# Broadcasting Video Content in Dense 802.11g Sports and Entertainment Venues

## Project Update

Last update: 9/24/2010

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

# Status Overview

- Analytic
  - Refer to the paper “Modeling Application Based Forward Error Correction”
- Empirical
  - Assessing and modeling the loss process in Clemson’s football stadium
- Simulation
  - Validating the APFEC simulation model
  - Validating the simulated loss process
  - Validating the multicast 802.11 model

DISCLAIMER: The results that are shown in this presentation are preliminary (we are still validating our models and methods).

# 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
- 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

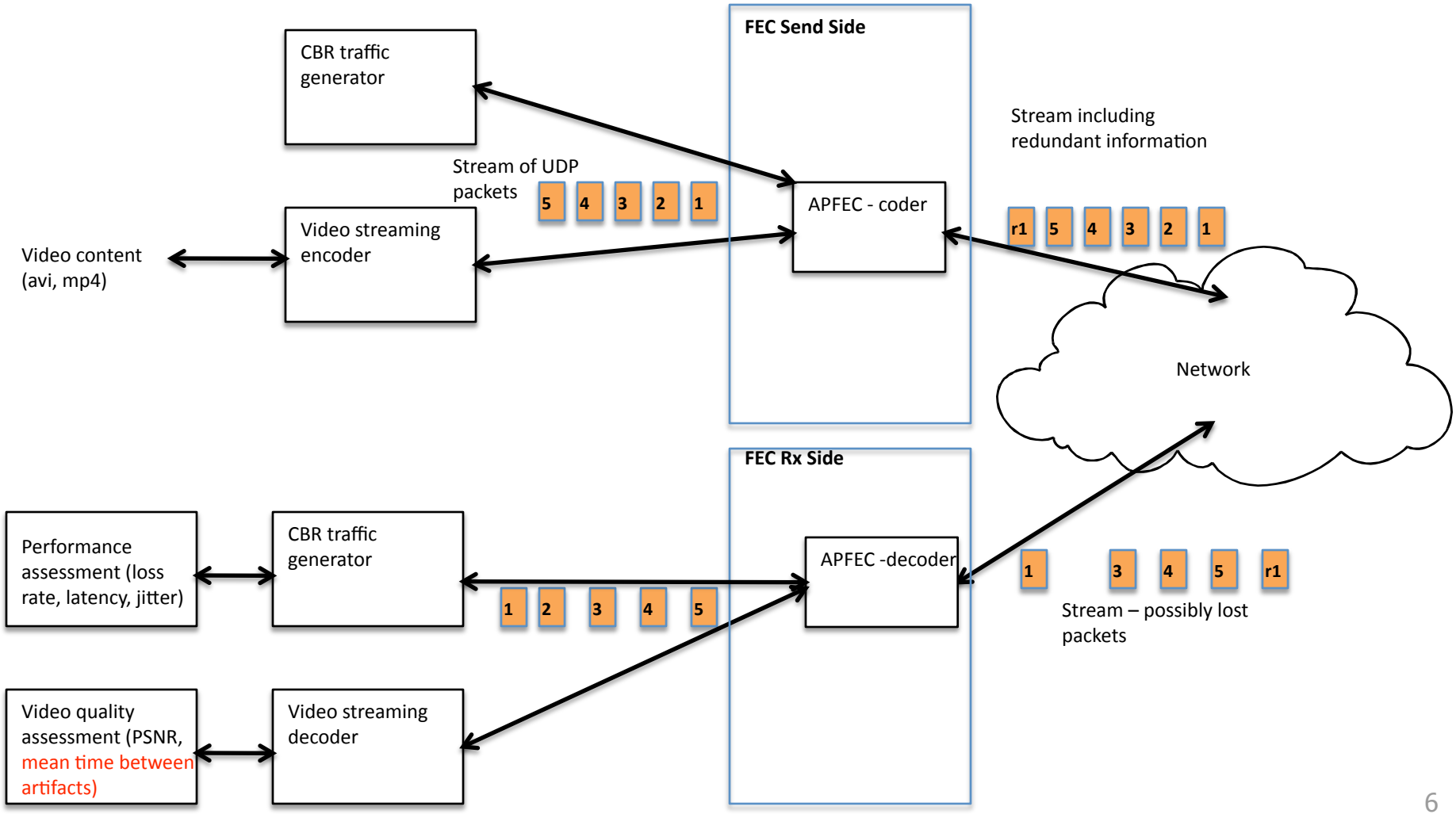
# Packet Loss Process

- Correctly modeling realistic loss process is crucial to our study. Two aspects:
  - Characterizing packet loss
  - Modeling a loss process in our simulation model
    - Artificially induced packet loss (model-based loss)
      - To help us study APFEC in a controlled manner, we can simplify the network to an ideal network (no loss, infinite capacity) and add model-based packet loss.
    - Propagation effects:
      - Packets are sent with a given Tx power level. They arrive at a station and are received with a Rx power level. The difference is caused by the path loss and fading propagation models that are active.
    - Congestion
      - Packets might be dropped by collisions over the channel. Or packets might be dropped if arriving traffic exceeds queue capacity of the radio.

# Highlights of "Modeling Application Based Forward Error Correction"

- For a given APFEC system parameterized by N,k
- Let r represent the redundancy (N-k)/N
- Let p represent the long term loss rate of the network
