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/recode 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 bitstreams[3, 4] are 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.

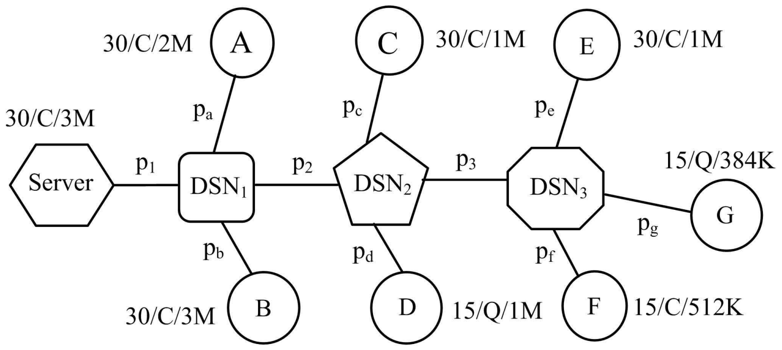


Figure 1. Intermediate adaptation of the video bitstream according to user video requests and network conditions by overlay data service nodes

In this example, the video server would store one bitstream (corresponding to the highest frame-rate/resolution/quality) and send it onto the network. Inside the network, the DSNs adapt the scalable bitstream to serve diverse users

*This work was partially supported by ARO contract W911NF0410300
†Yufeng Shan is now with Cisco Systems, e-mail: yshan@cisco.com

by forwarding appropriate parts 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 encodes the scalable video based on the highest user request and aggregated network conditions, then it sends the encoded bitstream onto the network. Inside the network, the DSNs adapt the FGA-FEC encoded bitstream to satisfy heterogeneous users by shortening and/or dropping packets. However, we assumed that there was no congestion on the network backbone, i.e. that the backbone available bandwidth was large enough to accommodate all user requirements. This assumption is reasonable for service-provider structured networks, where the congestion and packet loss mainly happen at the network edge.

On a multihop network, congestion could be anywhere inside the network, especially in an ad hoc and/or wireless network. Here we address the problem of extending 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/recode at each DSN for each user. By FGA-FEC decode/recode, we mean that a DSN decodes FGA-FEC of the received GOP, re-optimizes the multiple descriptions, and then recodes the GOP using FGA-FEC designed 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/recode 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/recode at DSN1 and do FGA-FEC adaptation at the remaining DSNs. Thus, we should 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 to do FGA-FEC decode/recode at the selected DSNs. The proposed distributed FGA-FEC scheme includes two parts: (1) a coordination method between individual FGA-FEC optimization processes running at neighbor nodes to reduce the optimization computation, and (2) application of OM-FEC[1] to reduce the number of FGA-FEC decode/recode nodes, i.e we use FGA-FEC adaptation where permitted and perform FGA-FEC decode/recode only at certain key DSNs. If there is no congestion over the backbone, we choose the end-to-end FGA-FEC scheme, i.e. no FEC decode/recode at intermediate nodes, just efficient adaptation. If each backbone link is congested, it is a heavyweight hop-by-hop FEC decode/recode scheme. In this paper we focus on SNR scalability, and leave extension to spatial-resolution and frame-rate scalability as a future topic.

2. DISTRIBUTED FGA-FEC

2.1. Overview the FGA-FEC scheme

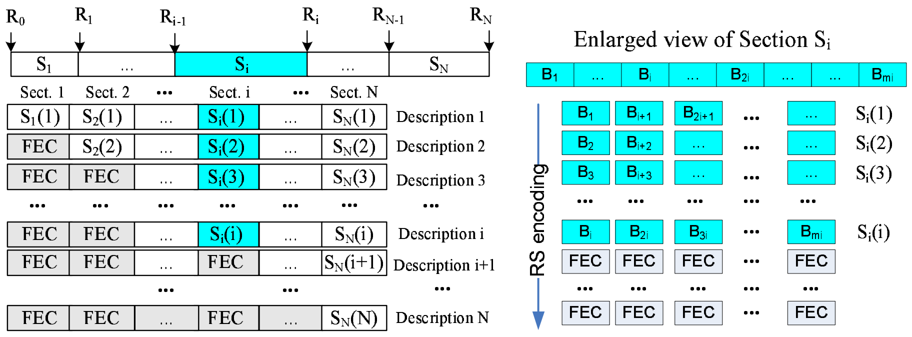


Figure 2. FGA-FEC encoding of one GOP. Here, FEC is added vertically at block level and each horizontal row of blocks is packetized into one network packet.

