Distributed Fine Grain Adaptive-FEC Scheme for Scalable Video Streaming

Yufeng Shan, John W. Woods, and Shivkumar Kalyanaraman
Department of ECSE, Rensselaer Polytechnic Institute, TROY, NY 12180-3590
Email: shany@rpi.edu; {woods,shivkuma}@ecse.rpi.edu

ABSTRACT

In this paper, we investigate a distributed fine grain adaptive FEC (FGA-FEC) scheme for scalable video streaming to heterogeneous users over a congested multihop network, where we do FGA-FEC decode/re-code at selected intermediate overlay nodes, and do FGA-FEC adaptation at remaining nodes. In order to reduce the overall computational burden, we propose two methods: (1) a coordination between optimization processes running at adjacent nodes to reduce the optimization computation, and (2) extension of our overlay multihop FEC (OM-FEC[1, 2]) to reduce the number of FGA-FEC decode/recode nodes. Simulations show that the proposed scheme can greatly reduce computation, and can provide near best possible video quality to diverse users.

Keywords: D istributed FEC, Video streaming, Scalable

1. INTRODUCTION

Scalable video bitstream[3, 4] is encoded to accommodate diverse user's requirements in terms of quality, frame rate and resolution. This kind of bitstream has the characteristic that the subsets corresponding to lower frame-rate/resolution/quality of the video are embedded in bitstreams corresponding to higher frame-rate/resolution /quality. Different sub-bitstreams can be extracted in a simple manner without transcoding, to readily accommodate a variety of users considering their display size, computing power, connection bandwidth and etc. While streaming, with the support of a service overlay network,[5, 6] a single scalable bitstream would be sufficient to satisfy the requirements of multiple diverse users as shown by an example in Fig. 1, where DSNs are data service nodes with certain service functions, such as bitstream adaptation, network monitoring and so on. Users "A" to "G" have different video preferences (shown as "frame-rate/resolution/bitrate"). Here C and Q represent the common CIF and QCIF formats, respectively, and p_a to p_g are the average packet-loss rates of the overlay virtual links.

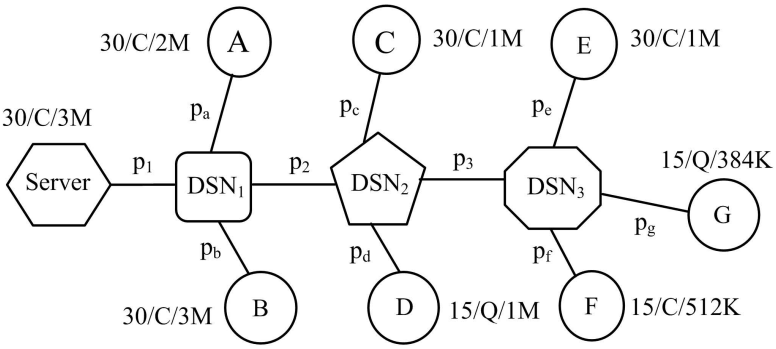


Figure 1. Intermediate adaptation of the video bitstream according to user video requests and network conditions by overlay data service nodes

In this example, the video server would store one bitstream (corresponding to the highest frame-rate /resolution /quality) and send it to the network. Inside network, DSNs would adapt the scalable bitstream to serve diverse users by forwarding the appropriate portion of the bitstream. To match such bitstream adaptation in lossy networks, we proposed a scalable error-correction coding method, called fine grain adaptive FEC (FGA-FEC).[7] Our FGA-FEC can encode scalable video in such a way that both the embedded bitstream and the error

correction codes can be easily and precisely adapted in a multidimensional way to satisfy diverse users without complex transcoding at intermediate nodes. The server first encoded the scalable video based on the highest user request and aggregated network conditions, then it sent the encoded bitstream into the network. Inside the network, the DSNs adapted the FGA-FEC encoded bitstream to satisfy heterogeneous users by shortening and/or dropping packets. We assumed that there was no congestion in the network backbone, i.e. that the backbone available bandwidth was large enough to accommodate all user requirements. This assumption is for service provider based structured networks, where the congestion and packet loss mainly happen at the edge of network or at the last mile connection.

