1. Introduction

Twenty years ago, the only method you could watch a movie was to either go to a theatre or buy/rent a videotape. However, things have changed a lot since then. People now prefer watching everything online. YouTube becomes the place where we watch most video. Even the DVD movie rental companies, like NetFlix, are investing most of their money in expanding the online streaming business. What is the magic that makes all this revolution happen? Network Streaming!

In the first MP, you have already learned how to capture the raw images and raw audio data from external devices, and got some experience in video/audio playback. Therefore, we now focus on how to stream the captured video/audio contents from one machine to another machine through connecting network. In this MP, you will achieve a deep understanding of the concepts like bandwidth, QoS, and why they matter so much. Moreover, you will have all important components ready for your next MP: building a multi-view surveillance system.

You can use the Linux workstations in the EWS lab room SC216 and SC220. A group directory and an SVN account have been set up for everyone. You can use your own machine as well. You still need the Logitech Webcam you have borrowed (you can also use your own webcam) and use your own microphone/headphone for the audio recording/playback.

2. Problem Description

2.1 System Overview

For this MP, you have to create a server and a client component. They should run on separate machine. However, to implement the bonus functionality, your media server should support multiple clients running concurrently. The functionalities of server and client components are given below.

Server

The server always listens to a port for any incoming client request. A camera is connected to the server to capture live audio and video streams at a fixed rate. We use 30fps for the video capture and 8000Hz for the audio capture. (Note: you have to capture at these fixed rates even though a client may request for lower rates of audio and video. You will need this so that multiple clients [bonus functionality] can...
request in different rates $\leq 30$fps). However, if your server supports only one client, then you can use client’s requested rates to capture audio and video data. Figure 1 best describes the setup required for MP2.

![Server-Client communication architecture](image)

**Client**

A client is a separate machine with Graphical User Interface that can send a connection request to the server for video and/or audio streams of different rates. The client works in two modes: *active mode* and *passive mode*. In active mode, the client requests for both audio and video streams. The video rate may vary between 15fps to 25fps depending on the resource availability and the audio rate is fixed to 8000Hz. Also, the active mode requests for high resolution video (e.g., 640X480). However, in passive mode, the client requests for only video streams (no audio) with 10 fps, and the resolution of the video streams are low (e.g., 320X240). The GUI should allow the user to change between modes. The client should display the video on the screen and send audio to the sound card. In active mode, the audio and video should be properly synchronized.

### 2.2 Functional Components

As a part of this MP, you will have experience in implementing different multimedia concepts such as Resource Admission, Resource Reservation, QoS Negotiation, QoS Enforcement and Real-time streaming and Session Control. Below we describe the required functional components for this MP.

**Client Resource Admission**

Each client maintains a “resource.txt” file that stores the available application bandwidth (ABa) of the local machine. Your client GUI should allow us to change this value. When a client is ready to send the streaming request to the server, it first performs the resource admission at the local machine. The admission process should consider defined media rate (by the user), the estimation of media frame size
(you can consider average frame size to allow optimistic allocation) and the available bandwidth at the client (can be obtained from resource.txt). Note that depending on the client mode, the average media frame size ($M_A$), frame rate ($F_A$) and media type (video with audio or without audio) vary. A streaming request is sent to the server only if the admission process is successful. The admission process becomes successful if $B_A \leq AB_A$, where $B_A$ is the required application bandwidth. If the admission process is not successful, show the rejection to the users. Remember that, in active mode, the admission process considers both the audio and video streams, while in the passive mode, the admission process considers only the video stream.

Resource Negotiation

After the admission is successful at the client, it sends a connection request to the server that contains a list of requested media (client can request in audio and video active mode or only video in passive mode) data, the required resolution (to compute the frame size) and the required rates. Depending on these values, the server should perform resource admission as follows.