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 $R_0 \le R_1 \le R_2 \le ... \le R_N$ 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 nodes, 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

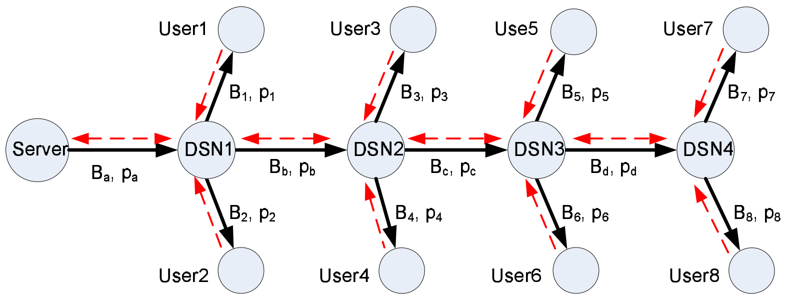


Figure 3. Streaming video from server to users through DSNs, red-dotted arrows are overhead information flows, black-solid arrows are coded video flows.

We outline our idea in a simplified form 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, edge DSNs (eg. 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 (eg. 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 the DSNs to determine the amount of FEC that should be applied to the video and then sends the encoded video onto the network. Inside the network, certain pre-selected DSNs will decode, redo the FGA-FEC design, and recode FEC for some users, but the other DSNs are just adaptation nodes. We consider two kinds of flows in our distributed algorithm, upstream overhead information flow (shown via red-dotted arrows in 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 with the optimization processes running at neighbor nodes to reduce the computational burden, as well as to decide which nodes that will be involved in the FGA-FEC decode/recode. We apply the idea of OM-FEC to minimize the number of involved FGA-FEC decode/recode nodes while still maintaining near optimal video quality for each user, as measured by PSNR. We turn next to how to reduce the optimization computation.

2.2.1. Coordination Between Algorithm Processes Running at Neighbor Nodes

Here we run the FGA-FEC optimization algorithm at both the DSNs and the 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 the next GOP. The only difference in the optimization algorithms running at DSNs and server are their input parameters. In this study, the optimization time interval is one GOP. Here, we briefly overview our optimization algorithm.

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

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

subject to:

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

where N is the number of descriptions encoded in one GOP, r_i is the source rate of each subsection in 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 and R_{total} is the total available bandwidth (bitrate) 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 for coordinating among neighbors comes from the following considerations: (1) video statistics between adjacent GOPs usually do not change rapidly (except for scene cuts), and (2) server and parent DSNs will have already calculated the optimization information from their child DSNs for the same GOP, with only different B and p. Again, these B and p values are usually highly correlated. Therefore, we wish to utilize this previous optimization information. Edge DSNs initialize optimization for a new GOP. There, and at internal DSNs, 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 own prior 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*. For reference, we also will consider a *full-search* method, wherein each node runs the optimization algorithm independently. There, the upstream communication between nodes only consists of the aggregated video requests. The optimization parameters shared between nodes are λ's and rate break points R_i's.

2.3. Finding Congested Links

An extreme case of our distributed FGA-FEC would be hop-by-hop FGA-FEC decode/recode, i.e do FGA-FEC decode/recode at each DSN. This method will provide the best quality for the users on a congested backbone, since the protection is specifically optimized for each individual link. However, it should not be necessary to do the FGA-FEC decode/recode at every 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 propose to do FGA-FEC decode/recode at only selected nodes, while still providing a similar video quality to hop-by-hop FGA-FEC decode/recode.

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

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

Algorithm 1: Distributed FGA-FEC Algorithm

```
1  Edge DSNs initialize new GOP optimizations for their children
   using *search with previous GOP.*
2  Edge DSNs send video requests and optimization information
   ${D, D_{\max}, \lambda, B, p, p_{\max}}$ to their parent DSNs.
3  Intermediate DSNs run optimization for their children using
   *search with neighbor.*
4  Intermediate DSNs tests if their children DSNs have enough available bandwidth $B$
   for an FGA-FEC adaptation.
   If yes, intermediate DSNs set their children DSNs as adaptation nodes.
   Otherwise, we have a congested link, and child DSNs will do
   FGA-FEC decode/recode.
5  Intermediate DSNs send video requests and optimization
   parameters to their parents.
6  This process is repeated at all intermediate DSNs and eventually reaches the server.
   Thus, the congested links are detected along with the FGA-FEC
   decode/recode nodes.
```

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 $B_7 < R_{\text{total}7}$, the 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' requests, in this case, it is $D = \min(D_{\min 7}$ or D'_7, $D_{\min 8}$ or $D'_8)$, given both users have a valid data request. The maximum tolerable distortion will be $D_{\max} = \max(D_{\max 7}, D_{\max 8})$ (at least we 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 parameter 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 (i.e. it can provide the video quality D with FGA-FEC), DSN4 is set as an FGA-FEC adaptation node, otherwise, the link between DSN3 and DSN4 is congested, and both DSN3 and DSN4 need to do FEC decode/recode.

