Broadcasting Video Content in Dense 802.11g Sports and Entertainment Venues

Project Update

Last update: 8/29/2010

Principal Investigators

Dr James Martin
School of Computing
Clemson University
Clemson, SC
jim.martin@cs.clemson.edu
Phone: 864 656 4592

Dr James Westall
School of Computing
Clemson University
Clemson, SC
westall@cs.clemson.edu
Phone: 864 656 5868
Project Progress: Developing the Models

- 802.11 Protocol
  - 802.11g PHY behaviors: refer to Experiments 1 and 2 documented in projectUpdate-7-8-2010.pdf
  - Adaptive modulation/coding: model supports basic rate and one statically defined rate for data – it’s possible to add AMC in the future
  - Multicast: model supports multiple multicast streams – broadcasts at basic rate over the air (under test)

- Packet loss process
  - Loss metrics: mean burst loss length (MBL), mean interloss distance (MILD)
  - Model-based loss: Bernoulli loss, Hidden Markov models, Gilbert-Elliot model
  - Propagation effects: propagation models, Rx packet capture
  - Protocol effects: congestion, collisions

- Application:
  - Video streaming traffic generator (encoder/decoder)
  - Application FEC Model
  - Application oriented quality assessment
Outline of this Status Update

• Overview the design of the application FEC traffic generator

• Demonstrate results using a simple unicast CBR-based stream
Application FEC Model

1. Video content (avi, mp4)
2. Video streaming encoder
3. APFEC - coder
4. FEC Send Side
5. Stream including redundant information
6. Network
7. FEC Rx Side
8. APFEC - decoder
9. Stream – possibly lost packets

Performance assessment (loss rate, latency, jitter)

Video quality assessment (PSNR, mean time between artifacts)
APFEC Video Streaming

• Send side highlights
  – Configured with an N and K
  – Maintains an index identifying the block number
    • Maintained as an unsigned integer, first block of a new session is always 1 (counter wraps at $2^{32}$). This is sent in each packet.
  – Redundant packets always sent at the end of the block
  – Variable size data packets:
    • Find the average packet size of packets sent in the block. Set all FEC packets to this size
  – BlockTimer pops if K data packets don’t arrive within a timeout amount of time. Two choices:
    • Pad the block to ALWAYS send N packets
    • Scale down the N and K for this block (this is what the code currently does)
  – Each packet will contain:
    • blockNumber, blockIndex (place in the current block), and the N and K that are in effect for the current block
    • Only data packets have sequence numbers, FEC ordering is done using the blockNumber and blockIndex.
APFEC Video Streaming

- Rx side highlights
  - N and K for a given block are determined on a block by block basis (from the N_ and K_ contained in each packet)
  - blockTimer required in the event K packets do not arrive in a timely manner
  - Two queues are maintained
    - blockQ: queues all packets that arrive that are associated with the current block being collected
    - OutOfOrderQ: queues all packets that arrive that are NOT associated with the current block being collected
  - In order packets that arrive are added to the blockQ and a duplicate packet is created and passed up to the application
  - When 1 or more data packets in the current block do not arrive, the block will be completed once any of the following occurs:
    - Once K or more packets have arrived
    - Once a threshold number of packets from next blocks have arrived
    - Once the blockTimer pops
class AppFecAgent : public Agent {
public:
  AppFecAgent();
  virtual void sendmsg(int nbytes, const char *flags = 0) {
    sendmsg(nbytes, NULL, flags);
  }
  virtual void init(int streamID, int N, int K);
  virtual void sendmsg(int nbytes, AppData* data, const char *flags = 0);
  virtual void blockTimerHandler(int);
  virtual void command(int argc, const char*const* argv);
  virtual void dumpFinalStats(char *outputFile);
private:
  int blockindex_;
  int numberBytesInBlock;
  int numberPacketsInBlock;
  int streamID;
  int seqno_;
  int blockno_;
  int N_; 
  int K_; 
  int typeFEC;
  int streamType;
  char traceString[32];
  FILE *fp;
  int state; // -1 uninited, 1 inited
  double blockTimerDelay;
  int numberFECPackets;
  AppFecBlockTimer *myTimer;
};