- Let p' represent the long term loss rate experienced by the flow AFTER APFEC
- Assuming ideal coding such as Reed-Solomon (RS)
- And if we assume uncorrelated packet loss in the network, we expect:
  - If r>p, then p' →0 as N goes to infinity
  - If r=p, then p'→p as N goes to infinity
  - If r<p, then p'->p/2 as N goes to infinity
- If we assume correlated packet loss, then it depends on the details of the process (e.g., the intensity and level of correlation or loss events).
  - In general, you want to operate the system so that r is larger than p…..
  - In addition, a larger block size offers better correction capability, however this causes packet delay to grow
  - Given that p is likely to change over time, the optimal system adapts based on current conditions

# Application FEC Model



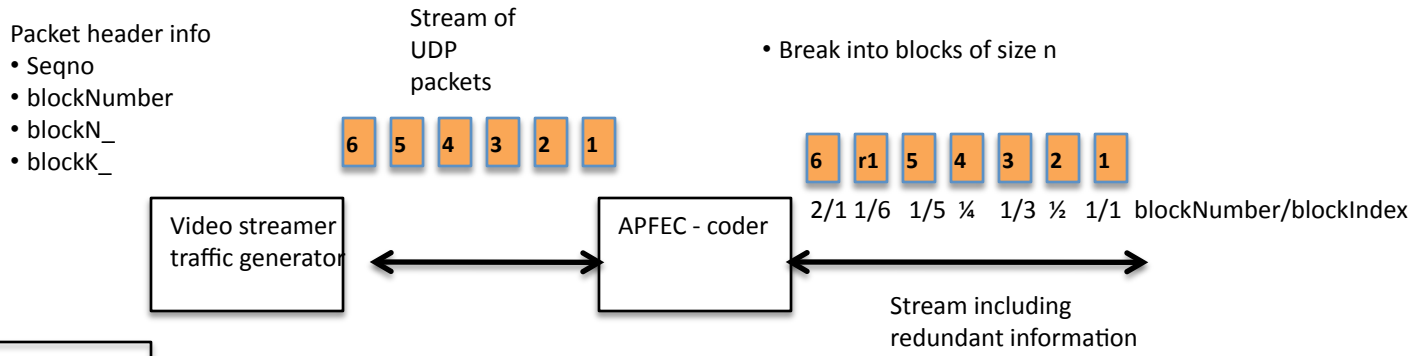
# APFEC Video Streaming

- 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 2EXP32). 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

# Application FEC Send Side

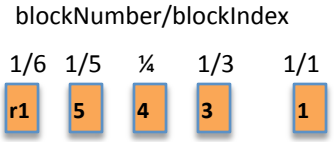



```
constructor:
 bind("N_", &N_);   //tcl param
 bind("K_", &K_);   //tcl param
 blockTimerDelay = 0.200;
```

```
* Function: void AppFecSourceAgent::sendmsg(int nbytes, AppData* data, const char* flags)
*
*  pseudo code:
*  set n to number of packets required to send the message
*  for n packets
*  {
*    if this packet is the FIRST data packet in a block
*      start the blockTimer
*
*    increment the seqno_
*    increment the blockIndex_
*    allocate and setup the packet (set contentType to VIDEO)
*    send the packet
*
*    if (blockIndex_ >= K_)  (if this packet is the LAST data packet in a block)
*      call sendFEC(numberFECPackets) to send the FEC data
•      stop the blockTimer
•    }
```

