End-to-End Processing for Streaming Applications (Addenda)

Ying-zong Huang (2010)

Part I
Delay sensitive on-demand streaming

In [1], the problem of on-demand streaming was modeled as follows:

A pre-encoded and subset-decodable packet store with a playback rate of K packets per unit time is to be streamed across a packet erasure channel with a nominal (i.e. error-prone) throughput of N packets per unit time. With known probability p, some packets are erased i.i.d. The sender has access to encoder-computed importance indicators of the packets, where packet i contributes an additional additive distortion D_i if not decoded. The sender can choose to discard packets, send them as is, or code them into checksums and send the checksums. The paper derived the sender's optimal choices given this model. This work did not consider delay, and so realistically N and K cannot be very large.

If instead there is a fixed delay of T packets allowed before decoding, then we can extend the solution to account for this. We do this by choosing checksums that do not span more than T consecutive packets, but choosing the disposition of packets over all of them.

1 Greedy algorithm for assigning checksums

Let the maximum allowed delay be T packets. We describe a several-pass (suboptimal) algorithm to lower the expected distortion incurred at the decoder.

For notation, we index the packets as $1, 2, ..., T; T+1, T+2, ..., 2T; ...$

We let D_i be the distortion of packet i; let D_S be the total distortion of packets in S; let $\bar{D}_S$ be the average of the same; let $\bar{D}_i^j$ be the average distortion of packets $i, ..., j$.

Let X be the number of errors. Define $p_i = \mathbb{P}[X > i]$ as the residual error rate when i or fewer errors can be transparently handled, e.g. with a properly designed Reed-Solomon code. For instance, if $X \sim B(x; n, p)$, where $B(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$, then let the residual error rate be:

$$p_i(n) = \sum_{x=i+1}^{n} \binom{n}{x} p^x (1-p)^{n-x}$$

where p is the raw error rate.

For each packet j in a block of T packets $i+1, ..., i+T$, of which a subset S is kept, we may consider the following possibilities, along with their expected cost with respect to packet j.

- Discard packet j, i.e. a (0,1) code: $p_{-1}(0)D_j = D_j$
- No protection, i.e. a (1,1) code: $p_0(1)D_j$
- Protect S with 1 check, i.e. an $(|S|+1, |S|)$ code: $p_1(|S|+1)D_j$
- ...
- Protect S witih k checks, i.e. an $(|S|+k, |S|)$ code: $p_k(|S|+k)D_j$

There are more possibilities, but we will stick to these limited options. Next we describe the algorithm. Suppose we begin with K data packets (divisible by T) and N (assume $\geq K$) total packet slots.

Step 0 (initialization): Initialize $S_1, ..., S_k$ by setting $S_j = \{(j-1)T+1, ..., jT\}$, so $\bar{D}_{S_j} = \bar{D}_{(j-1)T+1}^{jT}$. Define $C_j(s)$ to be the expected cost of block S_j when protected with s_j checks. Initialize the protection amount to $s_j = 0$ (no protection) for all k bins and initialize the costs for bin j to $C_{S_j}(0) = p_0(1)D_{S_j}$.

Step 1 (allocate checks): Determine the marginal cost reductions $\Delta C_{S_j} = C_{S_j}(s_j) - C_{S_j}(s_j+1)$ across the k bins by computing

$$(p_0(1) - p_1(|S_j|+1))D_{S_j}$$

Choose

$$j* = \arg\max_j \Delta C_{S_j}$$

and update the amount of protection on bin $j*$ by $s_{j*} \leftarrow s_{j*} + 1$. Update also $N \leftarrow N - 1$. Stop when $N = K$. Go to Step 2.

Step 2 (bin optimize): Within each bin, use a variation of the original method to determine if some protected packets should be left unprotected by finding the marginal cost reduction

$$\Delta C_{S_j} = C_{S_j}(s_j) - C_{S_j \backslash i}(s_j) - C_i(0)$$

by choosing the lowest cost packet i from S_j. If $\Delta C_{S_j} > 0$, then $S_j \leftarrow S_j \backslash i$. If $\Delta C_{S_j} \leq 0$, then go to Step 3.

Step 3 (choose discards): For each packet i, compute the cost $C_i(s_j) - C_i(-1) + C_{S_j}(s_j) - C_{S_j \backslash i}(s_j)$ of discarding packet i. Choose the lowest discard cost packet i. Suppse it belongs in S_j, evaluate the marginal cost reduction test

$$\Delta C_i = C_i(s_j) - C_i(-1) + C_{S_j}(s_j) - C_{S_j \backslash i}(s_j) + C_{S_{j*}}(s_{j*}) - C_{S_{j*}}(s_{j*}+1)$$

where $j*$ is chosen by the same method as in Step 1, and it may even be the same bin, in which case $S_{j*} = S_j \backslash i$. If $\Delta C_i > 0$, then $N \leftarrow N + 1$, $S_j \leftarrow S_j \backslash i$, and then run Step 1. If $\Delta C_i \leq 0$, then stop.

