15-441: Computer Networks - Project 3
Reliability and Congestion Control protocols for MPEG

Project 3 Lead TA: Umair Shah (umair@cs.cmu.edu)

Assigned: Friday, October 18, 2002.
Due Date: Friday November 8, 2002.

1 Abstract

In this assignment, you are going to implement different reliability and congestion control protocols for MPEG-1 and analyze their performance. Please remember to read the complete assignment handout a couple of times so that you know what is being provided and what you are being assigned to do.

1.1 Suggested Checkpoints and Deadlines

Below is a table of *suggested* checkpoints. While it is not required that you meet all of the requirements by the specified dates, you should keep these deadlines in mind to gauge your progress. This assignment is due on **Friday November 8th by 11:59PM**. The late policy is explained on the course website.

Date	Description
October 16	Assignment handed out. **PLEASE START EARLY!**
October 20	Familiarize with the provided code and modify `sender` to receive ACKs.
October 24	Have your 100% reliability protocol running
October 28	Have your flow control protocol running
November 4	Analyze the performance and come up with your own best video protocol. Have it up and running.
November 6	Assignment due date. You will be turning in all of your source code as well as a Makefile and a tests.txt file explaining how you tested your program and your design of Problem 2 and 3. The handin directory is: `/afs/cs.cmu.edu/academic/class/15-441-f02-users/group-XX`.

1.2 Submission Checklist

- All your source code
- A Makefile
- A tests.txt file explaining how you tested your program and which files contain what code and what ack type you want us to use for the mpeg-client for different parts of the assignment. It should also describe briefly the rationale of your schemes for problem 2 and problem 3 and the performance metric you output in problem3.txt.
- Handin directory is `/afs/cs.cmu.edu/academic/class/15-441-f02-users/group-XX`

2 Where to get help

If you have a question, please please please do not hesitate to ask us for help, that's why we're here! General questions should be posted to the class bulletin board, **academic.cs.15-441**. If you have more specific questions (especially ones that require us looking at your code), please drop by our office hours. The TA's office hours are listed below: Since

TA	Office	Hours
Umair Shah	Wean 3710	Wednesday 2-3pm
Justin Weisz	Wean 3604	Tuesday 5-6pm
Xavier Appe	Wean 3108	Friday 2-3pm
Julio Lopez	Wean 8205	Monday 2:30-3:30pm

Umair is the Lead TA for this Project, he will hold extra office hours for this project on Thursday 2-3pm in addition to the normal office hours.

3 Project Outline

During the course of this project, you will do the following:

- Become familiar with MPEG framing and frame types
- Implement a 100% reliability protocol for MPEG streams modifying *only* the MPEG sender that we have provided you.
- Implement an MPEG rate control protocol
- Analyze the performance of the reliability protocols across different performance indices.
- Come up with your own protocol and its implementation that optimizes performance for the given performance metric.

4 Project specification

4.1 Background

MPEG stands for Moving Picture Experts Group. It is the name of family of standards used for coding audio-visual information (e.g., movies, music) in a digital compressed format. The main reason for its popularity over other video and audio coding formats is that MPEG uses sophisticated compression techniques to compress video into much smaller files. MPEG-1 represents the video in frames. The frames can be encoded in three types: intra-frames (I-frames), forward predicted frames (P-frames), and bi-directional predicted frames (B-frames).

In this project you will be implementing reliability and rate control protocols for an MPEG-1 streaming server. We have provided you with all the MPEG-1 parsing functions that you should need. However, the following resources may prove to be useful for your project.

1. **MPEG Video Resources and Software**
   `http://www.mpeg.org/MPEG/video.html`
   Very good links to all that you might want to know about MPEG video encoding.
2. **MPEG Video Software Decoder**
   `http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html`
   We have provided you with a modified version of this Berkeley MPEG Player.

3. **MPEG-1 Frames**
   `http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_overview.html`
   Simple explanation of MPEG-1 frame (I, B and P) encoding.

4. **How The MPEG Process Works**
   `http://www.pixeltools.com/pixweb2.html`
   A basic tutorial on how MPEG encoding works. Good to know but not really required for the assignment.

