

Faculty of Technology and Society

Department of Computer Science

Fall 2016

# Client-server system

# Alex Baldwin, [Alex.Baldwin@mah.se](mailto:Alex.Baldwin@mah.se)

Johan Holmberg, [Johan.Holmberg@mah.se](mailto:Johan.Holmberg@mah.se)

Zahra Ghaffari, [Zahra.Ghaffari@mah.se](mailto:Zahra.Ghaffari@mah.se)

# Description of the system

# Overall system description

The system contains two applications, a server application and a client application. The idea is to stream images from the camera, that is the server, to several client applications.

## Server application

We programmed the Server application in C-language in Linux, using the AXIS Embedded Development SDK. The server application is installed on an AXIS camera. The server listens to requests from clients. On receipt of a request, the server uses the fork() command to create a new process on which the client and server can communicate. Communication between client and server is done using sockets and the TCP protocol.

Upon connecting to the server, the client must send an initial message containing a byte specifying the length of the message and then a string containing a VAPIX option string with the FPS and resolution of the image stream. Using this string, a stream is opened using capture\_open\_stream from the capture.h header in the SDK.

The next actions occur in a loop until the client disconnects:

* The server captures frame data from the stream and sends it to the client (preceded by the size of the data).
* The server reads a response “status” message from the client, providing instructions about how to proceed:  
    
  0 is a request to disconnect  
  1 is OK (the server should continue sending images as normal)  
  2 is a request to change the stream settings – in this case the client will send a new VAPIX option string containing the updated settings. The server opens a new stream with these properties.

If the client disconnects, the process associated with it is terminated.

## Client application

We programmed the Client application in Java in the Eclipse IDE. The client allows the user to specify the server’s IP-address and port, select a frame rate and resolution from a list of predefined options and then connect to the server. The client follows the process described above:

* First send an option string with the FPS and resolution
* Then, in a loop:
  + Read an image and display it
  + Send a status update according to the protocol defined above.

The stream settings can be updated at any time by changing the selected options and clicking the “update” button. Communication with the server takes place on a new thread to enable interaction with the GUI while the client is communicating with the server, since the read/write methods are blocking.

## Reflections

On the suitability of our client/server solution:

AXIS’s VAPIX library provides built-in functionality for streaming images and video over HTTP – a solution that is almost certain to be better implemented than ours. In a real video streaming application, using VAPIX would probably be preferable. If, on the other hand, we wished to perform some kind of image processing on the server, it might be better to use a custom solution like ours.

The client should probably send FPS and resolution separately instead of sending the whole VAPIX option string itself.

We experienced some challenges during the work with the application as listed in below.

Challenges during programming server application:

* We had some trouble sending the size of the image from the server to the client, not realising that the byte order needed to be changed using the htonl() function. It was also difficult to know which data types in the server application corresponded to the data types used in the client.

Challenges during programming client application:

* To read the size of the images that the client received
* Managing program client application to receive multiple images
* At first we managed to send a request and receive a text from the server application. Next step was to try receiving an image that could be written to a file. When the previous part was successfully done, we modified the program to receive multiple images.

# References