Partitioning of Multiple Fine-Grained Scalable Video Sequences Concurrently Streamed to Heterogeneous Clients

Cheng-Hsin Hsu and Mohamed Hefeeda, *Member, IEEE*

***Abstract*—Fine-grained scalable (FGS) coding of video streams has been proposed in the literature to accommodate client heterogeneity. FGS streams are composed of two layers: a base layer, which provides basic quality, and a single enhancement layer that adds incremental quality refinements proportional to number of bits received. The base layer uses nonscalable coding which is more efficient in terms of compression ratio than scalable coding used in the enhancement layer. Thus for coding efficiency larger base layers are desired. Larger base layers, however, disqualify more clients from getting the stream. In this paper, we experimentally analyze this coding efficiency gap using diverse video sequences. For FGS sequences, we show that this gap is a non-increasing function of the base layer rate. We then formulate an optimization problem to determine the base layer rate of a single sequence to maximize the average quality for a given client bandwidth distribution. We design an optimal and efficient algorithm (called FGSOPT) to solve this problem. We extend our formulation to the multiple-sequence case, in which a bandwidth-limited server concurrently streams multiple FGS sequences to diverse sets of clients. We prove that this problem is NP-Complete. We design a branch-and-bound algorithm (called MFGSOPT) to compute the optimal solution. MFGSOPT runs fast for many typical cases because it intelligently cuts the search space. In the worst case, however, it has exponential time complexity. We also propose a heuristic algorithm (called MFGS) to solve the multiple-sequence problem. We experimentally show that MFGS produces near-optimal results and it scales to large problems: it terminates in less than 0.5 s for problems with more than 30 sequences. Therefore, MFGS can be used in dynamic systems, where the server periodically adjusts the structure of FGS streams to suit current client distributions.**

***Index Terms*—Fine-grained scalable coding, multimedia communication, quality optimization, video streaming.**

I. Introduction

VIDEO streaming over the Internet is increasingly getting very popular as higher bandwidth links and more powerful machines are becoming more affordable for end users. Users typically seek the highest possible video quality. Users, however, are quite heterogeneous in terms of network bandwidth and processing capacity. A conventional nonscalable coded stream only supports one decoding rate, which is insufficient in such a heterogeneous environment. This is because supporting clients with different bandwidth requires storing and serving multiple versions of each video stream. To cope with this heterogeneity, various scalable coding techniques have been proposed in the literature. A scalable coded stream consists of various representations of the original video sequence, with different resolutions, frame rates, or quality.

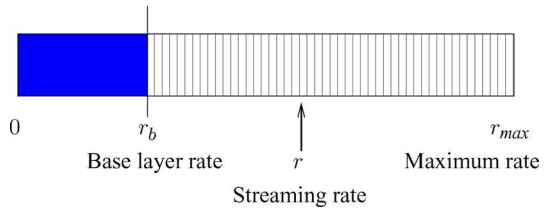


Fig. 1. Simple representation of an FGS-coded stream. The stream can be decoded at any rate between r_b and r_{max}.

Scalable coders are roughly categorized into two classes: coarse-grained scalable and fine-grained scalable. Coarse-grained scalable (CGS) coders divide a video stream into multiple layers. They provide limited rate scalability at the layer level: clients receiving incomplete layers cannot use them to enhance quality. In contrast, fine-grained scalable (FGS) coders provide finer rate scalability and better error resiliency [1]–[3]. An FGS encoder compresses video data into two layers: a base layer which provides basic quality, and a single enhancement layer that adds incremental quality refinements proportional to the number of *bits* received. As shown in Fig. 1, arbitrary truncation (at the bit level) of the enhancement layer to achieve a target rate is possible for FGS coding. This in turn enables streaming servers to fully utilize available bandwidth of individual clients, which results in better video playback quality and ultimately higher user satisfaction.

The fine rate scalability of FGS, however, comes at an expense of coding efficiency. That is, an FGS stream results in lower quality compared to a nonscalable coded stream when both streams are reconstructed at the same bit rate. Previous research indicates that this coding efficiency gap can be as high as 2 dB in MPEG-4 FGS coders [4]. The two main causes of

Manuscript received May 29, 2007; revised November 29, 2007. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant #313083 and RTI Grant #344619. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Madjid Merabti.

C.-H. Hsu is with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada (e-mail: cha16@cs.sfu.ca).

M. Hefeeda is with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada and also with Mansoura University, Mansoura, Egypt (e-mail: cha16@cs.sfu.ca; mhefeeda@cs.sfu.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2008.917365

this coding efficiency gap are: 1) less accurate motion compensation as only base layer is used for motion estimation[1] and 2) unexploited correlation between base layer and enhancement layer. The coding efficiency gap is more significant in video streams coded with lower base layer rates because less information is contained in the base layer in this case, which leads to higher motion estimation error. Furthermore, video sequences with lower temporal redundancy result in a smaller gap because motion compensation does not provide much quality gain for these sequences [1], [4].

While the temporal correlation is fixed for a given sequence, the base layer rate is a configurable parameter. Therefore, content providers may code a sequence at higher base layer rate to reduce the coding efficiency gap and achieve higher quality. This may increase perceived quality for some clients, which could allow the provider to charge higher service rates.[2] On the other hand, a higher base layer rate may disqualify other clients from receiving the complete base layer stream. Since the base layer is nonscalable, these disqualified clients cannot even render basic quality and effectively they are denied access to the video stream, even though the server may have enough bandwidth to serve them. This may lead to under-utilization of server bandwidth. Hence, there is a trade-off between coding efficiency (and the resulting client perceived quality) on one hand, and the number of clients that can receive the stream (and the resulting server bandwidth utilization) on the other.

To FGS encode a given video sequence, content providers have many options for the base layer rate. Each base layer rate determines the average perceived quality for all clients, which can be used as a metric for user satisfaction. In streaming systems where multiple video sequences are concurrently streamed to diverse sets of clients, content providers have even more choices for the base layer rates of individual sequences. We refer to the problem of determining the base layer rates of FGS streams as *structuring FGS streams*, because the base layer defines the structure of stream (as shown in Fig. 1). Unfortunately, there are no systematic ways in the literature to aid content providers in choosing the optimal structure of FGS streams that would maximize the average quality for all clients over all sequences. In this paper, we propose efficient and optimal algorithms to solve the stream structuring problem.

A. Paper Contributions

Our contributions in this paper can be summarized as follows:

- We experimentally analyze the coding efficiency gap using diverse video sequences. For FGS sequences, we show that this gap is a nonincreasing function of the base layer rate. This analysis could be of interest in its own right for streaming systems that employ FGS encoding.
- We formulate an optimization problem to determine the base layer of a single sequence to maximize the average quality for a given client bandwidth distribution. We design an optimal and efficient algorithm (called FGSOPT) to solve this problem. Preliminary results of this part appear in our previous work [6].
- We extend our formulation to the multiple-sequence case, in which a bandwidth-limited server concurrently streams multiple FGS sequences to diverse sets of clients. The objective is to efficiently utilize the server bandwidth while maximizing perceived quality for all clients. We prove that this problem is NP-complete. We design a branch-and-bound algorithm (called MFGSOPT) to compute the optimal solution. MFGSOPT runs fast for many typical cases because it intelligently cuts the search space. In the worst case, however, it has exponential time complexity. Therefore, MFGSOPT is more suitable for off-line scenarios where the server has pre-estimates on the client distribution.
- We propose a heuristic algorithm (called MFGS) to solve the multiple-sequence problem. We experimentally show that MFGS produces near-optimal results and it scales to large problems: it terminates in less than 0.5 s for problems with more than 30 sequences. Therefore, MFGS can be used in dynamic systems, where the server periodically adjusts the structure of FGS streams to suit current client distributions.

Our proposed algorithms systematically and optimally choose the best base layer rate for individual video sequences. This is in contrast to manual, error-prone, rule-of-thumb techniques currently used by system administrators. A schematic diagram showing where our algorithms fit in streaming systems is given in Fig. 2. As the figure indicates, our algorithms can be used during the encoding process as in live streaming systems, or they could be used to transcode already-encoded streams to maximize the quality of the current clients. In both cases, the algorithms can be invoked periodically (e.g., every 5 min) to cope with the dynamic changes in client distributions.

B. Paper Organization

The rest of this paper is organized as follows. In the next section, we summarize the related work. In Section III, we analyze the coding efficiency of FGS streams. We formulate and solve the optimization problem for single-sequence systems in Section IV. In Section V, we prove that structuring multiple sequences is an NP-complete problem, and we present a branch-and-bound algorithm and an efficient heuristic algorithm to solve it. We evaluate our algorithms in Section VI, and we conclude the paper in Section VII.

II. Related Work

The coding efficiency gap of MPEG-4 FGS coders are studied in [1], [4]. The authors investigate the relationship between the FGS coding efficiency gap and the video temporal correlation. They found that the correlation coefficient between an enhancement layer frame and its motion-compensated reference frame is a good indication of the FGS coding efficiency. We study the efficiency gap of the state-of-the-art H.264 coders.

[1]We note that motion-compensated fine-grained scalable coding tools, such as the representative progressive fine granularity scalable (PFGS) coding [5], have been proposed. Employing those coding tools would reduce the coding inefficiency gap between FGS coded streams and nonscalable ones.

[2]In this work, we consider problems to maximize user-perceived quality that depends on video encoding rates and available bandwidths. We interchangeably refer to perceived quality as reconstructed quality or simply as quality.