Server Resource Admission

The server also maintains a “resource.txt” file that stores the available application bandwidth of the local machine. The admission process should consider client defined rates, the estimation of media (audio and/ or video) frame size, the local media capture rate and the available bandwidth at the server (can be obtained from resource.txt). The request is accepted only if the admission process at the server is successful. If the admission process is not successful, the request is rejected and the rejection is informed to the client.

Resource Reservation

Both server and client should maintain a resource table storing states describing the allocated resources for each client session. For multiple clients (optional feature) this table is important since, you cannot allocate resources that already been allocated to the other clients. If $BA_A$ is the allocated bandwidth to other clients and $AB_A$ is the available application bandwidth (given in resource.txt), then when a new client request comes, the available application bandwidth to support the client request is $AB_A - BA_A$. Remember, you do not need to consider transport packet size, RTP header size or transport packet payload in your application level bandwidth computation.
QoS Enforcement

Once the request is accepted at the server, it uses a rate control mechanism to enforce the negotiated rate in the data communication (or streaming). You can implement token bucket, leaky bucket, gstreamer specific rate control or any simple queuing mechanism to control rates.

Data Plane Communication

The audio and video data between the server and the client should consider RTP (or RTP like) and UDP. The RTP header should include media (audio and video) capture timestamp (the time when the data is captured from the webcam). You can use “rtpbin” of gstreamer for data communication or you can build your own RTP-like protocol over UDP.

Audio Video Synchronization

If a client receives both audio and video data from the server (in active mode), the playback should synchronize the media data. To synchronize, you need the capture timestamp from the RTP header of the video and audio frames. The details can be found in Lecture 23.

Session Control

You should implement session control functionalities at the client such as START session, STOP session, PAUSE session, RESUME streaming, and SWITCH streaming modes. STOP session button should teardown the ongoing session, while PAUSE session button only stop the streaming but maintains the control connection. Therefore, the RESUME functionality does not require further negotiation of resource availability. However, START session button will start from the initial connection setup (Resource Admission, Negotiation and Resource Reservation). Session control command can be given at any time during the streaming.

Session Adaptation

Also, users can change the “resource.txt” (using GUI) at the client and the updated resource should be reflected in the current communication if necessary. For example, if we lower the machine bandwidth at the client machine, there should be again resource admission, and QoS negotiation both at the server and client. Below we show an example case, where the change in “resource.txt” causes an update in playback rate at the client.
Session Monitoring
You should monitor the session at the client. The session monitoring should include: 1) Incoming bandwidth, 2) Synchronization skew, 3) Media Jitter (both for video and audio if any) 4) Failure rate. You will get again 10 points bonus to implement to show the entire graph visually on the screen with respect to time.
Machine Problem 2: Video/Audio Streaming  
CS414 Spring 2012: Multimedia Systems  
Instructor: Klara Nahrstedt

Posted: Mar 12, 2012  
Due: 5:00pm April 7, 2012

Figure 2 shows the placement of different functional components during the connection setup and streaming phase. Figure 3 shows the flow of session control during the streaming. We only require stop and resume. The stop command from the users should stop the streaming and the resume command should re-start the streaming. The resume happens from the current video frame.

Remember that, any changes in the resource.xml should perform the re-admission control and QoS negotiation. The streaming in active mode should be done with the highest possible rate (between 15 to 25fps) considering the client bandwidth defined in resource.txt.

3. Required Features

**Video Streaming (10 points):** The client should be able to display the video from the server depending on the mode. Playback in active mode requires video with 640X480 resolution and 15 to 25 fps. Playback in passive mode requires video with 320X240 resolution with 10 fps.

**Audio Streaming (10 points):** The client should be able to play the audio from the server. Note that, in passive mode, there is no audio.

**A/V Synchronization (15 points):** In active mode of playback, the audio and video captured from the server should be properly synchronized. You will be judged depending on your synchronization perfection.

**Resource Admission and Negotiation (10 points):** You should implement a simple resource admission and negotiation protocol mentioned above. The server and client bandwidths are given in resource.txt file at each machine.

