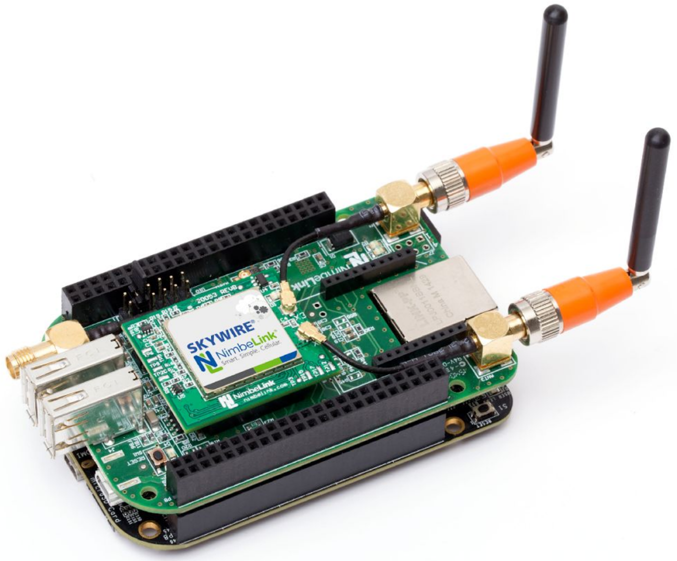
Streaming Video Over LTE with Skywire® Modem & Beaglebone Black

NimbeLink Corp

Updated: December 2015



While efforts are made to ensure accuracy, typographical and other errors may exist in this document. NimbeLink reserves the right to modify or discontinue its products and to modify this and any other product documentation at any time.

All NimbeLink products are sold subject to its published Terms and Conditions, subject to any separate terms agreed with its customers. No warranty of any type is extended by publication of this documentation, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose and non-infringement.

Table of Contents

1. Introduction

1.1 Overview

This document provides information for how to set up a remote camera that streams high-definition (720p or 1080p) video over a 4G LTE CAT 3 or CAT 4 connection using a Skywire® modem and a Beaglebone Black. With this setup, it is easy to make a remote-monitoring solution that can be used any place there is a cellular signal and power.

1.2 Orderable Parts

Orderable Device	Description	Manufacturer	Carrier	Network Type
NL-AB-BBBC	Skywire Beaglebone Black Cape	NimbeLink		
NL-SW-LTE-TSVG	Skywire 4G LTE CAT 3 Cellular Modem	NimbeLink	Verizon	LTE CAT 3
NL-SW-LTE-S7588	Skywire 4G LTE CAT 4 Cellular Modem	NimbeLink	Verizon/ AT&T	LTE CAT 4
BB-BBLK-000	Beaglebone Black Rev C	Circuitco/TI		
960-000764	HD Pro Webcam C920	Logitech		

1.3 Additional Resources

- NimbeLink’s Skywire Verizon 4G LTE CAT 3 Modem Product Page
- NimbeLink’s Skywire Dual-Mode 4G LTE CAT 4 Modem Product Page
- NimbeLink’s Skywire Beaglebone Black Cape Product Page

1.4 Tested Hardware and Software

This guide has been tested and verified working on the following hardware and software:

Hardware

BeagleBone Black Rev C

Skywire NL-SW-LTE-TVSG 4G LTE CAT3 Modem

Software

Beaglebone Black Debian 7.9

Beaglebone Black Debian 8.4 Kernel 4.4.x (non-RT Kernel)

Note: There is an issue with the RT version of Kernel 4.1.x. This guide will work, but you will have an intermittent stream and bandwidth issues with the USB connections.

2. Getting Started

2.1 Prerequisites

This application note assumes the following:

1. You have your Skywire 4G LTE CAT 3 or CAT 4 modem installed and working on your Beaglebone Black using a Skywire Beaglebone Black Cape.
2. You can successfully use your Skywire modem to establish a PPPd connection.
3. You have a Public Static IP address on your Skywire modem. If you do not, and NimbeLink handled your Skywire activation, please contact NimbeLink to get that changed. If you activated your Skywire modem on your own account, please contact Verizon to get that changed.

Note: You will need to know your Public Static IP address, as that will be the address of the video server's stream.

Please consult the following documents for information if you do not meet the above requirements.