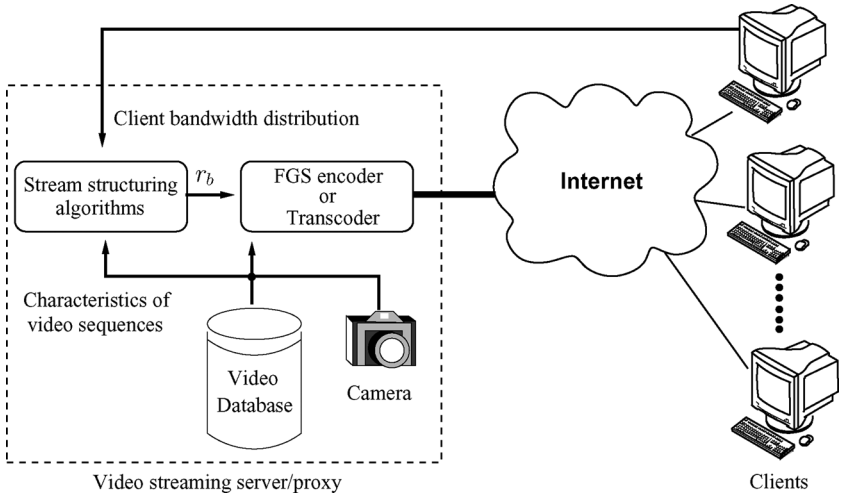


Fig. 2. System architecture of a video streaming server or proxy.

Streaming systems, e.g., [7]–[11], account for the coding efficiency of scalable coders using a layering overhead, which represents the bit rate that does not contribute toward the video quality. Similarly, we model this overhead by a coding efficiency gap function, and we empirically estimate this function.

The authors of [12] experimentally show that properly choosing base layer rates of FGS streams can improve perceived video quality for clients. This work, however, does not propose systematic methods to choose base layer rate. In contrast, our work optimally computes base layer rates based on client distributions and video sequence characteristics to maximize average perceived quality. In our previous work [13], we considered optimal coding of a single stream that has multiple layers with different granularities. The algorithm in [13] can also be used in streaming systems with multiple nonscalable versions of the stream to compute the optimal rate of each version. The work in the current paper is different because it considers streams with only one FGS layer, and more importantly, it presents a solution to the optimal structuring of multiple sequences being streamed concurrently to diverse client sets, which is more general. We are not aware of any other works in the literature that propose structuring algorithms for fine-grained scalable streams.

The performance of layered streams versus nonscalable streams is studied in [7]. The authors formulate a dynamic programming problem to compute the rate of each layer such that the average perceived video quality is maximized. The square root rate-distortion model [14] is used to estimate the coding efficiency of the layered coding. In [11], the authors consider broadcasting multi-layer video streams in a wireless cellular system with a given number of channels and client capacity distribution. They determine the optimal rate of each layer to maximize the average perceived quality. Unlike our work, these two works target coarse-grained streams which provide limited flexibility compared to fine-grained streams.

The authors of [15] study multicast streaming systems with many receivers. They partition receivers into several groups to maximize a system-wide utility function. A video stream used in such systems can be encoded into multiple cumulative layers. Several versions with different rates of the same stream can also be created. This work does not consider fine-grained streams, nor does it account for the layering overhead. Several papers [8]–[10] have approximated layering overhead for performance comparison of layered streams and multiple version streams. For example, the work in [10] proposes a linear layering overhead function, which is inspired by the experimental results in [1], whereas the works in [8], [9] employ a fixed layering overhead.

III. Characteristics of FGS-Coded Streams

A fine-grained scalable (FGS) video stream is composed of two layers: base layer and enhancement layer. As depicted in Fig. 1, the base layer is nonscalable and must be received in its entirety to provide basic quality, while the enhancement layer can be truncated at arbitrary bit positions. Therefore, an FGS-coded stream can support a wide range of streaming rates, and thus many heterogeneous clients. For a given video sequence, a maximum bit rate r_{max} is determined by the administrator. r_{max} corresponds to the maximum possible quality of the video stream, and it is specified by the resources (storage and bandwidth) allocated to that video sequence. We denote the bit rate of the base layer as r_b, where $0 \leq r_b \leq r_{max}$. An FGS-coded stream can be served at any bit rate r, where $r_b \leq r \leq r_{max}$.

Our problem in this paper is to determine the best base layer rate r_b so that the average quality is maximized for all clients. To solve this problem, we need to study the implications of varying r_b. We design the following experiments to analyze these implications. We use the Joint Scalable Video Model (JSVM) reference software version 8.0 [16] in our experiments. A brief description of this software and how we configured it is given in Section VI. We chose five diverse video sequences: City, Mobile, Soccer, Harbour, and Crew. The first four sequences are in CIF format with 30 frames per second, while the last one is in high-resolution 4CIF format. We set r_b at a specific value and encode the whole stream with a maximum rate $r_{max} = 3000\,\text{kbps}$ for CIF sequences and $r_{max} = 10000\,\text{kbps}$ for 4CIF sequences. Then we determine the quality that would be perceived by various clients decoding the stream at different rates.

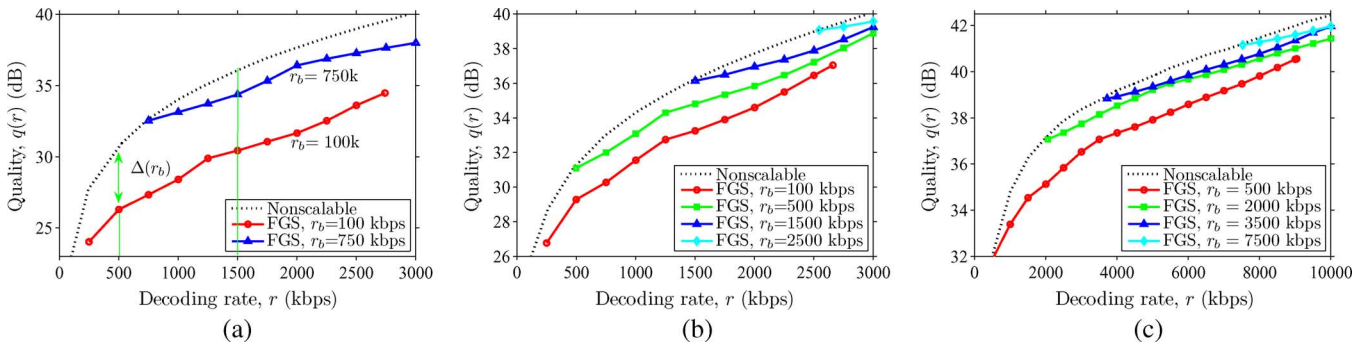


Fig. 3. Coding efficiency gap between FGS and nonscalable streams. The gap decreases as the base layer rate r_b increases. Increasing r_b, however, limits the number of clients that can receive the stream. Moreover, different video sequences pose different quality gaps. Sample results shown for CIF sequences (a), (b), and for a 4CIF sequence (c), while similar results are observed for all other sequences. (a) Mobile. (b) Harbour. (c) Crew.

We consider clients in the range between 250 and 3000 kbps with a step of 250 kbps for CIF sequences, and between 500 and 10000 kbps with a step of 500 kbps for 4CIF sequences. The quality is determined by decoding the stream and computing the peak signal to noise ratio (PSNR) in dB. We repeat the whole experiment for several values of the base layer rate, and for all five sequences. These are computationally intensive experiments and each took many CPU processing hours to complete.

The sample results of two CIF sequences and one 4CIF sequence are presented in Fig. 3, while similar results are observed for all other sequences. Several observations can be drawn from this figure. First, FGS streams have lower coding efficiency. For example, Fig. 3(a) indicates that decoding a nonscalable stream at rate 500 kbps results in 30 dB video quality, while decoding an FGS stream (with $r_b = 100\,\text{kbps}$) results in 26 dB video quality. We model this difference in coding efficiency by a *quality gap* $\Delta(r_b)$ function, which is defined as follows.

Definition 1 (Quality Gap $\Delta(r_b)$): The quality gap $\Delta(r_b)$ is defined as the quality difference between a nonscalable stream and a fine-grained scalable stream coded with base layer rate r_b, when both streams are decoded at the same bit rate.

The quality gap can be explained by the additional overhead and unexploited video redundancy caused by the scalable coding structure. A second observation we can make from Fig. 3 is that higher base layer rates lead to smaller quality gaps. For example, Fig. 3(a) shows that at a decoding rate of 1500 kbps, an FGS stream with $r_b = 750$ kbps results in about 1-dB quality gap compared to nonscalable stream, while an FGS stream with $r_b = 100$ kbps results in a 6-dB quality gap. These differences can be explained by the fact that more temporal redundancy can be exploited if the base layer contains more information, i.e., is coded at a higher rate. This observation indicates that the quality gap $\Delta(r_b)$ is a nonincreasing function of the base layer r_b. We further validate this property in the evaluation section. We will use this nonincreasing property in solving the quality optimization problem in the next sections.

A third observation is that sequences with different characteristics lead to different quality gap $\Delta(r_b)$. For instance, Figs. 3(a) and (b) show that at decoding rate 1500 kbps, FGS streams (with $r_b = 100$ kbps) for Mobile and Harbour sequences result in about a 7- and 3-dB quality gap, respectively. This observation suggests that there is no single quality gap function $\Delta(r_b)$ that is suitable for all video sequences. Therefore, we need to consider heterogeneous gap functions when designing streaming systems that concurrently serve multiple video sequences.