5. **MPEG Pointers and Resources**
   `http://www.mpeg.org/MPEG/index.html`
   Useful resources and links.

6. **XGraph tutorial**
   `http://www.eng.buffalo.edu/ tareen/projects/xgraphtut.html`
   A simple tutorial on using xgraph for plotting graphs. **xgraph** is a general purpose x-y data plotter with interactive buttons for panning, zooming, printing, and selecting display options. It will plot data from any number of files on the same graph and can handle unlimited data-set sizes and any number of data files.

4.2 Programming Guidelines

Your MPEG server must be written in the C programming language. You are not allowed to use any custom socket classes or libraries, only the standard libsocket and the provided libmpeg. C++ submissions will not be accepted. **Please note that unlike previous projects, you can only use the Andrew Linux machines (i.e. linux.andrew.cmu.edu), for compiling and testing your program in this project because we are providing you with a binary.** So you are responsible for making sure your program compiles and runs correctly on these machines. We recommend using `gcc` to compile your program and `gdb` to debug it. You should also use the `-Wall` flag when compiling to generate full warnings and to help debug. For this project, you will also be responsible for turning in your Makefile. A sample Makefile for this assignment is provided.

4.3 Provided Files

We have provided you with the following three programs:

- `mpeg-client` - A very basic streaming MPEG player.
- `router` - A simple wide-area network router with variable bandwidth and buffer.
- `sender` - An MPEG sender with limited capability as a starting point for this assignment.

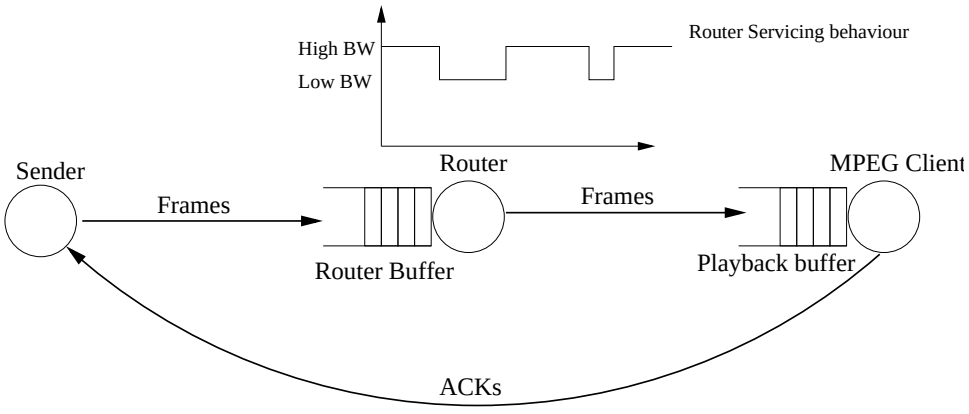

Figure 1: Topology for mpeg-client, router and sender

These programs work together as shown in Figure 1. The sender opens an MPEG file and sends it to the router. The router forwards the frames to the mpeg-client. The mpeg-client returns the ACKs to the sender.

4.4 The MPEG client: `mpeg-client`

We have provided you with an MPEG client (`mpeg-client`) that has the following capabilities.

