encoding rates that jump around over time, from high through medium to low density, and back again, according to variations in the network's capacity to deliver [2][3][8]. In a similar way, to minimize transmission costs, packet losses and network latencies, servers are now being deployed, at no inconsiderable cost, in complex caching hierarchies in which streaming content emanates from origin servers down to proxy servers located within POPs, at the edge of the Internet [4][7][6].

The engineers at Burst.com, who have been engaged in complementary efforts, have long challenged the conventional wisdom upon which present streaming systems are built. In particular, they have argued [11][12] that greater use should be made of available memory or disk space within players (typically PCs) to accommodate multi-media content delivered early, ahead of the real-time schedule. By so doing, they reasoned, one could offer a jitter or pause free viewing experience while making the best possible use of the server or network resources at hand.

The approach that we have advocated rests upon three fundamental insights

1) As any present-day viewer of streaming content well knows, a performance is typically delayed for several seconds while content accumulates in a local buffer, so as to decouple viewing from the vagaries of content delivery. The evident benefits of this simple practice with respect to short clips can be extended to long performances also, through *more extensive and systematic use of media buffers*.

2) *Streaming rate != encoding rate*. Present day streaming systems deliver CBR content at a streaming rate that equals the encoding rate. This need not be the case if one can store content delivered ahead of schedule in a large media buffer, as previously discussed.

3) A *system view* must be adopted for effective control over these added degrees of freedom if they are to deliver on their full potential, without causing mayhem instead.

Given these new possibilities, the problem we have posed ourselves is this: according to what rationale and algorithms should a server admit new clients, set content flow rates to its clients, and exploit available buffer capacity within client players?

### 1.2 Optimization Algorithms

One time-honored way of designing algorithms of the class required here is to re-cast the problem to be solved as an optimization problem, in which one seeks to maximize a designated *objective function* moment-by-moment, subject to a set of real-world *operational constraints*, which will typically vary over time. Accordingly, given a set of clients and associated sessions, an *optimal flow algorithm* continuously establishes content flow rates from the content server to each of its clients such as to maximize aggregate value according to the governing value function.

This approach holds several advantages: 1) optimization problems are well understood, and tractable by a large and diverse collection of computational methods; 2) if it exists, the global solution that is obtained is arguably optimal by construction, and thus superior or equal to all other.

More formally, in linear programming one seeks to optimize a linear objective function of state variable  $x$

$$c^*x = c_1*x_1 + \dots + c_n*x_n$$

subject to a set of linear inequality constraints

$$A*x \leq b$$

where  $x^T = (x_1, \dots, x_n)$ ,  $c = (c_1, \dots, c_n)$  are the state variable & cost vectors,  $A$  is an  $n \times m$  matrix,  $b^T = (b_1, \dots, b_m)$  is the constraint vector, and the operator  $*$  stands for matrix or scalar multiplication.

A number of well-known and generally effective computational solutions (e.g. the Simplex Method) exist for problems of this type. The Simplex method has worst case complexity of  $O(\exp(n))$ . Interior methods are known to converge on a solution in polynomial time. Neither would be acceptable in our case given the potentially high dimensionality of our problem. As we shall see, owing to the simple geometry of our specific problem, we have been able to obtain an algebraic method that yields a solution in  $O(n)$  time.

### 1.3 Optimized Server Architecture

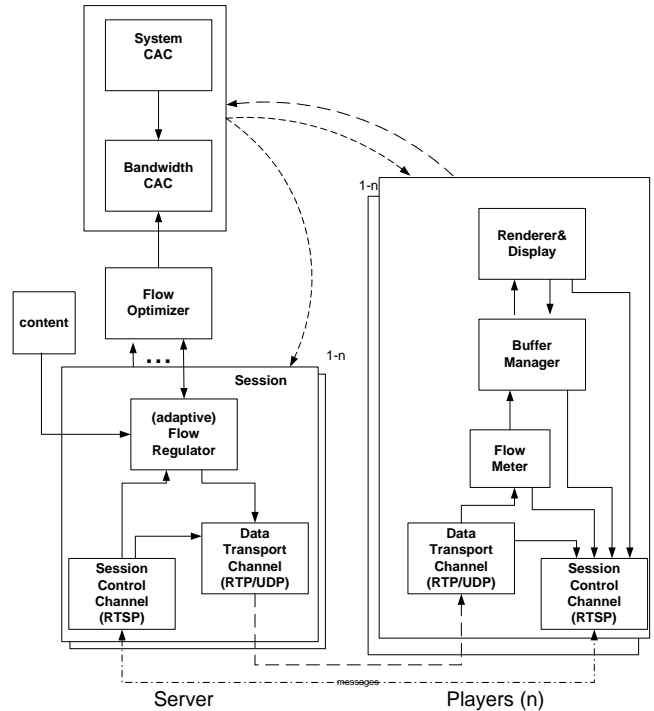
Figure 1 depicts how an optimization sub-system might be dropped within the engine of a modern streaming server. Also shown is a software block diagram for a typical player.

Every client has an associated session control stack within the server by means of which flow rate of content from server to client is effected and regulated to a given flow rate target. The Flow Optimizer manages bandwidth utilization *across all sessions* that have been admitted by the System Call Admission Control (CAC) and the Bandwidth CAC blocks. Specifically, the flow optimizer modulated the flow rate target of every active session so as to maximize aggregate flow or, more generally, cost.

A Flow Regulator [1] obtains content data from a cache or disk, at the current offset into the content file, which it forwards as packets to the Data Transfer Channel. The size and pacing of the packets are determined by the flow regulator in order to achieve

the flow rate target imposed by the Flow Optimizer. The flow regulator is also responsible for determining the channel flow capacity to the client, and delivered flow rate, in coordination with a flow meter subsystem in the client.

The Data Transport channel is responsible for transporting data packets received from the Flow Regulator to its peer entity within the client. The mode of communication between the peer entities is typically datagram oriented (e.g. UDP), enhanced with acknowledgements to facilitate rapid recovery from occasional packet loss, out-of sequence delivery, and duplication. TCP is also used.



**Figure 1: Optimized Server Architecture. The optimizer acts as a supervisory controller over all session control stacks**

Peered Session Control Channels permit configuration and status data, together with miscellaneous VCR-style commands, to be sent from client to server and vice-versa. The Buffer Manager accepts data from the Flow Meter, which data is retained until it is consumed by the Renderer prior to display on the screen.

The flow optimizer is continuously informed of channels flow capacities, buffer levels and capacities within clients, as well as relevant VCR commands initiated by the viewer.

