Lab 2 Multimedia Systems

Group

Marc Dahlem
Jon Martin Mikalsen

1 Problem Specification

1.1 Description

In this lab there shall be one server that will supervise video streams sent from at least three other client applications. On this server there shall also be a list over connected clients, and a alarm function that switch to a given stream if there is some motion detected on that stream. On the clients there shall be a connect button that connects the video stream to the server, and a settings page where server ip and port can be changed.

1.2 Assumptions

To do what is described above, there are several things that has to work. To begin with the motion detection has to be implemented into the first lab that was developed and delivered. Then the actual connection between client to server has to be researched, and then figure out how to manage multiple clients.

1.3 Extra features

The client has a graphical interface to connect to the server. He can change settings like the host to connect to and the port on which the server is delivering the service.
On the serverside are all extra features of lab 1 still available. The incoming stream with the lastest recognized motion is set to a playback window, which can be set to fullscreen.
The userinterface lets the user select and change the directory for the recording.
Also are messages shown on the playback window, when new clients connect or a motion is detected or stopped.

2 Usage and user's guide

Here the instructions on how to get the application running is stated.

2.1 Requirements

To get this application up and running there are several requirements that as to be in place.

- Have the knowledge on how to run a given java Eclipse project/program
- Have GStreamer installed on computer
- Have the ability to run other GStreamer applications developed in Java
- Have Eclipse running and working on the computer
- Have knowledge on setting environment paths
- Basic knowledge of (auto)make.

2.2 How to get it running

This section describes two different ways to get the program running.
For just running the application a runnable jar-file was created and attached to the fronter submission. The second way is to download the source files from the setup git reposity.
If you want to create an own project with the source file, than you have to follow the manual instruction to setup the eclipse project.

2.2.1 Runnable .jar file

To run the application one can use pre-build jar files delivered with this report.
To execute them run in the command line 'java -jar client.jar' or 'java -jar server.jar'.
The order of this two executions doesn't matter.
Hint: If one doesn't want to connect to localhost one has to open port 5000 on the server side.

2.2.2 Git repository with code

A already predefined eclipse project is available on the git repository

- https://github.com/MarcDahlem/ANSUR

It includes a fully setup eclipse project and can be imported into the eclipse workplace.

In the git repository is all the code for the labs1 and 2 in the main folder "ANSUR". Inside this folder there is another folder called "Multimedia_Lab2" that is the second lab. Here one can find the folder with the eclipse project, or a folder containing all delivered files like javadoc, this pdf, some screenshots and the runnable jar files.

One way to get the eclipse project up and running is to create one new java projects in eclipse, called "Multimedia_lab2" and then copy / paste the contents into these project folders. Or right click in the "Package Explorer -> import -> (general) Existing projects into workspace" and then just browse the project that was downloaded from Github.

To make sure everything is ok, the gstreamer-java.jar and jna.jar, as well as the swt.jar have to be added in the project build path. To check this, right click on the mulitmedia_lab2 project, and go to properties. See illustration 1

