IJCSC 0973-7391 Volume 7 • Number 1 Sept 2015 - March 2016 pp.69-75 available online at www.csjournalss.com

Wearable live streaming gadget using Raspberry pi

Akash Dhamasia
Nirma University
Ahmedabad, India
akash.dhamasia12@gmail.com

Kunal Prajapati
Nirma University
Ahmedabad, India
11bce111@nirmauni.ac.in

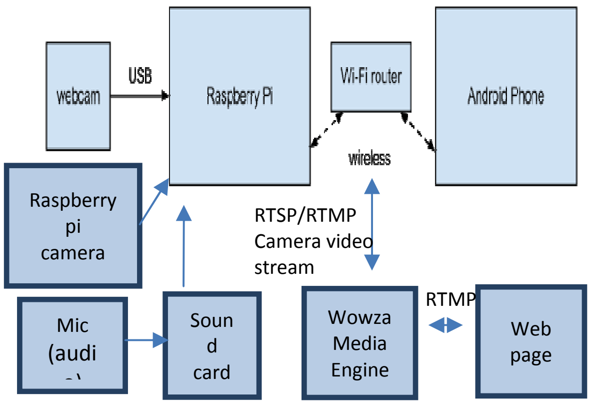
Prof. Parita Oza
Nirma University
Ahmedabad, India
parita.prajapati@nirmauni.ac.in

Abstract --This paper presents a raspberry pi based wearable device capable of streaming live audio and video on to the dedicated server and to android based mobile phones and tablets using gstreamer technology. This device is made with the consideration to provide security to the user such that in case of any danger, the user would be able to send panic signal to the server and through live footage of the scene, the response team can react accordingly. The gadget is accessible to the user through its application made currently for android enabled smart devices. The application provides the functionality of clicking, sharing, and publishing pictures and videos. In addition to that the application utilizes the gps present inside the smart device and send its location to the dedicated server through the device network on user demand.

Keywords: Raspberry Pi, Gstreamer, Security, Live audio and video streaming.

1. INTRODUCTION:

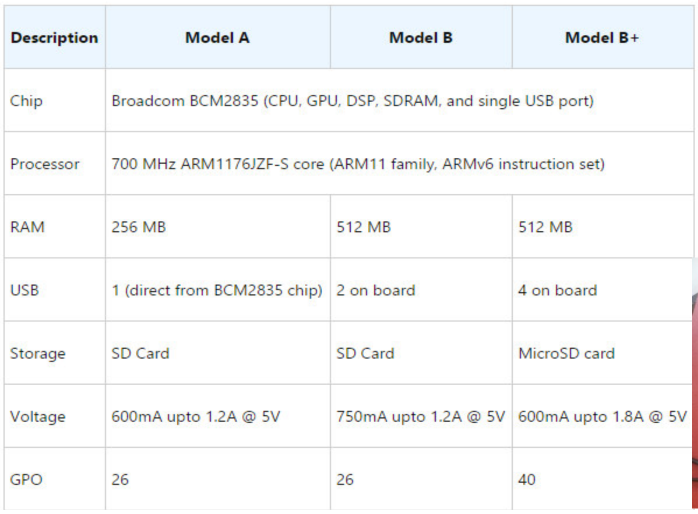
Capturing unexpected moments, Stream live video footage, Sharing capability, Webcam connectivity are some prime features of the gadget which makes it multipurpose. There are variosu applications in which the gadget can fit like Remote Surveillance, Robotic applications, Streaming live event, Remote Class Room teaching, GPS tracking, Security applications etc. The gadget is made using raspberry pi. Raspberry pi is a programmable single board computer and can be used to perform multi-processing tasks. We used raspberry pi as a mediator between camera and the end devices. It processes the data received from the camera, encode it, build a pipeline, and send it via udp or tcp network protocol. Gstreamer technology is used to build pipeline for sending and receiving camera feeds. GStreamer is an open source framework for constructing pipeline consisting of media handling elements.