## 2. Server Optimization for VOD

### 2.1 Problem Statement

#### 2.1.1 Definitions

**Table 1: Definition of terms**

Variable Name	Meaning
k	Session identifier
$\Delta t$	Time between recalculation of target session flows by optimizer

$t_k$	Time since the start of session k
$f_k(t)$	Target content flow rate for session k
$f_k^{CAP}(t)$	Physical flow capacity of channel serving session k at time t
$f_k^{MIN}(t), f_k^{MAX}(t)$	Minimum/Maximum effective constraint on session k flow
$f_k^{RES}(t)$	Reserve flow rate for session k
$f_{Svr}^{MAX}$	Server aggregate flow capacity
$l_k^{MAX}$	Media buffer capacity of player for session k
$l_k(t)$	Current media buffer level within player for session k. Note $l_k(t) \leq l_k^{max}$
$T_k$	Duration of play for content being delivered over session k
$L_k^{TOGO}(t)$	Content to go for session k, i.e., as yet undelivered by server. Initialized to $f_k * T_k$
$f_e$	Encoding (play) rate of content for session k
$C_k$	Cost of flow for session k (= 1 for max flow)
$\beta$	Burst tuning factor
$f_k^{JIT}$	Just-in-time flow rate for session k
IsPaused(k)	1 if session k consumption (e.g. playing) has been paused.

### 2.1.2 Objective Function

The objective function we seek to maximize is a linear function of session flow rates, as follows

$$(1) \quad \text{Cost}(t) = f_1(t) * C_1 + \dots + f_n(t) * C_n$$

Where  $C_k$  is the unit cost of bandwidth for session k.

### 2.1.3 Maximum Flow Constraints

The maximum constraints on aggregate session flow and also individual session flows are these:

$$(2) \quad f_1(t) + \dots + f_n(t) \leq f_{Svr}^{MAX}$$

i.e., the sum of all target session flows shall not exceed the server's flow capacity;

$$(3) \quad f_k(t) \leq f_k^{CAP}(t) \text{ for every } k$$

i.e., the target session flow shall not exceed the channel capacity, which may vary over time according to changing network conditions;

$$(4) \quad f_k(t) \leq (l_k^{MAX} - l_k(t)) / \Delta t + (1 - \text{IsPaused}(k)) * f_e$$

i.e., the target flow should not cause a media buffer overflow before the next recalculation. In particular, whenever the buffer is full, session flow cannot exceed the consumption rate, which might be zero if the session has been paused.

$$(5) \quad f_k(t) \leq L_k^{TOGO}(t) / \Delta t$$

i.e. the target flow cannot deliver beyond the end of the content.

### 2.1.4 Reserve Flow Constraint

In our search for a suitable lower bound on session flow we have been guided by the following considerations

- A lower bound should define, over time, a *feasible* content flow trajectory that delivers all the CBR content in the time available under ideal network conditions. Clearly a lower bound of zero does not meet this requirement.
- The lower bound should be a monotonic non-increasing function of time, in order to ensure that a session's worst-

case needs never exceed those known to the CAC algorithm at the outset.

We term a minimum flow constraint that employs a lower bound that meets these requirements a *reserve flow constraint*, and the minimum value it allows,  $f_k^{RES}(t)$ , is termed the *reserve flow rate*.

#### 2.1.4.1 Just-In-Time Flow

The just-in-time flow rate at a given time t is such that if it were imposed for the balance of the session (i.e., over the time interval [t, T]), the last bit of content would be delivered just in time, at the last possible moment.

A formula for the just-in-time flow rate at any time t between [0, T] is given by

$$(6) \quad f_k^{JIT}(t) = L_k^{TOGO}(t) / (T_k - t_k)$$

that is to say, the constant flow rate that delivers what content remains in the server over remaining play time. At t=0 we have

$$f_k^{JIT}(0) = f_e * T_k / T_k = f_e$$

i.e., the just-in-time flow rate equals the encoding rate at the start of a session. Equivalently, we can express the just-in-time flow rate in terms of the session buffer level, as follows:

$$(7) \quad f_k^{JIT}(t) = f_e - l_k / (T_k - t_k)$$

i.e. the jit flow rate equals the encoding rate reduced by the flow needed to drain the media buffer in the time remaining<sup>1</sup>.

**Proposition:**

$$(8) \quad f_k(t) \geq f_k^{JIT}(t)$$

is a reserve flow constraint.

The first requirement is true by construction of  $f_k^{JIT}$ . To see that the JIT flow rate is monotonic non-increasing, we need only observe that if (8) is satisfied, then

$$\begin{aligned} f_k^{JIT}(t+\Delta t) &= L_k^{TOGO}(t+\Delta t) / (T_k - t_k - \Delta t) \\ &= [L_k^{TOGO}(t) - f_k(t) * \Delta t] / (T_k - t_k - \Delta t) \end{aligned}$$

Multiplying each side by  $(T_k - t_k - \Delta t) / (T_k - t_k)$  & applying (6) to the RHS, we obtain

$$f_k^{JIT}(t+\Delta t) * (1 - \Delta t / (T_k - t_k)) = f_k^{JIT}(t) - f_k(t) * \Delta t / (T_k - t_k)$$

After substitution of  $f_k^{JIT}(t)$  for  $f_k(t)$ , by (8) we obtain inequality

$$f_k^{JIT}(t+\Delta t) * (1 - \Delta t / (T_k - t_k)) \leq f_k^{JIT}(t) - f_k^{JIT}(t) * \Delta t / (T_k - t_k)$$

Finally,

$$f_k^{JIT}(t+\Delta t) \leq f_k^{JIT}(t)$$

Regardless, should the actual<sup>2</sup> flow rate fall below the JIT flow rate, the JIT flow rate will rise, as it depends by (7) on the actual buffer level, which will sag.

**Proposition:** the JIT reserve flow constraint given by (8) is the lowest possible reserve constraint.

By definition of the JIT flow rate, by (6) a lesser monotonic non-increasing flow rate cannot deliver all the required content in the time available.

<sup>1</sup> (7) can be derived from (6) by simple algebraic manipulation.

<sup>2</sup> As distinct from the *target* flow rate.

The minimum flow constraint obtained by combining (8) with expression (7) may be thought of as the *terminal buffer underflow constraint*. The minimum it imposes always exceeds the minimum imposed by the traditional buffer underflow constraint,

$$f_k(t) \geq f_{e_k} - l_k(t) / \Delta t$$

#### 2.1.4.2 Tunable Reserve Constraint

We have employed a parameterized family of reserve flow constraints defined as follows:

$$(9) \quad f_k(t) \geq f_k^{JIT}(t) * (1+\beta) = f_k^{RES}(t)$$

where  $\beta \geq 0$ .