Finally, we note that similar scalable coding inefficiencies were observed in MPEG-4 FGS coders [1]. This is consistent with our observations on the recent H.264 coders.

IV. Single-Sequence Formulation

In this section, we formulate the quality optimization problem for a single video sequence. We also present an optimal algorithm to solve it. This problem is important for streaming systems in which: 1) the sever is broadcasting a single FGS stream to many clients or 2) the server pre-allocates a fixed bandwidth for each stream that it serves. We extend the formulation to multiple sequence systems in Section V.

A. Problem Formulation

We formulate and solve an optimization problem for a single video sequence. This optimization problem searches for the best base layer rate r_b for a given video sequence that achieves the highest average perceived quality for all clients. We consider heterogeneous client populations. We model this heterogeneity by dividing clients into C classes. All clients belonging to the same class c $(1 \le c \le C)$ have the same bandwidth b_c. We assume that $b_1 < b_2 < \cdots < b_C$. The fraction of clients in each class c is given by a probability mass function f_c, where $\sum_{c=1}^{C} f_c = 1$. No assumptions are made on the number of client classes or on the probability function. Without loss of generality, we assume that $b_C \le r_{max}$. If otherwise, we combine clients with bandwidth larger than r_{max} in a class with bandwidth equal to r_{max}. We can do that because no matter how large the client bandwidth is, it cannot receive more than the maximum rate r_{max}.

We write the single sequence optimization problem that maximizes the average perceived quality as follows:

$$\max_{r_b} \sum_{c=1}^{C} q(b_c) f_c, \text{ where } r_b \in [0, r_{max}] \tag{1}$$

where $q(b_c)$ is the quality (measured as PSNR in decibels) achieved by clients in class c.

A naive approach to solve the above problem is to try all possible values for r_b in the range $[0, r_{max}]$. This is very costly

because FGS coders allow for too many possibilities for r_b. We propose an optimal algorithm that takes at most $O(C)$ steps in the following.

B. Optimal and Efficient Algorithm

We develop an efficient, yet optimal, algorithm to find the best base layer rate that maximizes average perceived quality for a given video sequence. Our approach is enabled by the following theorem.

Theorem 1: An optimal solution r_b^* for the base layer rate that maximizes the average perceived quality for all users can be found at one of the rates b_c, where $1 \leq c \leq C$.

Proof: Referring to Fig. 3(a), we can re-write the quality of the FGS stream $q(b_c)$ for clients in class c as

$$q(b_c) = \begin{cases} 0, & b_c < r_b \\ q_{ns}(b_c) - \Delta(r_b), & b_c \geq r_b \end{cases} \quad (2)$$

where $q_{ns(b_c)}$ is the quality achieved by the nonscalable encoder at rate b_c, and $\Delta(r_b)$ is the quality gap between the FGS and nonscalable streams as defined in Section III. Notice that the quality for clients in any class c is zero if these clients do not have enough bandwidth to receive the complete base layer, i.e., if $b_c < r_b$.

We divide the search range $[0, r_{max}]$ into non-overlapping intervals $(b_{c-1}, b_c]$, where $c = 1, 2, \ldots, C$ and $b_0 = 0$. Now assume that the optimal base layer rate r_b occurs in an arbitrary interval $(b_{z-1}, b_z]$. Since all classes with $b_c \leq b_{z-1}$ receive quality of zero, the maximization problem becomes

$$\max_{r_b} \sum_{c=z}^{C} [q_{ns}(b_c) - \Delta(r_b)]\, f_c, \text{ where } r_b \in (b_{z-1}, b_z]. \quad (3)$$

Notice that the only term that depends on r_b in the above equation is the quality gap $\Delta(r_b)$. Thus, to maximize quality, we need to minimize $\Delta(r_b)$. Recall that in Section III we argued that $\Delta(r_b)$ is non-increasing function of r_b, we validate this argument in Section VI. Since $\Delta(r_b)$ is non-increasing in the interval $(b_{z-1}, b_z]$, no point in that interval could make the quality gap smaller than $\Delta(r_b = b_z)$. Thus, an optimal solution for r_b occurs at b_z. ∎

The above theorem tells us that to find an optimal base layer rate r_b^*, it suffices to check only the rates b_c, where $c = 1, 2, \ldots, C$. A straightforward approach to implement this lemma is to compute (3) at $c = 1, 2, \ldots, C$ and choose the rate that corresponds to the maximum quality. This would require computing the summation at every iteration, which makes the time complexity of the algorithm $O(C^2)$. A better approach is to iteratively compute each term from $c = C$ towards $c = 1$, and every iteration only adds the difference in quality to the quality computed in the previous iteration. The difference d in quality between class c and $c+1$ is given by the following equation:

$$d = [q_{ns}\,(b_c)) - \Delta(b_c)]\, f_c - \sum_{i=c+1}^{C} f_i\, [\Delta(b_c) - \Delta(b_{c+1})]\,.$$

The first term represents the quality improvement because clients with bandwidth b_c is capable to receive coded streams. The second term represents the quality degradation of all clients that have bandwidth larger than b_c because of a larger coding efficiency gap.

FGSOPT

```
1.  q_1 = q_2 = ··· = q_C = 0;
2.  q_max = q_C = [q_ns(b_C) − Δ(b_C)] f_C;
3.  r_b^* = b_C;
4.  f_rcv = f_C;
5.  for c = C − 1 to 1
6.      d = [q_ns(b_c) − Δ(b_c)] f_c − f_rcv[Δ(b_c) − Δ(b_{c+1})];
7.      q_c = q_{c+1} + d;
8.      if q_c > q_max
9.          q_max = q_c;
10.         r_b^* = b_c;
11.     f_rcv = f_rcv + f_c;
12.  return r_b^*;
```

Fig. 4. Efficient algorithm to compute the optimal FGS base layer rate for a video sequence.

Using the above idea and Theorem 1, we propose an efficient algorithm for our single sequence formulation, called FGSOPT, that computes an optimal value for the base layer rate. The pseudo code of the algorithm is given in Fig. 4. The inputs to the algorithm are: 1) a probability mass function f_c that describes the bandwidth distribution of different client classes; 2) a rate-distortion function $q_{ns}(r)$ that yields the expected quality when decoding the nonscalable video stream at rate r; and 3) a quality gap function $\Delta(r_b)$ that describes the reduction in quality if the video stream were to be encoded in FGS manner with base layer rate r_b. In Sections VI-E and VI-F, we discuss how rate-distortion and quality gap functions can be estimated. The output of the algorithm is the optimal base layer rate. The time complexity of the algorithm is clearly $O(C)$.

V. Multiple-Sequence Formulation

In this section, we extend our formulation to structure multiple FGS video sequences. This is a general optimization problem applicable to servers that are concurrently streaming multiple FGS sequences to diverse client communities. We first present the formulation of the problem and show that it is, unfortunately, NP-complete. Then, we present a branch-and-bound algorithm that finds the optimal solution. This algorithm is fast in many typical cases, but its worst-case time complexity is exponential. Thus, it is not possible to use it in dynamic real-time systems. Rather, it could be used for off-line cases in which the server has estimates on future client distributions and can therefore produce optimal FGS streams for them apriori. For dynamic cases, we propose a heuristic algorithm that runs significantly faster than the branch-and-bound algorithm and produces near-optimal results.

A. Problem Formulation and Hardness

We consider a streaming server with a given network bandwidth B_{max}. The server has S sequences to encode and serve to diverse client communities. The sequences are assumed to have different popularities. Our objective is to determine the base layer rate of each video sequence such that the server maximally utilizes its capacity and achieves the best perceived

quality for all clients receiving the video sequences. To formulate this problem, we generalize the notations used in the single sequence problem by using a superscript to refer to the sequence number.

Let N^s be the number of clients that receive sequence s, where $s = 1, 2, \ldots, S$[3]. We model the heterogeneous client population by dividing all N^s clients into C^s classes. All clients in the same class c allocate the same bandwidth b_c^s to receive sequence s, where $c = 1, 2, \ldots, C^s$. Without loss of generality, we assume $b_1^s < b_2^s < \cdots < b_{C^s}^s$ for all s. The clients of each sequence s are distributed over C^s classes according to the probability mass function f_c^s, where $\sum_{c=1}^{C^s} f_c^s = 1$. Let r_{max}^s be the maximum bit rate of sequence s. For any sequence s, the base layer rate r_b^s must be no larger than r_{max}^s. As in the single-sequence case, we assume that $b_{C^s} \leq r_{max}^s$ without loss of generality.

We denote the stream structuring policy for all sequences by the vector $\mathbf{r}_b$, where $\mathbf{r}_b = \{r_b^s, 1 \leq s \leq S\}$. Clearly, we have $0 \leq \mathbf{r}_b \leq \mathbf{r}_{max}$, where $\mathbf{r}_{max} = \{r_{max}^s, 1 \leq s \leq S\}$. The multiple sequence problem that maximizes the average perceived quality for all clients over all sequences can be written as

$$\max_{\mathbf{r}_b} \quad \sum_{s=1}^{S} \left\{ N^s \sum_{c=1}^{C^s} q_c^s f_c^s \right\} \tag{4a}$$

$$\text{s.t.} \quad 0 \leq \mathbf{r}_b \leq \mathbf{r}_{max}; \tag{4b}$$

$$\sum_{s=1}^{S} \left\{ N^s \sum_{c=1}^{C^s} r_c^s f_c^s \right\} \leq B_{max}. \tag{4c}$$

In the above formulation, r_c^s represents the decoding rate of sequence s for clients in class c, and q_c^s denotes the quality for sequence s achieved by clients in class c. Since clients in classes with bandwidth less than r_b^s cannot even receive base layer, We have

$$r_c^s = \begin{cases} 0, & b_c^s < r_b^s \\ b_c^s, & b_c^s \geq r_b^s \end{cases} \tag{5}$$

and, consequently

$$q_c^s = \begin{cases} 0, & b_c^s < r_b^s \\ q_{ns}^s(b_c^s) - \Delta^s(r_b^s) & b_c^s \geq r_b^s \end{cases} \tag{6}$$

where $q_{ns}^s(b_c^s)$ is the quality achieved by coding sequence s with nonscalable coders at rate b_c^s, and $\Delta^s(r_b^s)$ is the quality gap function between the FGS and nonscalable streams for sequence s that is FGS-coded with base layer rate r_b^s.

