A Survey and New Results in Renegotiated Service

Ellen W. Zegura *Scott McFarland* *Ojas Parekh*

Networking and Telecommunications Group
College of Computing
Georgia Tech
Atlanta, GA 30332-0280

Abstract

To use a network service class, an application must determine which service is suitable and provide a concise description of traffic characteristics required by the service class. Applications may find it difficult to do this if accurate *a priori* knowledge of traffic is not available or if the traffic characteristics will change over time. Both of these difficulties can be addressed by a *renegotiated* service that supports changes in the traffic contract over time.

We focus in this paper on applications that have known, but changing, service requirements, thus they can use off-line processing. We survey existing work and present new algorithms for determining *when* to renegotiate the service and *what* new traffic contract to supply. We evaluate the performance of our work and the work of others using a common set of application streams.

1 Introduction

In recent years, much effort has been devoted to the development of new network service classes. For example, a constant bit rate (CBR) service provides an application with network resources capable of transmitting at a constant (specified) bit rate, usually with low loss and delay. A variable bit rate (VBR) service provides an application with network resources capable of transmitting at a rate that varies, within constraints. In using a network with a variety of service classes, an application must determine which service is suitable, provide a concise description of its traffic characteristics, and request the service from the network. If the request is granted, the application transmits data over the supplied channel, then releases the network resources.

For several reasons, an application may find it difficult to use a basic network service as described above. First, the application may not have accurate *a priori* knowledge of the traffic which it will generate, particularly if that behavior is user-driven or otherwise depends on run-time activity. Second, the traffic characteristics may change over time. If a service is negotiated that meets the worst case requirements, the network may be severely underutilized (and the user may be significantly overcharged) during much of the connection time.

Both problems can be partially addressed by a *renegotiated* service that supports modifications to the traffic contract during the lifetime of an application [1]. An application that does not have accurate a priori knowledge of traffic can use a renegotiated service in an on-line, real-time fashion. An application with known, but changing, traffic can use off-line methods to determine when to renegotiate. We focus in this paper on the use of a renegotiated service for applications in the latter category. The main problems to be addressed are *when* to renegotiate the service and *what* traffic contract to negotiate.

The purpose of this paper is twofold. First, we survey results on the use of renegotiated service for off-line applications. The results are described with respect to a common framework, intended to clarify the relationships between different efforts in the renegotiated services area. Second, we present several new results in renegotiated services, with emphasis on efficient methods to find renegotiation points. We also present a performance comparison for a selected set of methods on common application streams.

2 Problem Framework

Figure 1 depicts the components of the renegotiation problem. For simplicity of exposition, we assume the application is real-time compressed video, however the problem and solutions are more generally applicable to any real-time, variable bit-rate, stored application.

The playout requirements of the application stream are represented as bytes per time unit, where the time unit is chosen based on the application (e.g., one frame in video). The first processing step is responsible for generating a *feasible transmission schedule*. That is, given the playout requirements and the client buffer size B, this step produces a schedule consisting of the transmission rate per time unit such that the client does not suffer from starvation or buffer overflow during playout. The transmission rate may lag or exceed the playout rate in a given time unit, since the buffer can absorb short term differences. The example shows a transmission schedule that sends at a high rate early in the transmission (storing excess data in the client buffer), then reduces the rate (with playout using data in the client buffer).

The second processing step produces a schedule for the renegotiation of the network service. That is, given a feasible transmission schedule, this step produces a schedule consisting of the times when renegotiation is required, and the network service parameters for each service interval. The renegotiation schedule must ensure that sufficient network resources are allocated at each time unit to support the transmission schedule. The example demonstrates a renegotiation schedule with one renegotiation point, between frame times four and five.

As we will see in the survey, it is common for the solution to the two steps to be combined into a single algorithm that simultaneously finds a feasible transmission schedule with an associated renegotiation schedule. We will call this a *feasible renegotiation schedule*.

Beyond the constraints mentioned above, the renegotiation schedule is generally designed to optimize some criteria. From a user perspective, the renegotiation schedule should minimize the *cost* incurred by the use of the service. This cost is usually expressed as the sum

[1] Alternatively, a renegotiation entity can provide an interface between the application and the (non-renegotiation-based) basic services.