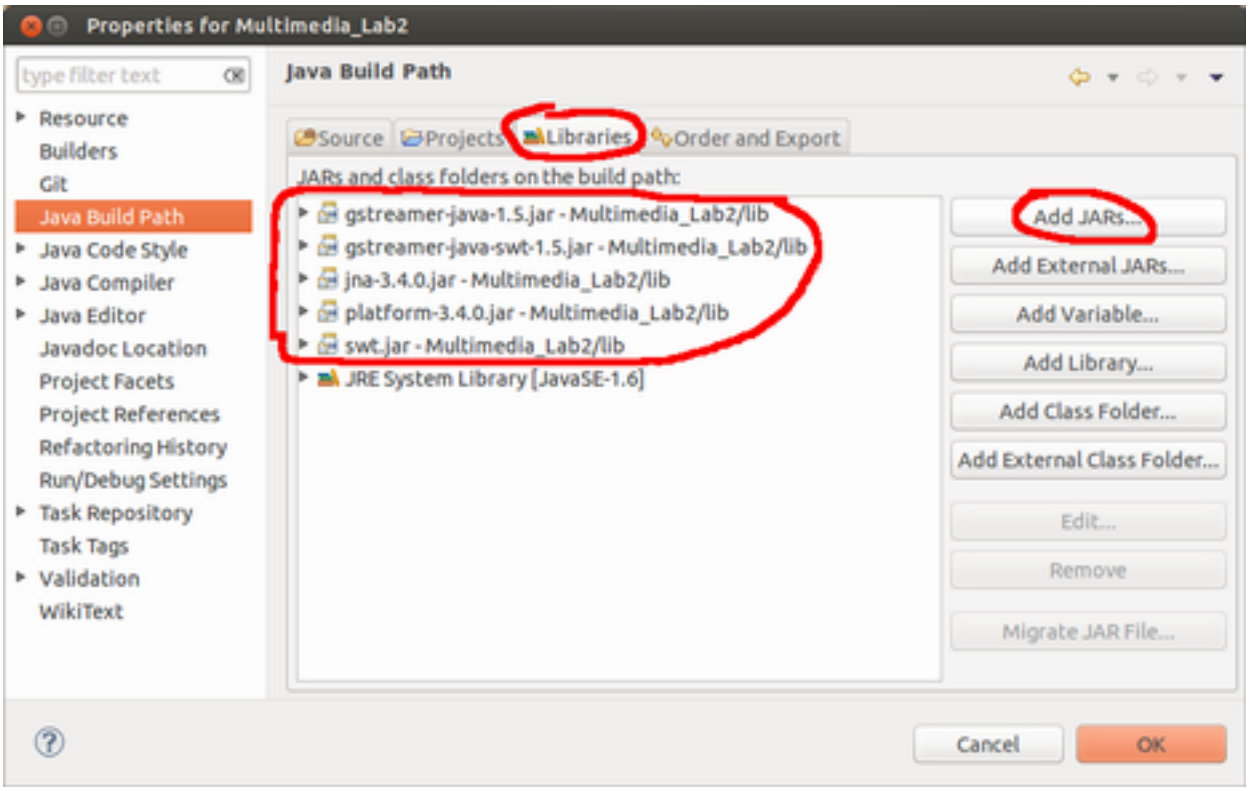

Illustration 1: libraries that has to be in the project java build path,

Now when the project settings are set up, the motion detection plug-in has to be installed. This has only been tested on a Linux computer and the following packages have to be installed

- libopencv-dev
- libopencv
- libconf
- automake

When these things are ok, the actual plug-in can be downloaded from a git repository:

- https://github.com/codebrainz/motiondetector

Follow these steps to install the plug-in:

1. If downloaded as zip/jar/tar files extract motiondetector folder to wherever pleased
2. Open a terminal and “cd” into the folder “motiondetection”
3. type “sh autogen.sh” to run a script that will start the first part of the installation
4. When the script is done, type “make”.
5. Now type “sudo make install“, that will install the plugin as root to the /usr/local/lib/ gstreamer-10/ folder.
6. After the make install is done, export the necessary path by opening the .profile file in your home direcoctory and adding the following lines at the end of this file. Don’t forget to save after that ;-)

```
#adding path for the gstreamer plugins
GST_PLUGIN_PATH=$GST_PLUGIN_PATH:/usr/local/lib/gstreamer-0.10/
export GST_PLUGIN_PATH
```

7. Reboot computer to load the profile
8. The plugin should now be installed, and the lab code should be runnable
9. To test the installation try ‘gst-inspect motiondetector’ in the commandline. That should give you detailed information about possible variables etc of the plugin.

Troubleshooting:

1. Can’t run “make” ? The answer to this is that there could be some packages missing on the computer, to check this simply run “./configure” inside the “motiondetector” folder to check, and then install missing packages.

2.2 Result

2.2.1 Client

When starting up the client the screen shown in illustration 2 will appear.

Illustration 2: Client start-up page

In the client there is a possibility to change the server ip address and it’s port, and to do this just click the “settings” button, and the following page will show (Illustration 3).