[Fig:1 - Block diagram]

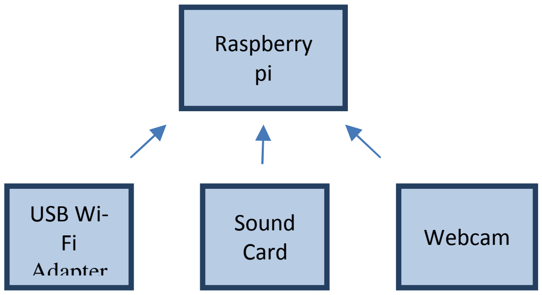
2. DEVICE COMPONENTS: SOFTWARE AND HARDWARE

Raspberry pie: Raspberry pi is a programmable single board computer and can be used to perform multi-processing tasks. the Pi comes in three configurations. Below is a table that gives the details about all three models namely A, B and B+.



Description	Model A	Model B	Model B+
Chip	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, and single USB port)		
Processor	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)		
RAM	256 MB	512 MB	512 MB
USB	1 (direct from BCM2835 chip)	2 on board	4 on board
Storage	SD Card	SD Card	MicroSD card
Voltage	600mA upto 1.2A @ 5V	750mA upto 1.2A @ 5V	600mA upto 1.8A @ 5V
GPO	26	26	40

Fig 2- Raspberry pi Models



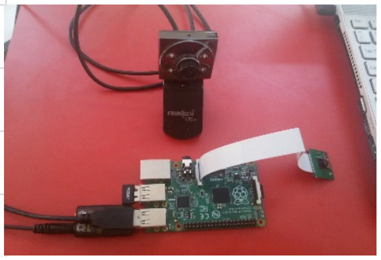


Fig 3- Device Components

As shown in figure, 3 USB ports are required, connection involving raspberry pie. Thus we choose Model B+ otherwise all Model would have been able to work.
Camera module*:* The Raspberry Pi camera module can be used to take high-definition video, as well as stills photographs. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi.
USB Wi-Fi adapter:To connect Raspberry pi to network
Mic + USB sound card:The Raspberry Pi comes with a 3.5mm analogue stereo audio output, but no input. For audio input a usb sound card is required.
Gstreamer:Open source library for constructing graphs of media-handling elements.
FFmpeg:It's a complete, cross-platform solution to record, convert and stream audio as well as video.
ALSA:Advanced Linux Sound Architecture is a software framework that provides API for sound card device drivers.
Gstreamer SDK: Android application which is a complete package for handling different types of gstreamer pipeline.
Wowza media streaming Engine:Wowza Media Engine is very popular streaming engine which can stream high quality video and audio.

3. WORKING PRINCIPLE:

Audio and Video Streaming:Streaming technology refers to sending large streams of data between systems. Because the data is too big to send in one go it is cut in to smaller packets of data. These packets are then send sequentially. In order to decrease the size of the data it is often compressed. The operating principle of video streaming is the same. Basically a video is compressed and then send in packets through a transport.
Methods of compressing video data:The first is 'Inter-Frame' based compression. Think of this as saving every image in the video as a JPEG image. An example compression algorithm that works accordingly is Motion-JPEG. Other examples are DV and HuffYUV.The second method is 'Intra-Frame' based compression and uses the the differences in images. If you start with an image the 'Intra-Frame' based method only tracks the differences in the following frames. Some highly sophisticated algorithms have been developed over the years of which the most used one is H.264. Other examples include Theora, Xvid and Divx. Compression algorithms for video are often referred to as a 'codec'.
Methods of transmitting video data:To transport the stream of video data packets there are many possibilities. In TCP/IP networks an UDP transport is the most simple solution. The RTP protocol is a transport protocol on top of UDP. Nowadays HTTP is also often used as a transport for streaming video

Gstreamer:To use the Gstreamer framework it's easiest to install it on a Linux system. In this example we are using Ubuntu but the steps should be similar on other platforms. To make sure the framework is installed run the following command in the terminal:sudo apt-get install gstreamer1.0-tools \

```
 gstreamer1.0-plugins-base \
 gstreamer1.0-plugins-good \
 gstreamer1.0-plugins-bad \
 gstreamer1.0-plugins-ugly
```