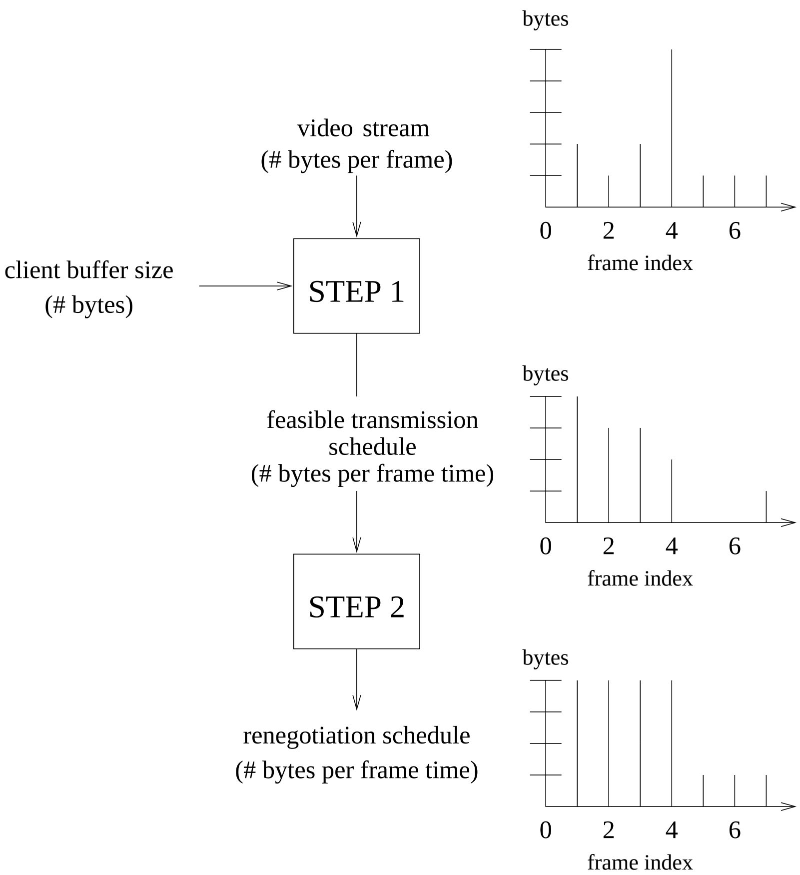

Figure 1: Renegotiation problem framework

Terminology	
Original Stream	
N	total number of frames
f_i	number of bytes in frame i, $0 \leq i < N$
Transmission Schedule	
B	size (in bytes) of client buffer
a_i	number of bytes sent in frame time i, $0 \leq i < N$
Renegotiation Schedule	
R	number of renegotiations
e_i	index of last frame in i-th interval, $0 \leq i \leq R$
l_i	length (in frames) of i-th interval, $0 \leq i \leq R$
r_i	constant rate during i-th interval, $0 \leq i \leq R$

Table 1: Renegotiation Algorithm Terminology

of a constant cost for each renegotiation and a cost for the network resources allocated over time. From a network operator perspective, the efficient use of the network resources is of importance. The idea is that if each application uses the resources efficiently, then the network will be able to accomodate more applications. Most algorithms optimize the cost of the service and then analyze network efficiency as part of evaluation. We will see one algorithm in the survey that explicitly optimizes network efficiency.

3 Survey

Conceptually, a renegotiated service could support arbitrary changes to the traffic contract, including changes in the basic service. However, most of the work in the area of renegotiated service has focused on renegotiating within the constant bit rate class.

Following common practice, we assume a discrete-time model at the level of frames. All of the algorithms could be adapted in a straightforward way for a different level of granularity. Table 1 contains terminology used by renegotiation algorithms.

The work by Salehi et al. [7] contains a clean separation between the two processing steps in Figure 1. Three main algorithms are described. The first (denoted Algorithm A in [7]) finds a feasible transmission schedule that optimizes a network efficiency measure called the "smoothness" of the schedule. Informally, a smooth schedule is one which minimizes the variability in rate over time. The precise definition uses the mathematical concept of majorization [4].