One problem still remains: in a multihop network, congestion could be anywhere inside the network, especially in an ad hoc wireless network. How should we modify FGA-FEC to work with a congested back-bone? Here, a congested link is defined as a link whose available bandwidth is less than the minimum required bandwidth to accommodate a user's video request. One solution to address this problem is a hop by hop based solution.[8] We can optimize FEC protection for each individual link and apply FGA-FEC decode/re-code at each DSN for each user. By FGA-FEC decode/re-code, we mean that a DSN decodes FGA-FEC of the received GOP, re-optimize the multiple descriptions and then re-codes the GOP with new designed FGA-FEC for its downlinks. This would be a heavyweight hop-by-hop computationally intensive method if done at every overlay node. Here, we argue it may not be necessary to do FEC decode/re-code at each DSN. For example, in Fig. 1, if the link between the server and the DSN1 is congested, and other links are not, we may only do FGA-FEC decode/re-code at DSN1 and do FGA-FEC adaptation at the remaining DSNs. We need to identify the congested links in the backbone and apply the appropriate transformation at each DSN. Still, running the full FGA-FEC optimization at even some DSN nodes may be computationally demanding. So, here we describe a distributed algorithm, where we do FGA-FEC decode/re-code at the selected DSNs. The proposed distributed FGA-FEC scheme includes two parts: (1) a coordination method between FGA-FEC optimization processes running at nearby nodes to reduce the optimization computation, and (2) we apply OM-FEC to reduce the number of FGA-FEC decode/re-code nodes, i.e we use FGA-FEC adaptation where permitted and perform FGA-FEC decode/re-code only at certain key DSNs. This design thus lies between the end-to-end and hop-by-hop paradigms. If there is no congestion over the backbone, we choose end-to-end FGA-FEC scheme, no FEC decode/re-code is needed at intermediate nodes, but efficient adaptation. If each backbone link is congested, it is a heavyweight hop-by-hop FEC decode/re-code scheme. For this more advanced distributed algorithm, we only focus on SNR scalability, and leave extension to resolution and frame-rate scalability as a topic of future work.

2. DISTRIBUTED FGA-FEC

2.1. Overview the FGA-FEC scheme

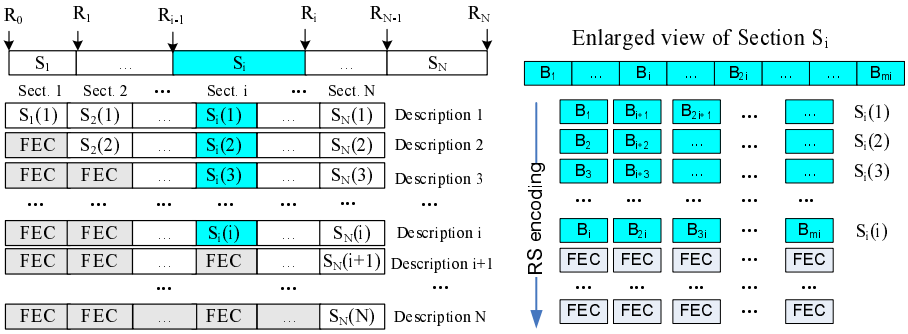


Figure 2. FGA-FEC encoding of one GOP. Here, FEC is added vertically at block level and each horizontal row of blocks is packetized into one network packet.

Our FGA-FEC encoding method (Fig. 2) extends MD-FEC[9] by adding scalability (adaptation) features. Given a GOP of scalable-coded video bitstream organized from MSB (R_0) to LSB (R_N), shown at the top in Fig. 2, suppose we want to encode this GOP into N descriptions, we first run an optimal bit allocation scheme

and divide the bitstream into N sections S_i, $(i \in [1, N])$, marked with source-rate break points $R_0, R_1, R_2, ..., R_N$, where $R_0 \leq R_1 \leq R_2 \leq ... \leq R_N$ and $R_0 = 0$. Section S_i $(i \in [1, N])$ is further split into equal size subsections with each subsection i blocks. These subsections are encoded by an RS(N, i) code vertically at block level to generate parity blocks. Since each block column is independently coded, at intermediate node, we can adapt the bitstream by easily removing related columns and/or dropping descriptions,[7, 10] both source data and parity bits, to satisfy diverse users. In this FGA-FEC scheme, no transcoding is needed, the adaptation method is very efficient and near optimal.[10]

2.2. Distributed FGA-FEC

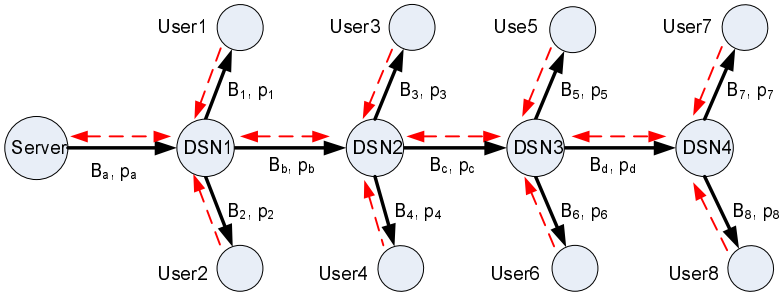


Figure 3. Streaming video from server to users through DSNs, red-dotted arrows are overhead information flows, black-solid arrows are video flows.