1. It listens to a UDP port for receiving an MPEG file frame by frame.
2. It has two play modes. In the pause mode (enabled with -pm command line flag) it will pause to wait for the next in sequence packet. In the play mode (default) it will continue playing skipping missed packets.
3. It has a no display mode which does not show the frames of the movie. However it will open and initialize the window. This mode is useful if you want to try high frame rates and the X display is becoming the bottleneck. **You don't want to have a remote X display for the frame rates because it will generate uncontrolled delays for your system.**
4. It has a buffer to store incoming video frames before playing them out to screen at the specified frame rate. The buffer size can be changed by a command line option.
5. mpeg-client would exit after receiving the final frame or 20000 frames whichever happens first.
6. It has different acknowledgement mechanisms builtin which the user can choose using different command line options. These include
   - NONE: No feedback mode
   - ACK: send an ACK for the packet received i.e. given a packet with sequence number n, sends an ACK with sequence number n.
   - CUMACK: sends cumulative ACKs for packets i.e. on receiving a packet with sequence number n, it sends an ACK with sequence number k such that
     - **-pm flag** - all packets with sequence number $0...k$ have been correctly received.
     - **no -pm flag** - all packets with sequence number $l...k$ have been correctly received where l is the last frame displayed by mpeg-client.
   - SACK: sends a selective acknowledgement on receiving a packet i.e. on receiving packet with sequence number n, it sends an ACK with sequence number ($k : b_{k+1} : ... : b_{k+i} : ... : b_{k+16}$). So the higher order 16 bits contain the sequence number k calculated the same way as the CUMACK option and the lower 16 containing a bitmask where $b_{k+i} = 1$ when the $k + i$th packet has been received correctly by the receiver and 0 otherwise. Hence a SACK acknowledgement provides an acknowledgement for 17 packets from packet k to $k + 16$.

   Note: The sequence numbers are in frame numbers and not byte sequence numbers
7. It has a statistics module which measures various performance statistics and reports them at the end of the MPEG file stream
   - Total Dropped, Ignored(already played or duplicate) and Received frames as well as bytes for I, B and P frames (explained in Table 2).
   - Number of frames skipped/missed in the video.
   - Buffer queue length against time (produces bufqlen.txt)
   - Buffer Play/Pause state against time (explained in Section 4.4.1) - (produces buffer.txt)

Note: You can not modify the mpeg-client code. You can only alter its behaviour by using the different command line parameters.

4.4.1 How the playback buffer works?

The buffer has three states. `STOP, PLAY and PAUSE`. Initially it is in the `STOP` state. When it gets 50% full, it goes into the `PLAY` state and starts playing the frames at the specified frame rate. When the buffer becomes empty, it goes into the `PAUSE` state and stays there until the buffer gets 50% full. The buffer keeps all the packets **inorder** and plays them one by one according to the specified frame rate. With the `-pm` flag it will not play the next packet in the buffer (and remain in `PAUSE` state) until it receives the packet with the next sequence number. If the buffer does not have enough space and an earlier frame (i.e. earlier than some of the frames at the end of the buffer) is received, it is inserted into the buffer knocking out only enough of the later frames that make enough space for this frame.

4.4.2 `mpeg-client`: Command line options

Usage: mpeg-client [-a ack_type] [-l loss_rate] [-b buffersize] [-s address:port] [-p port] [-r frame_rate] [-nodisplay] [-pm] [-d]

Option	Description
-a ack_type	Specify the acknowledgement mechanism. ack_type can be one of the following : NONE, ACK, CUMACK and SACK. Default is NONE
-b buffersize	Buffer size in bytes to use for storing the video frames before playback. Default is 100,000 bytes.
-nodisplay	No Display mode is for disabling the X-windows displays of the frames. Note the window will still be create. Disabled by default.
-pm	Pause Mode enabled (disabled by default). In Pause mode it will wait for the next in sequence frame and not display the next in buffer frame. Otherwise by default it will skip missing frames.
-p port	Listen port for MPEG frames. Default value is 10002
-s address:port	The MPEG sender IP address and port number to use for providing feedback (ACKs etc.). Default values are 127.0.0.1:10003
-r frame_rate	Frame rate in frames/sec. This is the rate at which the frames will be played out during the buffer `PLAY` state. Default value is 1 frame/sec

Table 1: mpeg-client command line options

4.4.3 How the mpeg-client works?

The mpeg-client waits for MPEG data. The first packet it must receive is the initialization packet which contains information about the picture size etc. The initialization frame will not be lost or dropped by mpeg-client. In rare instances it can be dropped by the network and such instances will be ignored from the grading point of view. When it receives this packet, you should see the display window size adjust to the movie size. On receiving data, it first checks whether the packet should be lost or not based on the loss rate. If it is not lost the buffer size is checked if it can hold this packet. If the buffer does not have enough space it drops the packet. If the buffer has enough space, it adds it to the buffer and generates an ACK based on the specified ack model. Based on the loss rate the ACK can be lost as well. The buffer plays out packets based on its `STOP-PLAY-PAUSE` states and the specified frame rate. When the file finishes, it outputs the various summary statistics.

