Assignment 2

CS414, Multimedia Systems (Instructor: Klara Nahrstedt)

Posted: February 9, 2009

This is the second assignment for your Peer-to-Peer Streaming (p2-stream) project where you start to build the network and P2P fabric of your video-on-demand (VOD) server with your group members. This assignment will have three goals: (a) bootstrap the p2-stream system, i.e., register p2-stream client, setup a session, build the network connectivity among the peers, and build the network connectivity between the p2-stream client and the p2-stream service, (b) build distributed streaming on top of the network fabric implementing PLAY operation from peers to p2-stream client, (c) work with buffer management needed for PLAY operation. All assignments will be implemented on the Linux Dell Machines in 0216 SC lab.

Assignment Description:

1. P2-Stream Architecture

The P2-stream System will consist of four peers (P1, P2, P3, P4), where P1 peer is also the video server where the session management and video/movie database will reside, P2 and P3 are proxy peers where some of the cached requested movie content will reside and P4 is the client peer where the streamed content/movie will be displayed. The p2-stream architecture is shown in Figure 1. In this MP, we assume that one chunk has one ffmpeg packet.

![Diagram showing the p2-stream architecture](image)

Figure 1: p2-stream architecture

The peer P1 will include two major functions: (a) session control (session setup, session management) and (b) streaming of requested video to proxy peers P2 and P3. We will assume that the video server includes a set of movies (videos, stored in MJPEG format – so you need to
convert your video to mjpeg format). So the control functions and data plane functions that will implement through MP2, MP3 and MP4 are shown in Figure 2.

<table>
<thead>
<tr>
<th>Data Plane</th>
<th>Control Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream (Play, Fast Forward, Rewind) Video</td>
<td>Keep state about streams within session man.</td>
</tr>
<tr>
<td>Setup and maintain connections between peers</td>
<td>Session setup – register every peer with session manager (keep peer list) and keep list of movies</td>
</tr>
<tr>
<td></td>
<td>Session update - new peers</td>
</tr>
<tr>
<td></td>
<td>Leave or join, new files, what chunks are distributed</td>
</tr>
<tr>
<td></td>
<td>UDP/TCP/IP protocol stack</td>
</tr>
</tbody>
</table>

**Figure 2: Protocol Architecture on Client/Peer/Server to implement the P2-Stream Functions**

To achieve this type of architecture, in this MP2, you need to build step by step various protocols on top of the TCP or UDP/IP network protocol stack as shown in Figure 2. You can choose to work with either TCP or UDP sockets.

**2. Building Control Plane for P2-Stream Service**

**2.1. Establishing Connectivity among peers (Bootstrapping the system)**

To bootstrap the p2-stream system, you need to execute the following steps:

1. Start the Video server peer P1 and its session manager (as shown in Figure 3).
2. Each other peer P2, P3, P4 connect/register to P1 server. P1 session manager component will set up a peer table who is currently participating in the P2P network (in this MP, you do not need to maintain the topology of the network).
3. Session manager also initializes the (a) chunk table to start to keep track which peer has (caches) what chunks, and (b) the movie table to keep track what movies are currently available for P2P streaming. Note: you can maintain a single table to keep track of the movies available and what chunks of what movie are distributed/cached at what peer.
4. Once the registration, connectivity between P1 and P2, P3, P4 are finalized, you should pre-fetch few chunks to peer P2 and peer P3 from P1 (we will talk about streaming below). **Note:** since in MP2 we are not yet building a full-fledged P2P system, we need to warm up the p2-stream system and move few chunks to peer P2 and peer P3. We will assume (as shown in Figure 3) that the peer P4 will display video. Peers P2 and P3 will be proxies caching chunks and forwarding them to peer P4 upon request. Your chunk table should keep track what chunks are pre-fetched from P1 to peer P2 and what chunks are pre-fetched from P1 to peer P3. I would recommend that you move 1 second of chunks to peer P2 and P3, i.e., if your recording rate is 20 frames per second, and 1 a
chunk is one ffmpeg packet, then you should have at least 10 chunks at peer P2 and at least 10 chunks at peer P3. Notice that here we don’t require peer P2 to cache odd chunks and P3 to cache even chunks, but they should keep different chunks.