- NimbeLink's Skywire Beaglebone Black Cape User Manual
- NimbeLink's Skywire Beaglebone Black LTE PPPd Application Note

In addition, we will be downloading utilities and programs to get the streaming to work. It is highly recommended that you have your Beaglebone Black connected to an Ethernet connection or a Wifi connection through this process. If you download the utilities over the cellular PPPd connection with your Skywire, you could use your data allowance and incur overage charges.

Note on documentation: This guide is using command run as root. If you are not logged in as root on your Beaglebone Black, make sure to include sudo before your commands as necessary.

2.2 Updating the Beaglebone Black

Power on the Beaglebone with the USB camera disconnected.

As mentioned in Section 1.4, there is an issue using RT Kernel 4.1 on Debian 8.4 for Beaglebone Black. Therefore, it is recommended that you update your kernel to a newer version. As of the writing of this application note, we recommend Kernel 4.4.x if you are using Debian 8.x.

First, login to the Beaglebone Black and get your Kernel version:

```
# uname -r
```

and you will see the kernel version that you have. Use the following section for your version:

Version 4.1.x

If you are running 4.1.x, you will need to update your Kernel.

First, update and search for the latest Kernel version:

```
# apt-get update
```

```
# apt-cache search linux-image | grep bone
```

This will return a long list of the current available Kernels for BeagleBone Black. As of the time of this writing, we recommend the Kernel 4.4.x non-RT. For instance, here is an example entry, and the one we recommend:

```
linux-image-4.4.9-bone10 - Linux kernel, version 4.4.9-bone10
```

Next, install the Kernel. Using the above listing:

```
# apt-get install linux-image-4.4.9-bone10
```

Once the install is complete, reboot and verify you have the correct version:

```
# uname -r
```

If it looks good, run the Kernel upgrade script:

```
# cd /opt/scripts/tools
```

```
# ./update_kernel.sh --bone-kernel --lts-4_4
```

Once you have finished the upgrade, reboot, and update and upgrade:

```
# apt-get update
```

```
# apt-get upgrade
```

Proceed to Section 2.3

Version 4.4.x

If you are running Kernel 4.4.x, you just need to run the Kernel update script:

```
# cd /opt/scripts/tools
```

```
# ./update_kernel.sh --bone-kernel --lts-4_4
```

Once you have finished the upgrade, reboot, and update and upgrade:

```
# apt-get update
```

```
# apt-get upgrade
```

Proceed to the next Section.

2.3 Setting Up the Beaglebone Black

To begin, we will need to install some packages onto the Beaglebone Black to get the video streaming working.

Log into the Beaglebone Black as root, and install the following:

```
# apt-get install cmake make libssl-dev libav-tools
```

```
# apt-get install libcurl4-openssl-dev pkg-config git
# apt-get install v4l-utils
```

```
# pkg-config openssl --cflags --libs
```

These commands will install and setup the necessary packages for building the streaming server and streaming software.

2.4 Installing the CRTMP Server

CRTMP is the video stream server that we will use. It is a free, C++-based RTMP server. To begin, clone the repository:

```
# cd /opt
# git clone https://github.com/shiretu/crtmpserver.git
```

And navigate to the following folder:

```
# cd crtmpserver/builders/cmake/cmake_find_modules
```

Next, we need to edit the following files to add an ARM toolchain for compiling and running the program. Add the following line:

```
 /usr/lib/arm-linux-gnueabihf
```

above the `/usr/lib64` line in each of the following files and sections:

- `Find_openssl.cmake`
  - `PATHS` section of `'ssl'`, `'crypto'`, and `'z'`
- `Find_pcap.cmake`
  - `PATHS` section of `'pcap'`
- `Find_dl.cmake`
  - `PATHS` section of `'dl'`
- `Find_lua.cmake`
  - `PATHS` section of `'lua'`

Once you have added the above lines to their respective files, go to the cmake directory:

```
# cd /opt/crtmpserver/builders/cmake
```

and start the build process:

```
# ./run
```

The build process will take about fifteen minutes or more to run. Once it is complete, the server will try to start but will fail, giving an error that the IP address it is trying to use is already in use. This is expected.