4.4.4 Performance Statistics

A brief explanation of the performance statistics being gathered is given in Table 2. Here is a sample output from the program.

```
Stats Frames Summary:
        Total   I       P       B
Total   477     306     169     0
Drop    0       0       0       0
```

```
Ignore  0       0       0       0
Recd.   477     306     169     0

Stats Bytes Summary:
        Total   I       P       B
Total   2876031 2208480 662332  0
Drop    0       0       0       0
Ignore  0       0       0       0
Recd.   2876031 2208480 662332  0

Video Play States time:
STOP:   1.190
PLAY:   56.546
PAUSE:  11.208
Missed Frames :         37
```

Statistic	Description
Drop/Ignore/Received Total, I, B and P frames	Displays number of frames(and bytes) of each type handled by the mpeg-client Drop = Dropped due to buffer overflow Ignore = Either duplicate or an earlier frame which can not be played Received = Frames played Total = All frames received (sum of all the above)
Missed frames	Frames skipped while playing a video.
Buffer queue length against time	Outputs the current time and buffer queue length in bytes at time interval k into the bufqlen.txt file where $k = 1/frame\_rate$. You can plot bufqlen.txt by typing "xgraph bufqlen.txt".
Buffer Play/Pause state against time	Whenever a change of state occurs, outputs an entry to the buffer.txt file. 1 represents the play state and 0 represents the pause or stop state. You can plot buffer.txt by typing "xgraph buffer.txt".

Table 2: Performance Statistics

In about a week we will also provide you with another index Video Quality Index (VQI) which you will have to use for Problem 3. The VQI would include different factors that affect video quality like the number of frames missed of different types and the time the player spends in pause state.

4.5 The Router: `router`

We have provided you with a simple router that has the following capabilities:

- It queues incoming packet into its fixed size buffer.
- It has two serving rates, a high one and a low one. After a random time interval it switches from high to low or low to high. If you want a constant rate, you can set the two to be equal.
- It has an incoming link latency which you can set.
- The router drops the whole frame when buffer overflows. This is a bit unrealistic because in the real world the frame will be fragmented into multiple packets by the IP layer. However for this project you should assume that the frame is contained completely in one UDP packet.
- It has the ability to dump all new (non-duplicate) frames' header data into a specified dump file.

- The router outputs the summary statistics of the frames and bytes received and dropped by it in a format similar to mpeg-client.
- Whenever it receives the "init" MPEG frame it resets all its statistics counters and starts the frame dump file again. Whenever it handles the "final" MPEG frame it displays its statistics. Hence you can run the router once and use it over multiple file runs.

4.5.1 `router` : Command line options

Usage: `router` [-f filename] [-p port] [b buffersize] [-s address:port] [-bl low_bandwidth] [-bh high_bandwidth] [-del delay] [-d] [-h]

Option	Description
-f filename	file for frame header output. If not specified then the header information of the frames is not output.
-d	Enable debug output. Disabled by default.
-b buffer	Buffer size to use for the router (bytes). Default is 80000.
-bh bw	Higher level serving rate (bytes/sec). Default is 100000
-bl bw	Lower level serving rate (bytes/sec). Default is 80000
-del delay	Link latency to be used in seconds. Default is 100msec or 0.1sec.
-p port	Listen port for MPEG frames. Default is 10001
-s address:port	The MPEG client IP address and port number to use for sending the MPEG file. Default values are 127.0.0.1:10002

Table 3: router command line arguments.

4.6 The MPEG sender: `sender`

We have provided you with a barebones MPEG sender. It has the following capabilities :

- It handles all the command line options required for this project.
- It can open the specified MPEG file and transmit it to a specified receiver over UDP.
- It can receive packets (i.e. the socket exists) from the mpeg-client but currently it does not have any packet handling capability.
- Currently it sends packets at a certain frame rate using the `usleep()` call which you obviously would have to change!!

You have to add all your code to this sender. **All your problem solutions should be in one program and not in 3 different programs**.