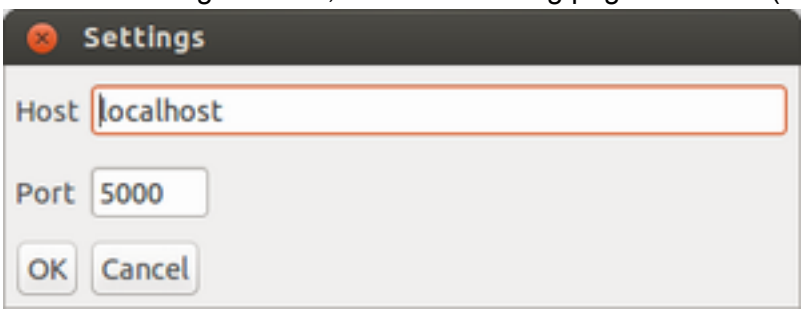

Illustration 3: Client settings page

2.2.2 Server

When starting the server the application should look like illustration 4 shows.

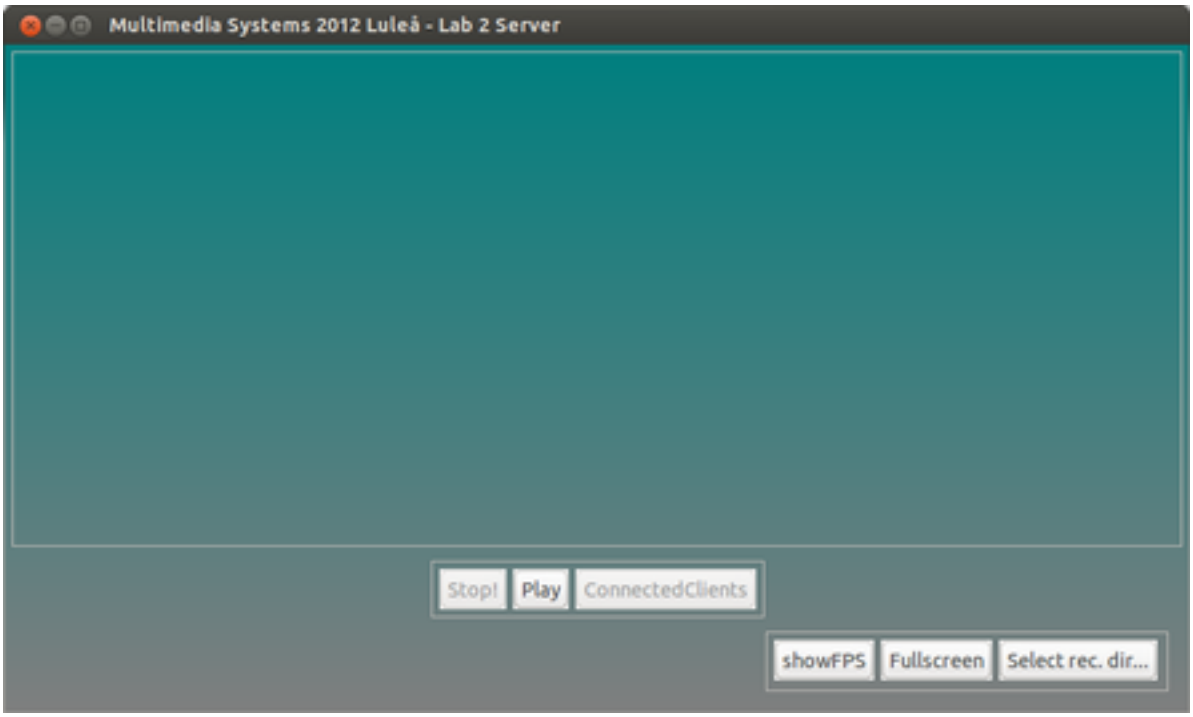

Illustration 4: Server start page

When the server is started the user is prompted with a file-browser that will choose where motion detection movies will be recorded to.