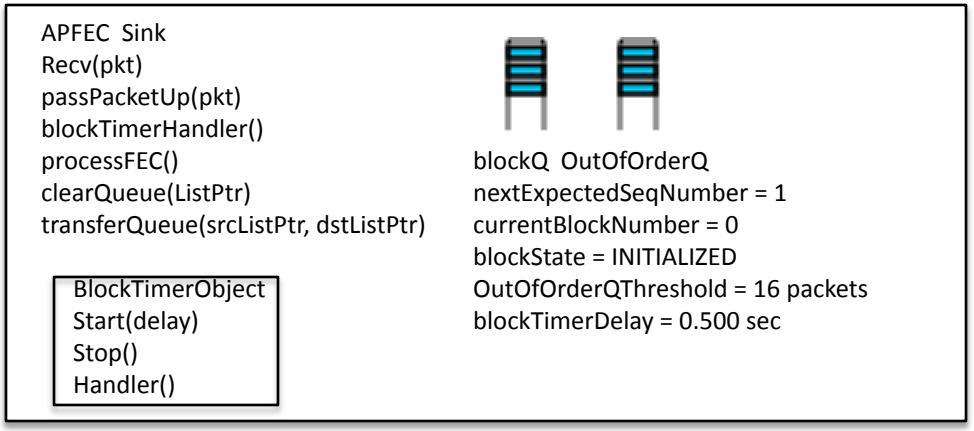
```
void AppFecSourceAgent::sendFEC(int numberFECPackets)
*This method completes the FEC process for the current block
*  If do not have a full block
*    if fixedBlockMode_ is FALSE
*      scale N_ and K_ (numberFECPackets might be reduced)
*    else
*      keep K_ and N_ to the configured values
*  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




Packet header info
- Seqno
- blockNumber
- blockIndex
- blockN_
- blockK_

- 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
- *




blockTimerHander()
*numberBlockTimeouts++;
*processFECBlock();

clearQueue(ListPtr)

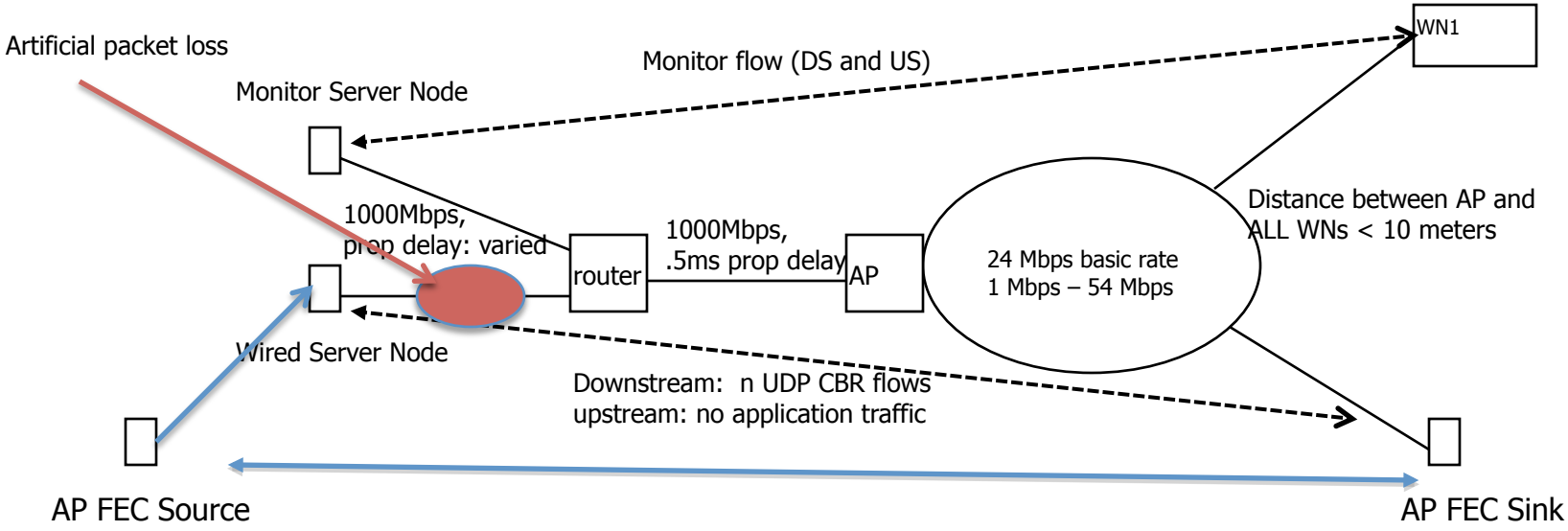
passPacketUp(pkt)

transferQueue(srcListPtr,dstListPtr)

# 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 transfered > 0
*        start block timer
*        set blockState to in progress
*        Get current blocks K
*        if (blockQ->getListSize < K_)
*         break;
*  }
```