The following theorem shows that the multiple-sequence problem in (4) is an NP-complete problem. The proof idea is to reduce a well-known NP-complete problem, multiple-choice knapsack problem (MCKP) [17, Ch. 11], to our problem. Details are given in the technical report [18] due to space limitations.

Theorem 2: Determining the base layer rates of multiple FGS sequences concurrently streamed by a server with limited bandwidth to maximize the average perceived quality for all clients over all sequences is an NP-complete problem.

[3]We should note that, in a more general sense, N^s can be seen as the importance the sequence s. This allows administrators to gauge the relative bandwidth allocation among all sequences.

MFGSOPT

```
1.  Compute y_c^s and w_c^s using (7) and (8), for all c and s;
2.  <r_s, Y> = init_assign(); // initial solution
3.  Construct subtree a_0; Initialize A = {a_0}; // A: a FIFO
4.  repeat until A = ∅
5.      a = removeHead(A);
6.      for c = 1 to C^{a.s}
7.          <Δy, Δw, r̂_{a.s+1}, r̂_{a.s+2}, ..., r̂_S> =
7.              BOUND(a, w_c^{a.s});
8.          if Δy + y_c^{a.s} + a.y ≤ Y // worse assignment
9.              // cut this branch
10.         elseif Δw ≤ a.B − w_c^{a.s} // better and feasible
11.             // update best known assignment so far
12.             Y = Δy + y_c^{a.s} + a.y;
13.             <r_1, r_2, ..., r_S> =
13.                 <a.r_1, a.r_2, ..., a.r_{a.s−1}>|
13.                 < b_c^{a.s}>|<r̂_{a.s+1}, r̂_{a.s+2}, ..., r̂_S>;
14              // cut this branch
15.         else // better but infeasible
16.             Create a new node a' based on a and c;
17.             addTail(A, a');
18. return <r_s, Y>;
BOUND(a, w)
1.  if a.s + 1 > S return <0, 0, 0>; // leaf node
2.  for s = a.s + 1 to S
3.      m^s = C^s; // initial value
4.      for c = C^s − 1 to 1
5.          if y_c^s > y_{m^s}^s and w_c^s ≤ a.B − w
6.              m^s = c; // b_{m^s}^s is the best rate
7.  r_s = b_{m^s}^s ∀s = a.s + 1, a.s + 2, ..., S;
8.  Δy = Σ_{s=a.s+1}^{S} y_{m^s}^s;   Δw = Σ_{s=a.s+1}^{S} w_{m^s}^s;
9.  return <Δy, Δw, r_s>;
```

Fig. 5. Branch-and-bound algorithm to compute the optimal base layer rates for multiple video sequences.

B. Optimal, Branch-and-Bound, Algorithm

We propose a branch-and-bound algorithm that performs a breadth first search on an incrementally constructed tree to find optimal base layer rates for the considered S sequences. The tree has S levels, each corresponds to a sequence. In level s, we create a node for each possible base layer rate assignment r_b^s for sequence s. An example tree is given in Fig. 6. The number of possible r_b^s is finite since Theorem 1 tells us that it is sufficient to check rates b_c^s, where $c = 1, 2, \ldots, C^s$ for an optimal base layer rate that maximizes average quality for sequence s. Clearly, searching the entire tree, even fully constructing it, takes exponential number of steps. Our algorithm tries to *cut*, or more accurately not to create, as many branches of the tree as possible without sacrificing the optimal solution. This is achieved using the BOUND function described below.

To search for the optimal base layer rates, we define two variables: 1) y_c^s is the average quality improvement contributed by sequence s, when the base layer is coded at rate b_c^s and 2) w_c^s is the server bandwidth consumption of all clients of sequence s, when the base layer is coded at rate b_c^s. Using (4), (5), and (6), these two variables can be computed as

$$y_c^s = \sum_{j=c}^{C^s} \left\{ N^s \left[q_{ns}^s\left(b_j^s\right) - \Delta^s\left(b_c^s\right) \right] f_j^s \right\} \tag{7}$$

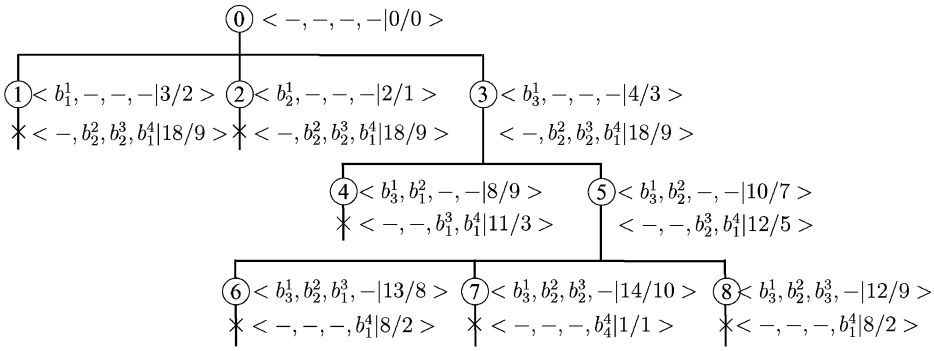


Fig. 6. Finding the optimal structuring of four sequences using our branch-and-bound algorithm. Using our BOUND function, the search space is significantly reduced without sacrificing the optimal solution.

and

$$w_c^s = \sum_{j=c}^{C^s} \{N^s b_j^s f_j^s\}. \tag{8}$$

Now, the goal of the algorithm is to maximize the average quality of all sequences: $Y = \sum_{s=1}^{S} y_c^s$ without consuming more than the server bandwidth: $W = \sum_{s=1}^{S} w_c^s \leq B_{max}$. This is achieved by: 1) traversing through the tree and memorizing the best known average quality so far and 2) using a bounding function to determine whether a subsequent branching may lead to a better assignment without fully expanding the branch.

The idea of the BOUND function is to relax the constraint on the server bandwidth to compute an *upper bound* on the achievable quality from a branch. If this upper bound is smaller than the current best-known solution, this branch is cut. Computing the upper bound is not costly, because the interaction among different sequences (hence the combinatorial explosion) is avoided. The pseudocode of the BOUND function is given in Fig. 5. The BOUND function is called before expanding any branch. The function is passed the maximum allowable bandwidth consumption w for all sequences in that branch. We allow each sequence to consume bandwidth as much as w. By doing so, we search for the maximum quality contribution from each sequence without worrying about other sequences (the for-loop in lines 4–6). This is clearly an upper bound on the quality which may or may not be feasible to achieve, but it can be used to judge whether the branch is worth full expansion. The BOUND function returns the upper bound on the quality from the branch, as well as the associated bandwidth consumption and the base layer rates of all sequences in the branch.

As described in Fig. 5, the MFGSOPT algorithm uses the return values from the BOUND function as follows. First, if the upper bound is worse than the best known solution so far, we cut this branch (lines 8–9). Second, if the upper bound is better then the best known solution *and* the total bandwidth consumption of all sequences happens to be less than the server bandwidth, we take the upper bound from this branch as the best known solution (lines 10–14). We also cut this branch because we have already found the best solution in it that branch which happens to conform to the original bandwidth constraint even though we relaxed this constraint during the computation of this solution. Third, if the upper bound is better than the best known solution so far but it requires more bandwidth than the server bandwidth, we need to expand this branch (lines 15–17).

TABLE I
AVERAGE QUALITY IMPROVEMENT y_c^s AND CONSUMED BANDWIDTH w_c^s OF THE ILLUSTRATIVE EXAMPLE

	client class					client class			
y_c^s	1	2	3	4	w_c^s	1	2	3	4
1	3	2	4		1	2	1	3	
2	4	6			2	6	4		
3	3	4	2		3	1	3	2	
4	8	3	4	1	4	2	4	6	1

Finally, the branch-and-bound algorithm needs an initial assignment of the base layer rates. This solution can be set arbitrarily. We use our heuristic algorithm (described in Section V-C) to start the branch-and-bound algorithm to begin with an informed guess. This significantly reduces the running times in typical cases.

The BOUND subroutine runs in polynomial time complexity $O(CS)$, where $C = \max_{1 \leq s \leq S} \{C^s\}$ corresponds to the maximum number of user classes among all client sequences and S is the number of sequences. This is because the for-loops start from line 2 and line 4 iterate through at most S and C values, respectively. The worst-case time complexity of the MFGSOPT algorithm is $O(C^{S+2}S)$. This is because we have at most $O(C^S)$ nodes to be considered, and each node calls the BOUND subroutine $O(C)$ times (the for-loop between lines 6–17). Given that BOUND subroutine runs in time $O(CS)$, the running time of MFGSOPT algorithm is at most $O(C^S CCS) = O(C^{S+2}S)$.