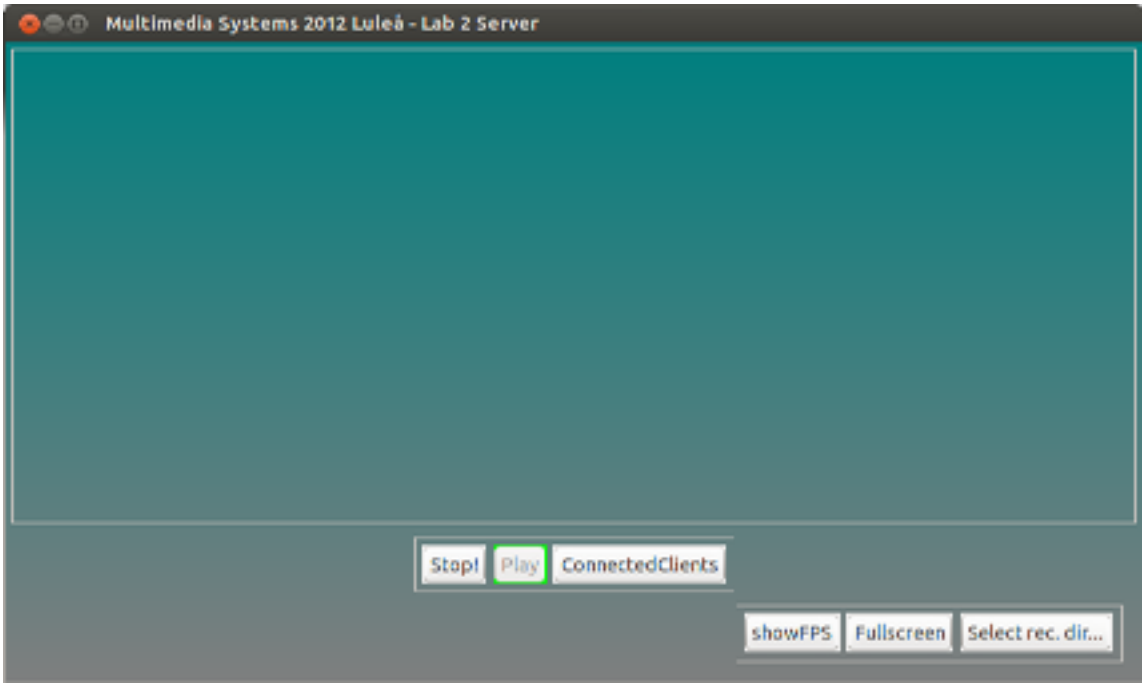

Illustration 5: Server after play is pushed

When a client connects to the server, a video-stream will show up in the main window.