# Simulation Model



# Experiment Definition: Simulation Parameters

- FEC: N, k
- Path properties (RF, artificial loss process, path loss, fading effects, channel interference)
- Stations: number, location, movement
- Workloads
  - Background traffic
  - Traffic under observation

- Experiment Notation:
  - EXP wxy.z (e.g., EXP311.1)
    - W: scenario that specifies top level scenario. This is used to group similar experiments
    - X: identifies dominant cause of loss process: Bernoulli, Gilbert-Elliot, RF propagation, or congestion
    - Y: An experimental parameter such as long term loss rate
    - Z: An experimental parameter such as FEC block size

# Experiment Definition

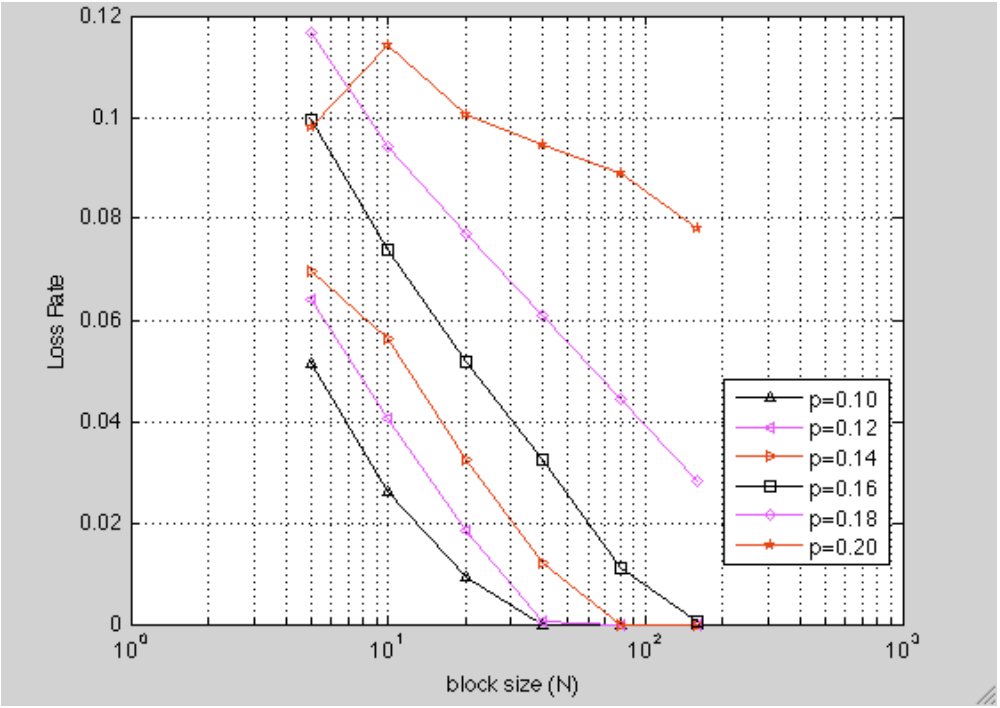
| Experiment | FEC parameters | Flow under observation | Background traffic | Loss process |
|---|---|---|---|---|
| 311 | k/N:0.8, vary blockSize, vary loss rate | 5 downstream CBR flows (3Mbps, packetsize 1460) | none | Bernoulli |
| 321 | k/N:0.8, vary blockSize, vary loss rate | 5 downstream CBR flows (3Mbps, packetsize 1460) | none | Gilbert-Elliot |
| 411 | k/N:0.8, vary blockSize, vary loss rate | 1 Multicast flow (3Mbps, packetsize 1460) | none | Bernoulli |
| 421 | k/N:0.8, vary blockSize, vary loss rate | 1 Multicast flow (3Mbps, packetsize 1460) | none | Gilbert-Elliot |
| 511 | k/N:0.8, vary blockSize, | 1 Multicast flow Variable Number of TCP Flows | Vary number of computing downstream TCP flows | Congestion |
| 512 | k/N:0.8, vary blockSize, | Multicast flow – vary number/ rates | 5 downstream TCP flows | Congestion |
| 611/612 | k/N:0.8, vary blockSize, | Multicast flow – vary RF factors | None/5 downstream TCP flows | Path loss/Propagation model |