Given a feasible transmission schedule and an upper bound on the number of renegotiations, the second algorithm (Algorithm B) uses dynamic programming to minimize the cost for a renegotiation schedule. If R denotes the actual number of renegotiations, the cost is defined as:

$$\alpha R + \sum_{i=0}^{R} \phi(r_i) l_i,$$

where α denotes the cost per renegotiation and $\phi(r)$ denotes the cost for reserving bandwidth r over one frame time. We will refer to the first term in the cost as the *renegotiation cost* and the second term as the *resource-time product*. A variation on this algorithm (Algorithm C) minimizes the cost using an arbitrary number of renegotiations. Thus, executing Algorithm A followed by Algorithm C will produce a renegotiation schedule of minimal cost for the transmission schedule that is optimally smooth.

Finally, Algorithm D combines the two steps to construct a feasible renegotiation schedule. That is, a feasible transmission schedule is determined in conjunction with finding a renegotiation schedule that minimizes cost. Smoothness of the transmission schedule is not taken into account.

Grossglauser et al. [3] also describe an algorithm for constructing a feasible renegotiation schedule that minimizes cost, defined as above with $\phi(r) = \gamma r$. That is, they assume a constant cost γ per allocated bandwidth over a time unit. They represent all feasible transmission schedules using a trellis graph structure; the shortest-path from a designated source vertex to destination vertex corresponds to the minimum cost feasible renegotiation schedule.

Feng and Sechrest [2] describe a Critical Bandwidth Allocation algorithm for constructing a feasible transmission schedule using an optimization criteria that anticipates (to an extent) the renegotiation step [2]. Specifically, their algorithm will minimize the number of *increases* in the transmission rate, while avoiding buffer starvation and overflow. For all schedules that achieve the minimum number of rate increases, they find the schedule with the minimum bandwidth allocation during the "segments" between rate increases. The minimum bandwidth allocation in a segment consists of monotonically decreasing "runs" of constant transmission rate. The algorithm does not attempt to minimize the number of rate decreases.

A straightforward renegotiation schedule can be derived from the Feng and Sechrest transmission schedule by placing a renegotiation point every time the transmission rate increases. This renegotiation schedule should have a small number of renegotiation points, but potentially a large cost due to network resource allocation. An alternative schedule can be derived by renegotiating every time the transmission rate changes (i.e., increases or decreases). This schedule may have a large number of renegotiation points, but will reduce (though not minimize) the cost due to network resource allocation.

From the Feng and Sechrest pseudocode it becomes evident that the Chang and Kung algorithm [1] takes a very similar approach, constructing a feasible transmission schedule that attempts to minimize the number of *changes* in the transmission rate. They describe a Piecewise Constant Bit Rate heuristic to construct a schedule, essentially using the same technique as in Feng and Sechrest. As in the Feng and Sechrest case, a renegotiation schedule could be constructed from the resulting transmission schedule, however Chang and Kung envision a network service that does call admission using the transmission schedules to determine whether the network resources (specifically, link bandwidth) will be exceeded. That is, the service will directly accomodate the changes in transmission rate. Thus, the optimization criteria for the overall system is the efficiency of network utilization, measured

[2]In the Appendix we provide pseudocode for the Feng and Sechrest algorithm. The exact processing is not laid out in detail in the original paper; further, there appear to be several minor typographical errors in the equations in the paper.

Reference	Step(s)	Optimization criteria	Runtime
Salehi A	1	smoothness	$O(N)$
Salehi B,C	2	cost	$O(RN^2), O(N^2)$
Salehi D	1,2	cost	$O(Bm^2N)$
Grossglauser	1,2	cost	not given
Feng	1,2	# rate increases	$O(N)$
Chang	1,2	# renegotiations	$O(N)$
McManus	1,2	client buffer space, other	$O(N^2C^2(R+N))$

Table 2: Algorithm comparison

as the percentage of bottleneck link bandwidth that is allocated.