We outline our idea in a simplified example as shown in Fig. 3, where a server streams video to 8 diverse users through DSNs over a congested backbone. Before the streaming session, each end user sends its ideal video request (D_{min} in terms of distortion) and maximum tolerable distortion (D_{max}) to its directly connected DSN. During the streaming, at each time interval (1 GOP or multiple GOPs), edge DSNs (DSN4, whose downlinks have only end users) initialize optimization processes for each child to figure out what kind of bitstream it needs to request from its parent DSN (DSN3). This request is based on its children's link conditions and their video requests. The combined video request of its child nodes along with the optimization result is sent to DSN3 as overhead information. DSN3 then runs optimizations for its own children, including DSN4 (DSN3 treats DSN4 as one ordinary user), and generates the requested information to its parent DSN2. This process is repeated until we arrive back at the server. The server then runs the same algorithms as DSNs to determine the amount of FEC that should be applied to the video and then sends the encoded video into network. Inside the network, some selected DSNs will decode, redo the FGA-FEC design and recode FEC for some users, the other DSNs are only adaptation nodes. There are two kinds of flows in the distributed algorithm, upstream overhead information flow (shown via red-dotted arrows at Fig. 3) and downstream video data flow (shown via black arrows). Each DSN only exchanges optimization information with its direct parent or children, generating only local overhead information traffic. The DSNs use this information to coordinate optimization processes running at nearby nodes to reduce the computational burden, as well as to decide which nodes that will be involved in the FGA-FEC decode/re-code. We apply the idea of OM-FEC to minimize the number of involved FGA-FEC decode/re-code nodes while still maintaining the near optimal video quality for each user. We first focus on how to reduce the optimization computation.

2.2.1. Coordination Between Algorithm Processes Running at Adjacent Nodes

The FGA-FEC optimization algorithm is run at both DSNs and video server. A DSN runs optimization for its children to figure out what kind of bitstream it needs to request from its parent DSN or server. The server runs optimization to design the FEC and to encode a GOP. The only difference in the optimization algorithms running at DSNs and server are the input parameters. In this study, the optimization time interval is one GOP. Here, we briefly overview the optimization algorithm.

The optimization goal is to find the optimal bitrate partition $R = \{R_1, R_2, \cdots, R_N\}$ of a GOP, which minimizes the end-to-end mean distortion $E[D(R)]$ over a channel with available bandwidth B and packet loss

probability p.

$$E[D(R)] = \sum_{i=0}^{N} q_i D(R_i), \tag{1}$$

subject to:

$$\begin{cases} 0 \le R_1 \le R_2 \le ... \le R_N \\ R_{\text{total}} \le B \\ R_i - R_{i-1} = r_i \times i, \qquad r_i \ge 0, \quad \forall\, i \in [1, N] \end{cases}$$

where N is the number of descriptions encoded in one GOP, r_i is the rate of each subsection at section i ($i \in [1, N]$) of the bitstream (see Fig. 2). The probability that any i out of N packets are successfully delivered is q_i, R_{total} is the total bandwidth (bitrate) available for both FEC and video data.

Solving (1) is a constrained optimization problem. To find the optimal solution, we can use the Lagrange multiplier method and construct the function

$$F(R_1, \cdots, R_2, \lambda) = \sum_{i=0}^{N} q_i D(R_i) + \lambda(\sum_{i=0}^{N} \alpha_i R_i - B). \tag{2}$$

Taking the partial derivative of (2) with respect to R_i, $i = 0, 1, \cdots, N$, and setting them to 0, then, we can use a bisection search to find the appropriate λ and the corresponding rate break points $R = \{R_1, R_2, \cdots, R_N\}$, and the $E[D]$ value.

The motivation of coordination typically is from the following: (1) Video statistic information between adjacent GOPs does not change rapidly. (2) Server and parent DSNs have the optimization information from their child DSNs of the same GOP, with only different B and p. Therefore, the problem can be simplified into how to utilize the previous optimization information as network condition and video statistics change. We will use two coordination methods: (1) search with previous GOP results at this DSN, and (2) search with current GOP result from child node. Edge DSNs (a DSN whose children are all end users) initialize optimization for a new GOP. There, we can use optimization information from the previous GOP, we call this method "search with previous GOP". Intermediate DSNs and the server have local information not only of the same GOP from child DSNs but also have their previous GOP optimization result. Thus, they can use information of either of these GOPs to initialize their optimization search. Using the optimization information from child DSN, will be called"search with neighbor". We also consider a full search method, where each node runs the optimization algorithm independently. There, the upstream communication between nodes is only the video request. The optimization shared between nodes are λs and rate break points R_is.

2.3. Coordination to Reduce Number of FGA-FEC Decode/re-code Nodes

An extreme case of the distributed FGA-FEC is hop-by-hop FGA-FEC decode/re-code, i.e do FGA-FEC decode/re-code at each DSN. This method can provide the best possible video quality for diverse users in a congested backbone, since the protection is specifically optimized for each individual user. One may argue that it is not necessary to do the FGA-FEC decode/re-code at each DSN, if only part of the network is congested. For example, we already have shown that if the network backbone is not congested, our simpler FGA-FEC adaptation can also provide a near optimal solution if the user diversity is not too great.[7, 10] Combining these two ideas together, we do FGA-FEC decode/re-code at some selected nodes, while still providing similar video quality to hop-by-hop FGA-FEC decode/re-code. So here, we apply our OM-FEC concept to the network backbone to divide the network into segments and hence minimize the number of FGA-FEC decode/re-code nodes.