This family converges on (8) when  $\beta$  approaches 0. A  $\beta$  value of .1, for example, ensures a minimum flow rate that exceeds the JIT flow rate by 10%. Thus, non-zero values of  $\beta$  force early delivery of content to a player's media buffer, with beneficial effects on the viewing experience, as will be demonstrated in a subsequent section.

#### 2.1.4.3 Effective Min/Max Constraints

In practice, we must ensure that the lower constraints never exceed the upper constraints. For example, constraint (4) may impose a maximum on flow that falls below the reserve flow rate implied by (9) if a media buffer is small and full. The bandwidth capacity of the channel will also vary over time. A serious problem may arise should it ever fall below the reserve rate. Such an occurrence may be indicative of one of two possibilities: 1) a short-term congestion condition that will lift in time, or 2) a fundamental mismatch between the selected encoding rate & channel capacity. The second scenario will announce itself early in a session, and must be dealt by selection a lower encoding rate, or by pausing the session, and allowing the media buffer to fill till the reserve rate drops again below channel capacity, when the session is resumed<sup>3</sup>.

The *effective minimum constraint* on session flow at any time  $t$  is given by

$$(10) \quad f_k(t) \geq f_k^{MIN} = \min \{ f_k^{RES}, f_k^{CAP} \\ (l_k^{MAX} - l_k(t)) / \Delta t + (1-IsPaused(k))*f_e \}$$

With respect to maximum constraints, only one of constraints (3), (4) and (5) will be effective at any one time. The *effective maximum constraint* at any time  $t$  is the one imposed by the least most of (3) (4) and (5), as follows:

$$f_k(t) \leq f_k^{MAX} = \text{Min} \{ f_k^{CAP}(t), \\ (l_k^{MAX} - l_k(t)) / \Delta t + (1-IsPaused(k))*f_{e_k}, \\ L_k^{TOGO}(t) / \Delta t \}$$

## 2.2 Call Admission Control

The traditional CAC algorithm compares the flow rate requirement of a prospective client with the *unused bandwidth* (obtained by subtracting the aggregate flow target from the server flow capacity) and grants service if unused bandwidth is not exceeded. This procedure breaks down in our case, as the optimizer causes all available bandwidth to be used if it can,

thereby reducing unused bandwidth to zero in the best case, even when a new client could be accommodated with a healthy margin to spare.

Nevertheless, a simple variation on this idea can suit our needs. We define *available bandwidth* as the difference between server flow capacity and the sum of reserve flow rates for each session computed according to (9), as follows:

$$(11) \quad f_{available} = f_{Svr}^{MAX} - (f_1^{RES} + \dots + f_n^{RES})$$

Service is granted if the computed reserve flow rate of the prospective client is less than available bandwidth, even when unused bandwidth is zero!

Given the monotonic decreasing nature of session reserve flow rates,  $f_{available}$  can readily be seen to be a *monotonic non-decreasing* function of time over the intervals separating client admissions, at which points it undergoes a drop as a new load is taken on. This all-important characteristic implies the following

**Proposition:** Provided all flows exceed their reserve levels, any client admitted for service based on the present value of available bandwidth is guaranteed to have sufficient bandwidth at its disposal over the entire future course of the session.

## 2.3 VCR controls

A typical VCR supports controls such as Pause, Fast-Forward and Rewind. As a consequence, viewers have come to expect similar functionality with media players. Fast Forward is potentially problematic when it moves beyond the last of the pre-buffered content. Even if, as in [2] we use frame skipping or thinning to hold the flow rate steady during the rapid advance, we must confront the problem of the approaching deadline, which causes  $f_k^{JIT}$  to rise according to (5), and with it  $f_k^{RES}(t)$  according to (9), in violation of our monotonicity requirement on  $f_k^{RES}(t)$ . One solution that will typically succeed without delay is to force prioritized re-admission of the session at the new content offset, with a new deadline, and revised buffer level as applicable.

Pausing, which may cause a notch in  $f_k^{MIN}$  when the media buffer fills up, can be dealt with in the same manner when the session resumes.

## 2.4 An Algorithm to Maximize Flow

The objective function to maximize server flow is

$$(12) \quad \text{Cost}(t) = f_1(t) + \dots + f_n(t)$$

The cost vector in this instance is the  $n$ -dimensional unit vector, where  $n$  is the number of active sessions.

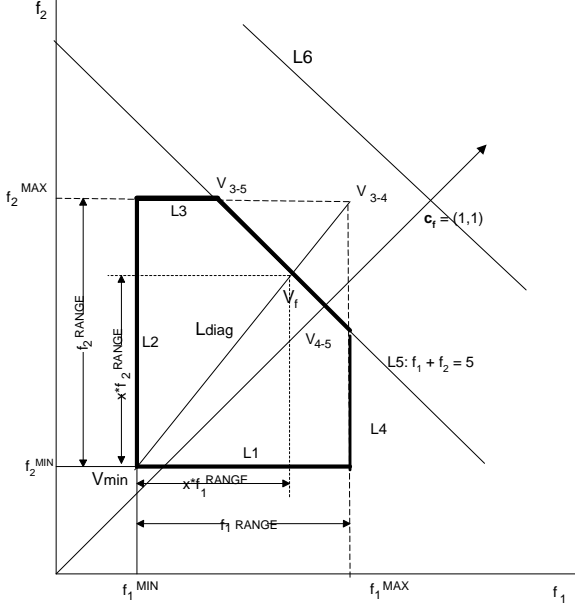
$$c = (1, 1, \dots, 1)$$

A representation of our optimization problem for 2 sessions is depicted in figure 2, below. We will use this special case to reason by analogy for the  $n$ -dimensional case.

1. The effective minimum and maximum constraints on individual session flows (e.g.  $f_1$  and  $f_2$ ) collectively define a hyper-rectangle. In our figure, the rectangle of interest is bounded by the rectangle formed by points  $(f_1^{MIN}, f_2^{MIN})$ ,  $(f_1^{MAX}, f_2^{MIN})$ ,  $(f_1^{MAX}, f_2^{MAX})$ ,  $(f_1^{MIN}, f_2^{MAX})$ .
2. It is well known from Linear Programming that points (e.g.  $(f_1, f_2)$ ) whose coordinate sum (e.g.  $f_1 + f_2$ ) cannot exceed a given maximum lie to the origin side of an associated hyper-plane orthogonal to the unit vector. For  $f_{Svr}^{MAX} = 5$ , the

<sup>3</sup> Such a practice is possible only if the media buffer is sufficiently large.

hyper-plane (i.e. line) is L5 in figure 2, whereas for  $f_{Svr}^{MAX} = 8$ , this hyper-plane is L6.