2.2 Implementation of Request for PLAY Function

If peer P4 wants to play a movie, it first needs to issue a request to the session manager at P1. Note, that the session manager address is widely known to start the overall p2-stream service (e.g., via websites). To implement the request for video, the following protocol could be implemented:

1. Peer P4 sends a message to peer P1 that hosts the session manager (P4 knows the IP address of P1 ahead of time). The message should include (a) the name of the movie, (b) operation type – in our case PLAY, and (c) duration how long you want the movie to be played.
2. Peer P1 listens on a specific port dedicated for incoming requests and once the request arrives, message is forwarded to the session manager.
3. Session manager looks up in the movie table if the movie is available, and it looks up in the chunk table if this movie is already cached in other neighbor peers to peer P4. Note: Since in this MP2 we assume that the neighbor peers for P4 will be P2 and P3, and we have already pre-fetched few chunks to P2 and P3 (see section 2.1), then the session manager will create a response message to P4, informing it if the movie is available, and if it is available what are its neighbor peers (e.g., P2 and P3) and what chunks they have. In our case the neighbor peers are P2 and P3 and chunks will correspond to 1 second. In

Figure 3: Role of Session Manager during establishment of the p2-stream session.
particular, P1 will return the IP addresses and ports of the sockets at P2 and P3 so that P4 can contact them for the video content.

4. Session manager sends the message out to P4, P4 receives the message and establishes connectivity to peers P2 and P3. Note: The session manager should also send to P4 the recorded rate of the movie so that the playback/display on the client P4 site can be prepared for the display rate (the goal is that the ‘display rate = recorded rate’).

5. When the client is done playing for the video duration, it sends a “stop” message to all peers P1, P2, P3 to ensure graceful stop. This means that all movie buffers at P1, P2, and P3 should be emptied, chunk table updated, and any other bookkeeping metadata updated at the P1 session manager and at the proxy peers P2 and P3.

3. Building Data Plane for p2-stream Service

To build up the PLAY streaming service and deliver video to P4 in smooth fashion, we will use two types of modes of operation (see Figure 1 as well).

3.1 Push-based streaming approach

During the setup, few chunks were pre-fetched from P1 to P2 and P3. However, during the streaming phase when peer P4 should receive chunks from P2 and P3, new chunks have to arrive from P1 to P2 and P3. In the demonstration of the MP2, we will start the peers P1, P2, P3 first so that we can prefetch some video from P1 to P2 and P3. Then, we start P4 and display video content at P4. We recommend that you setup a UPD/IP connection between P1 and P2/P3 and stream chunks between P1 and P2/P3.

(a) Streaming: means that P1 will push independently chunks of the requested movie to P2 and P3, i.e., P2 and P3 do not have to request each chunk. It is requested that you implement a rate-control when you push chunks into the network from P1 to P2 and P3.

(b) Rate Control: means that P1 will not send the chunks to P2 and P3 as fast as possible, but it will pace itself. It means that it will send a chunk, and then sleep (use usleep command or SDL_Delay()), and then wake up and send another chunk, etc. For example, if the frame rate of your video is 20 frames per second and 1 frame=1 packet=1 chunk, then you consider chunk sending rate 20 chunks per second and send one chunk, then sleep for (slightly less than or equal to ) 50 ms (since we have network transmission delay), then again send another chunk, etc. Since you have two peers (P2 and P3) to stream data from P1 (see Figure 1), you will have half of the rate for streaming P1-P2 and half of the rate P1-P3.

(c) Multi-Threading at peer P1: In peer P1, you should have one “file reading” thread reading from the requested movie and write into a buffer. Then you should have two “network” threads that are reading from the buffer to the two UPD/IP sockets that lead to peers P2 and P3. Note: You cannot (should not) decode (decompress) the MJPEG packets in the “file reading” thread since you don’t want to send uncompressed video over the network. This defeats the purpose of compression. You want to have the data compressed until you reach peer P4. Note: you must have a buffer per movie. The “file reading” thread is shared thread per movie (i.e., if we will have multiple clients in later MP4 that request the same movie, the “file reading” thread will read the chunks from the movie file). There will be multiple networking threads. There exist multiple possible designs: (1) you can have one network
thread per peer (one network thread reads from multiple movie buffers), (2) you can have one network thread per peer and per movie.