We use the topology of Fig. 3 to illustrate the idea. In Fig. 3, if there is no congestion in the backbone, we can directly encode a video using FGA-FEC only at the server and then use the simpler FGA-FEC adaptation inside the network. If some links in the backbone are congested, we need to identify them and apply FGA-FEC decode/re-code functions at the boundary nodes of these congest links as in Algorithm 1. We still use local information to decide the congested links.

Algorithm 1: Distributed FGA-FEC Algorithm

1 Edge DSN initializes new GOP optimization for its children using "search with previous GOP".
2 Edge DSN sends video requests and the optimization information $\{D, D_{\max}, \lambda, B, p, p_{\max}\}$ to its parent DSN.
3 Intermediate DSN runs optimization for its children using "search with neighbor".
4 Intermediate DSN tests if its child DSN has enough bandwidth B for an FGA-FEC adaptation.
If yes, intermediate DSN informs its child DSN as an adaptation node.
Otherwise, this is a congested link, child DSN will do FGA-FEC decode/re-code.
5 Intermediate DSN sends video request and optimization information to its parents.
6 This process is repeated through intermediate DSNs back up to server.
Thus, the congested links are detected and so are the FGA-FEC decode/re-code nodes.

Referring to Fig. 3, the network conditions are listed for each link. Before the streaming session, each end user sends its video requirement to its parent DSN as overhead information. This requirement can be described as a quality range $[D_{\min}, D_{\max}]$, where $D_{\min}$ is the user's ideal video preference and $D_{\max}$ is his/her maximum tolerable distortion. The optimization starts from the edge DSN (DSN4) and proceeds up to the server using a bottom up procedure through all the intermediate DSNs, in detail as below.

1. DSN4 runs optimization algorithm for its child User7, the result will be $D_{\min 7} + \lambda_7 R_{\text{total}7}$, given $B_7 \geq R_{\text{total}7}$, we can only use part of the available bandwidth, since $D_{\min 7}$ is satisfied. If $B_7 < R_{\text{total}7}$, link is congested, and the result will be $D'_7 + \lambda'_7 R'_{\text{total}7}$, where $D_{\min 7} < D'_7$ and $R'_{\text{total}7} = B_7$, i.e. we use up all the available bandwidth to get the best quality possible for User7. If $D_{\max 7} < D'_7$, no video is sent to User7.

Similarly for User8, the result will be $D_{\min 8} + \lambda_8 R_{\text{total}8}$ given $B_8 \geq R_{\text{total}8}$. If $B_8 < R_{\text{total}8}$ the result will be $D'_8 + \lambda'_8 R'_{\text{total}8}$, where $D_{\min 8} < D'_8$, and $R'_{\text{total}8} = B_8$.

2. DSN4 generates its video request to send upstream which is the union of both users' request, in this case, it is D =min($D_{\min 7}$ or D'_7, $D_{\min 8}$ or D'_8), given both users have valid data request. The maximum tolerable distortion will be $D_{\max}$ =max($D_{\max 7}, D_{\max 8}$) (at least should satisfy one user), the requested bitstream range will be at [D, $D_{\max}$]. DSN4 sends this combined video request to its parent DSN3, along with its optimization information $\{\lambda, B, p\}$. Here, (B, p) corresponds to D. The total request is the set $\{D, D_{\max}, \lambda, B, p, p_{\max}\}$, where $p_{\max}$ is the maximum loss probability among its children links and is used for FGA-FEC optimization test.

3. DSN3 runs the optimization algorithm for its children (including DSN4 with video request [D, $D_{\max}$] and network conditions (B_d, p_d), based on the optimization information λ from DSN4 (DSN3 uses the optimization information to start its own process).

4. DSN3 does one FGA-FEC optimization test, it optimizes FGA-FEC based on B_d and the aggregated loss probability, roughly $1-(1-p_d)(1-p_{\max})$. If B_d is large enough for FGA-FEC encoding (can provide the video quality D with FGA-FEC), DSN4 is notified as an FGA-FEC adaptation node, otherwise, the link between DSN3 and DSN4 is congested, both DSN3 and DSN4 need to do FEC decode/re-code.

5. DSN3 sends its own request $\{D, D_{\max}, \lambda, B, p, p_{\max}\}$ to DSN2.

6. This process is repeated upward to server through DSN2 and DSN1. The backbone is then divided into segments and appropriate nodes are selected for FEC decode/re-code. Similarly as OM-FEC, the partition result could be an end-to-end scheme (no intermediate FEC decode/re-code is performed, only FGA-FEC adaptation) if no congestion is detected in the backbone, or at the other extreme it could be heavy weight hop-by-hop FEC decode/re-code if every hop is congested in the backbone, or anywhere in between.