**Figure 2: Problem geometry for max flow in 2 dimensions**

3. The points that satisfy all constraints lie in the intersection of the hyper-rectangle and the half-space cleft by the maximum hyper-plane. As depicted, L5 clips the hyper-rectangle whereas L6 lies well above.
4. It is also well known from Linear Programming that equal-cost points for a given cost vector lie on a hyper-plane orthogonal to the vector itself. In our case the cost vector is *also* the unit vector. It follows that all points along the maximum hyper-plane have the same value according to the objective function (12). Further,
  - Whenever the maximum hyper-plane does clip our hyper-rectangle (as depicted for L5) any point on the hyper-plane contained at the intersection also maximizes the objective function.
  - If, as for L6, there is no intersection between our rectangle and the maximum hyper-plane, only one point in the rectangle maximizes the objective function, namely  $V_{3-4} = (f_1^{MAX}, f_2^{MAX})$  at which  $f_1$  and  $f_2$  both assume their respective maximum.
5. In the former instance, which alone presents a challenge, we can find a point that maximizes the objective function at the intersection of the diagonal line segment  $L_{diag}$  of the rectangle bounded by lines L1 through L4, starting at vertex  $V_{min}$  and ending at point  $V_{3-4}$ . This intersection between  $L_{diag}$  and L5, indicated by  $V_f$ , is optimal on the same basis as any other point lying along the intersection of L5 with the rectangle bounded by L1 through L4, as previously discussed.

**Proposition:** For every flow  $f_i$ , the optimal flow rate is obtained by interpolation between minimum and maximum flows

$$(13) \quad f_i^{OPT} = f_i^{MIN} + \alpha * (f_i^{MAX} - f_i^{MIN})$$

where

$$\alpha = \min \{1, (f_{Svr}^{MAX} - (f_1^{MIN} + \dots + f_n^{MIN})) / ((f_1^{MAX} - f_2^{MIN}) + \dots + (f_n^{MAX} - f_n^{MIN}))\}$$

Based on the foregoing discussion, this result can be obtained by elementary vector geometry and algebraic manipulation, as follows.

Referring to our figure, we seek scale factor  $\alpha$  such that the vector sum of  $(f_1^{MIN} \dots f_n^{MIN}) + \alpha (f_1^{RANGE} \dots f_n^{RANGE})$  intersects the maximum hyper-plane for capacity  $f_{Svr}^{MAX}$ . The point of intersection is obtained by solving equation

$$(f_1^{MIN} + \dots + f_n^{MIN}) + \alpha (f_1^{RANGE} + \dots + f_n^{RANGE}) = f_{Svr}^{MAX}$$

Solving for  $a$ , we obtain

$$\alpha = [f_{Svr}^{MAX} - (f_1^{MIN} + \dots + f_n^{MIN})] / (f_1^{RANGE} + \dots + f_n^{RANGE})$$

For any given session, the optimal flow rate is given by

$$f_i = f_i^{MIN} + \alpha f_i^{RANGE}, \text{ if } \alpha < 1,$$

or

$$f_i = f_i^{MIN} + f_i^{RANGE}, \text{ if } \alpha > 1,$$

where by definition  $f_i^{RANGE} = f_i^{MAX} - f_i^{MIN}$

This solution is of great interest by virtue of its simplicity and of the efficiency of the calculation, which is of complexity  $O(n)^4$ , and therefore scales very well as the number of sessions grows large.

## 2.5 An Algorithm to Maximize Charges

The objective function for this case is given by (1). An efficient  $O(n)$  solution is obtained by applying the previous method to a series of equal-cost sub-problems for every cost category  $C_k$ , each with associated session flows  $f_{kj}$ , as follows:

```

// We assign worst-case flow targets to each session
f_k^OPT = f_k^MIN for every k= 1 .. n
// We compute the swing bandwidth (BW)
f_swing = f_Svr^MAX - (f_1^MIN + ... + f_n^MIN)
for each C_k from most to least costly do
  // calculate scale factor for sub-problem
  alpha = f_swing / ((f_k1^MAX - f_k1^MIN) + ... + (f_kn^MAX - f_kn^MIN))
  if (alpha < 1)
    // this category exhausts remaining BW
    f_kj = f_kj^MIN + alpha * (f_kj^MAX - f_kj^MIN)
    for all j = k1..kn
      done;
  else
    // this category driven to maximum
    f_kj = f_kj^MAX for all j = k1..kn
  // reduce swing BW by amt allocated
  f_swing = f_swing - ((f_k1^MAX - f_k1^MIN) + ... + (f_kn^MAX - f_kn^MIN))
endif
endfor

```

<sup>4</sup> The evaluation of  $\alpha$  is  $O(n)$ , and need be performed only once per update of vector  $f^{OPT}$ . Given  $\alpha$  the latter update is also  $O(n)$ . That the entire calculation is  $O(n)$  follows from the fact that  $O(n) + O(n) = O(n)$ .

This algorithm allocates swing<sup>5</sup> bandwidth to higher paying customers before any is given to lower paying customers. The former will be pushed to the limit of their estimated channel capacity (which estimate is prone to error), while others are made to operate well below capacity. This significant drawback is overcome in a subsequent section by another approach to this problem.

### 3. Buffer Level Dynamics

#### 3.1 Analysis

In this section we develop a number of useful lower bounds for the media buffer level within players.