When the iterative algorithm terminates, we have a set of D discarded packets, and a set of protected packets in each bin numbering no more than T, with total number of check packets equal to $N - K + D$. While this algorithm is almost certainly suboptimal, it is a reasonable first approach for fixed coding delay transcoding. An improvement would be to remove the sharp coding boundaries.

Part II

Stream decodability computation

In [2], the sender in the proposed system for lossless streaming can base its decisions on a calculation of whether certain source packets are likely to be decodable. After the sender consumes source packets $\mathbf{s}[1], ..., \mathbf{s}[j]$ and transmits network packets $\mathbf{x}[1], ..., \mathbf{x}[k]$, and receives feedback in the form of $\mathcal{P}$ (set of known received network packets), $\mathcal{N}$ (set of known lost network packets), and $\mathcal{U}$ (packets of unknown status), it can figure the probability of every source packet being sequentially decodeable. The structure of sequential decodability allows the algorithm for computing these probabilities in complexity $\mathcal{O}(j^2)$ rather than $\mathcal{O}(2^{|\mathcal{U}|})$. The following describes how this is done.

1 Data Structure

Decodeability when $\mathbf{x}[1], ..., \mathbf{x}[k]$ are sent is governed by two numbers, which are states in the success probability calculation algorithm: $r(k)$, the number of received network packets out of the k that the sender supposes the receiver has; and $n_{\mathbf{G}}(k)$, the number of source packets that have been consumed in the production of these $r(k)$ network packets (and hence the maximum potentially decodeable ones).

The algorithm begins by building the states $r(k)$ into a trellis indexed by k and assigning transition probabilities. Each state "time" k is associated with the network packet $\mathbf{x}[k]$. Since whenever $r(k) \geq n_{\mathbf{G}}(k)$, decoding up to $\mathbf{s}[n_{\mathbf{G}}(k)]$ succeeds.[1] The state value $n_{\mathbf{G}}(k)$ therefore corresponds to a special "success" state at index k, meaning $\mathbf{x}[1], ..., \mathbf{x}[k]$ together can decode $\mathbf{s}[1], ..., \mathbf{s}[n_{\mathbf{G}}(k)]$. The initial states are $r(0) = 0$, $n_{\mathbf{G}}(0) = 0$.

For each network packet $\mathbf{x}[k] \in \mathcal{U}$, two values of $r(k)$ branch from each value of $r(k-1)$ depending on whether $\mathbf{x}[k]$ is supposed as received or not: $r(k) = r(k-1)$ with probability p (not received), and $r(k) = r(k-1) + 1$ with probability $1 - p$ (received). For each $\mathbf{x}[k] \in \mathcal{P}$, $r(k) = r(k-1) + 1$ with probability 1, and for each $\mathbf{x}[k] \in \mathcal{N}$, $r(k) = r(k-1)$ with probability 1.

A path through the trellis (i.e. a sequence of values $r(0), r(1), ..., r(k)$) represents a *non-decodeable path* for the source packet $\mathbf{s}[j]$, if $r(i) < n_{\mathbf{G}}(i)$ for all $i \in \{1, ..., k\}$ where $n_{\mathbf{G}}(i) \geq j$. The probability that $\mathbf{s}[j]$ is sequentially decodeable is then the complement of the sum probability of all such non-decodeable paths.

2 Computation

Let $p_i(r(i-1), r(i))$ denote the transition probability from state value $r(i-1)$ to state value $r(i)$. The probability of a path $r(0), r(1), ..., r(k)$ is the product of the transition probabilities along it, namely

$$\prod_{i=1}^{k} p_i(r(i-1), r(i))$$

[1]Except when the receiver matrix $\hat{\mathbf{G}}$ is redundant, but this can be pre-detected and avoided by an intelligent sender.

This can be factored into two pieces:

$$\prod_{i=1}^{k'} p_i(r(i-1), r(i)) \times \prod_{i=k'+1}^{k} p_i(r(i-1), r(i))$$

which are respectively the probability of the left partial path into $r(k')$, which we call $\alpha_{k'}(r(1), ..., r(k'))$, and the probability of the right partial path out of $r(k')$, which we call $\beta_{k'}(r(k'+1), ..., r(k))$. These can be computed recursively as

$$\alpha_{k'}(r(1), ..., r(k')) = p_{k'}(r(k'-1), r(k'))\alpha_{k'-1}(r(1), ..., r(k'-1))$$

and

$$\beta_{k'}(r(k'+1), ..., r(k)) = p_{k'+1}(r(k'), r(k'+1))\beta_{k'+1}(r(k'+2), ..., r(k))$$

The initial conditions are $\alpha_0(0) = 1$ and $\beta_k(r(k)) = 1$.