**QoS Enforcement (Rate control) (15 points):** You should create a proper rate control mechanism so that the network always gets the packet on the negotiated rate.

**Session Control (10 points):** The client GUI should implement the stop and resume functionality. You can also change the mode of the client at any time.

**Session Adaptation (10 points):** We should be able to update resource.txt at the client machine. Depending on the modification, the rate of the current session should be re-admitted and re-negotiated.

**Session Monitoring (10 points):** Display the monitoring output on the GUI in the text form.

**Report Writing (10 points):** Write a clean report describing your approach, algorithm and assumptions.
4. Bonus Features

**Graphical Plot of Monitoring (10 Points):** Similar to MP1, showing the GUI for the monitored output (graphs, plots) will be awarded 10 points as bonus points. Remember that to get the bonus point full, you have to implement all monitoring graphs and append them nicely in your video display GUI.

**Supporting Multiple Clients (20 points):** We will provide 20 bonus point if your code supports multiple concurrent clients watching videos in different modes.

5. Data Plane Design Architecture

Here we present a basic architecture for the implementation. *You can use your own design. The only required part for MP2 is that you must have all the functional components explained above. Note: this example does not contain the design of the control plane.*

![Data Plane Design Architecture](image)

**Figure 4. The server implementation architecture**

5.1 Server Architecture

Server receives audio and video from the webcam and stores them in the separate queues. You can use mjpeg or mpeg4 encoder for the video and vorbis for the audio. Before putting them in the queue, we have to create an RTP header and store the
capture timestamp along with the frame. The Rate Control component reads from
the queues depending on the negotiated rate. (You can also use a loop with timer,
e.g., 10 fps can be implemented with 100ms sleep in the rate control thread. You are
also free to use leaky bucket and token bucket.) Then the frames are sent to the
network. The implementation architecture is shown in Figure 4.

5.2 Client Architecture
Client receives requested audio and video frames from the network using UDP. The
frames are stored in the separate queues. In the active mode, the client performs AV
synchronization depending on the capture timestamps. Selected frames from the
queue are then sent to the application source, which creates a pipeline to forward
them to the appropriate sink. The architecture is shown in Figure 5.

![Figure 5. The client implementation architecture](image)

[NOTE: You can use any grstreamer functionalities or you can implement your
own RTP-like, RTCP-like to RTSP-like protocols to implement this MP]

6. Submission

Pack all source codes and documents into a zip file or tar ball and submit them
through Compass. Do not submit your solutions through email unless there are
technical problems with the Compass system. The submission deadline is April 7th
at 5:00 pm. You can get up to two days bonus for each MP, but please remember you
can use only 3 bonus days throughout the semester. Further late submissions are
not accepted and will get 0 point.
6.1 Source Code
You should only submit your source codes (.java, .c or .cpp files), Makefile, and any open source libraries (or jar files) you use in your solution into compass. Do not include any pre-compiled obj files (.o), binary execution, or pre-recorded media files in the directory.

6.2 Documentation
Your documentation should include two parts: user manual and development manual. The user manual should include all instructions on how to compile and install your source code, and how to run your program and test all features. If you have designed any GUI, it is better to attach some screen shots to explain. The development manual should have the implementation details of all features. The implementation details include program flow, key data structures; media file formats, important algorithms, and so on.

7. Evaluation
The evaluation will be done by face-to-face interview with each group. You will run your program. We may ask you to show the source code implementing different functionalities. Your solution is evaluated based on how many features you have implemented and demonstrated. In order to get full score (100 points) of this MP, you need to implement all 100-point required features. 30-point optional features will be used to offset your lower marks at MPs, homework's or exams. Evaluation will happen on Monday April 9th at lab SC216 at 5:00 pm. A sign-up sheet will be provided (by email or using compass) to schedule your demonstration slot.

Please Attend MP2 Lecture on Friday March 16th, 2012. TA will discuss various design alternatives and gstreamer functionalities that you can use for implementing this MP.]