Illustrative Example: We describe the idea of the MFGSOPT algorithm using a simple example, which is illustrated in Fig. 6. We assume there are four sequences, each of them has 3, 2, 3, and 4 client classes, respectively. The total server bandwidth is given as 11 units. For clarity of the example, we assume that the quality function $q_{ns}^s(b_c^s)$, quality gap function $\Delta^s(r_b^s)$, and the client bandwidth distribution function f_c^s are given, such that the computed variables y_c^s and w_c^s (using (7) and (8)) are as shown in Table I. The initial best known quality is set to zero at the root node as we have no knowledge on the optimal base layer rate assignment. We denote this unknown assignment as $\langle -, -, -, -|0/0\rangle$, where the first four elements represent base layer rates of the four sequences in order, and the last element, 0/0, denotes total quality over total bandwidth of

this assignment. We create three child nodes (nodes are labeled with the order of creation), where each node represents possible a base layer rate for sequence 1. For example, node 1 represents coding sequence 1 at base layer rate b_1^1 where rates for sequences 2, 3, and 4 are not determined yet. Using Table I, we know that sequence 1 consumes bandwidth $w_1^1 = 2$ and produces average quality improvement $y_1^1 = 3$. Before expanding the branch under node 1, we use the BOUND function to find the upper bound on the quality from that branch, which is 18. The BOUND function also returns the associated bandwidth consumption as 9, and the base layer rates of sequences 2, 3, 4 as b_2^2, b_2^3, and b_1^4. Notice that, the BOUND function provides a *feasible* and *better* assignment as the total bandwidth consumption is $9+2 \leq 11$ and average quality is $18+3 > 0$. Hence, we memorize this rate assignment and update the best known average quality to 21. Most importantly, the BOUND function finds an optimal assignment for sequences 2, 3, and 4 without even creating any of the 32 nodes below node 1.

We then check the branch under node 2. The BOUND function tells us that the upper bound on the quality is $18 + 2 = 20$. We cut this branch as well, because 20 is smaller than the best known quality. We next check node 3. This time, the BOUND function returns an upper bound on the quality as $18 + 4 > 21$ that consumes bandwidth $9 + 3 > 11$. Notice that this assignment, while produces higher quality than the best known solution, is not a feasible one as it requires more bandwidth than the server bandwidth. Therefore, we need to expand this branch. We create child nodes 4 and 5 for node 3. We repeatedly use the BOUND function to reduce the search space until all branches are either investigated or cut. Memorizing best known rate assignment and using the BOUND function allow us to traverse through only eight nodes yet cover the complete search space, which has 72 leaf nodes. We find the optimal assignment as $\langle b_1^1, b_2^2, b_2^3, b_1^4 | 21/11 \rangle$.

C. Efficient Heuristic Algorithm

We propose a heuristic algorithm to solve the multiple-sequence stream structuring problem. Our experimental results (in Section VI) show that it produces near-optimal solutions, and runs much faster than the branch-and-bound algorithm. The core idea of this algorithm is to incrementally allocate more bandwidth to sequences that are expected to increase the total quality by higher margins for each bandwidth unit consumed from the server bandwidth.

The pseudocode of the algorithm is given in Fig. 7. The algorithm maintains a two-dimensional array $\mathbf{t}$, where the s row corresponds to sequence s. There are C^s columns for each s row. Each element t_c^s of the array is a triplet of the form $\langle y_c^s, w_c^s, b_c^s \rangle$, where y_c^s is the average quality improvement given by (7), w_c^s is the bandwidth consumption given by (8), when the base layer of sequence s is encoded at rate b_c^s. The algorithm sorts each row based on y_c^s in increasing order (line 2). After sorting, the algorithm removes all array entries that would clearly produce inferior solutions (line 3). This is the case when $w_{c+1}^s \leq w_c^s$ because y_{c+1}^s is greater then y_c^s, i.e., the element t_c^s uses more bandwidth than t_{c+1}^s yet contributes smaller value to the overall quality. After removing these inferior entries, each row of the array $\mathbf{t}$ is sorted in increasing y and w.

MFGS

1. Compute array $\mathbf{t}$ using Eqs. (7) and (8).
2. Sort each row of $\mathbf{t}$ in ascending order based on y_c^s.
3. Remove all t_c^s, where $w_{c+1}^s \leq w_c^s$.
4. $B_r = B_{max}$; $k^s = 0$, for all s;
5. **while** (True)
6. $\quad \hat{s} = 0$;
7. $\quad$ **for** $s = 1$ to S
8. $\quad\quad$ Compute $l_c^s(B_r)$ using Eq. (9).
9. $\quad\quad$ **if** $k^s < C^s$ and $w_{k^s+1}^s - w_{k^s}^s \leq B_r$ and
9. $\quad\quad$ $\left(\hat{s} = 0 \text{ or } l_{k^s+1}^s > l_{k^{\hat{s}}+1}^{\hat{s}}\right)$
10. $\quad\quad\quad$ $\hat{s} = s$;
11. $\quad$ **if** $\hat{s} = 0$ **return** $b_{k^s}^s$, where $s = 1, 2, \ldots, S$;
12. $\quad$ $B_r = B_r - \left(w_{k^{\hat{s}}+1}^{\hat{s}} - w_{k^{\hat{s}}}^{\hat{s}}\right)$;
13. $\quad$ $k^{\hat{s}}$ ++ ;

Fig. 7. Effective heuristic algorithm to compute base layer rates for multiple video sequences.

TABLE II
UPDATED ARRAY $\mathbf{t}$ AFTER SORTING AND REMOVING ENTRIES THAT WOULD CLEARLY YIELD INFERIOR SOLUTIONS

t_c^s	0	1	2	3	4
1	0	$< 2, 1, b_2^1 >$	$< 3, 2, b_1^1 >$	$< 4, 3, b_3^1 >$	
2	0	~~$< 4, 6, b_1^2 >$~~	$< 6, 4, b_2^2 >$		
3	0	~~$< 2, 2, b_3^3 >$~~	$< 3, 1, b_1^3 >$	$< 4, 3, b_2^3 >$	
4	0	$< 1, 1, b_4^4 >$	~~$< 3, 4, b_2^4 >$~~	~~$< 4, 6, b_3^4 >$~~	$< 8, 2, b_1^4 >$

The while loop (lines 5–13) repeatedly allocates more bandwidth to the sequence with the highest quality improvement per bandwidth usage l_c^s, which is computed as

$$l_c^s(B_r) = \begin{cases} 0, & \text{if } w_c^s - w_{c-1}^s > B_r \\ \frac{y_h^s - y_{c-1}^s}{w_h^s - w_{c-1}^s}, & \text{otherwise} \end{cases} \quad (9)$$

where B_r is the remaining (i.e., unallocated yet) server bandwidth, and h is the largest integer such that $w_h^s - w_{c-1}^s \leq B_r$. Let k^s be the index of the current base layer rate for sequence s, i.e., the base layer rate of s is $b_{k^s}^s$. We start from $k^s = 0$ for all sequences (line 4), and search for the sequence $\hat{s}$ with the highest $l_{k^s}^s(B_r)$ value among all sequences. We allocate more bandwidth to that sequence $\hat{s}$, i.e., we encode sequence $\hat{s}$ at base layer rate $b_{k^{\hat{s}}+1}^{\hat{s}}$ rather than $b_{k^{\hat{s}}}^{\hat{s}}$ (line 13). We update the remaining bandwidth (line 12), recompute $l_{k^s}^s(B_r)$ for all sequences (line 8), and find the sequence that leads to the highest overall quality margin again. We allocate more bandwidth to that sequence. We stop once B_r is not sufficient to further increase the average quality (line 11).

The time complexity of MFGS algorithm is $O(C^2 S)$, where $C = \max_{1 \leq s \leq S} \{C^s\}$ is the number of client classes and S is the number of sequences. This is because it takes: 1) $O(C^2 S)$ steps to compute y_c^s and w_c^s as indicated by (7) and (8); 2) $O(C \log CS)$ to sort rows of $\mathbf{t}$ and remove inferior entries of $\mathbf{t}$; and 3) $O(CL)$ iterations to find the best $\hat{s}$.

Illustrative Example: We use the illustrative example developed in Section V-B to explain the operation of MFGS algorithm. The variables y_c^s and w_c^s are shown in Table I. The algorithm first sorts each row of the array $\mathbf{t}$ and removes inferior array entries as described above, which leads to the updated array shown in Table II, where removed entries are crossed out.

For example, t_1^2 can be removed because coding sequence 2 with b_2^2 leads to quality improvement 6 with bandwidth consumption 4, which is clearly better than encoding that sequence with b_1^2 that results in quality improvement 4 with bandwidth consumption 6.

Initially, all sequences are allocated zero bandwidth, and the remaining bandwidth is $B_r = 11$. The potential quality improvement per bandwidth unit usage can be computed using (9) as: 2/1, 6/4, 3/1, and 1/1 for sequences 1, 2, 3, and 4, respectively. We set $k^3 = 1$ as sequence 3 leads to the highest l_c^s value, thus it is expected to increase the total quality by the highest margin. Accordingly, we update the remaining bandwidth to be 10. We then recompute $l_c^s(B_r)$ and get 2/1, 6/4, 2/2, and 1/1. Since sequence 1 leads to the highest l_c^s value, we assign more bandwidth to it by letting $k^1 = 1$. We repeat this iteration seven times until the remaining bandwidth reaches 0, when $k^1 = 3$, $k^2 = 1$, $k^3 = 1$, and $k^4 = 2$. The correspondent base layer rate assignment is $\langle b_3^1, b_2^2, b_1^3, b_1^1 \rangle$, and the average quality is 21. In this example, MFGS algorithm finds an optimal solution, which is shown to be 21 by MFGSOPT algorithm in Section V-B.