Initially, McManus and Ross [5] used the client buffer space as the optimization criteria. Given the number of renegotiations R, they use a dynamic programming algorithm to find a feasible renegotiation schedule that minimizes the necessary client buffer. The cost of the network resources are not taken into account. They also describe a more efficient heuristic to reduce (but not necessarily minimize) client buffer space. More recently, they have demonstrated that the dynamic programming paradigm can be used to optimize a variety of criteria, including minimizing the number of renegotiations subject to a given client buffer constraint [6].

Table 2 compares the various approaches based on optimization criteria and run time. Several parameters are particular to certain algorithms, including m, which is the number of discrete reservable bandwidths, and C, which is the number of fixed-size packets in the stream. The runtime of the algorithms that combine Steps 1 and 2 to optimize cost can be quite long if N is large. Salehi and McManus reduce the runtime by discretizing the solution space in various ways (e.g., restricting the number of allowed bandwidths or the particular locations in the transmission when renegotiation is allowed).

4 New Results

In this section, we present new results that address an unexplored aspect of renegotiated service. Specifically, we consider approximation algorithms for Step 2. Thus, the input to our algorithms is a feasible transmission schedule; this stream may be the original application stream or it may be the output from a Step 1 algorithm. Our approximation algorithms find a set of renegotiation points with faster runtime than an optimal algorithm.

4.1 Optimal Algorithm

We begin by stating our version of the optimal Step 2 algorithm, using a shortest-path formulation. Following the terminology of the previous section, let a_i denote the number of bytes transmitted in frame time i, $0 \leq i < N$. We assume that the transmission sequence represented by the a_i's is feasible for the given client buffer. We construct a graph corresponding to the problem instance as follows. (See Figure 2.) For each frame time i,

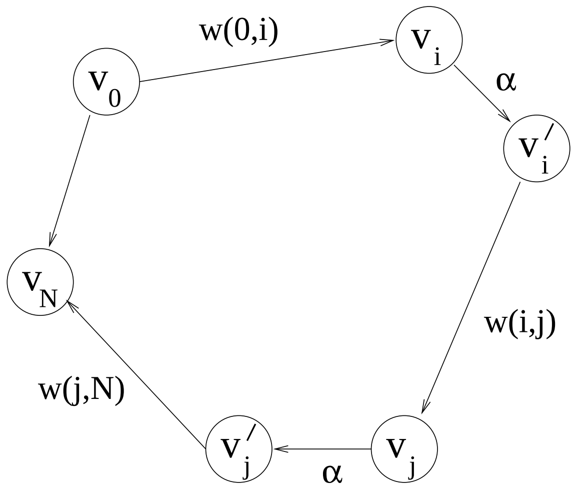

Figure 2: Shortest path formulation

$0 < i < N$, we associate two vertices v_i and v'_i. A directed edge (v_i, v'_i) is included for all i, with weight α. (Recall that α expresses the cost for each renegotiation point.) We also have a vertex v_0 to represent time 0 and v_N to represent time N. For each i and j such that $j > i$, we include a directed edge (v'_i, v_j). The weight on the edge is given by:

$$w(i,j) = (j - i + 1) max_{i \leq k \leq j} a_k$$

Thus the weight is the cost for reserving sufficient bandwidth over the interval from frame time i to frame time j, assuming that the cost is the product of the bandwidth reserved and the length of the interval [3].

Given this problem formulation, a shortest path from vertex v_0 to vertex v_N represents an optimal Step 2 solution. Each vertex v_i in the path means that a renegotiation point is placed just before frame time i in the transmission schedule; the cost of the path is the cost of the renegotiation schedule. The runtime of the shortest-path computation is $O(N^2)$.

If the number of renegotiation points R is given, we can simplify the graph formulation to collapse the vertex pairs (v_i, v'_i) into a single vertex. We also must restrict the shortest path to contain no more than R edges. The Bellman-Ford-Moore algorithm, which finds paths using breadth-first search, will find the shortest path containing at most R edges in time $O(RN^2)$. Note that these runtimes match the Salehi Algorithm B and C runtimes; we present the shortest-path formulation because it provides some intuition about the process of constructing a renegotiation schedule of low cost.