The last step for installing the CRTMP server is to modify the `.lua` file that starts the server. First, navigate to the directory:

```
# cd /opt/crtmpserver/builders/cmake/crtmpserver
```

and make a backup of the current `lua` script there:

```
# mv crtmpserver.lua crtmpserver.lua.bak
```

Create the crtmpserver.lua file:

```
# touch crtmpserver.lua
```

and using your favorite text editor, paste in the following information:

```
-- Start of the configuration. This is the only node in the config file.
-- The rest of them are sub-nodes
configuration=
{
        -- if true, the server will run as a daemon.
        -- NOTE: all console appenders will be ignored if this is a daemon
        daemon=false,
        -- the OS's path separator. Used in composing paths
        pathSeparator="/",

        -- this is the place where all the logging facilities are setted up
        -- you can add/remove any number of locations

        logAppenders=
        {
                {
                        -- name of the appender. Not too important, but is mandatory
                        name="console appender",
                        -- type of the appender. We can have the following values:
                        -- console, coloredConsole and file
                        -- NOTE: console appenders will be ignored if we run the server
                        -- as a daemon
                        type="coloredConsole",
                        -- the level of logging. 6 is the FINEST message, 0 is FATAL message.
                        -- The appender will "catch" all the messages below or equal to this
level
                        -- bigger the level, more messages are recorded
                        level=6
                },
                {
                        name="file appender",
                        type="file",
                        level=6,
                        -- the file where the log messages are going to land
                        fileName="/tmp/crtmpserver",
                        --newLineCharacters="\r\n",
                        fileHistorySize=10,
                        fileLength=1024*256,
                        singleLine=true
                }
        },

        -- this node holds all the RTMP applications
        applications=
        {
                -- this is the root directory of all applications
                -- usually this is relative to the binary execuable
                rootDirectory="applications",

                --this is where the applications array starts
                {
                        -- The name of the application. It is mandatory and must be unique
                        name="appselector",
                        -- Short description of the application. Optional
                        description="Application for selecting the rest of the applications",

                        -- The type of the application. Possible values are:
                        -- dynamiclinklibrary - the application is a shared library
                        protocol="dynamiclinklibrary",
                        -- the complete path to the library. This is optional. If not provided,
                        -- the server will try to load the library from here
                        -- //lib.{so|dll|dylib}
                        -- library="/some/path/to/some/shared/library.so"

                        -- Tells the server to validate the clien's handshake before going
further.
                        -- It is optional, defaulted to true
                        validateHandshake=false,
                        -- this is the folder from where the current application gets it's
content.
                        -- It is optional. If not specified, it will be defaulted to:
                        -- //mediaFolder
                        -- mediaFolder="/some/directory/where/media/files/are/stored"
                        -- the application will also be known by that names. It is optional
                        --aliases=
                        --{
                        --      "simpleLive",
                        --      "vod",
                        --      "live",
                        --},
                        -- This flag designates the default application. The default application
                        -- is responsable of analyzing the "connect" request and distribute
                        -- the future connection to the correct application.
                        default=true,
                        acceptors =
                        {
                                {
                                        ip="0.0.0.0",
                                        port=1935,
                                        protocol="inboundRtmp"
                                },
                                {
                                        ip="0.0.0.0",
                                        port=8081,
                                        protocol="inboundRtmps",
                                        sslKey="server.key",
                                        sslCert="server.crt"
                                },
                                {
                                        ip="0.0.0.0",
                                        port=8080,
                                        protocol="inboundRtmpt"
                  },
                        }
                },
                {
                        description="FLV Playback Sample",
                        name="flvplayback",
                        protocol="dynamiclinklibrary",
                        aliases=
                        {
                                "simpleLive",
                                "vod",
                                "live",
                                "WeeklyQuest",
                                "SOSample",
                                "oflaDemo",
                        },
                        acceptors =
                        {
                                {
                                        ip="0.0.0.0",
                                        port=6666,
                                        protocol="inboundLiveFlv",
                                        waitForMetadata=true,
                                },
                                {
                                        ip="0.0.0.0",
                                        port=9999,
                                        protocol="inboundTcpTs"
                                },
                                {
                                        ip="0.0.0.0",
                                        port=10000,
                                        protocol="inboundUdpTs"
                                },
                                --[[{
                                        ip="0.0.0.0",
                                        port=7654,
                                        protocol="inboundRawHttpStream",
                                        crossDomainFile="/tmp/crossdomain.xml"
                                }, ]]--
                                {
                                        ip="0.0.0.0",
                                        port=554,
                                        protocol="inboundRtsp"
                                },
                        },
                        externalStreams =
                        {
                                --[[
                                {

  uri="rtsp://fms20.mediadirect.ro/live2/realitatea/realitatea",
                                        localStreamName="rtsp_test",
                                        forceTcp=true
                                },
                                {
                                        uri="rtmp://edge01.fms.dutchview.nl/botr/bunny",
                                        localStreamName="rtmp_test",
                                        swfUrl="http://www.example.com/example.swf",
                                        pageUrl="http://www.example.com/",
                                        tcUrl="rtmp://edge01.fms.dutchview.nl/botr/bunny", --this
  one is usually required and should have the same value as the uri
                                        emulateUserAgent="MAC 10,1,82,76",
                                }
                                {

  uri="rtsp://animalhousenc.dvrdns.org:554/streaming/channels/0",
                                        localStreamName="PoolSide",
                                        forceTcp=true