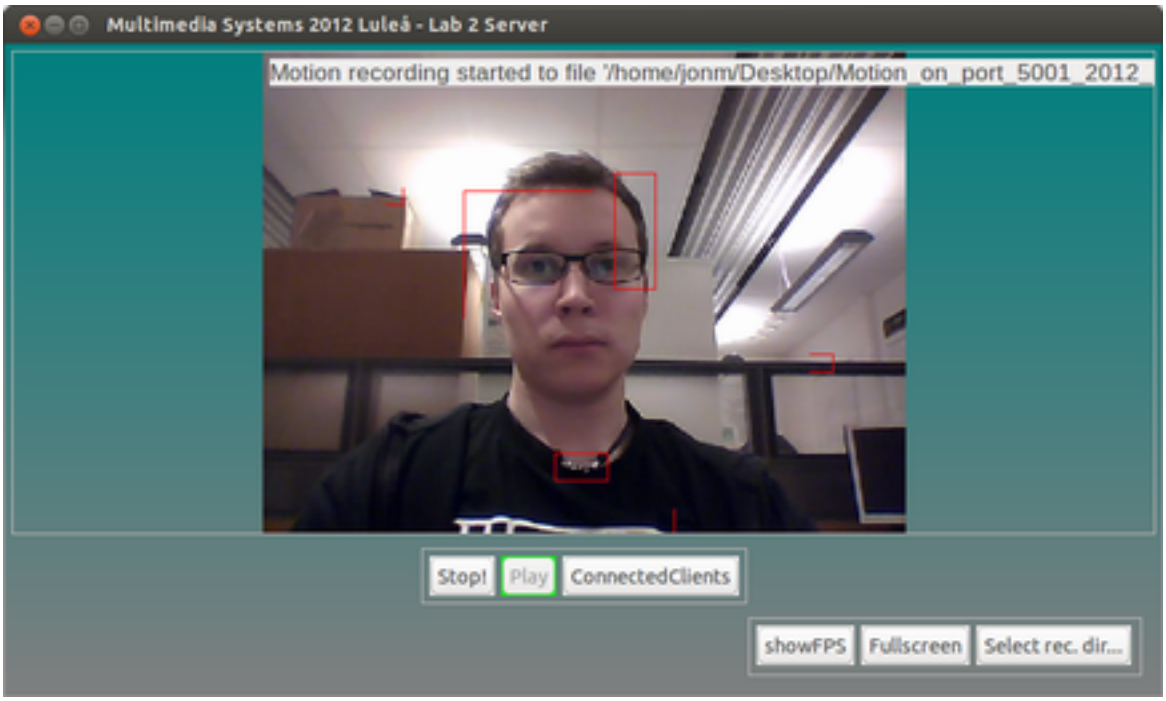

Illustration 6: Server when a stream is received

Also when a client connects, there is a warning about an internal data flow problem that is caused by a bug[1] in the gstreamer oggmuxer.

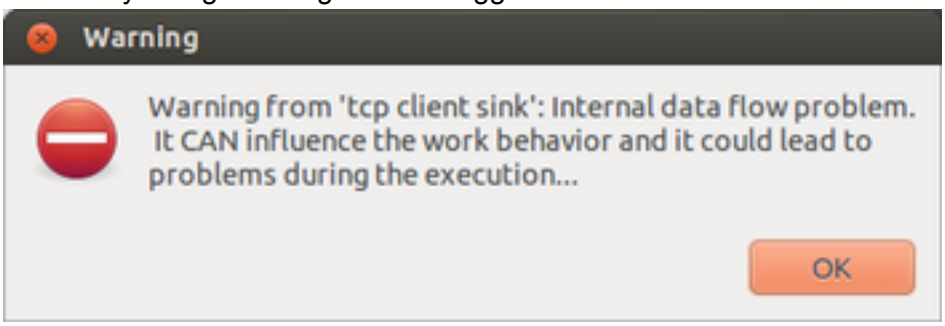

Illustration 7: A warning that appears when a client connects to the server

When a client then disconnects, the server will receive a EOS(End Of Stream) that notifies that a client has disconnected.

[1] https://bugs.launchpad.net/ubuntu/+source/gstreamer0.10/+bug/363282

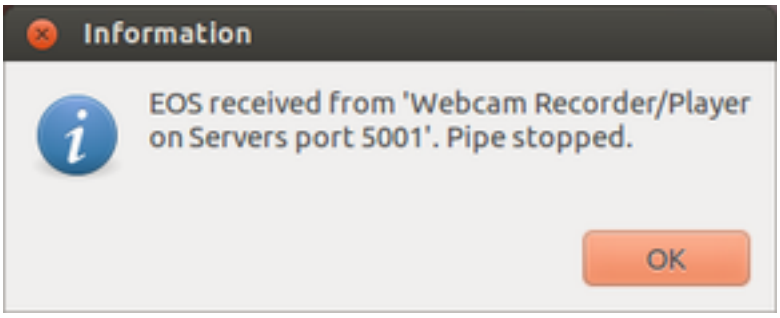

Illustration 8: A popupbox that comes when a client disconnects from server

3 System description

3.1 Internal design and structure

Here is the applications design and structure described with pictures of pipelines, handling of multiple clients on the server, switching to a stream where a motion is detected, and setup of the automatic recording.