4.6.1 `sender` : Command line options

Usage: `sender` -f filename [-b buffer_size] [-p port] [-n problem_number] [-s address:port] [-r frame_rate] [-d] [-h]

4.6.2 MPEG functions library

We have provided some simple MPEG wrapper functions and structures for you which you should use for your assignment. These are:

- `init_mpegplay(filename)` : Takes the MPEG filename and generates the MPEG initialization frame.
- `create_message(framesize, ip, port, frame)` : Generates a message with sequence number and frame size information ready to be sent to the mpeg-client.

Option	Description
-f filename	MPEG file to send to mpeg-client
-n problem_number	0 - for best effort reliability protocol i.e. doesn't care about ACKs (default). 1 - for 100% reliabile protocol 2 - for sending rate adjusting protocol 3 - for your own reliability protocol minimizing the video quality index
-b buffersize	The playback buffer size being used by the receiver.
-p port	Listen port for MPEG frame ACKs. Default is 10003
-s address:port	The MPEG client IP address and port number to use for sending the MPEG file. Default values are 127.0.0.1:10001
-r frame_rate	Frame rate in frames/sec. This is the rate at which the frames will be sent in Problem 1. In problem 2 and 3 this option should be ignored.

Table 4: sender command line arguments.

- `getNextFrame()` : Returns the next MPEG frame from the file.
- `send_msg(mailbox, mesg)` : Sends a prepared message containing the current frame on the outgoing UDP socket (in the mailbox).
- `get_next(mailbox)` : Gets the next message from the mailbox.

We also provide you with the following data structures which are used by both the sender and mpeg-client. So you should not modify them but you have to understand and use them. These data structures are:

```
/* Defined in mpegplay.h */
typedef struct _mpegframe {
  unsigned int seq;     /* frame sequence number */
  unsigned int type;    /* ACK, I, B, P frame type */
  unsigned int size;    /* Size of mpegframe structure in bytes. */
  unsigned int frame;   /* First 4 bytes of mpeg frame data. */
} mpegframe;

/* defined in message.h */
typedef struct _message {
  char* body;           /* Will contain the frame */
  int len;              /* Length of the message */
  int ip;               /* ip address for the receiver */
  short port;           /* port for the receiver */
  struct _message* next;
} message;

/* defined in mailbox.h */
typedef struct _mailbox {
  int udp_in_port;      /* UDP listen port */
  int udp_out_sock;     /* UDP sending socket */
  int udp_in_sock;      /* UDP receiving socket */
  struct sockaddr_in mailbox_addr;
} mailbox;
```

5 Project Tasks

You can run the existing provided code as follows.

- In one window run ./mpeg-client. You might want to enable debugging with the -d option and also play around with the loss rate and frame rate. A window should pop up. e.g. ./mpeg-client -d -r 0.5
- In the second window run the router, e.g. ./router -d -bh 50000 -bl 40000.
- In the third window on the same machine run the sender. You might want to enable the debugging flag as well and also the frame rate. Specify the same frame rate for both (e.g. 0.5 or 1 frame/sec) and you should see the movie frame by frame. e.g. ./sender -d -r 0.5 -f mpegfile.
- Note: If all of you are using the same machine, you might want to use your own port numbers to avoid conflicts. A good scheme might be to use 101XX, 102XX and 103XX where XX is your group number.
- Follow the debugging messages and try to understand what is happening.

We have provided you with some sample MPEG-1 files. You can download others and test your stuff with them as well. Now you should be ready to tackle the problems. *Remember to include in the tests.txt file, the ack_type to be used for each of the assigned problems.* **Note: For all performance values we will look at the average of 5 runs because the values can vary over time. So you should also test your performance numbers multiple times before being satisfied.**

5.1 Problem 1

Implement a 100% reliability protocol for MPEG files. You would want to enable the -pm flag of the mpeg-client so that it pauses and waits for the next in sequence frame. Test your solution under varying playback buffer sizes and frame rate conditions. Assume that you know the player frame rate and playback buffer size. We will announce in about a week the range of parameter values that we would use to test your solution.