D. Discussion

Our MFGSOPT algorithm targets video on-demand systems, where a video sequence is expected to be streamed to many clients over an extended period of time. Therefore, the cost of computing or estimating the inputs of the algorithm is justified by the quality improvement observed by the clients. Moreover, this cost is controllable: For videos with expected low demand, the administrator can quickly compute rough estimates of the inputs, while for popular videos more elaborate estimations can be done. One of the following three methods can be used to control the computational time. First, the quality gap $\Delta^s(r_b^s)$ can be roughly estimated by a line with negative slope that requires only two sample points. As we show in Section VI, a fourth order polynomial function is a better representation of $\Delta^s(r_b^s)$, but would require more sample points and higher computing power. Second, the rate-distortion function can be estimated by sophisticated analytic methods [19] or by simple curve fitting to a few empirical samples. In Section VI, we show that a quadratic function fairly accurately models the relationship between the rate and expected distortion. Third, the administrator may adopt various granularities of bandwidth estimations to gauge the maximum number of client classes among all video sequences (C). Choosing a coarse-grained bandwidth estimation method imposes lower network and computational overhead. This also leads to a smaller C value and thus reduces the running-time of our MFGSOPT algorithm.

Our MFGS algorithm targets on-line applications as it runs much faster than optimal algorithms that have exponential running time. We will verify the running time of MFGS algorithm in the evaluation section. Like MFGSOPT algorithm, the computational cost of MFGS algorithm is controllable through the choices of its inputs. As a final note, we employ MFGS to determine an initial assignment for MFGSOPT algorithm presented in the previous subsection.

VI. Evaluation

In this section, we rigorously evaluate our proposed algorithms. We start by describing our experimental setup. Then we demonstrate the potential quality improvement resulted by our algorithms. Then, we show that our heuristic algorithm for the multiple-sequence problem produces near-optimal results and runs substantially faster than our branch-and-bound algorithm. We also show that our heuristic algorithm scales with number of sequences and it terminates in a fraction of a second for systems with more than 30 sequences. Finally, we experimentally validate the non-increasing property of the quality gap function $\Delta(r_b)$ assumed by our algorithms, and we show that a simple quadratic function is a good approximation for rate-distortion models used in our algorithms.

A. Setup

Software: In our experiments, we augment and use the reference software of the Joint Scalable Video Model (JSVM) [16]. The reference software includes an implementation of the scalable video coding (SVC) extension to the H.264 standard [20]. The details of SVC can be found in [21], [22]. The JSVM reference software is implemented in C++ and contains several executables. We use the following executables: *H264AVCEncoderLibTest*, *BitStreamExtractor*, *H264AVCDecoderLibTest*, and *FixedQPEncoder*. The *H264AVCEncoderLibTest* is a configurable SVC encoder that can compress a raw video file into a global stream. This global stream consists of several embedded substreams, which deliver lower quality video representations at lower rates. The global stream is stored as a file. The *BitStreamExtractor* tool extracts a user-specified substream from an existing global stream and stores it in a new file. Further stream extractions from this substream file are possible as the syntax and semantics of the global stream and substream files are identical. The *H264AVCDecoderLibTest* is an SVC decoder that decompresses coded stream into a raw video file.

Since the *H264AVCEncoderLibTest* does not implement rate control algorithm for a user-specified rate constraint, we have to use quantization parameter (QP) to gauge the resulted stream rate. The *FixedQPEncoder* is a tool that searches the proper QPs to satisfy rate constraints. It iteratively calls *H264AVCEncoderLibTest* with estimated QP values, and stops when the resulted stream rate is within an acceptable range of the desired rate.

In addition, we have implemented the three algorithms proposed in this paper: FGSOPT, MFGS, and MFGSOPT. All algorithms are implemented in Java and the code is available from the authors.

Video sequences: We consider diverse video sequences. We choose five standard video sequences for our experiments: City, Mobile, Soccer, Harbour and Crew. The first four are in CIF format and the fifth is in 4CIF. We encode these sequences with the widely adopted IBBBPBBBP group of picture (GoP) structure at 30 frames per second. We first encode a sequence with single layer configuration using the *FixedQPEncoder* tool to get appropriate QP values for the target base layer rate. We then use the same QP values to code an FGS stream. We instrument the reference software to extract the rate-distortion characteristics of the nonscalable and FGS streams.

Clients: We consider large number of clients with network bandwidth distributed according to five representative distributions. The first is a normal distribution with mean at 1000 kbps and standard deviation of 100 kbps. The second is a bimodal

TABLE III
STREAMING SCENARIOS USED IN THE EXPERIMENTS

#	Sequence(s)	Clients	# of Clients
I	⟨Mobile, City⟩	⟨Normal, Bi-modal (right)⟩	⟨100, 200⟩
II	⟨Mobile, Soccer⟩	⟨Normal, Bi-modal (left)⟩	⟨100, 200⟩
III	⟨Mobile, Harbour⟩	⟨Normal, Multi-modal⟩	⟨100, 300⟩
IV	⟨City, Soccer⟩	⟨Bi-modal (right), Bi-modal (left)⟩	⟨200, 200⟩
V	⟨City, Harbour⟩	⟨Bi-modal (right), Multi-modal⟩	⟨200, 300⟩
VI	⟨Soccer, Harbour⟩	⟨Bi-modal (left), Multi-modal⟩	⟨200, 300⟩
VII	⟨Mobile, City, Soccer, Harbour⟩	⟨Normal, Bi-modal (right), Bi-modal (left), Multi-modal⟩	⟨100, 200, 200, 300⟩
VIII	⟨Mobile⟩	⟨Uniform⟩	⟨100⟩

distribution that consists of two normally-distributed peaks with means at 250 and 1000 kbps, and standard deviations of 25 and 100. This bimodal distribution is skewed to the right: 80% of client classes are from the normal distribution with mean 1000 kbps. The third is a bi-modal distribution with the same setting, except that it is skewed to the left: 80% of client classes are from the normal distribution with mean 250 kbps. The fourth is a multi-modal distribution with three normal distributions, which represents a typical client distribution in today's Internet: 50% of clients are equipped with dial-up connections, which have a normal distribution with mean 40 kbps and standard deviation of 25 kbps; 35% of clients use DSL services, where the average bandwidth is 1000 kbps with standard deviation of 100 kbps; and 15% of clients have high-speed connections with average bandwidth 2000 kbps and standard deviation of 200 kbps. The last distribution is uniform between 35 and 3005 kbps. In all these client bandwidth distributions, we consider the client bandwidth estimation accuracy is 5 kbps.

Streaming Scenario: We define eight representative streaming scenarios as illustrated by Table III. These streaming scenarios have different R-D characteristics, which are captured by classifying them into several categories of video sequences. For example, a sports event sequence would have similar R-D characteristics to the Soccer sequence. Streaming scenarios also have diverse groups of viewers and viewing popularity, which are captured by client bandwidth distributions and number of clients. We use these scenarios in our experiments to evaluate our algorithms.

B. Average Quality Improvement

As mentioned in the introduction section, we are not aware of similar algorithms in the literature that optimize the average quality by controlling the base layer rate. Therefore, we compare the results of our algorithm to the results of heuristic methods. That is, we choose two *reasonable* rates for the base layer and compare the resulting quality against the quality produced by our algorithms. Indeed, there are too many other choices and we cannot cover all of them in our experiments. This is not really an issue because our algorithms are provably optimal, and the best that heuristic methods can do is to approach our algorithms by trial and error.

We first evaluate the quality improvement achieved by our single sequence algorithm, which is denoted by FGSOPT in the plots. We present a sample result with streaming scenario VIII defined in Table III, other scenarios yield similar results. We run

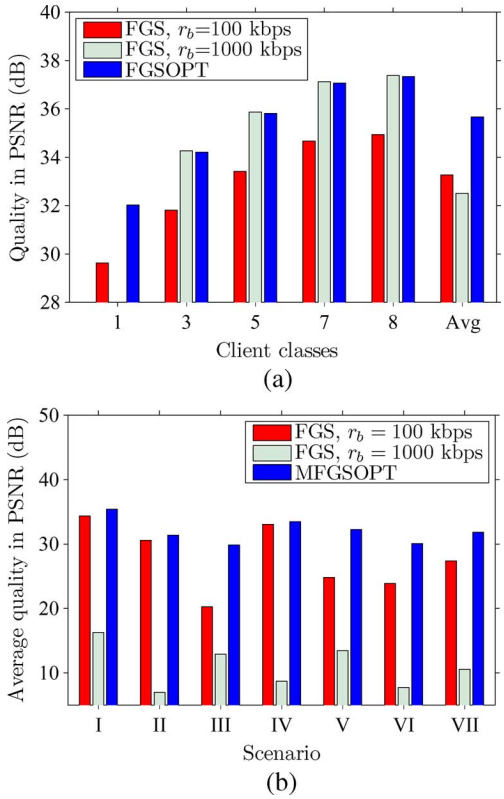


Fig. 8. Potential quality improvement using our streaming structuring algorithms in (a) single-sequence systems and (b) multiple-sequence systems.

the FGSOPT algorithm to compute the optimal base layer rate. We choose two base layer rates for comparison: 100 and 1000 kbps. We compute the perceived quality for each client class and the average quality over all classes. The result is shown in Fig. 8(a). The figure clearly shows that the average quality over all classes has been improved using our FGSOPT algorithm. Across all client classes, the average quality improvement is more than 2 dB.