**Proposition:** For each  $k$ , and  $t$  in  $[0 T_k]$

$$(14) \quad l_k(t) \geq f_{e_k} * T_k * (1-t_k/T_k) [1 - (1-t_k/T_k)^\beta]$$

Proof: the rate of change of buffer level equals the net flow of content into it, that is to say the difference between the flow rate from the server and the consumption rate:

$$d/dt(l_k(t)) = f_k(t) - f_{e_k}$$

Let us define the reserve buffer level  $l_k^{RES}(t)$  to be the media buffer level that would result from an inflow at the reserve flow rate. From (9) we know that

$$d/dt(l_k(t)) \geq f_k^{RES}(t) - f_{e_k} = d/dt(l_k^{RES}(t))$$

Applying (9) again we obtain

$$d/dt(l_k(t)) \geq d/dt(l_k^{RES}(t)) = \beta * f_{e_k} - (1+\beta) * l_k(t)/(T_k - t_k)$$

It follows that  $l_k(t)$  must be bounded from below by the solution to differential equation

$$d/dt(l_k^{RES}(t)) + (1+\beta) * l_k^{RES}(t)/(T-t) - \beta * f_e = 0$$

This is a time varying ordinary differential equation that can be solved by applying method 16.316 in [13], followed by laborious integrations and algebraic manipulations. The solution is

$$(15) \quad l_k^{RES}(t) = f_e * T * (1-t/T) [1 - (1-t/T)^\beta]$$

**Proposition:** the following facts are true of  $l_k^{RES}(t)$  and  $l_k(t)$  over  $[0 T]$ :

$$(16) \quad \text{Avg}(l_k(t)) \geq \text{Avg}(l_k^{RES}(t)) = f_e * [1/2 - 1/(2+\beta)]$$

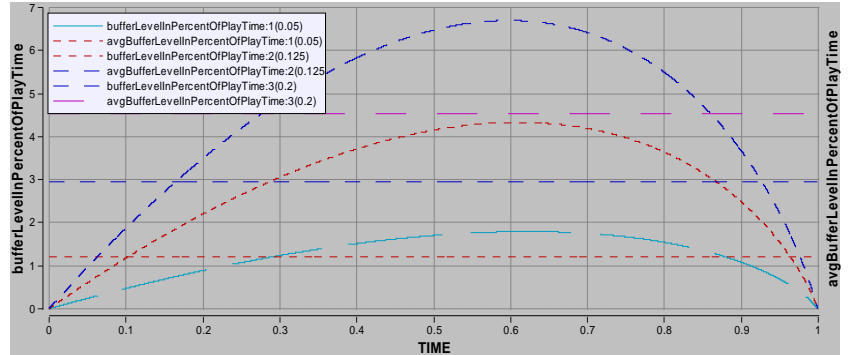
$$(17) \quad t^{MAX} = T * [1 - (1+\beta)^{-1/\beta}]$$

$$(18) \quad l_k^{RES}(t^{MAX}) = f_e * T * (1+\beta)^{-1/\beta} * [1 - 1/(1+\beta)]$$

(16) is obtained by integrating (15) over interval  $[0, T]$  and dividing the result by  $T$ . Equation (16) is obtained by solving for the root of the derivative of (15). Plugging the time value into (15) we obtain the peak value of  $l_k^{RES}(t)$ . Figure 3 depicts a graph of reserve media buffer level over time ( $t/T$ ) alongside the average level for three separate values of  $\beta$ : .05, .125 and .2. Buffer levels are expressed in the time required to drain the buffer at the play rate, assuming all inflow has ceased. As shown, the reserve level rises quickly and exhibits a robust arch, reminiscent of a ballistic trajectory, with the average sitting well above the

median. The greater the value of  $\beta$ , the more pronounced the arch, and the reserve levels of content achieved. During the early phases of a session content accumulates in a media buffer at a minimum in rough proportion to  $\beta$ . Thus, for content lasting sixty minutes, and  $\beta$  of .125, a reserve buffer level of at least one minute is achieved in slightly more than 8 minutes (i.e.  $1/.125$ ) from the start of play. The level peaks at  $.6$  of  $T$ , after which it declines sharply as inflow falls below the consumption rate.

Figure 4 depicts the reserve flow rates that gave rise to the buffer levels graphs presented in figure 3. Sessions are launched with an initial flow rate equal to  $(1+\beta)*f_e$ . The flow rate declines over time, crossing below the encoding rate line at the point where the



**Figure 3. Graphs of reserve buffer level and average buffer level over normalized time ( $t/T$ ) for three values of  $\beta$  (0.5, .125, .20). Levels are expressed in percent of play time  $T$ , representing the time needed to drain the buffer at the play rate  $f_e$  should all further inflow cease.**

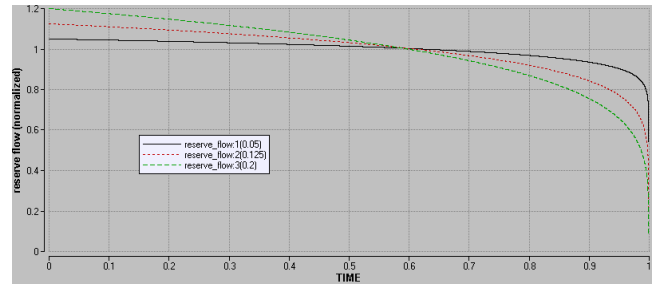
reserve level curves begins its decline.

#### 3.2 Observations

It is generally agreed that the quality of a multi-media experience (QOS) from a technical standpoint hinges on at least three factors:

1. Avoidance of long start-up delays required to pre-fill a player's media buffer
2. Absence of disruption or quality degradation during the session.
3. Richness of the content (i.e., it's encoding bit-rate).

We submit and will now endeavor to justify that the methods presented thus far enhance all of these factors.



**Figure 4. Graph of reserve flow rate over time (normalized  $t/T$ ) for three values of  $\beta$  (i.e., .05, .125, .2). The reserve flow rate has been normalized to the encoding/play rate. Note that the initial flow rate at  $t=0$  is  $(1+\beta)*f_e$**

<sup>5</sup> Swing bandwidth typically equals available bandwidth, but may exceed it.

A start-up delay, lasting several seconds, is typically required to permit a player to buffer sufficient content to avert content shortages resulting from unavoidable fluctuations in the delivery rate of content. Longer clips typically require longer wait times, in proportion to the increase in delivery rate and level dispersion one can expect over the protracted period of play. In practice viewers are unwilling to tolerate lengthy start-up delays, thereby causing undesirable pauses to occur whenever longer clips are viewed.

In our case, the level of buffered content rises quickest at the start of a session, and continues to rise till well past the half way time mark, achieving levels that effectively isolate viewers from virtually all network disturbances and congestion events over the entire duration of the viewing period. Consequently a very modest start-up delay will typically suffice for long and short clips alike.

The encoding rate of content that can be viewed over a given channel (e.g., DSL, cable, dial-up) is typically well below the channel capacity, as the latter is typically subject to fluctuations that would compromise timely content delivery at the real-time rate, were the encoding rate and the capacity too closely matched. In our case, any excess bandwidth (which need only exceed  $\beta \cdot f_e$  on average) is relentlessly exploited to isolate the viewer from the network, and to enhance server efficiency through early delivery of content whenever possible.