4.2 Approximation Algorithms

We now turn to approximation algorithms for the problem of finding a renegotiation schedule. In each case, we assume that the number of renegotiation points, R, is given. In the

[3] Note that if the a_i's contain "runs" of constant rate, we need only consider renegotiating at the boundaries between changes in transmission rate.

Algorithm	Runtime
Optimal (arbitrary R)	$O(N^2)$
Optimal (R given)	$O(RN^2)$
Sequential	$O(RN)$
Concurrent	$O(N \log R)$
Sequential-K	$O((R/K)N \log K)$

Table 3: Runtimes for Approximation Algorithms

Sequential algorithm, the R renegotiation points are placed one at a time in the transmission schedule. Each point is placed so as to maximally reduce the current resource-time product. In the simplified graph formulation, this corresponds to maintaining a current solution, initially consisting of the edge from v_0 to v_N. At each step, the algorithm considers each possible location for a new renegotiation point. In the graph, introducing a new renegotiation point at location k amounts to replacing a particular edge in the current solution and replacing it by two edges, using vertex v_k as an intermediate point. The edge that is removed is the one whose corresponding interval contains location k. The running time of the Sequential algorithm is $O(RN)$.

In the **Concurrent** algorithm, each potential renegotiation point is evaluated for the effect that it would have on reducing the resource-time product, *were it to be the only renegotiation point*. The R points with the greatest effect are selected as the renegotiation points. In the simplified graph formulation, this corresponds to evaluating the length of the two-edge path consisting of (v_0, v_i) and (v_i, v_N) for all i. The R shortest two-hop paths determine the R renegotiation points. The running time of the Concurrent algorithm is $O(N \log R)$.

In the **Sequential-K** algorithm, K points are placed in each sequential step. These K points are chosen by finding the K points that each maximally reduce the resource-time product of the current solution. When $K = 1$ this becomes the Sequential algorithm; when $K = R$ it is the Concurrent algorithm. The running time of the algorithm is $O((R/K)N \log K)$. In the performance analysis, we examine the Sequential-2 case.

To summarize, Table 3 gives the running times of the optimal and approximation algorithms.

5 Performance Comparison

We conclude this paper with a comparison of the performance of the algorithms. Algorithms for Step 1 are combined with Algorithms for Step 2, and these combinations are compared to an algorithm that solve Steps 1 and 2 simultaneously. Specifically, we consider Salehi Algorithm A and Feng as Step 1 algorithms. Note that though Feng was originally designed to solve Steps 1 and 2 simultaneously, it can also be used as a Step 1 algorithm to find a feasible transmission schedule that is then passed to an algorithm to find a renegotiation schedule.

We consider the Optimal algorithm (ours or Salehi Algorithm C), the Sequential Algorithm and the Concurrent Algorithm as Step 2 algorithms. We compare to the Feng algorithm that solves both steps simultaneously.

The application streams that we consider are the first five minutes of two longer video streams: Advertisements and CNN, and the first 2.5 minutes of The Princess Bride [4]. We use a client buffer size of 64 Kbps.