                                },
                                        {

  uri="rtsp://animalhousenc.dvrdns.org:556/streaming/channels/0",
                                               localStreamName="BoneYard",
                                               forceTcp=true
                                },
                                {

  uri="rtsp://animalhousenc.dvrdns.org:557/streaming/channels/0",
                                               localStreamName="BigPool",
                                               forceTcp=true
                                },
                                        {

  uri="rtsp://192.168.1.186:554/mpeg4/media.amp?videocodec=h264&streamprofile=high",
                                               localStreamName="nerd",
                                               forceTcp=true
                                },
                                        {
                                               uri="rtsp://192.168.1.190:554/0",
                                               localStreamName="leopard",
                                               forceTcp=true
                                }, ]]--
```

```
		},
		validateHandshake=false,
		--enableCheckBandwidth=true,
		--[[authentication=
		{
			rtmp={
				type="adobe",
				encoderAgents=
				{
					"FMLE/3.0 (compatible; FMSc/1.0)",
					"My user agent",
				},
				usersFile="users.lua"
			},
			rtsp={
				usersFile="users.lua"
			}
		}, --]]
		mediaStorage = {
		--[[	namedStorage1={
				--this storage contains all properties with their
				--default values. The only mandatory property is
				--mediaFolder
				description="Some storage",
				mediaFolder="/Volumes/Storage/media/",
				metaFolder="/tmp/metadata",
				enableStats=false,
				clientSideBuffer=15,
				keyframeSeek=false,
				seekGranularity=0.1,
			},
			namedStorage2={
				mediaFolder="/Volumes/Storage/media/mp4",
				metaFolder="/tmp/metadata",
				seekGranularity=0.2,
				enableStats=true,
			},
			namedStorage3={
				mediaFolder="/Volumes/Storage/media/flv",
				metaFolder="/tmp/metadata",
			},
			{
				--this one doesn't have a name
				mediaFolder="/Volumes/Storage/media/mp3",
			} --]]
		},
	},
	{
		name="samplefactory",
		description="asdsadasdsa",
		protocol="dynamiclinklibrary",
		aliases=
		{
			"httpOutboundTest"
		},
		acceptors =
		{
			{
				ip="0.0.0.0",
				port=8989,
				protocol="httpEchoProtocol"
			},
			{
				ip="0.0.0.0",
				port=8988,
				protocol="echoProtocol"
			}
```

```
                        },
                        validateHandshake=false,
                        --default=true,
                },
                {
                        name="vptests",
                        description="Variant protocol tests",
                        protocol="dynamiclinklibrary",
                        aliases=
                        {
                                "vptests_alias1",
                                "vptests_alias2",
                                "vptests_alias3",
                        },
                        acceptors =
                        {
                                {
                                        ip="0.0.0.0",
                                        port=1111,
                                        protocol="inboundHttpXmlVariant"
                                }
                        },
                        validateHandshake=false,
                        --default=true,
                },
                {
                        name="admin",
                        description="Application for administering",
                        protocol="dynamiclinklibrary",
                        aliases=
                        {
                                "admin_alias1",
                                "admin_alias2",
                                "admin_alias3",
                        },
                        acceptors =
                        {
                                {
                                        ip="0.0.0.0",
                                        port=1112,
                                        protocol="inboundJsonCli",
                                        useLengthPadding=true
                                },
                        },
                        validateHandshake=false,
                        --default=true,
                },
                {
                        name="proxypublish",
                        description="Application for forwarding streams to another RTMP server",
                        protocol="dynamiclinklibrary",
                        acceptors =
                        {
                                {
                                        ip="0.0.0.0",
                                        port=6665,
                                        protocol="inboundLiveFlv"
                                },
                        },
                        abortOnConnectError=true,
                        targetServers =
                        {
                                --[[{

targetUri="rtmp://x.xxxxxxx.fme.ustream.tv/ustreamVideo/xxxxxxx",
                                        targetStreamName="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
                                        localStreamName="stream1",
                                        emulateUserAgent="FMLE/3.0 (compatible; FMSc/1.0
http://www.rtmpd.com)"
                                }]]--,
                                --[[{
                                        targetUri="rtmp://gigi:spaima@localhost/vod",
                                        targetStreamType="live", -- (live, record or append)
                                        emulateUserAgent="My user agent",
                                        localStreamName="stream1",
                                        keepAlive=true
                                },]]--
                        },
                        externalStreams =
                        {
                                --[[{

uri="rtsp://fms20.mediadirect.ro/live2/realitatea/realitatea",
                                        localStreamName="stream1",
                                        forceTcp=true,
                                        keepAlive=true
                                },]]--
                        },
                        validateHandshake=false,
                        --default=true,
                },
                {
                        name="stresstest",
                        description="Application for stressing a streaming server",
                        protocol="dynamiclinklibrary",
                        targetServer="localhost",
                        targetApp="vod",
                        active=false,
                        --[[streams =
                        {
                                "lg00","lg01","lg02","lg03","lg04","lg05","lg06","lg07","lg08",
                                "lg09","lg10","lg11","lg12","lg13","lg14","lg15","lg16","lg17",
                                "lg18","lg19","lg20","lg21","lg22","lg23","lg24","lg25","lg26",
                                "lg27","lg28","lg29","lg30","lg31","lg32","lg33","lg34","lg35",
                                "lg36","lg37","lg38","lg39","lg40","lg41","lg42","lg43","lg44",
                                "lg45","lg46","lg47","lg48","lg49"
                        },]]--
                        streams =
                        {
                                "mp4:lg.mp4"
                        },
                        numberOfConnections=10,
                        randomAccessStreams=false
                },
                --[[{
                        name="vmapp",
                        description="An application demonstrating the use of virtual machines",
                        protocol="dynamiclinklibrary",
                        vmType="lua",
                        script="flvplayback.lua",
                        aliases=
                        {
                                "flvplayback1",
                                "vod1"
                        },
                        acceptors=
                        {
                                {
                                        ip="0.0.0.0",
                                        port=6544,
                                        protocol="inboundTcpTs"
                                }
                        }
                },]]--
```

```
		--#INSERTION_MARKER# DO NOT REMOVE THIS. USED BY appscaffold SCRIPT.
	}
}
```