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

6. This process is repeated upwards to the server through DSN2 and DSN1. The backbone is then divided into segments and appropriate nodes are selected for FEC decode/recode.

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.

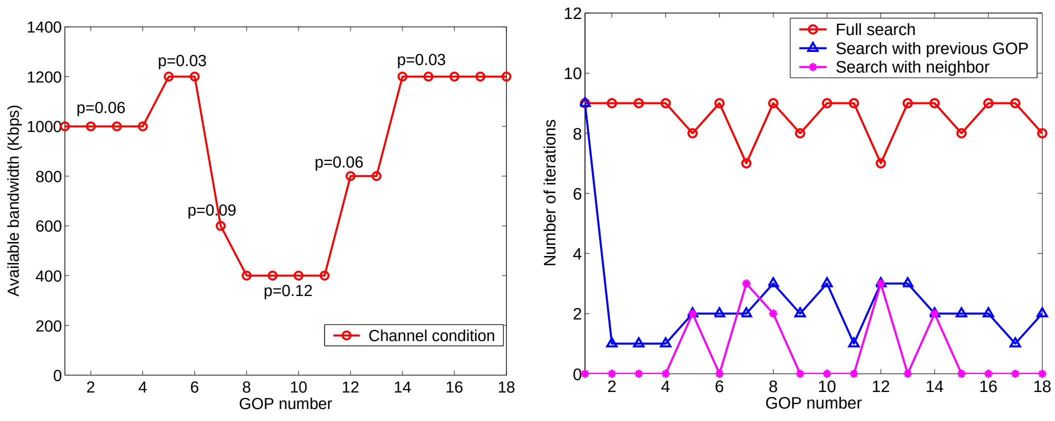
3. EXPERIMENTS AND SIMULATIONS

We did experiments and simulations to show the efficiency of our proposed distributed FGA-FEC scheme using test clips *Foreman* CIF, 18 GOPs, *Mobile*, SIF, 8 GOPs and *Football*, SIF, 7 GOPs, all with 16 frames/GOP.

The source encoder is the highly scalable and embedded MC-EZBC,[3] $N = 64$. The proposed scheme includes two approaches (1) a coordination method between optimization processes running at adjacent nodes to reduce computation, (2) the OM-FEC concept to reduce the number of FGA-FEC decode/recode nodes while still maintaining near optimal video quality, measured in terms of PSNR. Regarding the first approach, we compare the number of iterations needed to reach the optimization stoping point using *full search*, *search with previous GOP*, and *search with neighbor*. For the later approach, we compare with hop-by-hop FEC decode/recode scheme and show that we can get similar video quality, but using far fewer nodes involved in full FEC decode/recode. We use CPU time as a measure of algorithm efficiency.

3.1. Optimization Performance

We solve the optimization problem using a bisection search to find the best λ value. For 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 should be fairly constant, where ε is a given 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 $R_{\text{total}} = B$. Experimentally, we found that $\varepsilon = 1 \times 10^{-5}$ is a reasonable choice, with almost negligible quality loss.[10]



(a) Channel condition (b) Number of iterations

Figure 4. Dynamic link (channel) conditions: full-search algorithm vs. proposed *search with previous GOP* and *search with neighbor*, in terms of number of iterations for a dynamic channel, (a) channel conditions varying with GOP number, (b) the number of iterations to reach 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 dynamicly changing link, where the link (channel) condition changes versus GOP as shown in Fig. 4(a). The corresponding number of iterations to reach stopping point 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 always starts from this initial λ. In the *search with previous GOP* method, the first GOP is the same as full search, and we start from the same λ value. However, 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 conditions. In *search with neighbor*, we use the same GOP's information in prior network condition at the from child DSN. For the *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/recode

In this section, we compare hop-by-hop FGA-FEC decode/recode versus the distributed FGA-FEC scheme in a congested multicast scenario. We use the ns-2[11] network simulator with network topology the same as[8] and shown in Fig 5. The network backbone is congested with 1.15 Mbps between source and node 1. Packet-loss rates were not specified in,[8] but here we set the probability of packet drop for each link to $p = 0.01$.