Next, we evaluate the quality improvement achieved by our multiple sequence algorithm, which is denoted by MFGSOPT in the plots. We consider seven streaming scenarios, where scenarios I through VI consists of two simultaneous streaming sequences and scenario VII consists of four. In this experiment, we assume the server bandwidth is not the bottleneck. We run the MFGSOPT algorithm to compute the optimal base layer rate for each sequence. We also choose two base layer rates 100 and 1000 kbps for comparison. We compute the average quality over all clients across sequences. As shown by Fig. 8(b), a significant quality improvement can be achieved using our MFGSOPT algorithm. For example, in scenarios III, our algorithm produces up to 10 dB improvement in quality. This reveals that intelligent-chosen base layer rates can greatly improve average quality for all clients across all sequences.

C. Performance of the Heuristic MFGS Algorithm

We compare the achieved average quality of our heuristic algorithm, denoted as MFGS in the plots, against the achieved average quality of our branch-and-bound algorithm. We first assume that server bandwidth is not the bottleneck. We compute

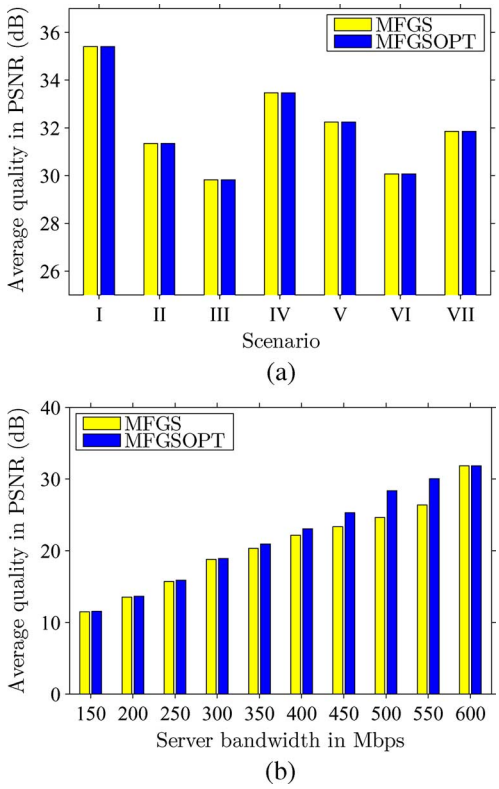


Fig. 9. Our heuristic MFGS algorithm produces near-optimal average quality when compared against our branch-and-bound MFGSOPT algorithm under: (a) various streaming scenarios and (b) different server bandwidth.

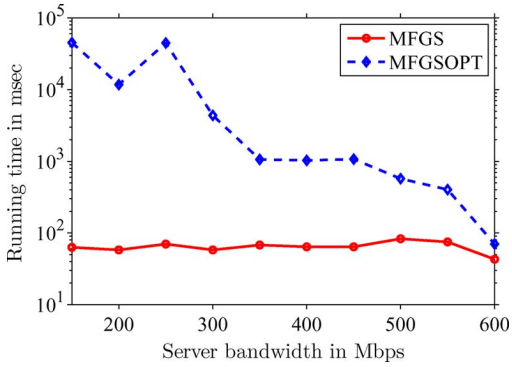


Fig. 10. Running times of the heuristic MFGS algorithm versus the branch-and-bound MFGSOPT algorithm.

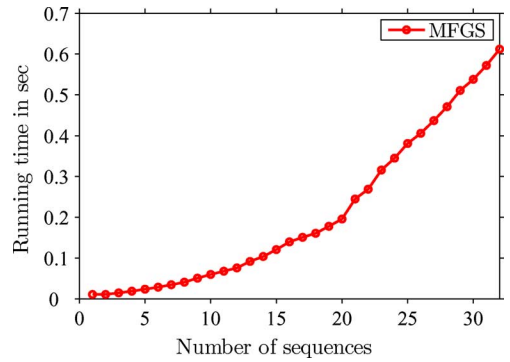


Fig. 11. Scalability of the heuristic MFGS algorithm as the number of sequences increases.

the base layer rates for all sequences using MFGS, and compute the average quality for all clients across all sequences. We repeat the experiment using the MFGSOPT algorithm. Fig. 9(a) compares the results of both algorithms. The figure shows that our heuristic algorithm produces base layer rates that lead to almost-optimal average quality.

Second, we consider scenarios where server bandwidth is the bottleneck. We employ streaming scenario VII defined above. We vary server bandwidth B_{max} from 150 Mbps to 600 Mbps at a 50 Mbps step. We run both MFGS and MFGSOPT algorithms, and compare their average quality over all clients across all sequences. We plot the results in Fig. 9(b). The figure again verifies that our heuristic algorithm yields near-optimal results.

D. Time Complexity and Scalability

We study the running times of the MFGS and MFGSOPT algorithms in different setting. We first choose streaming scenario VII, which consists of four streaming sequences. We vary the server bandwidth B_{max} from 150 to 600 Mbps at a 50 Mbps step. We run both algorithms and measure the running times on a machine with 2.66-GHz processor and 2-GB memory running Linux. Fig. 10 presents the results, which show that MFGS algorithm always terminates in less than 0.2 s, while MFGSOPT takes up to 45 s to complete. We notice that MFGSOPT runs significantly faster with higher server bandwidth. This is because with higher server bandwidth, the rate assignments returned by BOUND subroutine are likely to be feasible. This enables MFGSOPT to locate an optimal solution without inspecting a subtree, and thus reduces its running time.

We next investigate the scalability of the heuristic MFGS algorithm as the number of sequences increases. We generate 32 normally-distributed client distributions with random mean values between 50 and 2500 kbps and standard deviation of 50 kbps. We assume each client distribution has ten clients. We then assign each of these client distributions to a streaming sequence with R-D characteristic randomly chosen from the following four sequences: Mobile, City, Soccer, and Harbour. We run MFGS with different numbers of sequences: from 1 to 32. Fig. 11 shows the results. We can see that our MFGS algorithm solves a multiple sequence problem with 32 sequences in about 0.6 s. We also observe that the running time of MFGS is roughly linear in the number of sequences. For example, it terminates in about 0.2, 0.35, and 0.5 s with 20, 25, and 30 sequences, respectively. This indicates that our algorithm is scalable with respect to number of sequences.

E. Quality Gap Function

The FGSOPT and MFGSOPT algorithms assume that the quality gap $\Delta(r_b)$ is a non-increasing function of base layer rate r_b. To validate the accuracy of this assumption, we compute the quality gap at various base layer rates. We use the reference software to encode the test sequences with base layer rates between 100 and 3000 kbps with an increment of 250 kbps. Each base layer rate results in a unique FGS coded stream that supports decoding rates between r_b and 3000 kbps. To quantify the coding efficiency gap at a specific base layer rate r_b, we decode the stream at many decoding rates between r_b and 3000 kbps

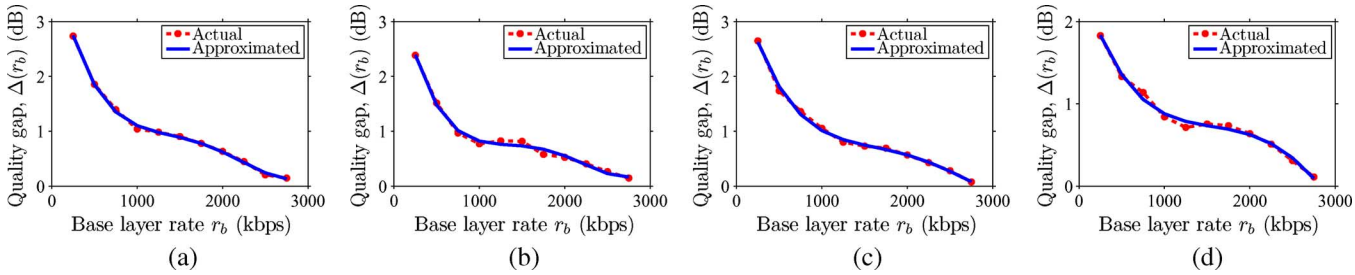


Fig. 12. Coding efficiency gap $\Delta(r_b)$ between FGS and nonscalable streams. The figure shows that $\Delta(r_b)$ is nonincreasing function, and it can be modeled by a fourth-order polynomial function. (a) Mobile. (b) City. (c) Soccer. (d) Harbour.