In the following, suppose $j = n_{\mathbf{G}}(k') > n_{\mathbf{G}}(k'-1)$. Let $S_{k'}(r(k'))$ be the set of non-decodeable paths for $\mathbf{s}[j]$ through $r(k')$. Let $A_{k'}(r(k'))$ be the set of all partial paths that enter $r(k')$. Let $B_{k'}(r(k'))$ be the set of all partial paths that leave $r(k')$ and do not pass through a "success" state value. Then, $S_{k'}(r(k')) = A_{k'}(r(k')) \otimes B_{k'}(r(k'))$. The sets are recursive, in that

$$A_{k'}(r(k')) = \{A_{k'-1}(0), ..., A_{k'-1}(n_{\mathbf{G}}(k'-1))\} \otimes r(k')$$

and

$$B_{k'}(r(k')) = r(k') \otimes \{B_{k'+1}(0), ..., B_{k'+1}(n_{\mathbf{G}}(k'+1)-1)\}$$

Now, we can further factor the sum probability of the non-decodeable paths for $\mathbf{s}[j]$ through $r(k')$ by noting

$$\begin{aligned} & \sum_{r(1),...,r(k) \in S_{k'}(r(k'))} \alpha_{k'}(r(1), ..., r(k')) \times \beta_{k'}(r(k'+1), ..., r(k)) \\ = & \sum_{r(1),...,r(k') \in A_{k'}(r(k'))} \alpha_{k'}(r(1), ..., r(k')) \times \sum_{r(k'),...,r(k) \in B_{k'}(r(k'))} \beta_{k'}(r(k'+1), ..., r(k)) \end{aligned}$$

and the recursion

$$\begin{aligned} \alpha_{k'}(r(k')) & \triangleq \sum_{r(1),...,r(k') \in A_{k'}(r(k'))} \alpha_{k'}(r(1), ..., r(k')) \\ & = \sum_{r(k'-1)=0}^{n_{\mathbf{G}}(k'-1)} p_{k'}(r(k'-1), r(k')) \sum_{r(1),...,r(k'-1) \in A_{k'-1}(r(k'-1))} \alpha_{k'-1}(r(1), ..., r(k'-1)) \\ & = \sum_{r(k'-1)=0}^{n_{\mathbf{G}}(k'-1)} p_{k'}(r(k'-1), r(k'))\alpha_{k'-1}(r(k'-1)) \end{aligned}$$

and likewise

$$\beta_{k'}(r(k')) \triangleq \sum_{r(k'),...,r(k) \in B_{k'}(r(k'))} \beta_{k'}(r(k'+1), ..., r(k))$$

$$= \sum_{r(k'+1)=0}^{n_{\mathbf{G}}(k'+1)-1} p_{k'+1}(r(k'), r(k'+1)) \sum_{r(k'+1),\ldots,r(k)\in B_{k'+1}(r(k'+1))} \beta_{k'+1}(r(k'+2), ..., r(k))$$

$$= \sum_{r(k'+1)=0}^{n_{\mathbf{G}}(k'+1)-1} p_{k'+1}(r(k'), r(k'+1))\beta_{k'+1}(r(k'+1))$$

So the final marginal probability of the sequential decodeability of $\mathbf{s}[j]$ is simply

$$q(j) \triangleq 1 - \sum_{r(k')=0}^{n_{\mathbf{G}}(k')-1} \alpha_{k'}(r(k'))\beta_{k'}(r(k'))$$

This immediately gives a recursive algorithm for computing the decodeability probabilities $q(1), ..., q(j)$ of all source packets $\mathbf{s}[1], ..., \mathbf{s}[j]$ at once. Beginning at the index $k' = 0$, and increasing, recursively compute all values of the left probabiliites $\alpha_{k'}(r(k'))$ of the non-success state values at index k'. Beginning at the index $k' = k$, and decreasing, recursively compute all values of the right probabilities $\beta_{k'}(r(k'))$ of the non-success state values at index k'. Then, at each index k' where $j = n_{\mathbf{G}}(k') > n_{\mathbf{G}}(k' - 1)$, sum over the products of left and right probabilities at the non-success state values, and take the complement.

References

[1] Huang, Apostolopoulos, "A Joint Packet Selection/Omission and FEC System for Streaming Video."

[2] Huang, Mehrotra, Li, "A Hybrid FEC-ARQ Protocol for Low-Delay Lossless Sequential Data Streaming."