(d) **Multi-threading at peer P2 and P3:** In peers P2 and P3, you should have one “receiving” thread reading from the socket that connects you to P1, and which writes into a joint buffer, and one “sending” thread, reading from the movie buffer to socket that connects P2/P3 to P4. Note: If we have a full-fledged P2P system, there will be n “receiving” and m “sending” threads depending to how many peers one is connected. In our MP2 case, since P2 and P3 are connected only to P1 and P4, and the traffic flows from P1 through P2/P3 to P4, you will have one receiving and one sending thread.

(e) **Buffer size at P2 and P3:** Assume that there is unlimited buffer at P2 and P3.

### 3.2 Pull-based approach

Once the chunks are streamed via push-based approach from P1 to P2/P3, they are cached in the movie buffer at P2/P3. Now, instead of pushing the chunks to P4, P2P systems at the final client destination pull the chunks. It means, peer P4 will send request to P2/P3 and request each chunk for display. Upon request of a chunk, P2/P3 send one chunk and wait for the next chunk request.

(a) **Multi-threading at P4:** You should use the Case 2 from MP1, since you should have two “network” threads communicating with P2/P3 and getting the chunks into the chunk buffer, and then you should have one “display” thread that will take the chunks from the buffer, decode the frames in the chunk and display them.

(b) **Scheduling at P4:** At the beginning, you will get from P1 the information what chunks which peer has. As you start the display, you should keep communicating with P1 control connection what chunks are in P2/P3. Once you have the information where the chunks reside in either P2 or P3, the peer P4 will make request for corresponding chunks at corresponding peers and put them in the buffer and display.

(c) **Buffering at P4:** we will explore two cases. One case will be that the chunk buffer at P4 is unlimited, the second case will be that the chunk buffer at P4 can only hold 2 second of data for display.

### 4. Implementation Cases for data plane

You will build MP2 in steps and implement three cases:
- **First case:** implement push-based streaming between P1 and P2. You should be able to stream the data, store the chunks at P2 and then display the received data (at peers P1 and P2 you should be able to use the Case 1 multi-threaded implementation from MP1)
- **Second case:** implement push-based streaming and pull-based approach between P1, P2, P3, P4 as discussed in Section 3 under the assumption that all buffers are unlimited at P2, P3, and P4.
- **Third case:** implement push-based and pull-based approaches between P1, P2, P3, P4 as discussed in Section 3 where P2, P3 have unlimited buffers, but P4 has a limited buffer of 1 second.

### 5. Comments:

- For the limited buffer, use circular buffer.
- Use mutex, condition variables to synchronize threads
- Define your communication packet format carefully so that you can send different control packets by using the same packet structure
- You must get the frame rate from the video file and display at P4 according to this rate (convert your video file to mjpeg)
- The PLAY request should be started from the client P4 using a command line as we had in MP1. Note that in addition to the PLAY operation, name of the movie and duration of the movie, you may also need to include server P1 address to starts operation, so that you can start the whole process of session connection/negotiation and movie streaming.

**Delivery**

Each group delivers:

- source C, or C++, or Java of your **p2-stream** program in the particular group directory. The source code evaluation will be based on how well is your code documented. If you use some code you found on the web (You must understand the code you found and include in your code, not just blindly copy the code!!!) or in a local system directories, document it. It is very important that you give credit to people who developed the previous code. Your own code should include the following information at the beginning of each C/C++ /Java file.
  - Each major source file should include
    - File Name: Name of the File
    - Description: Short description what the file includes (general description, what kind of functions are embedded in the file).
    - Version: version of your code. You start with version 0 and as you improve the code, at some point you increase the version.
    - Programmer’s Name: your name(s) who developed the code
    - Company/University Name: you put the name of the course, department and university you implemented the code for;
    - Date:

  - Each function in your C/C++ /Java file should have a header with information:
    - Function Name: Name of the Function
    - Description: Short description what the function does.
    - Arguments: Specification of each input argument parameter entering the function and its description.
    - Results: Specification of returning parameters exiting the function and their description.
    - Comments: some special system issues connected with this function

- Group representative(s) comes at the scheduled time (we will have a sign-up sheet) between **5pm and 7pm on Monday, March 1** and shows a demo of the required programs in 0216 Siebel Center.