Save and close the file.

2.5 Download and Install boneCV

To handle the video streaming, we will use Derek Molloy’s boneCV program. Type the following into the command prompt:

```
# cd /opt
```

```
# git clone git://github.com/derekmolloy/boneCV
```

This will download the program to our Beaglebone Black.

Change into the boneCV directory, and build the `capture` program:

```
# gcc capture.c -o capture
```

Once the building is complete, you will have the necessary capture program you need to stream from the webcam.

Open a text editor, and type (or copy/paste) the following:

```
#!/bin/bash
echo "Video Streaming for the Beaglebone - derekmolloy.ie"
echo "Piping the output of capture to avconv"
#1080P mode
v4l2-ctl --set-fmt-video=width=1920,height=1080,pixelformat=1
#720P mode
#v4l2-ctl --set-fmt-video=width=1280,height=720,pixelformat=1

# Pipe the output of capture into avconv/ffmpeg
# capture "-F"   My H264 passthrough mode
#     "-o"   Output the video (to be passed to avconv via pipe)
#     "-c0"  Capture 0 frames, which means infinite frames in my program
# avconv "-i -"  Take the input from the pipe
#     "-vcodec copy" Do not transcode the video

#1080P mode
#./capture -F -o -c0|avconv -re -i - -f alsa -ac 2 -i hw:1,0 -strict
experimental -threads 0 -acodec aac -ab 64k -ac 2 -vcodec copy -f  rtsp
-metadata title=teststream rtsp://127.0.01:554/live
#1080P mode, no audio
./capture -F -o -c0|avconv -re -i - -vcodec copy -f  rtsp -metadata
title=teststream rtsp://127.0.01:554/live
#720P mode
#./capture -f -o -c0|avconv -re -i - -f alsa -ac 2 -i hw:1,0 -strict
experimental -threads 0 -acodec aac -ab 64k -ac 2 -vcodec copy -f  rtsp
-metadata title=teststream rtsp://127.0.01:554/live
```

Save the file in the `/opt/boneCV/` directory as `streamVideoRTMP`.