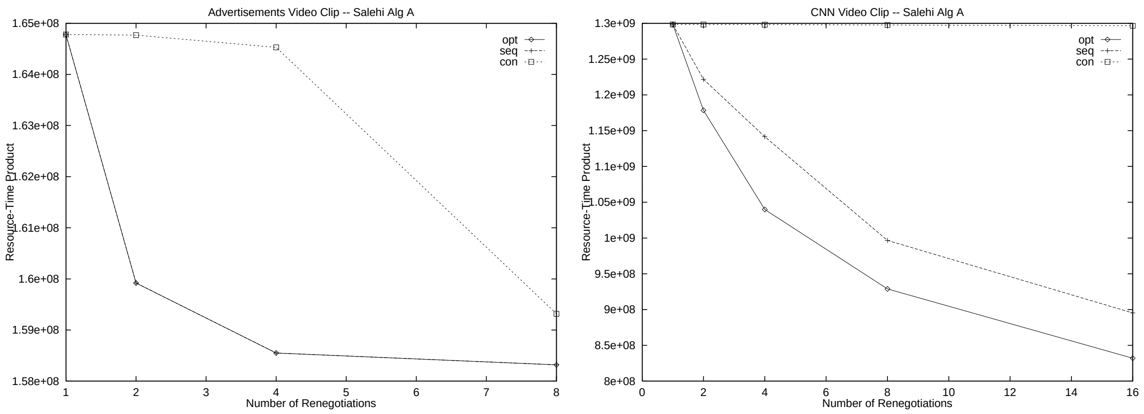

Figure 3: Salehi Algorithm A for Step 1

Figure 3 shows the resource-time product when using Salehi Algorithm A as Step 1 with all of the Step 2 algorithms, for the Advertisements trace (on the left) and the CNN trace (on the right). Each of these traces consists of 9000 frames, corresponding to 30 frames per second for 5 minutes. After smoothing, the Advertisements trace has only 11 different transmission rates, while the CNN trace has 1852. Because the transmission schedule for the Advertisements trace is so short, the Sequential algorithm is able to match the Optimal algorithm for Step 2. The Concurrent algorithm performs somewhat more poorly for a small number of renegotiations. For the CNN trace, the Optimal algorithm performs somewhat better than the Sequential algorithm, with the difference increasing as R increases. The Concurrent algorithm does quite poorly over the range of R in the plot.

Figure 4 shows the resource-time product with Feng as the Step 1 algorithm, again for the Advertisements trace (on the left) and the CNN trace (on the right). After running the Feng algorithm, the Advertisements trace has 562 different transmission rates, while the CNN trace has 856. The basic shape of the three curves is similar to Figure 3, with Sequential comparable to Optimal, and Concurrent performing relatively poorly. Comparing to Figure 3, we note that the Salehi smoothing algorithm makes a better Step 1 algorithm, particularly for the Advertisements trace.

Finally, Figure 5 compares the Salehi algorithm (on the left) and the Feng algorithm (on the right) for Step 1, with all Step 2 algorithms. We again see that the Salehi algorithm makes a better Step 1 algorithm. Though the Optimal appears to be doing somewhat better than the Sequential algorithm, the relative improvement is similar to the earlier plots, at

[4]The traces all come from the repository maintained by the Tenet group at UC Berkeley. See ftp://tenet.berkeley.edu/pub/dbind/traces.