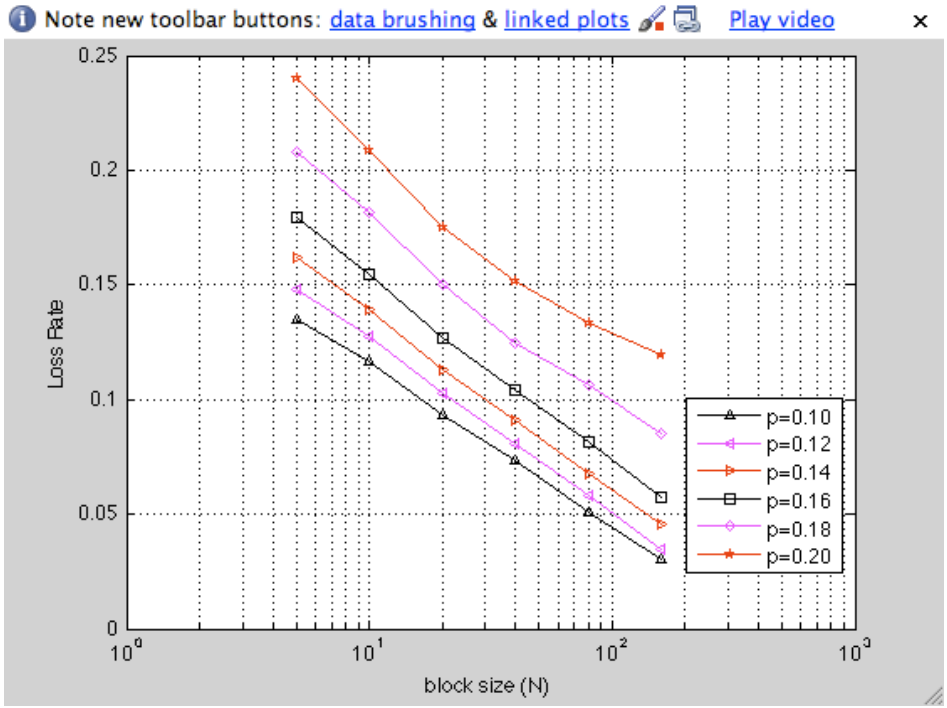
# Experiment Definition: Per Flow Metrics

- Coding efficiency: describes the FEC parameters as the ratio of k/N
- Effective loss rate: Receiver's estimate of loss count / Receiver's estimate of total amount of data (including FEC) sent
- Effectiveness: 1 – effective loss rate / true channel loss rate long term average
- Packet latency: end-to-end latency averaged for all packets in the flow
- Packet jitter: end-to-end jitter averaged for all packets in the flow
- Throughput: application throughput experienced by the flow