3.1.1 Pipelines

An overview on how the pipelines is built up can be found on github. Here is the link to the images. ***P.S. The pictures has to be downloaded.***

- **http://tinyurl.com/Lab2-pipeline**

The client pipeline is an easy pipeline to connect the webcam video to the server. It encodes the video with theoraencoder and muxes with an oggmuxer. The videofilter is used to fix the framerate of the webcam video. The actual server connection is made with an tcpclientsink. We used tcp to ensure, that the movie is fully available on the server to be able to do a motion detection without disturbing this procedure.

On server side is the stream received with a own pipeline per client. The received video is decoded and demuxed before it is put into the motion detector. After the motion detection the video is splitted into two different pipes: a playback bin and the recorder bin.
Both this bins handle the connection with dynamically changing pipelines from fakesinks to a real recorder pipe or videoplayback device.
This dynamically changing of the pipe was the hardest part of this program. But with this the pipeline can easily set to play the stream in the actual gui application, or to change dynamically from not recording to recording, if a motion event is detected.

3.1.2 Connection of multiple clients

When the server is started, it starts a socket listener that listen for incoming connections. If the server then gets multiple connections, it sets up a new streaming pipeline with a unique port for each connected client. When a client is connecting it opens a socket to the server by entering hostname and port (which is set in the settings menu), and send a string “get_port_and_start”

to the server. The server has a listener implemented that listens for this string, and return an available port to the client which it then connects to. Besides will the server start the motion detection pipeline for this client on the stated port.

3.1.3 Handling disconnections

When a client disconnects from the server, the streaming pipe for that client is thrown away on both the client and the server side. The disconnection is recognized by an EOSEvent or an ERROR on the pipelines bus.

When a server is stopped, it checks a list that contains all clients and then stop all pipes that are currently running. Every client will then receive a transmission error and stop its connection as well.

3.1.4 Switching to motion detection stream

To switch between shown streams on the Gui on a motion detection, the Gui has a listener that check for motion detection on the pipelines of the connected clients. When a motion is detected on a different stream from that one that is currently playing , it removes the video component (the playback sink for Gui) for the current pipeline shown in the Gui, and sets it to the one where the motion is detected. The switching is done by blocking the PAD in front of the playbacksink, flush an EOSEvent through the playback part and when this event is received on the last pad, the old playback part (f.e. fakesink) is removed from the pipe and replaced by the new playback part (f.e. VideoComponent-Element).

3.1.5 Recording

When a motion is detected on a video-stream it is recorded and stored on the server. The way that this is set up is that when the pipeline for a client is created, it has a fakesink[2] connected to it after a valve[3] element. When a motion is detected, the valve element is used to block the buffer flow, the fakesink is removed from the pipeline and replaced with a encoder, muxer and a filesink. Then the valve element is set back to let the buffers flow. This is then the starting of the recording. Stopping is the same procedure, but with the new recordbin is the fakesink and the old one is the real recording bin.

3.2 Classes

[2] **http://tinyurl.com/gstreamer-fakesink**

[3] **http://tinyurl.com/gstreamer-valve**

An overview of all the classes in the project can be found in with the code that is on Github. Here is a link to the two class diagrams. ***P.S. The pictures has to be downloaded.***

- **http://tinyurl.com/Lab2-Classdiagrams**

4 Problems and reflections

4.1 Limitations

4.1.1 Motion detection

When it comes to the motion detection, the limitations is the ones set by the developer of the motion detection plug-in. The plugin is not really fast and the motion detection can take a long time. That is why we used a framerate limitation on the client side, to fasten the motion detection up. But still we have a big delay between real motion and the detected one.

4.1.2 Client ports

The number of clients are limited by the number of ports available, we set a threshold for the ports to 99998 of theoretical connected clients. But this would of course on one hand be limited by the network bandwidth itself and on the other hand by the comptation power of the server.

4.1.3 Recording power

The buffers are limited and the computational power on a pc also. And the recording (encoding+muxing+writing to hard-drive) can be the limiting factor, when a lot of clients are connected to the same computer.