class AppFecSourceAgent : public AppFecAgent {
public:
  AppFecSourceAgent();
  virtual void init(int streamID, int N, int K);
  virtual void sendmsg(int nbytes, const char *flags = 0) {
    sendmsg(nbytes, NULL, flags);
  }
  virtual void sendmsg(int nbytes, AppData* data, const char *flags = 0);
  virtual void blockTimerHandler(int);
  virtual void command(int argc, const char*const* argv);
  virtual void dumpFinalStats(char *outputFile);
protected:
  int blockindex_;
  int numberBytesInBlock;
  int numberPacketsInBlock;
  int streamID;
  int seqno_;
  int blockno_;
  int N_; 
  int K_; 
  int typeFEC;
  int streamType;
  char traceString[32];
  FILE *fp;
  int state; // -1 uninited, 1 inited
  double blockTimerDelay;
  int numberFECPackets;
};
Application FEC Send Side

Packet header info
- Seqno
- blockNumber
- blockN_
- blockK_

Stream of UDP packets
- Break into blocks of size n

Video streamer
traffic generator

APFEC - coder

Stream including redundant information

constructor:
bind("N_", &N_); //tcl param
bind("K_", &K_); //tcl param
blockTimerDelay = 0.200;

void AppFecSourceAgent::sendFEC(int numberFECpackets)
* This method completes the FEC process for the current block
* If do not have a full block
* if fixedBlockMode_ is FALSE
* scale N_ and K_ (numberFECpackets might be reduced)
* else
* keep K_ and N_ to the configured values
* n = updatedN_ - updatedK_
* find avg size of data pkt sent in block (nBytesPerPacket = numberBytesInBlock / numberPacketsInBlock)
* For n packets
* { 
* create a packet (size : nBytesPerPacket)
* increment the blockindex_ (do not increment seqno_)
* setup the packet (rh->contentType = FEC_OVERHEAD)
* send the packet
* }
Application FEC Rx Side

blockNumber/blockIndex
1/6 1/5 ¼ 1/3 1/1

• Recv(pkt):
  * pseudo code (given packet pkt):
  * if configured for no fec
  *   pass pkt up
  * else if pkt associated with prior blocks
  *   throw out packet as it’s a dup or not needed
  * else if pkt is in the current block
  *   if pkt is a duplicate, just throw it away
  *   return;
  * if start of new block
  *   start block timer
  *   state = in progress
  *   insert in blockQ
  * if pkt is data, clone packet and passPacketUp(pkt)
  * if current block is complete (if at least K_ arrived)
  *   process FEC block
  * else if pkt is NOT associated with current block
  *   if pkt is a duplicate, just throw it away
  *   return
  *   insert in OutOfOrderQ
  *   if size of OutOfOrderQ exceeds threshold
  *   process FEC block //handles transferring Queues

Packet header info
• Seqno
• blockNumber
• blockIndex
• blockK_

APFEC Sink
Recv(pkt)
passPacketUp(pkt)
blockTimerHandler()
processFEC()
clearQueue(ListPtr)
transferQueue(srcListPtr, dstListPtr)

BlockTimerObject
Start(delay)
Stop()
Handler()

blockTimerHandler()
*numberBlockTimeouts++;
*processFECBlock();
clearQueue(ListPtr)

passPacketUp(pkt)
transferQueue(srcListPtr, dstListPtr)

blockQ_OutOfOrderQ
nextExpectedSeqNumber = 1
currentBlockNumber = 0
blockState = INITIALIZED
OutOfOrderQThreshold = 16 packets
blockTimerDelay = 0.500 sec
Application FEC Rx Side