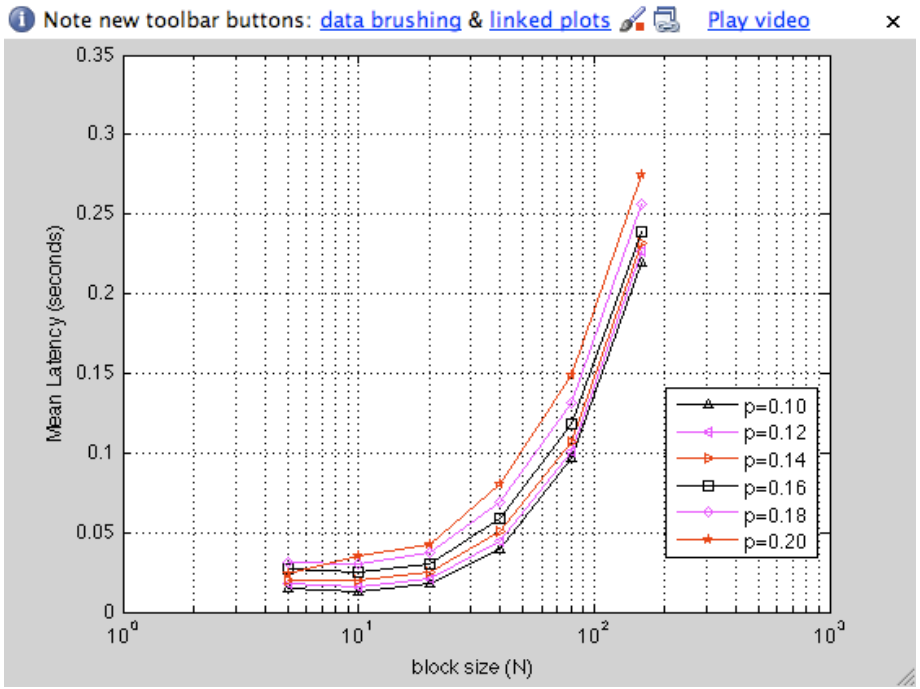
- Other metrics to be developed
  - Mean burst loss length (MBL) metric and interloss distance (ILD) metric as defined in the "Forward Error Correction Strategieis for Media Streaming Over Wireless Networks" paper (A. Nafaa, et. Al., IEEE COM Magazine, Dec 2007).
  - More accurate assessment of quality impacts on the streaming experience:
    - PSNR
    - Mean Time Between Artifacts
    - (1-alpha) tail of distribution of packet interarrival jitter samples