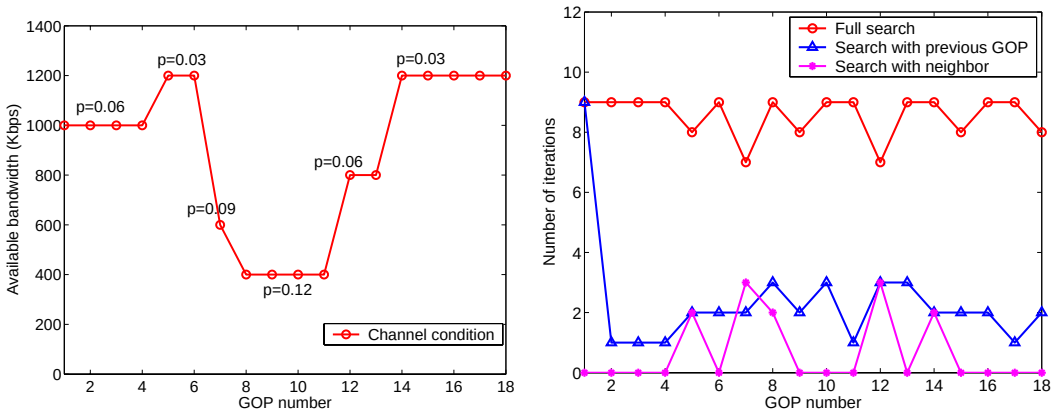
7. The server runs the optimization based on video request and optimization information from DSN1, encodes the GOP with FGA-FEC, and sends it to DSN1.

3. EXPERIMENTS AND SIMULATIONS

We did experiments and simulations to show the efficiency of our proposed distributed FGA-FEC scheme using videos *Foreman* CIF, 18 GOPs, *Mobile*, SIF, 8 GOPs and *Football*, SIF, 7 GOPs, with 16 frames/GOP in all three sequences. The source encoder is MC-EZBC, $N = 64$. The proposed scheme includes two approaches (1) a coordination method between optimization processes running at adjacent nodes to reduce computation, (2) using the OM-FEC concept to reduce the number of FGA-FEC decode/re-code nodes while still maintain near optimal video quality, measured in terms of PSNR. Regarding the first approach, we compare the number of iterations need to reach the optimization stop point using "full search", "search with previous GOP" and "search with neighbor". For the later approach, we compare with hop-by-hop FEC decode/re-code scheme and show that we can get similar video quality, but use fewer node involved in FEC decode/re-code. Finally, we measured the CPU time of using the distributed FGA-FEC algorithm to show the efficiency.

3.1. Optimization Performance

We solve the optimization problem using a bisection search to find the best λ value. We need find a stopping criteria. We use $|R_{\text{total}} - B| < \frac{1}{N} \times B$ and $|\lambda - \lambda_{\text{previous}}| < \varepsilon$, i.e the total rate should be close to the available bandwidth and λ is not changing much, where ε is a threshold. Intuitively, a larger threshold should correspond to coarser precision. After the optimization, $(N - \frac{1}{N} \times B) < R_{\text{total}} < (N + \frac{1}{N} \times B)$. If $R_{\text{total}} < B$, we need to allocate more video data to R_N to satisfy $R_{\text{total}} = B$. If $R_{\text{total}} > B$, we need to remove some video data from R_N to satisfy $R_{\text{total}} = B$. Experiments show that $\varepsilon = 1 \times 10^{-5}$ is a very good choice, the quality loss is almost negligible.[10]



(a) Channel condition (b) Number of iterations

Figure 4. Dynamic Channel Conditions: full search algorithm vs. our our proposed "search with previous GOP" and "search with neighbor", in terms of number of iterations at a dynamic channel, (a) channel conditions varying over GOP number, (b) the number of iterations to reach optimal stopping point.

In Fig. 4, we compare the full search algorithm with our proposed "search with previous GOP" and "search with neighbor" methods on a dynamic channel, where the channel condition changes over the GOPs as in Fig. 4(a). The corresponding number of iterations to reach stopping points for the three methods are shown in Fig. 4(b). Here one iteration is defined as one λ step calculation. Initially, we set $\lambda = 1 \times 10^{-3}$ in the "full search" method.[10] For full search optimization, the bisection search starts from the initial λ to the optimization stopping point. In the "search with previous GOP" method, the first GOP is the same as full search, we start from an initial λ value 1×10^{-3} and search to the optimization stopping point. After the first GOP, we use the previous GOP final λ (optimal point value) as our starting point to optimize the current GOP for the current network condition. In "search with neighbor", we use the same GOP information in previous network conditions from

child DSN. For "search with neighbor" method, if the network condition does not change, the optimization value can be used directly without optimization. From Fig. 4, we see that if the channel condition changes, both "search with previous GOP" and "search with neighbor" have similar performance, but when channel condition is statistically consistent, using "search with neighbor" gains over "search with previous GOP", saving about 2 iterations on average.