Note that there are options in the script for 720p and 1080p (the above script is set for 1080p). Make sure only one set is uncommented at a time.

Change permissions on the file to be able to run it:

```
# chmod a+x streamVideoRTMP
```

2.6 Start PPPd

If you do not have the PPPd files from the PPPd Application Note, follow the below instructions.

Change to the PPPd folder:

```
# cd /etc/ppp/peers/
```

Open a text editor and type (or copy/paste) the following:

For the Skywire CAT 3 Modem:

```
/dev/ttyUSB3
115200
connect "/usr/sbin/chat -v -f /etc/ppp/peers/verizon-chat"
noauth
defaultroute
usepeerdns
local
debug
updetach
```

For the Skywire CAT 4 Modem:

```
/dev/ttyACM0
115200
connect "/usr/sbin/chat -v -f /etc/ppp/peers/verizon-chat"
noauth
defaultroute
usepeerdns
local
debug
updetach
```

Save the file as `verizon`.

For the Skywire CAT 3 and CAT 4 Modem:

Open a text editor again, and type (or copy/paste) the following:

```
TIMEOUT 35
ECHO ON
ABORT '\nBUSY\r'
ABORT '\nERROR\r'
ABORT '\nNO ANSWER\r'
ABORT '\nNO CARRIER\r'
ABORT '\nNO DIALTONE\r'
ABORT '\nRINGING\r\n\r\nRINGING\r'
'' \rATZ
OK 'ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0'
OK AT+CGDCONT=3,"IP","mw01.vzwstatic","0.0.0.0",0,0
OK ATD*99***3#
CONNECT ''
```

Save the file as `verizon-chat`.

Now that you have the necessary script files created, you can start PPPd.

First, bring down the Ethernet and USB network interfaces:

```
# ifconfig eth0 down
```

```
# ifconfig usb0 down
```

Ensure that both are down by typing:

```
# ifconfig
```

If they are not down, run the down commands again.

Start PPPd:

```
# pon verizon
```

Once you have an IP address, ensure that your PPP connection is running by typing:

```
# ping google.com
```

Once it sends a few packets, use `CTRL-C` to stop the transmission.

2.7 Start the Video Server and Video Stream

Once you have your PPPd connection, plug in your webcam, and start your CRTMP server:

```
# killall apache2
```

```
# cd /opt/crtmpserver/builders/cmake/
```

```
# ./crtmpserver/crtmpserver --daemon --use-implicit-console-appender
./crtmpserver/crtmpserver.lua
```

You should be presented with a stream of output from `crtmpserver`, including a table of the connections and ports of your `crtmpserver`. You should see an entry for the following:

				Services
c	ip	port	protocol stack name	application name

tcp	0.0.0.0	554	inboundRtsp	flvplayback

This is the stream service we will be using for our port. This is already setup in the `streamVideoRTMP` script you created, but it is good to know for your reference.

Finally, start your video stream:

```
# cd /opt/boneCV
# ./streamVideoRTMP
```

Once your stream has started, you should notice that the activity light on your webcam has turned on, indicating a stream is in progress. To test out your stream, visit the following site (Note: this site requires Flash):

http://www.wowza.com/resources/3.5.0/examples/LiveVideoStreaming/FlashRTMPPlayer/player.html

Under “Server”, replace:

rtmp://localhost/live

with:

rtmp://[The static IP address of your Skywire Modem]/live

For the stream, replace:

myStream

with:

teststream

It will take up to twenty seconds for the stream to connect. Once connected, you should have a video stream on that website.

Note: Due to the CRTMP Server and network latency, there is about a 3-5 second delay in the video.

You now have a streaming video solution using a Skywire Modem and a Beaglebone Black.

2.8 References

The following outside references were cited in writing this application note. For more information, please consult:

Beaglebone Black Kernel Update Procedure:

http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#Expanding_File_System_Partition_On_A_microSD

CRTMP Server Setup:

http://nerdlogger.com/2013/11/09/streaming-1080p-video-using-raspberry-pi-or-beaglebone-black/

boneCV Setup:

http://derekmolloy.ie/streaming-video-using-rtp-on-the-beaglebone-black/