To have a basic understanding of the Gstreamer framework you need to think of it as a pipeline. The video data starts at the source and moves to the sink. Meanwhile you can do many things with the video data. Each chain in the pipeline is called an element. To construct a pipeline we have a very simple command line tool called 'gst-launch'.
To create this pipeline run the following command:gst-launch-1.0 videotestsrc ! autovideosink
The 'gst-launch-1.0' command uses the exclamation mark (!) to link elements to each in order to create a pipeline.
On the sending side we need to:

1. acquire the video data
2. compress the video data
3. cut the data into smaller packets
4. send the packets out through a network transport

On the receiving side we than want to:

1. receive the packets from the network transport
2. reassemble the packets into video data
3. decompress the video data
4. display the video

In general it is best to lookup the gstreamer plugin documentation to find the elements you need. In the case of a Motion-JPEG streaming setup using RTP we need the following elements:

1. 'videotestsrc' & 'autovideosink' to genereate and display an image
2. 'jpegenc' & 'jpegdec' to encode to and decode from JPEG
3. 'rtpjpegpay' & 'rtpjpegdepay' to create the RTP packets
4. 'udpsrc' & 'udpsink' to transport the RTP packets using UDP

Run the following two commands for the sender and the receiver:

```
Sender Side: gst-launch-1.0    !   videotestsrc !\
                 jpegenc   !\
                 rtpjpegpay  !\
                 udpsink host=127.0.0.1 port=5200
Receiver Side:gst-launch-1.0 udpsrc port=5200 ! \
                rtpjpegdepay ! \
                jpegdec ! \
                autovideosink
```

In this, the receiver will quit immediately with an error. This is because we need to tell the 'rtpjpegdepay' element some information about the data it will receive. This is called a 'capsfilter' in Gstreamer terms. A capsfilter is placed between the 'pads' of connecting elements. In Gstreamer events can transfer up and down the pipeline. These events can be used to pause the pipeline for example but it can also be used for exchanging the capabilities.Now that we have an example stream working we can replace the 'videotestsrc' with a webcam. On a Linux system we can use the 'v4l2src'. We will need a capsfilter to set the webcam's resolution.This gives us a nice feedback on the latency involved in this stream. The bandwidth used is about 1800 kbit/s. If we now try a default h.264 encoder we will notice difference.First, it might take very long for the image to show. This is because the stream first needs to receive a full keyframe. Second, your computer might not be able to cope with the encoding and decoding. Third, the latency is much higher (+5s) compared to the Motion-JPEG pipeline. On the other hand the bandwidth is much lower. About 300 kbit/s. We can tweak the parameters of the x264 encoder (sender) to make it more suitable for live streaming. You can find out which parameters you can set using the 'gst-inspect' command. Just try the following 'gst-inspect' command and match the output with the next 'gst-launch' example. Running 'gst-inspect' with no arguments will list all available elements.

```
gst-inspect-1.0 x264enc
```

```
gst-launch-1.0 v4l2src ! \
 video/x-raw,width=640,height=480 ! \
 x264enc tune=zerolatency byte-stream=true \
 bitrate=3000 threads=2 ! \
 h264parse config-interval=1 ! \
 rtph264pay ! udpsink host=127.0.0.1 port=5000
```

The bandwidth remains about the same (350 kbit/s) but the latency is much improved as well as the burden on the machine running it.To conclude these example pipelines we still need to know the latencies involved in the pipelines. Basically the recipe constructs a videostreaming setup which is displaying the image with a timestamp on the source computer as well as on a remote system. The difference in the timestamp corresponds with your latency of the pipeline.