**Demonstration Scenario for Case 1**

Step 1: start server P1 and listen for registration/request of peer P2
Step 2: start peer P2 and register with P1 – establish negotiation control connection to send requests/control messages
Step 3: send request from P2 to P1 about PLAY of the movie, name of the movie and the movie duration;
Step 4: server P1 finds movie and start to push chunks to peer P2 according to the frame rate of the video file
Step 5: peer P1 receives the chunks and displays them
Step 6: when P2 played the chunks of the movie for the duration time, it stops and sends a “stop” message to P1 to stop the streaming from P1.

Demonstration Scenario for Case 2 and 3

Step 1: start server P1, listen for the registration of peers/requests
Step 2: peers P2, P3 and P4 register with P1, P1 creates a peer list (or peer table), and P2, P3, P4 create connections to P1 for transmitting control (session control) messages.
Step 3: peer P4 (client) sends a request message over the control connection to P1 with information (a) operation type PLAY, (b) movie ‘name’, (c) movie duration.
Step 4: server P1 opens movie file of ‘name’, and starts to push/stream chunks to peers P2 and P3 for 1 second. Alternately, P1 can load the movie file to a buffer and then push the content from the buffer to the proxies.
Step 5: server P1 prepares/sends a response message to P4. The response message includes information (a) peers (proxies) addresses which have chunks of the movie ‘name’, (b) chunk table that are stored on peer P2 and P3.
Step 6: client P4 starts to pull chunks from the peers P2 and P3 and displays the chunks in order on the screen. In parallel, P1 streams/pushes new chunks to P2 and P3.
Step 7: client simultaneously requests/gets control messages from the server P1 about new chunks (or chunk table) on peers P2 and P3
Step 8: after P4 played the movie for ‘x’ seconds duration, it stops the display, and sends a “stop” message to peers P1, and P2, and P3 to stop the streaming, and gracefully stop the overall operation (clean up all your structures).
Grading Criteria and Evaluation of the Assignment (100 Points)

Please follow the demonstration scenarios closely to setup your system

1. Bootstrap the p2-stream service (discussed in Section 2.1) - **25 Points in total**
   - Server (P1) starts sockets, loads video file into the buffer, obtains frame rate of the video file (5 points)
   - Proxies (P2,P3) register with Server (P4) and the server puts proxies into the peer table – peer registration (5 points)
   - Server initializes and manages the chunk table (5 points)
   - Client (P4) connects to the server, server adds client into the peer table and returns the proxy information (5 points)
   - Questions – 5 (points)

2. Connect for PLAY Request (discussed in Section 2.2) – **10 Points in total**
   - Client connects to proxies and asks for the video chunks (5 points)
   - Questions – 5 (points)

3. Case 1 implementation of push-based approach (discussed in Section 3.1) – **15 Points in total**
   - Rate control (5 points)
   - Multi-threading (5 points)
   - Questions – 5 (points)

4. Case 2 implementation (discussed in Section 3.2) – **20 points in total**
   - Push-based between P1 and P2/P3 with rate control (5 points)
   - Pull-based between P4 and P2/P3 (5 points)
   - Multi-threading at the client P4 (5 points)
   - Questions – 5 (points)

5. Case 3 implementation (discussed in Section 3.2) – **20 points in total**
   - Limited buffer at P4 (7 points)
   - Display rate, display duration and graceful stop (8 points)
   - Questions – 5 (points)

6. Possible Point Deduction:
   - Error at the display side when it displays the video
   - Error at the proxies when fetching, forwarding video data
   - No frame rate at the display side
   - No limited buffer for case 3
   - No multi-threaded program
   - No thread synchronization
   - Can not answer questions about your own code – you should unstand what you write
     (Note: we will ask the representatives of the group any questions about the code)
   - Others may come if needed

7. DOCUMENTATION - **10 Points** Each group should write a README_g<Your group number>.pdf file (no more than 5 pages, please!). For example, README_g1.pdf is for the group 1. You should follow the instruction in this link [http://www.cs.uic.edu/class/sp10/cs414/mp/mpDocumentationGuidelines.pdf](http://www.cs.uic.edu/class/sp10/cs414/mp/mpDocumentationGuidelines.pdf) to write the README_g<Your group number>.pdf file. Please don’t repeat what is in this assignment writeup. Write the README file so that we can re-compile and run your code incase we need to do so to grade your MP. Email the README document to the TA longvu2@illinois.edu, and also
store it in your group directory by mid night March 1st, 2010. **IMPORTANT:** No late README submission is accepted.