The results in this section show that the coordination between adjacent nodes can greatly reduce the optimization computation.

3.2. Comparison of Distributed FGA-FEC with Hop-by-hop FGA-FEC Decode/re-code

The distributed FGA-FEC scheme uses OM-FEC to partition congested networks into segments and appropriately selects FGA-FEC decode/re-code nodes while trying to maintain a near optimal delivered video quality. In this section, we compare hop-by-hop FGA-FEC decode/re-code versus the distributed FGA-FEC scheme in a congested multicast scenario. We use the ns-2[11] network simulator. The network topology is the same as in Puri and Ramchandran's paper[8] shown in Fig 5, where the network backbone is congested (1.15 Mbps between source and node1). In this topology, we set the probability of packet drop of each link to $p = 0.01$.

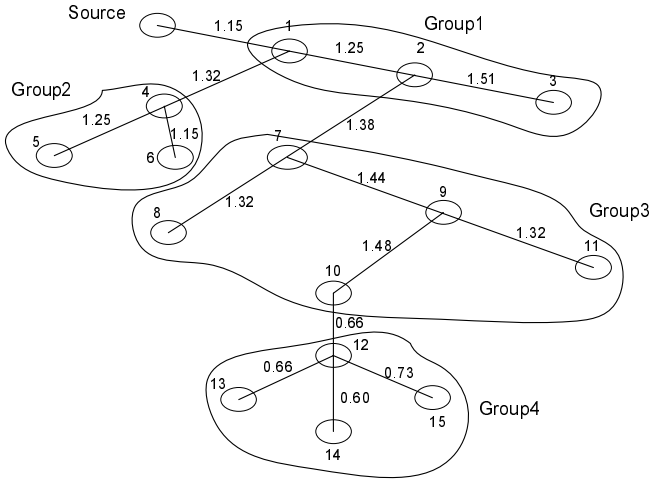
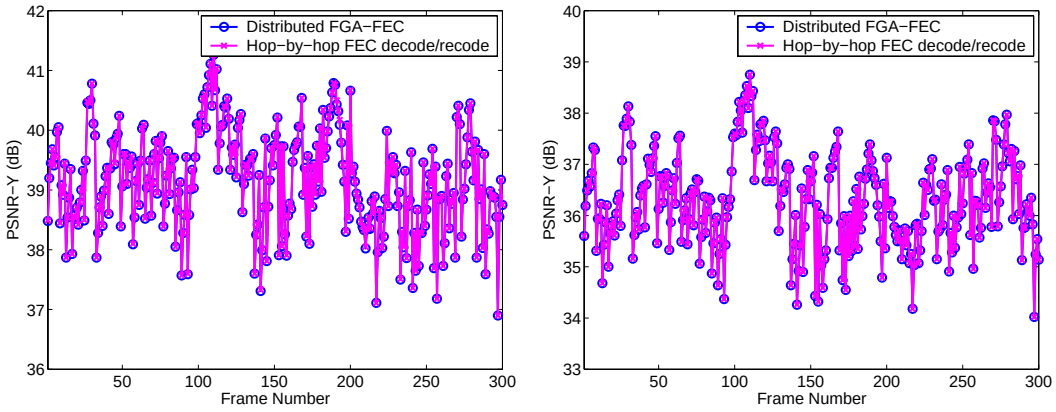


Figure 5. Network topology for a network of 16 nodes (link bandwidths are in Mbps, each link has packet-loss probability of 0.01). The backbone is congested, with smaller bandwidth than some end users.

In the hop-by-hop FGA-FEC decode/re-code, all intermediate nodes are involved in FEC decode /recode for their direct children. In the distributed FGA-FEC scheme, the congested network is partitioned into segments and only appropriate FGA-FEC decode/re-code nodes are selected. In this topology, distributed FGA-FEC would identify three congested links in the backbone: (1) from source to node 1, (2) from node 1 to node 2, and (3) from node 10 to node 12. Thus, nodes 1, 2, 10, 11 are FGA-FEC decode/re-code nodes. In the hop-by-hop FGA-FEC decode/re-code method, all intermediate nodes are involved in FEC decode/re-code, for a total of 7 nodes. Fig. 6 shows the PSNR quality delivered to receivers 5 and 12, respectively. For receiver 5, hop-by-hop FGA-FEC decode/re-code is about 0.01 dB better on average than the distributed FGA-FEC, with the relative loss mainly caused by the better performance of FGA-FEC decode/re-code at node 1. For hop-by-hop FGA-FEC decode/re-code, the received video is FGA-FEC re-coded at node 4 with packet loss probability $p = 0.01$. For the distributed FGA-FEC, the video is FGA-FEC re-coded at node 1 with a aggregated loss probability of about $p = 0.02$. Regarding receiver 12, these two schemes perform about the same, since they both do FGA-FEC decode/re-code at node 10 specifically for node 12.