As depicted in figure 3, the levels of reserve content rise sharply as  $\beta$  is increased. Nevertheless, a point of diminishing return is reached (typically around .2) beyond which increases in  $\beta$  impose limitations on the encoding rate of viewable content, as well as the number of concurrent sessions a server can take on, while failing to further increase isolation of a viewer from network disturbances.

### 3.3 Toward a Tunable QOS

In the preceding discussion we have assumed that all clients share the same value of  $\beta$ . One might also consider a scheme whereby a client receives a  $\beta$  value according to the class/cost of service that client has signed up for: the greater the cost (and expected quality of service, or QOS) the greater the value of  $\beta$ .

Instead of maximizing charges by the method described (), we propose a variation on the algorithm to maximize flow, whereby the reserve flow computation for every session takes into account the individual  $\beta$  value of the associated client. In this fashion a single algorithm can serve either end. According to the algorithm to maximize charge in section 2.5, the optimizer first ensures that every session receives its required minimum bandwidth (which would now typically depend on  $\beta$  and elapsed session time) before allocating any excess bandwidth more or less evenly to all by interpolation.

This scheme is simpler computationally than our first approach to the problem of maximizing charge; it is also arguably more fair and stable with respect to the allocation of available bandwidth.

## 4. Extensions

### 4.1 Deadline-Driven Content Distribution

In delivering viewable or audible content to a player, the optimization algorithm imposes a reserve flow rate that ensures that all content is delivered by the implied deadline, which is the last moment of a play session. Notwithstanding, the optimizer

attempts to deliver content earlier, according to bandwidth availability within the server taken as a whole. A similar, though far simpler scenario arises relative to the distribution of content from an optimized server to a target device, whenever the delivery must be accomplished by a designated time, or equivalently, within a given elapsed time, and when overlapped non-blocking consumption of the content at a fixed rate is not required.

In adapting the optimizer to the needs of content distribution we must revisit the formulation of the flow constraints, as follows:

1. We are not concerned with media buffers, let alone a buffer overflow, and can assume that all content received is saved away in a file on disk, awaiting further use. Consequently, inequality 4 is not <sup>6</sup>relevant.
2. For these same reasons, we may safely content ourselves with a  $\beta$  value of zero, at which the reserve flow rate equals the just-in-time rate obtained according to (6) by dividing the content as yet undelivered by the time till deadline:

$$f_k^{JIT} = L_k^{TOGO}(t) / (T_k - t_k)$$

Given the arbitrary format of the file to be distributed,  $T$  no longer signifies the “duration of play” at encoding rate  $f_e$ ; rather,  $T$  represents the elapsed time to the deadline measured at the moment the delivery session was launched, and  $t$  represents the elapsed time since session launch.

### 4.2 Live Events

The methods presented in this report, *together with their benefits*, have been extended to the optimization of “live” and also time-shifted “near-live” events.

### 4.3 Session Bundles

A session bundle is a set of sessions that flow along a shared channel, and are thus subject to an aggregate flow constraint equal to the capacity of the shared channel<sup>7</sup>. One important application of session bundles is connected with ad insertion. In that application a session bundle comprises a session associated with the featured program together with sessions for every advertisement scheduled to interrupt the feature program at a designated time. Each session is endowed with a separate media buffer, which, in the case of ads, would typically be large enough to accommodate the ad in full<sup>8</sup>, thereby permitting a seamless transition to the ad at the appointed time.

It is not enough for us to treat each of the constituent sessions as independent with respect to the flow optimizer, for this practice might well result in session flow targets that do not exceed

<sup>6</sup> It can be effectively ignored by setting buffer capacity to the size of the content subject to distribution

<sup>7</sup> Regardless of whether the capacity has been estimated or imposed.

<sup>8</sup> Whenever content is being delivered in anticipation of future play, as is the case of an ad, the reserve flow rate may be set to the just-in-time rate calculated using the time to the deadline rather than the ad duration, as previously discussed in connection with content distribution.

channel capacity when considered singly, yet exceed this same shared capacity when summed together.

Our proposed solution to this problem involves two distinct steps:

1. The session bundle is treated as a *virtual session* by the server optimizer, subject to a minimum flow constraint obtained by summation of the individual session minima, and a maximum constraint that is the least of a) the sum of the least of constraints (4) and (5) for each session, and b) the shared channel capacity. The flow optimizer then generates an aggregate flow target for the virtual session by interpolation, using (12).
2. We now possess an achievable aggregate flow target for the session bundle that we must apportion optimally among the constituent sessions. Fortunately for us, our interpolation procedure (12) can be applied again, to the sessions in our bundle, but where the aggregate flow target generated in step 1 replaces the aggregate flow constraint  $f_{Svr}^{MAX}$ . Thus we have shown the following...

**Proposition:** For every flow  $f_{ki}$ , within session bundle  $k$  the optimal flow rate is obtained by interpolation between minimum and maximum session flows

$$(20) \quad f_{ki}^{opt} = f_{ki}^{MIN} + \alpha * (f_{ki}^{MAX} - f_{ki}^{MIN})$$

where

$$f_{ki}^{max} = \text{Min} \{ L_k^{TOGO} / \Delta t,$$

$$(l_{ki}^{max} - l_{ki}(t)) / \Delta t + (1 - \text{IsPaused}(k)) * f_{ki}\}$$

$$\alpha = [f_k^{bundle} - (f_{k1}^{MIN} + \dots + f_{kn}^{MIN})] / ((f_{k1}^{MAX} - f_{k2}^{MIN}) + \dots + (f_{kn}^{MAX} - f_{kn}^{MIN}))$$

and the value of  $f_k^{bundle}$  is obtained from a higher level optimization in which session bundle  $k$  is treated as a virtual session for which

$$f_k^{MIN} = f_{k1}^{MIN} + \dots + f_{kn}^{MIN}$$

$$f_k^{MAX} = \text{min}(f_{k1}^{MAX} + \dots + f_{kn}^{MAX}, f_k^{CAP})$$

It must be noted that session bundles are typically dynamic in nature, outliving many if not all of their constituent sessions. Before a session is added to the bundle two CAC decisions must be reached, the first to establish bandwidth availability within the bundle, and the second to bandwidth availability within the server.