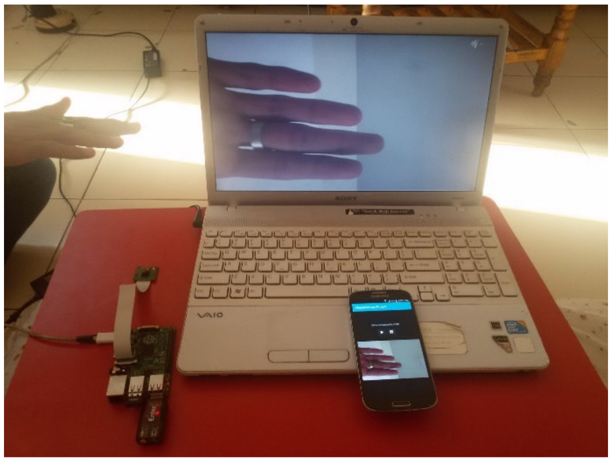
Sender:

```
gst-launch-1.0 v4l2src ! \
 video/x-raw,width=640,height=480 ! \
 timeoverlay ! \
 tee name="local" ! \
 queue ! \
 autovideosink local. ! \
 queue ! jpegenc! rtpjpegpay ! \
 udpsink host=127.0.0.1 port= 5000
```

Receiver:

```
gst-launch-1.0 udpsrc port=5000 ! \
 application/x-rtp,\
 encoding-name=JPEG,payload=26 ! \
 rtpjpegdepay ! jpegdec ! autovideosink
```

4. IMPLEMENTATION:



[Fig 4- Implementation]

We used following gstreamer pipeline to send audio and video to android and wowza media streaming engine.
Raspberry side:To Stream audio over the network: gst-launch alsasrc device=hw:Device ! audioconvert ! audioresample! 'audio/x-raw int,rate=8000,width=16,channels=1' ! udpsink host=x.x.x.x port=5001

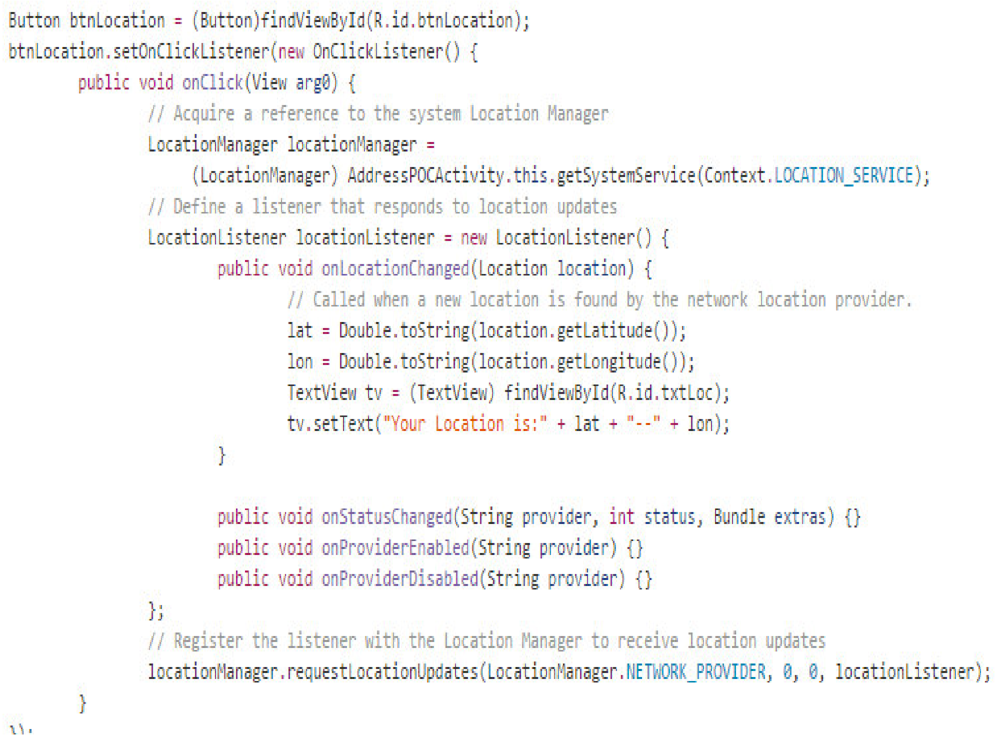
To Stream video over the network to Andriod device:raspivid -t 999999 -h 720 -w 1080 -fps 25 -hf -b 2000000 -o - | gst-launch -v fdsrc fd=0 ! h264parse ! rtph264pay ! udpsink host=192.168.0.193 port=8554
Andriod Side:receive video pipeline(Using Gstreamer SDK): data->pipeline = gst_parse_launch("udpsrc port=8554 caps=\"application/x-rtp, media=video, clock-rate=90000, encoding-name=H264,sprop-parameter-sets=\\\"J2QAFKwrQLj/LwDxImo\\\\=\\\\,KO4fLA\\\\=\\\\=\\\"\", payload=96\" ! rtph264depay byte-stream=false ! ffdec_h264 ! autovideosink sync=false", &error);