The results in this Section show that the distributed FGA-FEC algorithm can provide similar quality to hop-by-hop FGA-FEC decode/re-code, but with less than half the nodes involved in FEC computation, 7 nodes in FGA-FEC decode/re-code versus 4 in distributed FGA-FEC in the section's simulation.



(a) Receiver 5 (b) Receiver 12

Figure 6. Quality delivered in PSNR (dB) at two receivers. For receiver 5, distributed FGA-FEC average performance is less than 0.01 dB lower than the hop-by-hop FGA-FEC decode/re-code algorithm. At receiver 12, both schemes have about the same video quality since they both do transcoding at parent node.

3.3. Distributed FGA-FEC CPU-Time

In the distributed FGA-FEC algorithm, a DSN either does FGA-FEC adaptation or FGA-FEC decode/re-code to satisfy its users. Here, we compare the computational complexity of FGA-FEC adaptation vs. FGA-FEC decode/re-code. We measured the CPU time of both schemes using the topology of Fig. 7, where the DSN is a Dell desktop with Pentium 4, 1.6 GHz CPU, 256 MB Memory, running Red Hat Linux 8.2. We use test sequences *Foreman* and *Mobile*, 7 GOPs/sequence, and number of descriptions encoded for each GOP N =64. The $D(R)$ curve rate interpolation interval is 100 bytes. The number of $D(R)$ points actually transmitted is 21. Table 1 lists the measured items in both schemes.

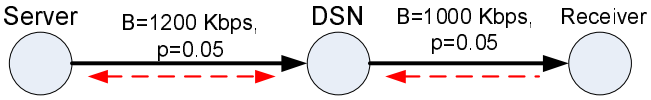


Figure 7. A simple topology video streaming to one user through DSN.

FGA-FEC decode/re-code	FGA-FEC adaptation
FGA-FEC decode time, FGA-FEC optimization time, and FGA-FEC recode time	The time to find the appropriate combination of dropping/shortening packets

Table 1. The measured items in FGA-FEC decode/re-code and FGA-FEC adaptation methods

3.3.1. FGA-FEC Decode/re-code Scheme

In the FGA-FEC decode/re-code scheme, the server first does optimization over a channel ($B = 1200$ Kbps and $p = 0.05$), then encodes the video using RS codes. At the DSN, RS decoding is first performed, then it does the optimization for its downlink channel ($B = 1000$ Kbps and $p = 0.05$), finally it does the RS encoding.

FEC optimization time

To optimize FEC protection for each GOP, the DSN needs to: (1) interpolate the $D(R)$ curve, (2) convexify $D(R)$ curve and calculate related parameters such as α_i, q_i, (3) perform bisection search at appropriate initial λ

value (one step of λ value calculation is called as one iteration), and (4) output the results. Table 2 shows the measured CPU time (in ms) of the different steps of the optimization algorithm for the two test clips.

Sequences	Total Optimization time/GOP (ms)	Bisection search time/GOP (ms)	Time/iteration (ms)
Foreman, FTFS	8.1	3.5	0.4
Foreman, HTFS	7.1	3.5	0.4
Foreman, FTHS	7.4	3.5	0.4
Mobile, FTFS	7.8	3.5	0.4

Table 2. Optimization CPU time. Here FTFS means full frame-rate full resolution, HTFS means half frame-rate full resolution and FTHS denotes full frame-rate half resolution. We show the average optimization time per GOP (sum of all four steps), the bisection search time, and the CPU time per iteration.

FGA-FEC decode/re-code time

Table 3 shows the measured CPU time (in ms) of RS decode/recode at the intermediate node for *Foreman*, only first 7 GOPs.

Sequences	decode time (ms)	recode time (ms)	total (ms)
Foreman, FTFS	28.7	15.8	44.5

Table 3. Measured CPU time (in ms) of RS decode/re-code at intermediate node. Results show that to perform FGA-FEC decode/re-code takes 44.5 ms on average per GOP.

3.3.2. FGA-FEC Adaptation Time

In the FGA-FEC adaptation, the server does the optimization and RS encoding based on aggregated network condition which is 1200 Kbps and $p = 0.1$. At the DSN, the FGA-FEC encoded bitstream is adapted for its downlink with 1000 Kbps, $p = 0.05$ only by shortening packets and dropping descriptions. Here, we measure the CPU time to find the appropriate combination of dropping/shortening packets. In this scheme, the DSN needs to: (1) interpolate the $D(R)$ curve, (2) find the appropriate combination of dropping/shortening packets, and (3) output the results. Table 4 shows the measured CPU time of FGA-FEC adaptation.