4.2 How to avoid limitations

4.2.1 Motion detection

The delay in the motion detection will we try to improve in the next lab by trying different parameters for the plugin. Another idea is to restructure the pipeline, so that the motion detection is done in parallel to the playback part. But the recording part has to be after the motion detection part so that the delay is not affecting the visible timeslots in recording part.

One hard way to avoid the limitations set by the motion detection plug-in, could be to develop an own plugin that detects motion or find another motion detection plugin that can be connected to the pipe and that is faster (maybe by using multiple threads?).

4.2.2 Client ports

This limitation can only be reduced, but not be solved by setting up multiple servers. For example one could set up a cloud with loadbalancers etc to control the connections.

4.2.3 Recording Power

The same idea of a cloud can be used to detect the motion and handle the recording. Also one can use faster drives to store the data to, to reduce this problem.

4.3 Problems during lab

4.3.1 Motion detection plug-in

Since the wish was to create a surveillance tool, a motion detection module had to be implemented. In the start the main focus was to find a good motion detection plug-in that was easy to implement and easy to use, but the the first plug-ins that was tried out did not work as wanted. It did not know the signals that were stated in the documentation. But without the ability to connect to signals on the motion detection plugin, this plugin is useless since no motion can be derived in the pipeline using this plugin.
We found 2 other plugins, but only one of them did signal a motion-start and a motion-end and did not destroy the colors of the video. But we are happy, that we finally found a motion detection plugin that does what we expected.

4.3.2 Server & Client connection refused

When trying to connect a client application to a server, there was some problems with connection refused. For this there was a very simple solution, by just using the “host”[4] property on the “tcpserversrc”. The host property is default set to ‘localhost’, but one has to set it to ‘0.0.0.0’ to be able to connect from everywhere.

Server:
Host=0.0.0.0 → Allows all ip’s to connect
Client:
Host=”Server-IP”

4.3.3 Server restart and client reconnect

The jna crashed when the server pipe was stopped and started again. That is the reason why we throw all pipelines away after they are stopped.

[4] **http://tinyurl.com/tcpserversrc-host**

4.4 Possible improvements

4.4.1 Client connection port

At the moment there is no function that resets a port number if a client disconnects. For example if a client connects to a server, and receive a message that he can connect on port 5001. If the client then disconnects, that port is not freed to other clients that will connect later, so the next connection will get port 5002.

4.4.2 Streaming delay

When streaming video to the server, the stream has a tendency to be delayed more and more while the time goes by. This is not seen as a big problem, since the motion will come eventually anyway, but this is something that could be optimized.

4.4.3 VideoComponent and switching in the GUI

At the moment the VideoComponent of the Gui is reused on every switch of the stream. That can lead to a data flow problem on this component. In a future release we will try to replace the old VideoComponent by a newly created one when a switch of the playback stream is happening.

4.4.4 More settings and further Gui improvements

At the moment there are a lot of settings hardcoded, like for example the encoder and decoder and the motion detection parameters. All these parameters can be used in a settings dialog in the Gui. Also the gui does not allow manual stream switching at the moment and the overlays shown on the VideoComponent are not well formatted. This can be improved in future releases.

5 Contribution

For making this lab possible there has to be given a big thanks to Matthew Brush who made the motion detection plug-in for gstreamer. Without his plug-in the lab would not have been what it is.

- https://github.com/codebrainz

Much of the final code was written by Marc when he sat programming in the university labrooms. But over day both Marc and Jon Martin sat together in the labs programming and discussing together on the same computer, similar to Pair Programming [5]. In the end there were also many late nights on school where both sat working together in hope to finish the lab in time

[5] http://en.wikipedia.org/wiki/Pair_programming

for the deadline.

Jon Martin also did a lot of research regarding possible connections to Android phones with videoplayback, notifications and stream handling in order to have it easier in the last lab.

When the program was complete, Jon Martin worked on the diagrams, illustrations, descriptions and on building the Lab 2 document that you are now reading.