To Stream video over the network to Wowza Media Streaming Engine:raspivid -n -mm matrix -w 1280 -h 720 -fps 25 -hf -vf -g 100 -t 0 -b 500000 -o - | ffmpeg -y -f h264 -i - -c:v copy -map 0:0 -f flv -rtmp_buffer 100 -rtmp_live live rtmp://107.170.xxx.xxx:1935/MyApp/myStream

Finally these pipelines are sent like:raspivid -n -mm matrix -w 1280 -h 720 -fps 25 -hf -vf -g 100 -t 0 -b 500000 -o - | tee >(ffmpeg -y -f h264 -i - -c:v copy -map 0:0 -f flv -rtmp_buffer 100 -rtmp_live live rtmp://107.170.xxx.xxx:1935/MyApp/myStream) | gst-launch -v fdsrc fd=0 ! h264parse ! rtph264pay ! udpsink host=192.168.0.193 port=8554 & gst-launch alsasrc device=hw:Device ! audioconvert ! audioresample ! 'audio/x-raw-int,rate=8000,width=16,channels=1' ! udpsink host=x.x.x.x port=5001

Audio:Audio is received on port and feed data received to AudioTrack object.
AudioTrack track = new AudioTrack(AudioManager.STREAM_MUSIC,
SAMPLE_RATE,AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT, BUF_SIZE, AudioTrack.MODE_STREAM);

Getting the user location: Get the location of the users device (via the NETWORK or GPS). This will be latitude and longitude.



```
Button btnLocation = (Button)findViewById(R.id.btnLocation);
btnLocation.setOnClickListener(new OnClickListener() {
        public void onClick(View arg0) {
                // Acquire a reference to the system Location Manager
                LocationManager locationManager =
                    (LocationManager) AddressPOCActivity.this.getSystemService(Context.LOCATION_SERVICE);
                // Define a listener that responds to location updates
                LocationListener locationListener = new LocationListener() {
                        public void onLocationChanged(Location location) {
                                // Called when a new location is found by the network location provider.
                                lat = Double.toString(location.getLatitude());
                                lon = Double.toString(location.getLongitude());
                                TextView tv = (TextView) findViewById(R.id.txtLoc);
                                tv.setText("Your Location is:" + lat + "--" + lon);
                        }

                        public void onStatusChanged(String provider, int status, Bundle extras) {}
                        public void onProviderEnabled(String provider) {}
                        public void onProviderDisabled(String provider) {}
                };
                // Register the listener with the Location Manager to receive location updates
                locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
        }
});
```

To get a location from the GPS you would just change this line:locationManager.requestLocationUpdates (LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);To:locationManager.requestLocationUpdates (LocationManager.GPS_PROVIDER, 0, 0, locationListener);