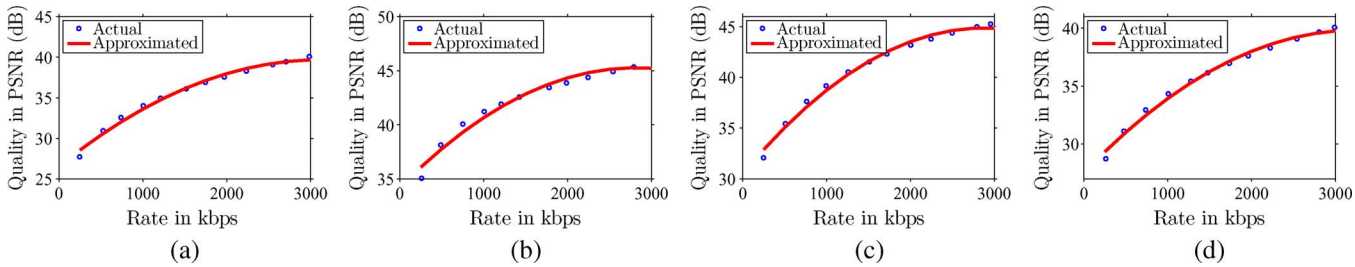


Fig. 13. Rate-distortion (R-D) function $q_{ns}(r)$ of nonscalable streams. The figure shows that a quadratic function provides a good approximation for the R-D function. (a) Mobile. (b) City. (c) Soccer. (d) Harbour.

and we take the average over all of them. We compute the reconstructed quality at each decoding rate by first extracting the substream that matches this rate. Then we decode the extracted substream and compare it against the original video stream. The results shown in Fig. 12 confirm the nonincreasing property of the quality gap function. On the same figure we plot a fourth order polynomial function that best fits the quality gap curve. We do this because our algorithms need to computes the quality gap $\Delta(r_b)$ at different base layer rates. Thus instead of empirically measuring the quality gap at too many base layer rates, which is computationally expensive, we estimate the polynomial function and employ it in the algorithms. Estimating the polynomial function requires measuring the quality gap only at a few base layer rates.

F. Rate-Distortion Function

Our streaming structuring algorithms require a rate-distortion (R-D) function that estimates the expected distortion at a given decoding rate when the stream is encoded in a nonscalable manner. Through extensive experiments, we have found that this R-D function can be approximated by a simple quadratic function. Fig. 13 shows some of the results, where we compute the R-D function at 12 sampling bit rates for four sequences. The figure also shows the best-fit quadratic function produced by the Matlab curve-fitting tool for the sample points. We note that the results in Fig. 13 provide guidelines for the administrator on the shape of the R-D functions and should be considered as a first approximation. Indeed, more elaborate R-D models can be found in the literature, but they are quite complex and expensive to implement. For detailed discussion and comparisons of various R-D models, see for example [19] and references therein.

VII. Conclusions

In this paper, we first investigated the characteristics of FGS coded video streams. We designed several experiments using the emerging H.264/MPEG-4 SVC coder to study the trade-off between the coding efficiency and the range of clients that can be supported. The base layer rate is the main controlling parameter: Larger base layer rates yield higher coding efficiency but support fewer client classes, and vice versa. Our experiments show that the coding efficiency gap is a non-increasing function of the base layer rate. Then, we formulated a single-sequence optimization problem to determine the base layer rate that achieves the best average video quality for a given client distribution. Solving this optimization problem is expensive, because there are too many possible choices for the base layer rate of FGS coded streams. We proposed a simple algorithm that runs in linear time. We proved that our algorithm yields the optimal base layer rate. Our proposed FGSOPT algorithm can be used in streaming systems in which: 1) the sever is broadcasting a single FGS-coded stream to many clients or 2) the server pre-allocates a fixed bandwidth for each stream.

We extended our formulation to multiple FGS video sequences, which is more general and applicable to servers that are concurrently streaming multiple sequences to diverse client communities. We formulated the problem and proved that it is NP-complete. Then, we proposed a branch-and-bound algorithm (MFGSOPT) that finds the optimal solution. This algorithm could be used for off-line cases in which the server has estimates on future client distributions and can therefore produces optimal FGS streams for them apriori. For dynamic cases, we proposed a heuristic algorithm (MFGS) that runs significantly faster than the branch-and-bound algorithm and produces near-optimal results.

We rigorously evaluated our proposed algorithms. We implemented our algorithms and compared their achieved average quality against rule-of-thumb coding structures, which is the current practice. Our results indicated that our optimal algorithms, FGSOPT and MFGSOPT, achieve better average perceived quality for all clients. We also showed the efficiency of

our heuristic MFGS algorithm, and verified that it scales to large number of sequences. Finally, we experimentally validated the non-increasing property of the quality gap function $\Delta(r_b)$ assumed by our algorithms, and we showed that a simple quadratic function is a good approximation for rate-distortion models used in our algorithms.

REFERENCES

[1] H. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP," *IEEE Trans. Multimedia*, vol. 3, no. 1, pp. 53–68, Mar. 2001.

[2] W. Li, "Overview of fine granularity scalability in MPEG-4 video standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 301–317, Mar. 2001.

[3] H. Schwarz, D. Marpe, and T. Wiegand, "The scalable H.264/MPEG4-AVC extension: Technology and applications," in *Proc. European Symp. Mobile Media Delivery (EuMob'06)*, Sardinia, Italy, Sep. 2006.

[4] M. van der Schaar and H. Radha, "Adaptive motion-compensation fine-granular-scalability (AMC-FGS) for wireless video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 6, pp. 32–51, Jun. 2002.

[5] F. Wu, S. Li, and Y. Zhang, "A framework for efficient progressive fine granularity scalable video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 332–344, Mar. 2001.

[6] C. Hsu and M. Hefeeda, "Optimal partitioning of fine-grained scalable video streams," in *Proc. ACM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'07)*, Urbana-Champaign, IL, Jun. 2007, pp. 63–68.

[7] I. Radulovic, P. Frossard, and O. Verscheure, "Adaptive video streaming in lossy networks: Versions or layers?," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME'04)*, Taipei, Taiwan, Jun. 2004, pp. 1915–1918.

[8] P. de Cuetos, D. Saparilla, and K. Ross, "Adaptive streaming of stored video in a TCP-friendly context: Multiple versions or multiple layers?," in *Proc. Int. Packet Video Workshop (PV'01)*, Kyongju, Korea, Apr. 2001.

[9] T. Kim and M. Ammar, "A comparison of layering and stream replication video multicast schemes," in *Proc. ACM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, Port Jefferson, NY, Jun. 2001, pp. 63–72.

[10] T. Kim and M. Ammar, "A comparison of heterogeneous video multicast schemes: Layered encoding or stream replication," *IEEE Trans. Multimedia*, vol. 7, no. 6, pp. 1123–1130, Dec. 2005.

[11] J. Liu, B. Li, Y. Hou, and I. Chlamtac, "Dynamic layering and bandwidth allocation for multi-session video broadcasting with general utility functions," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, Mar. 2003, pp. 630–640.

[12] H. Radha, M. van der Schaar, and S. Karande, "Scalable video transcaling for the wireless Internet," *EURASIP J. Appl. Signal Processing*, vol. 24, no. 2, pp. 265–279, Feb. 2004.

[13] C. Hsu and M. Hefeeda, "Structuring multi-layer scalable streams to maximize client-perceived quality," in *Proc. IEEE Int. Workshop on Quality of Service (IWQoS'07)*, Evanston, IL, Jun. 2007, pp. 182–187.

[14] M. Dai, D. Loguinov, and H. Radha, "Rate-distortion analysis and quality control in scalable Internet streaming," *IEEE Trans. Multimedia*, vol. 8, no. 6, pp. 1135–1146, Dec. 2006.

[15] Y. Yang, M. Kim, and S. Lam, "Optimal partitioning of multicast receivers," in *Proc. IEEE Int. Conf. Network Protocols (ICNP'00)*, Osaka, Japan, Nov. 2000, pp. 129–140.

[16] Joint Video Team, Joint Scalable Video Model Reference Software JSVM 8.0, Feb. 2007.

[17] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, 1st ed. New York: Springer, Jan. 2004.

[18] C. Hsu and M. Hefeeda, Partitioning of Multiple Fine-Grained Scalable Video Sequences Concurrently Streamed to Heterogeneous Clients Simon Fraser Univ., Vancouver, BC, Canada, Tech. Rep. TR 2007-02, Feb. 2007 [Online]. Available: http://www.nsl.cs.surrey.sfu.ca/projects/fgs/

[19] C. Hsu and M. Hefeeda, "On the accuracy and complexity of rate-distortion models for FGS-encoded video sequences," *ACM Trans. Multimedia Comput., Commun., and Appl.*, accepted for publication.

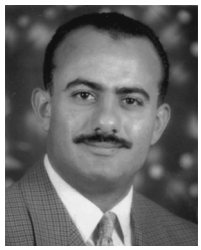
[20] Joint Video Team, Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC, Mar. 2005.

[21] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz, and M. Wien, Joint Draft 8 of SVC Amendment Joint Video Team, Hangzhou, China, Tech. Rep. JVT-U201, Oct. 2006.

[22] J. Reichel, H. Schwarz, and M. Wien, Joint Scalable Video Model JSVM-8 Joint Video Team, Hangzhou, China, Tech. Rep. JVT-U202, Oct. 2006.

Cheng-Hsin Hsu received the B.Sc. and M.Sc. degrees in 1996 and 2000, respectively, from National Chung-Cheng University, Taiwan, R.O.C., and the M.Eng. degree from the University of Maryland, College Park, in 2003. He is currently working toward the Ph.D. degree in the School of Computing Science, Simon Fraser University, Surrey, BC, Canada.

His research interests are in the area of multimedia networking and scalable video coding.



Mohamed Hefeeda (S'01–M'04) received B.Sc. and M.Sc. degrees from Mansoura University, Egypt, in 1997 and 1994, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2004.

He is an Assistant Professor in the School of Computing Science, Simon Fraser University, Surrey, BC, Canada, where he leads the Network Systems Lab. His research is funded by Canadian funding agencies and industry through several grants. His research interests include multimedia networking, peer-to-peer systems, and wireless sensor networks.

Dr. Hefeeda is a member of the ACM Special Interest Groups on Data Communications (SIGCOMM) and Multimedia (SIGMM).