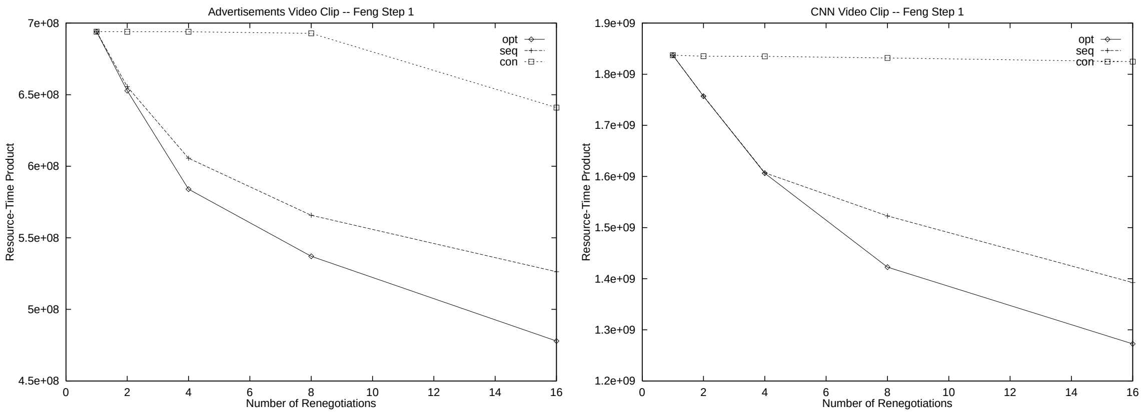

Figure 4: Feng Algorithm for Step 1

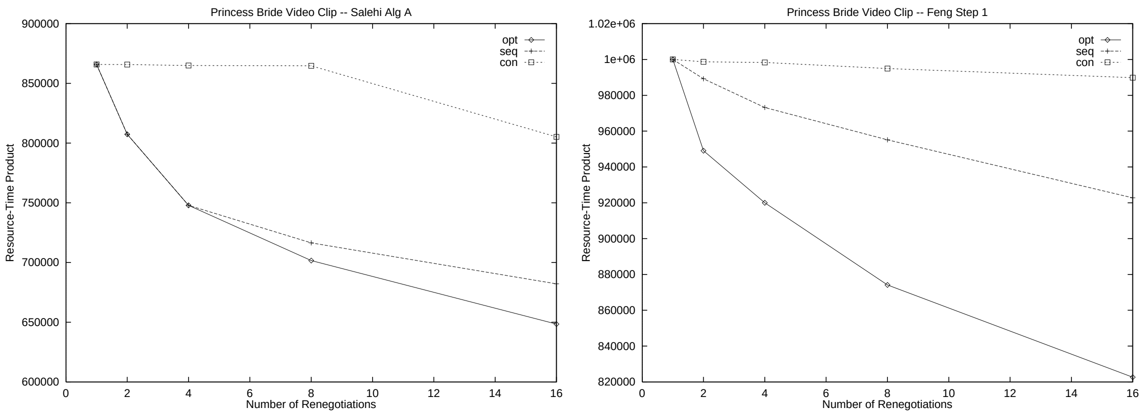

Figure 5: Princess Bride Trace

roughly 10%.

6 Appendix: Pseudocode for Feng and Sechrest

```
Global variables:
  f[i] - number of bytes in frame i
  N - total number of frames
  b - size (in bytes) of receiver buffer


---------------------------Main Routine---------------------------

Local variables:
```

```
  leftend - left end of region (i.e., index of first frame in region)
  rightend - right end of region (i.e., index of last frame in region)

Main {
    leftend = 1;
    while (leftend < N) {
        rightend = Findrightend(leftend);
        leftend = rightend + 1;
    }
}

------------------------Findrightend Routine-----------------------

Local variables:
  sum - running sum of frame sizes
  favg - running average of frame sizes
  lub - least upper bound on bandwidth for region
  glb - greatest lower bound on bandwidth for region
  lbi - index of frame that causes lower bound to be achieved
  r - bandwidth of region
  j - variable used to index through frames in region
  done - boolean variable to indicate if right endpoint has been found

Findrightend(left) {
    j = left; done = FALSE; sum = 0;
    lub = f[j] + b; lbi = j;
    glb = f[j];
    while (j < N && !done) {
        /* update all variables */
        sum += f[j];
        favg = sum/(j-left+1);
        lub = min(lub, favg + b/(j-left+1));

        /* Compute glb = max(glb, favg) and update lbi if necessary */
        if (glb <= favg) {
            glb = favg;
            lbi = j;
        }
        /* do comparisons to figure out if region ends here */
        if (favg > lub) {
            r = lub;
            right = j-1;
            done = TRUE;
        }
        else {
```

```
            if (glb > lub) {
                r = glb;
                right = lbi;
                done = TRUE;
            }
            else
                j++;
        }
    }
    printf ("Bandwidth of region from %d to %d is %d\n",left,right,r)
    return right;
}
```

References

[1] K. Chang and H.T. Kung. Efficient time-domain bandwidth allocation for video-on-demand. In *Proceedings of Int'l Conference on Computer Communication Networks (IC3N)*, 1996.

[2] W. Feng and S. Sechrest. Critical bandwidth allocation for delivery of compressed video. *Computer Communications*, October 1995.

[3] M. Grossglauser, S. Keshav, and D. Tse. RCBR: Simple and efficient service for multiple time-scale traffic. In *Proceedings of ACM Sigcomm*, 1995.

[4] A. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Applications.* Academic Press, New York, 1979.

[5] J. McManus and K. Ross. Prerecorded VBR sources in ATM networks: Piecewise-constant-rate transmission and transport. See http://www.systems.seas.upenn.edu/ross/PCRTT.ps, September 1995.

[6] J. McManus and K. Ross. A dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks. See http://www.systems.seas.upenn.edu/ ross/jean2.ps, January 1997.

[7] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Suporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proceedings of ACM Sigmetrics*, May 1996.