## 5. Experimental Results

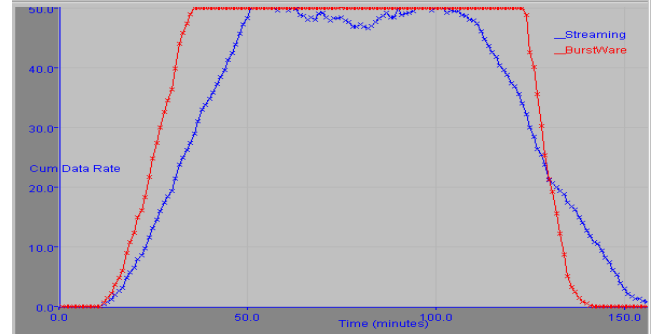
The methods described in this report have been subject to extensive experimentation within two separate settings: the Burstware VOD server simulator and the Burstware product.

### 5.1 BustSim

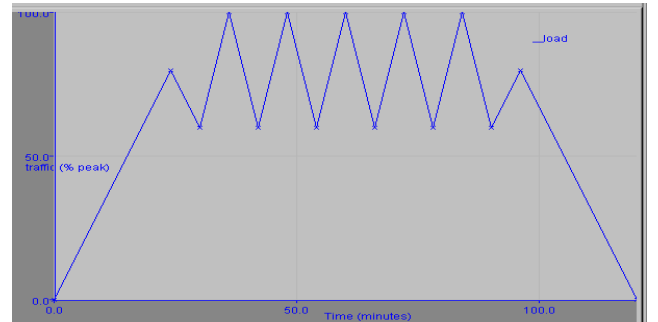
This java application was built on top of the JavaSim discrete event process-based simulation package from the University of Edinburgh. BustSim facilitates comparisons between optimized and traditional real-time streaming approaches to the delivery of VOD and live multi-media payloads over a network, across a range of server, network, and player configurations, load profiles and media content. Leaving real-time streaming aside, the application can also be employed as an aid to the deployment of optimized servers on networks: a proposed configuration is subjected to range of anticipated stimuli, the response is observed, the configuration adjusted accordingly, and so forth repeatedly until a satisfactory result is obtained.

Figures 5 below, illustrates how optimized content delivery differs from traditional streaming. The simulation run in question involved players equipped with DSL connections to the network (375 kbps in this instance) and endowed with 25-megabyte media buffers; traffic was generated according to the profile shown in figure 6, the content was 40 minutes in duration and encoded at 200kbps. The value of  $\beta$  was set to .125.

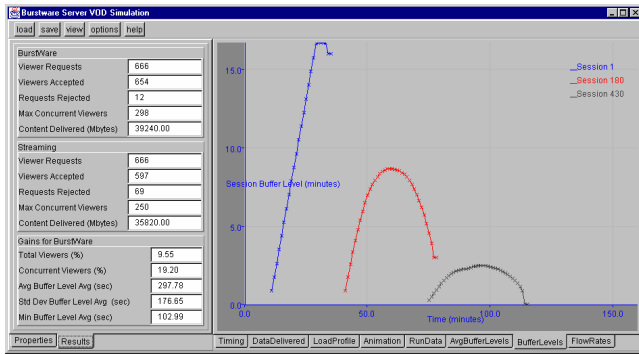
Optimized servers were able to service 9.5% more clients during the run, all the while maintaining an average buffer level across all players of nearly five minutes, indicative of a very high viewer QOS.



**Figure 5:** graph of aggregate flow for streaming versus optimized Burstware server in response to fluctuations in the arrival rate of clients depicted in the next figure. Server output leads the load and does not slacken during lulls in traffic, thereby increasing server responsiveness when traffic picks up again. Server efficiency gain = 9.5%.



**Figure 6.** Load profile curve used to modulate the arrival rate of client requests generated using an exponential distribution. Specifies for any given time the probability that a stochastically generated client request will be submitted to the CAC algorithm.



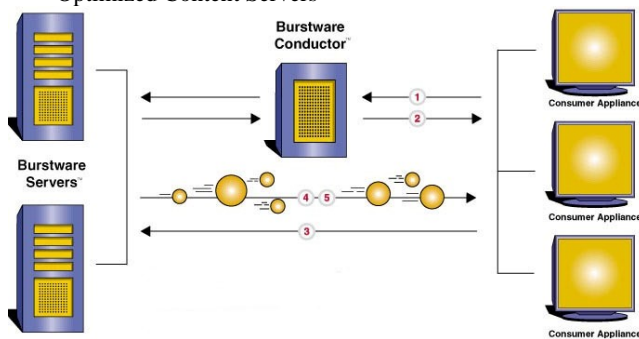
**Figure 7 Results panel to the left, and to the right buffer level trajectories for three sessions launched under progressively heavier load conditions, captured while the server is delivering session content.**

The buffer level trajectory to the right in figure 7 exhibits the characteristic arch of a session under heavy load conditions, exceeding by only a small margin the predicted reserve level trajectory for a  $\beta$  value of .125 depicted in figure 3. The leftmost trajectory ceased to increase only when the buffer saturated, unlike the middle trajectory whose session was forced to surrender bandwidth in mid-span, as server load increased.

## 5.2 Burstware Product

The Burstware Video and Live event delivery system is scalable and fault-tolerant. It comprises several tiers, depicted in figure 8, as follows:

- Client-side plug-ins for the Windows (WMP) and QuickTime media players, and also for a number of set-top boxes, that endow these with intelligent buffer/cache management
- Application Servers (Conductors), which are the first point of contact for service requests from clients. These platforms manage a server farm, functioning as load balancers that direct incoming service requests to the least loaded server.
- Optimized Content Servers



**Figure 8. Three-tiered Burstware architecture. 1) player requests service from conductor; 2) Conductor sends ordered list of servers to player; 3) player selects server from which to receive content; 4) Server begins to deliver data; 5) server receives status updates from the player; as shown the flow rate from server to client varies over time, slackening with increased server load, and increasing when load drops off.**

A leading digital media consulting firm, Approach, Inc. recently completed benchmark study [14] comparing the performance of BurstWare with Windows Media Services. The reader is referred

to that study for further details on the methods it employed and the conclusions – favorable to Burstware -- that were drawn.

## 6. Related Work

Our flow optimization methods should be viewed as constituting the lowest & perhaps only the first of many *supervisory control layers* to sit astride all streaming sessions, which comprise their own hierarchical stack of control functions: rate adaptive control sitting above a flow rate regulator [1] which relies on a transport entity. Our reliance on the last two of these three was discussed in an introductory section. To date we have not integrated our flow optimizer with a rate change facility of the kind discussed in [2][3], yet we believe there is much to be gained by doing so.

