Overview

There are actually three possible projects:

- create a movie server and a client that interacts with it (this is the “basic” project); then add one improvement from the list given below.

- do the “basic” project from the previous item, but instead of adding an improvement, create at least three original MPEG movies for use with the server (see below).

- negotiated project — propose either a variation of the project above, or a completely different network-related project, and get my approval for it.

0.1 “Basic” project

We saw (in the last lab and in the following class) that it is possible to interact with the VideoLAN program by directly piping an MPEG TS stream into the vlc program, like this:

```
java MPGClient | vlc -vvv stream/ts:///dev/stdin
```

where “MPGClient” is a program that prints out the contents of a sequence of UDP packets to standard output. (In lab 9, these UDP packets were sent from a separate server, MPGServer.)

It is possible to connect vlc directly to a stream of UDP packets using the following setup:

**Source:** MPGServer — located at host1; sends a stream of UDP packets (containing MPEG-TS data) to port 12346 of host2

**Destination:** — vlc, running on host2, receives a UDP packet stream at port 12346 and displays the movie. Command:

```
vlc -vvv udp://@:12346
```

Here, then, is the “basic” project: you will write a pair of Java programs. One, MPGClient.java residing at host2, will connect via TCP to another, MPGServer.java, at host1, and issue commands. MPGServer will then carry out the commands received via TCP from MPGClient. Valid commands are:
• **list** — server must respond, through the TCP socket, with a series of lines, each line containing the name of a movie. The list should be followed by a line containing the word “end”. The client should display this information on standard output.

• **play moviename** — the server will send a stream of UDP packets directly to a `vlc` process running on the client’s machine. (For purposes of this project, the `vlc` program may be started “by hand” before running the client, using a command similar to that given above; in other words, neither client nor server has the responsibility for “starting” `vlc`.)

• **quit** — client exits. (The server does not exit; in fact, it should be possible to start up another client instance and have it interact with the server.)

The client is responsible for validating commands (the server should never receive an illegal command).

The server should have at least three movies; they should all be in MPEG-TS format (if you are using the movies from the web site I showed you — “Crash cars,” etc., you should convert them from PS to TS as shown in class).

When you test your program you will probably do so as follows:

• In one terminal window, start `vlc` reading from a `udp:` source and using a specified port, e.g., 12345

• In a second terminal window, start your `MPGServer` with a known port number, e.g., 12346

• In a third terminal window, start your `MPGClient` and enter commands there, e.g., “list”, “play mummy”, etc.

**Improvements**

If you do the first option for the project, add one of the following improvements to the basic project above:

• Implement a command of the form “down moviename”. The server should send the movie to the client, via TCP, and the client should save it in a local file. The movie will not play. Note that you will be sending the file via TCP, not as a stream of UDP packets, and that the recipient will be the client program and not the `vlc` program.

• Allow multiple clients to simultaneously interact with the server. This will require the client to tell the server the port number of the associated `vlc` process. You will need to separately start up `vlc` with a different port number for each client and transmit the port to the server; you should be seeing two (or more) movies playing in two separate windows.

• Add at least one other enhancement to the client or server — obtain permission from me for this enhancement
Second Option

If you choose instead to do the second form of the project, you should create three substantially different MPEG files “from scratch” by creating a sequence of .ppm files and making them into MPEGs using a command like `ppmtompeg`. Each should consist of at least 300 frames; each image should be at least 50 by 50 in size. By “substantially different” I mean that you should not be able to make a trivial change in the code for one MPEG in order to create another.

Suggestions: for one of the MPEGs, create an artificial geometric animation like my “moving line” animation. For another, read in an existing PPM file and make gradual changes to it (e.g., each frame changes the “red” value by one each time). For the third, perhaps create a transition between one PPM image and another by taking weighted averages of the bytes in corresponding positions, where the weight changes with each frame.

Third Option

Propose a substantial project, either a programming project or a survey project (preferably one that has a “hands-on” component, e.g., development of a program or use of a network related utility). You must get permission from me for such an alternative project. If you choose this option, you will be required to give a formal presentation to the class on the final day of classes. Suggestions:

- running wireshark on your personal laptop, illustrate one of the network protocols that we did not spend much time on in class by dissecting a sequence of packets and pointing out all of the relevant fields related to the protocol and its relationship to other protocols and network behavior. Such a presentation would likely consist of both a PowerPoint presentation explaining the details of the protocol and a live demonstration, via packet captures, of the protocol in action. http://netpbm.sourceforge.net/doc/ppmtompeg.html

- Investigate multimedia networking further than our brief look at MPEG files; perhaps demonstrate via experiments some of the performance characteristics of different multimedia transfer protocols and research the finer points of some of these protocols.

- Write a simulation of some protocol in the book, e.g., a simulation of slotted ALOHA with a calculation of the efficiency over many random rounds of transmission, for varying numbers of hosts in the network. Compare these results to those in the book.