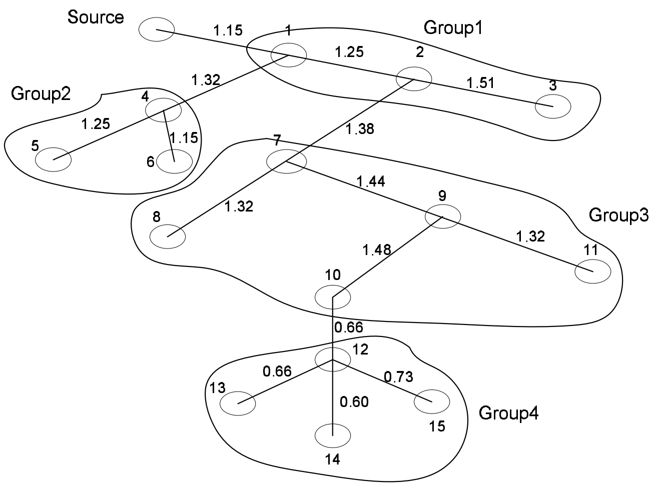


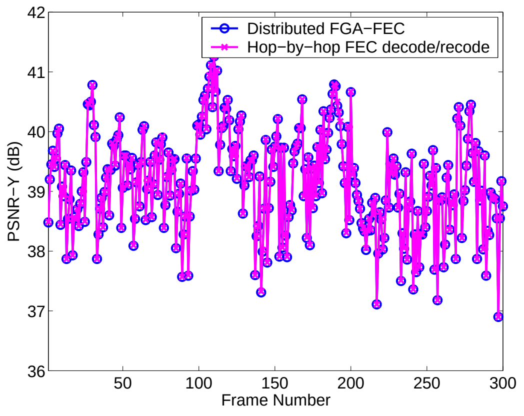
Figure 5. Network topology for a network of 16 nodes (link bandwidths are in Mbps, each link has packet-loss probability of 0.01). The backbone is congested, with smaller bandwidth than some end users.

In the hop-by-hop FGA-FEC decode/recode, all intermediate nodes are involved in FEC decode/recode for their direct children. In our distributed FGA-FEC scheme, the entire network is partitioned into segments and appropriate FGA-FEC decode/recode 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/recode nodes. In the hop-by-hop FGA-FEC decode/recode method, all intermediate nodes are involved in FEC decode/recode, 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/recode 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/recode at node 1. For hop-by-hop FGA-FEC decode/recode, the received video is FGA-FEC recoded at node 4 with packet-loss probability $p = 0.01$. For the distributed FGA-FEC, the video is FGA-FEC recoded at node 1 with a aggregated packet-loss probability of about $p = 0.02$. Regarding receiver 12, these two schemes perform about the same, since they both do FGA-FEC decode/recode at node 10 specifically for node 12.

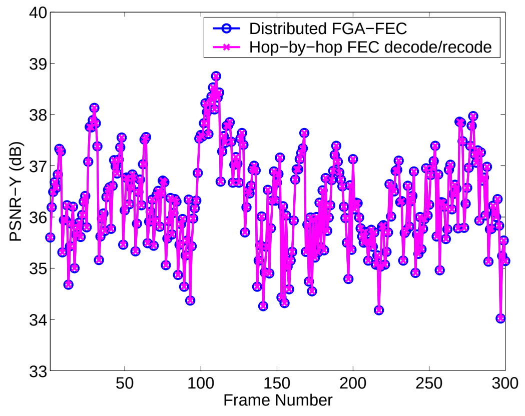
The results in this Section show that our distributed FGA-FEC algorithm can provide similar quality to hop-by-hop FGA-FEC decode/recode, but with less than half the nodes involved in FEC computation, 7 nodes in FGA-FEC decode/recode versus 4 in distributed FGA-FEC in the section's simulation.

3.3. Distributed FGA-FEC CPU-Time

In the distributed FGA-FEC algorithm, a DSN either does FGA-FEC adaptation or FGA-FEC decode/recode. Here, we compare the computational complexity of these two methods. 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 used 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, thus constituting a negligible overhead. Table 1 lists the measured items for both schemes.



(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/recode algorithm. At receiver 12, both schemes have about the same video quality since they both do transcoding at parent node.

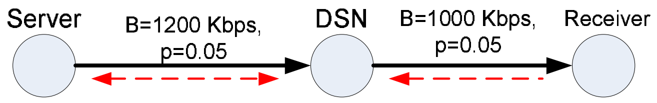


Figure 7. A simple topology video streaming to one user through DSN.

3.3.1. FGA-FEC Decode/recode Scheme

In the FGA-FEC decode/recode 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 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.

FGA-FEC decode/recode time

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

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

FGA-FEC decode/recode	**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/recode and FGA-FEC adaptation methods

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-temporal full-spatial resolution, HTFS means half temporal (frame-rate) full-spatial 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.

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/recode at intermediate node. Results show that to perform FGA-FEC decode/recode takes 44.5 ms on average per GOP.

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 (i.e. just shorten packets) is used, the CPU time to process one GOP is less than 10^{-2}ms.

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

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

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/recode case, 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/recode.

4. CONCLUSIONS

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 find 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/recode 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. V. Bajic, S. Kalyanaraman, and J. W. 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. V. Bajic, S. Kalyanaraman, and J. W. 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 internet 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/."