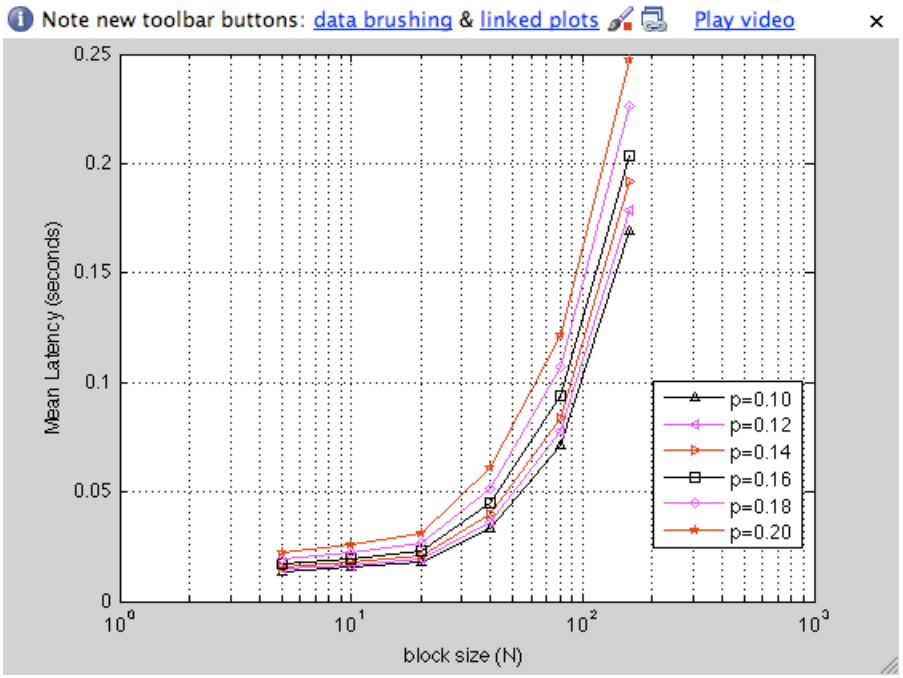
# Results: EXP311 and EXP321



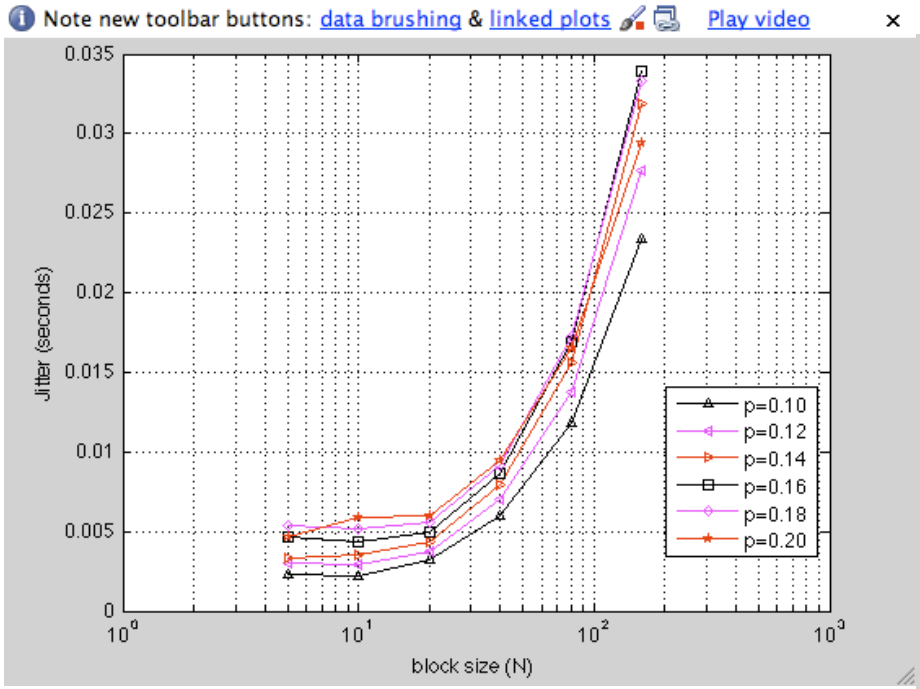



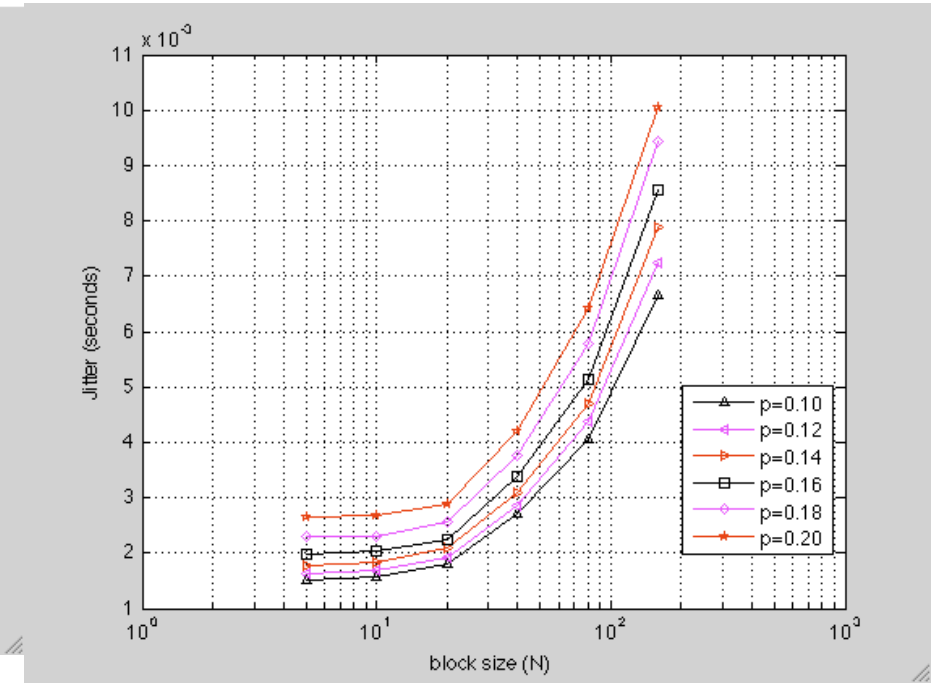
# Results: EXP311 and EXP321



# Results: EXP311 and EXP321



# Results: EXP311 and EXP321