5.2 Problem 2

Implement an MPEG rate control protocol. Basically assume your sender does not know what the router buffer size is and the bottleneck bandwidth. The sender should try to adapt its sending rate according to the network congestion.

1. Provide an estimate of the router link bandwidth from your sender code. You should implement a window-based congestion control algorithm to adapt the server transmission rate to the link bandwidth. You will probably want to start with TCP congestion control but may want to optimize the congestion control for this setup. Your program should generate an xgraph input files problem2a.txt - showing how your bandwidth guess varies over time.
2. **For extra credit,** estimate the router buffer size. You will want to do this using your congestion window, bandwidth estimate from the previous part and RTT estimate. Your program should generate an xgraph input file problem2b.txt - showing how your router buffer size guess varies over time. In tests.txt you must record the formula you are using to estimate the buffer size and how you are measuring the terms in your formula.

Set the low and high bandwidth of the network to the same value for this part so that you can model a constant bandwidth link. Test your solution under varying buffer sizes and frame rate conditions. We will announce in about a week the range of parameter values that we would use to test your solution.
Record your protocol design in the tests.txt file. We will announce in about a week some benchmark target results that your protocol should try to achieve.

5.3 Problem 3

Now come up with your own reliability and flow control protocol that provides the best value for the Video Quality Index (VQI). The VQI as outlined in Section 4.4.4 depends on the relative importance of different video frames and the amount of time the player spends in pause state. Assume that you neither know the router buffer size nor the bottleneck bandwidth at the router. We will announce in about a week the range of parameter values that we would use to test your solution.

Your program should generate an xgraph input file (problem3.txt) showing some appropriate graph that demonstrates your solution. As an example it can show how your available bandwidth guess varies over time if you think that the available bandwidth guess is the most important performance metric for you. Record your protocol design in tests.txt and an explanation of your graph. We will announce in about a week some benchmark target results that your protocol should be able to achieve.

5.4 Some hints for 2 and 3

In designing the solution to Problem 2 and 3 you might want to think about the following issues:

- To prevent overflowing of buffers you might want to control your sending rate. So you might want to pace your sending rate like TCP does using ACK clocking.
- You might want to have a variable window size rather than a fixed one so that you can adapt to the network congestion control.
- You might want to choose which frames are more important to send across. The buffers have limited space and so you would like to use that space optimally. So for example if the link bandwidth is too low to support the desired frame rate, you might want to send only I frames since they are more important. Similarly since you know the playback buffer size, if you can predict that if I send this packet right now, it will arrive too late for the player you might want to skip it anyway.
- You might not want to retransmit a packet too many times for two reasons. It will either create more congestion in the network or it might already be too late to send it because the player might already have played frames in future of this frame.

6 Testing

This section gives you some tips about how to evaluate and test your project and some basic stuff that you should be on the lookout for while writing the code.

6.1 Testing Tips

The goal of this section is to give you a better idea of how your project will be evaluated, and what you should be on the lookout for while writing your code. The following list in not guaranteed to be exhaustive:

- Try a wide range of values for loss rate, frame rate and buffer size.
- Use -d debugging flag to make sure what is being output makes sense.
- Try different MPEG files. Different files have different B, I and P frame distribution.
- Use the functions we provide you.

6.2 The tests.txt File

As part of this assignment, you are responsible for testing your submission and making sure everything works. You should document all of your testing strategies in a file named tests.txt and submit it along with the rest of your assignment. This file is worth 15% of your grade so make sure you don't miss anything. Here is a minimal set of things we would be expecting in the tests.txt file.

- Summary of your submitted code files i.e. which files have what functionality.
- -a ack_type to be used for Problem 2 and 3.
- any other command line arguments that you want us to use for Problem 1, 2 and 3.

- Design of the protocol for Problem 2 and 3.
- Explanation of the graph for Problem 3.
- Your testing strategy

Also, if you have any outstanding issues you know about, but are not able to fix, you should let us know about them and include them in tests.txt. Or, if you have any comments about this assignment, felt that it was too hard or too easy, please let us know.