Sequences	FGA-FEC adaptation time (ms)
Foreman, FTFS	2.9
Foreman, HTFS	1.8
Foreman, FTHS	1.9
Mobile, FTFS	2.6

Table 4. Measured CPU time (ms) of FGA-FEC adaptation

In summary, Table 5 compares the CPU time of running FGA-FEC decode/encode scheme and FGA-FEC adaptation on *Foreman* for the first 7 GOPs. If the FGA-FEC direct truncation method (how many bytes need to be truncated from each packet) is used, the CPU time to process one GOP is less than 10^{-2}ms.

In our distributed FGA-FEC method, we coordinate between optimization processes running at adjacent nodes to reduce the number of iterations needed to reach the stopping point. The DSNs usually need 2-3 iterations for each user in our distributed scheme vs. 8-10 iterations for a full search without coordination, thus we can save 30-40% CPU time in the optimization computation, or about 3 ms for each user. The optimization time saving is even more significant with FGA-FEC adaptation. The 3 ms saving is comparable to one FGA-FEC adaptation (2.9 ms) but much larger than direct truncation ($< 10^{-2}$ ms). In the FGA-FEC decode/re-code case,

Scheme performed in DSN	CPU time (ms)
FGA-FEC decode/re-code	52.6
FGA-FEC adaptation	2.9
FGA-FEC direct truncation	1×10^{-2}

Table 5. Intermediate node FGA-FEC decode/re-code vs. FGA-FEC adaptation in terms of CPU time.

FEC coding dominates the computation, about 52.6 ms. Since we only need to do one decoding and many encodings, for each encoding, the CPU time is about 16 ms, the total savings for one user is about $3 \div 16 = 20\%$. In addition to the coordination method, we apply the idea of OM-FEC to reduce the number of nodes involved in FEC decoding/recoding. The gain in the latter method is very significant, since the decoding/recoding time is nearly twenty times longer than that of FGA-FEC adaptation (52 ms vs. 2.9 ms). i.e. each DSN can support nearly twenty times as many users if using FGA-FEC adaptation than using FGA-FEC decode/re-code.

4. CONCLUSION

In this paper, we proposed a distributed FGA-FEC algorithm for video streaming to diverse users on a congested network. We proposed a distributed approach to greatly reduce the computational burden of optimization by exchanging overhead information between adjacent nodes. We also extended the idea of OM-FEC to determine the congested links and hence to reduce the number of needed FGA-FEC decode/encode nodes. Here we apply FGA-FEC adaptation whenever permitted and do FGA-FEC decode/re-code only at the edge of congested links. Simulations have shown the performance of the proposed scheme.

REFERENCES

1. Y. Shan, I. V. Bajic, S. Kalyanaraman, and J. W. Woods, "Overlay multi-hop fec scheme for video streaming," *Signal Processing: Image Commun.* **16**, pp. 710–727, September 2005.
2. Y. Shan, I. Bajic, S. Kalyanaraman, and J. Woods, "Overlay multi-hop fec scheme for video streaming over peer-to-peer networks," in *International Conference on Image Processing*, pp. 3133 – 3136, IEEE, Oct. 2004.
3. S. T. Hsiang and J. W. Woods, "Embeded video coding using invertible motion compensated 3-d subband/wavelet filter bank," *Signal Processing: Image Commun.* **16**, pp. 705–724, May 2001.
4. B.-J. Kim, Z. Xiong, and W. A. Pearlman, "Low bit-rate scalable video coding with 3-d set partitioning in hierarchical trees (3-d spiht)," *IEEE Trans. Circuits Syst. Video Technol.* **10**, pp. 1374–1387, Dec. 2000.
5. H. J. W. V. N. Padmanabhan and P. A. Chou, "Distributing streaming media content using cooperative networking," *Microsoft Corporation, Tech. Rep* **MSR-TR-02-37**, 2002.
6. X. Gu and K. Nahrstedt, "Qos-assured service composition in managed service overlay networks," in *International Conference on Distributed Computing Systems*, pp. 194 – 201, IEEE, May 2003.
7. Y. Shan, I. Bajic, S. Kalyanaraman, and J. Woods, "Joint source-network error control coding for scalable overlay streaming," in *International Conference on Image Processing*, pp. 11 – 14, IEEE, Sept 2005.
8. R. Puri, K. Ramchandran, K. Lee, and V.Bharghavan, "Forward error correction codes based multiple description coding for internent video streaming and multicast," *Signal Processing: Image Commun* **16**, pp. 645 – 657, May 2001.
9. R. Puri and K. Ramchandran, "Multiple description coding using forward error correction codes," in *Proc. 33rd Asilomar Conf. (ACSS)*, pp. 342–346, IEEE, Oct 1999.
10. Y. Shan, *Scalable Joint Source-Network Coding of Video.* PhD thesis, Rensselaer Polytechnic Institute, May 2007.
11. "The network simulator- ns-2; http://www.isi.edu/nsnam/ns/."