*processFEC()
* while (blockQ->getListSize() > 0)
  *
  * stop block timer
  * set blockState to completed
  * get this blocks N and K
  * if N is 1, this is an error
  * if all N received
  * loop through the blockQ
  * {
  *   if an element is not yet sent up, clone it and passPacketUp(pkt)
  *   delete the element
  * }
  * if K or more were received
  * loop through the blockQ
  * {
  *   if the element is in order
  *     if it has not been sent up yet, clone it and passPacketUp(pkt)
  *     delete the element from the Q
  *     if an element is missing in the Q, regenerate and passPacketUp(pkt)
  * }
  * if less than K were received
  * loop through the blockQ
  * {
  *   if an element is not yet sent up, clone it and passPacketUp(pkt)
  *   delete the element
  * }
  * advance currentBlockNumber++;
  * reset block state variables such as numberPacketsInBlock and numberBytesInBlock
  * if blockQ not empty, throw HARD ERROR
  * if OutOfOrderQ not empty
  * transfer any of the next block (if it exists) from the Outoforder to the blockQ
  * if transferred > 0
  * start block timer
  * set blockState to in progress
  * Get current blocks K
  * if (blockQ->getListSize < K)
  *   break;
  * }
Focus on Validating the Application FEC traffic generator. There is a single unicast CBR application flow (CBR rate 3Mbps). The main experimental parameters:

- $N, K,$ and $q$ (redundancy factor: $N-k/N$)
- Loss process: for now limited to a simple bernoulli process
- Packet size: this is the packet size before overhead. Each packet overhead will include octets for the following headers
  - IP: 20 octets
  - UDP: 8 octets
  - RTP: 32 octets
# Results

## Experiment Definition

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Description</th>
<th>Efficiency (FEC)</th>
<th>Efficiency (Total)</th>
<th>Effective Loss Rate</th>
<th>Average Packet Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>N,k: 10,8, vary loss rate (p)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Vary N, hold k at 5, p=0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Vary (N,k) – increase blocksize, maintain q of 0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>N,k:10,8, vary packet size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Experiment Results:

- **Efficiency (FEC):** assesses efficiency of just FEC: \( \frac{\text{totalDataBytesPassedUp}}{\text{totalDataBytesReceivedBySink}} \)
  - Note: both stats include all packet overhead and therefore cancels out in the ratio
- **Efficiency (Total):** assess overall efficiency: \( \frac{\text{Observed Application Goodput}}{\text{Total raw BW consumed by flow}} \)
  - Note: goodput measure is application data and does not include packet overhead.
- **Effective loss rate:** Receiver’s estimate of loss count / Receiver’s estimate of total amount of data (including FEC) sent
- **Average packet latency:** end-to-end latency averaged for all packets in the flow
Results

Experiment 3.1: vary p

<table>
<thead>
<tr>
<th>(N,K,q)</th>
<th>Loss Rate (bernoulli)</th>
<th>IP Packet Size (bytes)</th>
<th>Efficiency (FEC)</th>
<th>Efficiency (Total)</th>
<th>Effective Loss Rate</th>
<th>Average Packet Latency (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,8, 0.2</td>
<td>0.05</td>
<td>1520</td>
<td>0.835</td>
<td>0.769</td>
<td>0.009</td>
<td>0.006743</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.10</td>
<td>1520</td>
<td>0.863</td>
<td>0.768</td>
<td>0.034</td>
<td>0.012766</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.15</td>
<td>1520</td>
<td>0.879</td>
<td>0.768</td>
<td>0.074</td>
<td>0.022222</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>1520</td>
<td>0.885</td>
<td>0.769</td>
<td>0.125</td>
<td>0.035516</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.25</td>
<td>1520</td>
<td>0.878</td>
<td>0.769</td>
<td>0.181</td>
<td>0.049428</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.30</td>
<td>1520</td>
<td>0.865</td>
<td>0.768</td>
<td>0.249</td>
<td>0.064520</td>
</tr>
</tbody>
</table>

Results:
• Efficiency (FEC): assesses efficiency of just FEC: \( \frac{\text{totalDataBytesPassedUp}}{\text{totalDataBytesReceivedBySink}} \)
  • Note: both stats include all packet overhead and therefore cancels out in the ratio.
• Efficiency (Total) : assess overall efficiency: \( \frac{\text{Observed Application Goodput}}{\text{Total raw BW consumed by flow}} \)
  • Note: goodput measure is based on application data and does not include packet overhead.
• Effective loss rate: Receiver’s estimate of loss count / Receiver’s estimate of total amount of data (including FEC) sent
• Average packet latency: average end-to-end latency averaged for all packets in the flow.
## Results