To send to the Internet: Send that off to a server via Http GET

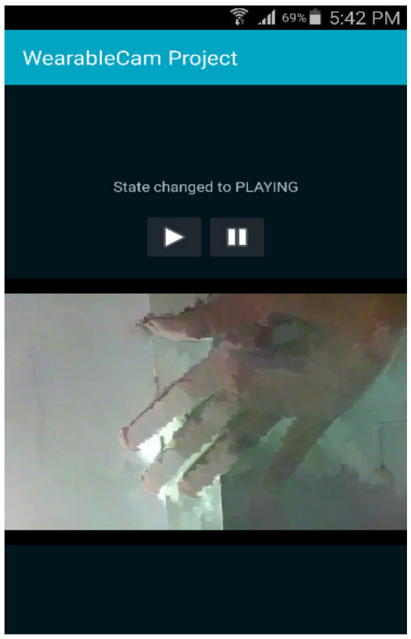
```
public void postData(String la, String lo) {
            // Create a new HttpClient and Post Header
            HttpClient httpclient = new DefaultHttpClient();
            HttpGet  htget = new HttpGet("http://<your_app_url>/Home/Book/"+la+"/"+lo);

            try {
                    // Execute HTTP Post Request
                    HttpResponse response = httpclient.execute(htget);
                    String resp = response.getStatusLine().toString();
                    Toast.makeText(this, resp, 5000).show();


            } catch (ClientProtocolException e) {
                    Toast.makeText(this, "Error", 5000).show();
            } catch (IOException e) {
                    Toast.makeText(this, "Error", 5000).show();
            }
    }
```

5. ISSUES AND OBSERVATION:

1. It has been found that on increasing the frame rate above 20 fps, the android application tends to drop packets.
2. On increasing resolution from 1080 by 720 to higher, latency also increases.
3. The product is runnable in devices with android api version 9 or higher.



[Fig 5- Distortion in streaming]

6. CONCLUSION:

Minimum lag and best frame rate is achieved using gstreamer framework in audio or video live streaming. The device is capable to stream live video and audio successfully to android devices and to web browser using wowza media stream engine over the network. The components used in making of the device are available open source in the market.

7. REFERENCES:

[1] http://www.codeproject.com/Articles/665518/Raspberry-Pi-as-low-cost-HD-surveillance-camera
[2] http://blog.miguelgrinberg.com/post/stream-video-from-the-raspberry-pi-camera-to-web-browsers-even-on-ios-and-android
[3] http://sanjosetech.blogspot.in/
[4] http://www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-camera/streaming-video-using-gstreamer
[5] http://www.androidhive.info/2014/06/android-streaming-live-camera-video-to-web-page/
[6] https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md
[7] http://www.raspberry-projects.com/pi/pi-operating-systems/raspbian/scripts
[8] http://www.daveconroy.com/turn-raspberry-pi-translator-speech-recognition-playback-60-languages/
[9] http://mutsuda.com/2012/09/07/raspberry-pi-into-an-audio-spying-device/
[10] http://asliceofraspberrypi.blogspot.in/2013/02/adding-audio-input-device.html
[11] http://blog.tkjelectronics.dk/2013/06/how-to-stream-video-and-audio-from-a-raspberry-pi-with-no-latency/
[12] http://labs.isee.biz/index.php/Example_GStreamer_Pipelines#Audio_RTP_Streaming
[13] http://rxwen.blogspot.in/2011/10/stream-audio-via-udp-on-android.html
[14] http://stackoverflow.com/questions/20759243/streaming-h264-using-raspberrypi-camera
[15] http://www.aurigma.com/docs/iu/WritingServerSideUploadCode.htm
[16] https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=73840
[17] http://stackoverflow.com/questions/25914467/android-streaming-live-camera-using-wowza-media-engine
[18] https://www.raspberrypi.org/forums/viewtopic.php?t=45368
[19] http://stackoverflow.com/questions/16238638/how-to-get-gstreamer-for-windows-and-for-android-work-together-with-h264-over-rt?rq=1
[20] http://stackoverflow.com/questions/23862218/gstreamer-raspberry-pi-and-android-how-to-get-streaming-video
[21] https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=95743
[22] http://pnetherwood.blogspot.in/2015/04/raspberry-pi-video-streaming-to-android.html
[23] http://www.linux-projects.org/modules/sections/index.php?op=viewarticle&artid=16#example11
[24] http://developer.android.com/guide/appendix/media-formats.html
[25] http://www.z25.org/static/_rd_/videostreaming_intro_plab/index.html