Channel capacity is known to vary widely over time. A capacity drop to below the reserve flow rate is problematic as it causes the latter to rise in violation of its monotonicity assumption. In a server with available bandwidth to spare such a rise can be accommodated by the flow optimizer (by bumping the session’s reservation to the higher value) provided the flow deficit can be made up in the near term by draining pre-delivered content from the media buffer. Should either of these resources approach depletion, stream degradation to a lesser bit-rate or frame rate may be the only viable option short of a pause. Whether up or down, any rate change is a change to  $f_e$ , which must be preceded by change to session’s reserve flow rate allocation by the optimizer.

As can be seen, the optimizer and rate adapter must be integrated with care. One possible approach would be to modify a rate adapter such that it renegotiates its proposed  $f_e$  value with the flow optimizer prior to effecting the rate change it deems appropriate based on information provided by the flow regulator. Rate increases might well be subject to deferral under heavy load conditions, awaiting bandwidth availability. The opposite is true of rate flow decreases: these would be forestalled as long as possible by the optimizer on a system that is not fully loaded, as previously discussed.

Variable bit-rate (VBR) streams are subject to burstiness over a number of time scales. Smoothing of VBR streams *by work-ahead* [9][10][8] involves early delivery of frames to a media buffer large enough to absorb the variability in the consumption bit-rate at a given frame rate. Smoothing algorithms adjust the incoming flow rate in stepwise fashion according to a *flow schedule* that ensures that the media buffer neither overflows nor underflows.

These techniques can be seen to closely resemble our own: both adjust the flow rate over time, employ the media buffer to absorb bursts and employ the same buffer overflow constraint. Our use of the reserve flow constraint (9) in lieu of the much less stringent buffer underflow constraint to bound flow from below ensures rapid content accumulation in the media buffer. Given a sufficiently high  $\beta$  value and large media buffer, it is highly probable<sup>9</sup> that the flow schedule imposed by the optimizer will be *VBR-feasible*, i.e., deliver all VBR frames without loss or pause. Anecdotal evidence collected thus far concerning the application of flow optimization to VBR streams confirms this view.

<sup>9</sup> In the absence of any a-priori knowledge of the content, this may be the best one can expect.

## 7. Conclusion

As the Internet continues to evolve, we can expect enormous increases in backbone capacity with concomitant decreases in the transmission costs. This development, which may or may not be imminent, will tend to undermine the economic as well as technical justifications for edge caching, and permit a return to more massive, centralized & mirrored deployment architectures, which, besides being far easier to manage, are in our opinion the best way of tapping the vast body of potentially available content at reasonable cost. The stage will then be set for the emergence of streaming services on a massive scale, provided servers can keep up, and players can deliver an experience that satisfies users.

Be that as it may. In the interim it is important that the limited capacity of streaming edge servers & caches be increased. One inexpensive way of doing this is to endow existing servers & proxies with a longer reach, such that a single copy or fragment of content, pulled (or pushed) in by one server on behalf of a first user, is shared by many drawn from a wider electronic neighborhood than is now possible. If fewer copies are required on the edge, then more titles can be held there, and service is improved.

Even so, given the massive size of streaming content, it is doubtful, especially in the near term, that the edge will be able to hold anything more than the most newsworthy or popular clips and the most popular movie titles. The vast majority of titles -- spanning re-runs of TV series, obscure movies of years past, to valuable content targeted at a narrow user base -- do not linger long on the edge for lack of viewers; instead these are best served end-to-end, directly from origin servers to clients -- those willing to pay or watch ads --, with as high a QOS as current technology can muster.

We believe that the methods presented in this report address these challenges, by improving the efficiency of origin or edge servers, increasing their reach, and lifting the quality of service offered to viewers. The methods are quite general, and as such they lend themselves to integration with present as well as future multimedia server platforms, to whose continued evolution they should contribute in a powerful way.

## 8. References

- [1] Kang Li, Jonathan Walpole, Dylan McNamee, Calton Pu, and David Steere. A Rate-Matching Packet Scheduler for Real-Rate Applications, in proceedings of Multimedia Computing and Networking 2001, San Jose, California, January, 2001
- [2] S. Pejhan. T. Chiang, Y. Zhang. Dynamic Frame Rate Control for Video Streams, Proc of ACM Multimedia 99
- [3] RealNetworks Corp. Surestream Whitepaper [http://www.realnetworks.com/devzone/documentation/wp\\_surestream.html?src=noref,rnhmpg\\_020701,rnhmt n,nosrc](http://www.realnetworks.com/devzone/documentation/wp_surestream.html?src=noref,rnhmpg_020701,rnhmt n,nosrc)
- [4] RealNetworks Corp. RealSystem Proxy White Paper [http://docs.real.com/docs/rn/RealSystem\\_Proxy8.pdf?src=r-realnetworks.proxy\\_011001](http://docs.real.com/docs/rn/RealSystem_Proxy8.pdf?src=r-realnetworks.proxy_011001)
- [5] G. Ghinea, J.P. Thomas, R.S. Fish Multimedia, Network Protocols and Users - Bridging the Gap. Proc of ACM Multimedia 99
- [6] M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang. Caching techniques for streaming multimedia over the internet. Technical Report BL011345-990409-04TM, Bell Laboratories, April 1999.
- [7] M. Andrews and K. Munagala. Online algorithms for caching multimedia streams. 8th European Symposium on Algorithms, 2000
- [8] S. Sen, J. Rexford, J. Dey, J. Kurose, and D. Towsley, Online Smoothing of Variable-Bit-Rate Streaming Video, Tech. Rep. 98-75, Department of Computer Science, University of Massachusetts Amherst, 1998
- [9] J. Rexford, S. Sen, and A. Basso, A Smoothing Proxy Service for Variable Bit-rate Streaming Video, in Proc. IEEE Global Internet Symposium, Rio de Janeiro, Brazil, Dec. 1999
- [10] J.D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley, Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing, IEEE/ACM Trans. Networking, vol. 6, pp. 397--410, August 1998
- [11] R. Lang et al. US patents 4,963,995 (issued 10/16/90), 5,057,932 (issued 10/15/91), 5,164,839 (issued 11/17/92) 5,995,705 (issued 11/30/99)
- [12] N. Polish. US patent 5,963,202 (issued 10/5/99)
- [13] L. GradshTEYN, L. Ryzhik. Table of Integrals Series and Products, Academic Press, 2000.
- [14] Approach Inc. Report at URL <http://burst.burst.com/f/approach.pdf>