7 Grading Criteria

When we grade your submission, we will run `make` to compile your code and then run a combination of tests and grading scripts to evaluate your submission. If your project generates compiler warnings during compilation, you will lose credit; if your server dumps core during our testing, you will lose substantial credit. Remember to compile with the `-Wall` flag, otherwise you will lose points if this is not in your Makefile. Poor design, documentation, or code structure will probably reduce your grade by making it hard for you to produce a working program, and making it hard for us to understand it. Remember the 15-212 slogan: *Think before you hack!*. Egregious failures in these areas will cause your grade to be lowered even if your implementation performs adequately. Putting all of your code in one module counts as an egregious design failure. It is better to have partial functionality working solidly than lots of code that doesn't actually do anything correctly.

The following is a checklist of all of the features that your mpeg-server must possess. As you write your programs, you should cross out each feature from the list once you have implemented and tested it. All of these features will be tested during grading.

Problem	Feature	Grade	
Problem 1			30
	Handle ACKs	10	
	Reliability	20	
Problem 2			30
	Flow Control	25	
	problem2a.txt	5	
Problem 3			25
	VQI optimized solution	20	
	problem3.txt	5	
tests.txt			15
	Design of Problem 2	5	
	Design of Problem 3	5	
	All Other Stuff	5	
Total			100
	Extra Credit (problem2b.txt)	10	
Net			110

7.1 Turning in

Below is a listing of all of the files you should submit. The handin directory is `/afs/cs.cmu.edu/academic/class/15441-f02-users/group-XX`, where XX is your assigned number on the course website (see the pictures page).
Please note that we will be checking timestamps in order to determine late submissions. Check the late policy on the course website and notify the TAs and professors **only** if you will be submitting this assignment late.

- Makefile

- All of your source code (files ending with .c or .h only, no .o files)
- Documentation of your test cases and any known issues you have in the file tests.txt. It should also explain which files contain what code, the ack_type to be used for running the mpeg-client and mpeg-server for problems 2 and 3, a brief description of your scheme for problem 2 and 3 and the performance metric being output in problem3.txt.

8 How to succeed in this assignment

Above all, as always please start early. We will be able to help you more if you start early. You will feel better if you start early. You will feel even better if you finish the assignment before it is due. Remember, lots of bad things can happen at 11:50pm the night the assignment is due. Keep in mind that this is the third of four assignments, and doing well on this assignment will ease your mind later on in the course.

Now that you are resolved to starting this assignment early, read over this handout three times. If anything is unclear, please send email to the bboard (academic.cs.15-441), or come to office hours. We are here to help!
Finally now that you understand the assignment, here is a suggested plan of attack. Note that we will not hold you to this outline. This is solely for guiding you through the assignment.

1. First try to understand the code that is provided to you. Look at sender.c and *.h files to understand the functions being called and the data structures being used.
2. Run some simple examples with the debugging flags switched on, on both the sender and mpeg-client side. Look at the messages to get a feel of what is happening.
3. Modify sender.c to implement receiving of ACKs. This should be pretty straightforward. You can try the simple ACK option first. Then you can modify it to work with CUMACK or SACK option.
4. Now try to implement 100% reliability given the ACKs you receive. Again if you might want to use the simple ACK option first and then try CUMACK and SACK options. Play around with loss rate, buffer size and frame rate to see the performance.
5. Now go to Problem 2. Start with implementing a TCP style congestion control mechanism. Then look at the performance and try to see where TCP is doing poorly. Think about how can you get an estimate of the available bandwidth using TCP style congestion control? Now think about how you can optimize the congestion control protocol for this. Write and document it in tests.txt as you think and keep on improving it.
6. For problem 3, once we give you the performance index VQI, think about what you would do to minimize it? Go over the hints given in Section 5.4. Implement those which seem easier first. Now think about what performance index would you want to measure for this part? Then implement that. Now think about how you can go about improving this? Try to minimize first a component of the index. Then try to minimize the other components of the index. Think about how would you minimize the components together. Implement and test them.
7. Take a deep breath and turn in your assignment. Congratulations! you have finished an MPEG-1 streamer.