### Experiment 3.2: $k=5$, vary $N$

<table>
<thead>
<tr>
<th>$(N, K, q)$</th>
<th>Loss Rate (bernoulli)</th>
<th>IP Packet Size (bytes)</th>
<th>Efficiency (FEC)</th>
<th>Efficiency (Total)</th>
<th>Effective Loss Rate</th>
<th>Average Packet Latency (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5, 5, 0.000</td>
<td>0.20</td>
<td>1520</td>
<td>1.000</td>
<td>0.959</td>
<td>0.200</td>
<td>0.004685</td>
</tr>
<tr>
<td>6, 5, 0.167</td>
<td>0.20</td>
<td>1520</td>
<td>0.888</td>
<td>0.800</td>
<td>0.14</td>
<td>0.041270</td>
</tr>
<tr>
<td>7, 5, 0.286</td>
<td>0.20</td>
<td>1520</td>
<td>0.822</td>
<td>0.687</td>
<td>0.094</td>
<td>0.019975</td>
</tr>
<tr>
<td>8, 5, 0.375</td>
<td>0.20</td>
<td>1520</td>
<td>0.749</td>
<td>0.599</td>
<td>0.067</td>
<td>0.011193</td>
</tr>
<tr>
<td>9, 5, 0.440</td>
<td>0.20</td>
<td>1520</td>
<td>0.682</td>
<td>0.532</td>
<td>0.051</td>
<td>0.007785</td>
</tr>
</tbody>
</table>
## Results

**Experiment 3.3:** increase N,k, fixed q=0.2, p=0.2

<table>
<thead>
<tr>
<th>(N,K, q)</th>
<th>Loss Rate (bernoulli)</th>
<th>IP Packet Size (bytes)</th>
<th>Efficiency (FEC)</th>
<th>Efficiency (Total)</th>
<th>Effective Loss Rate</th>
<th>Average Packet Latency (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>1520</td>
<td>0.885</td>
<td>0.769</td>
<td>0.125</td>
<td>0.035515</td>
</tr>
<tr>
<td>15,12, 0.2</td>
<td>0.20</td>
<td>1520</td>
<td>0.890</td>
<td>0.768</td>
<td>0.122</td>
<td>0.041608</td>
</tr>
<tr>
<td>20,16, 0.2</td>
<td>0.20</td>
<td>1520</td>
<td>0.899</td>
<td>0.769</td>
<td>0.118</td>
<td>0.042752</td>
</tr>
<tr>
<td>25,20,.02</td>
<td>0.20</td>
<td>1520</td>
<td>0.900</td>
<td>0.768</td>
<td>0.114</td>
<td>0.051264</td>
</tr>
<tr>
<td>30,24, 0.2</td>
<td>0.20</td>
<td>1520</td>
<td>0.904</td>
<td>0.768</td>
<td>0.113</td>
<td>0.061059</td>
</tr>
</tbody>
</table>
## Results

### Experiment 3.4: vary packet size

<table>
<thead>
<tr>
<th>(N,K, q)</th>
<th>Loss Rate (bernoulli)</th>
<th>IP Packet Size (bytes)</th>
<th>Efficiency (FEC)</th>
<th>Efficiency (Total)</th>
<th>Effective Loss Rate</th>
<th>Average Packet Latency (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>316</td>
<td>0.854</td>
<td>0.625</td>
<td>0.126</td>
<td>0.009881</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>572</td>
<td>0.866</td>
<td>0.701</td>
<td>0.125</td>
<td>0.015243</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>828</td>
<td>0.871</td>
<td>0.731</td>
<td>0.124</td>
<td>0.020493</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>1084</td>
<td>0.876</td>
<td>0.747</td>
<td>0.123</td>
<td>0.025807</td>
</tr>
<tr>
<td>10,8, 0.2</td>
<td>0.20</td>
<td>1260</td>
<td>0.876</td>
<td>0.754</td>
<td>0.124</td>
<td>0.029866</td>
</tr>
</tbody>
</table>