Enabling Green Video Streaming over Internet of Things

(8^{th} Quarter Deliverable)

Dr. Ghalib Asadullah Shah (PI)

Next-generation Wireless Networking (NWN) Lab,

Al-Khawarizmi Institute of Computer Science,

University of Engineering and Technology, Lahore

19-12-2015

About this Document

This document reports the activities performed in the 8th quarter of our project 'Enabling Green Video Streaming over Internet of Things' and the corresponding deliverable to be submitted to ICT R & D Fund.

Firstly, the report provides a performance comparison of the traditional video encoding technique with our proposed compressive sensing video encoding technique. The experimental analysis done in this deliverable, comprehensively evaluate the performance of the two encoding schemes. The aim of this work is to reduce video encoder complexity(both space and time), by the use of compressed sensing, while maintaining fairly low bit-rates (for transmission). In this deliverable we have implemented the CBS technique on the Raspberry Pi 2 with, 700 MHz Quad Core and Raspbian Debian (Linux) OS running on it. For performance comparison we are using the H.264 library in the ffmpeg software installed on the same Pi 2. The key idea behind this implementation was to make a hybrid codex, based on the theory of compressed sensing, while maintaining some useful features of existing codex, such as H.264.

All the proposed communication protocols proposed in the previous deliverables are implemented in the green camera node. For example, the proposed green routing protocol Green-RPL and the enhanced power save mode of IEEE 802.11 are implemented in the Contiki-OS. Moreover, the proposed compressive sensing based encoder CBS-Video Coding Technique is also implemented in the green camera node.

As part of this deliverable, the promised journal research paper is also attached which is ready for submission.

Contents

List of Figures

List of Tables

1 H.264

The H.264 standard was published in 2003 and is based on MPEG-4. There are a number of advantages provided by H.264 video coding standard [1]. H.264 provides support for bit rate adaptation that increases flexibility for packet level transmission at the network adaptation layer (NAL). Although it is a video compression method for converting real time digital video sequence into bit streams that requires less transmission capacity. The H.264 uses predefined tools available in decoding phase termed as profiles. Each profile of H.264 defines certain tools. In encoder phase also H.264 have variety of tools available. For transmission it provides number of support for illuminating transmission errors and compose packets in compressed format.

1.1 Coding Fragments of H.264

H.264 profile have following coding parts in general.

- Intra-coded slice (I slice)
- Predictive-coded slice (P slice)
- Context-based Adaptive Variable Length Coding (CAVLC) for entropy coding

The architecture of H.264 increases the capabilities offering solutions for encoding phase that support temporal, spatial and SNR qualities. The H.264 encoder is more complex than any previous video coding standard. There are computation and memory overheads in the development of embedded encoder/decoder. It does not offer any compatibility with the older versions or video coding standards. To generate scalable compressed video bit streams, punctured turbo codes are applied. To get overall performance improvement, usually following steps are performed

- Predict a new frame from a previous frame and only specify the prediction error.
- Prediction error will be coded using an image coding method.
- Prediction errors have smaller energy than the original pixel values and can be coded with fewer bits. In this standard the system is broken into two layers [2]: Network Abstraction Layer (NAL) and Video Coding Layer (VCL)

Video information is transformed into bitstreams through VCL, after the conversion the NAL layer maps the transformed bitstreams into NAL units those are HDLC-like and byte-oriented transportation-layer before delivery. Usually the NAL header is composed

of three fields. NAL unit header is the first byte after the NAL unit code prefix. The bits are categorized as following.

- Forbidden-bit (1 bit)
- NAL-storage-idc (2 bit)
- NAL-unit-type (5 byte)

forbidden-bit is used to indicate whether the NAL unit is corrupted or not. NAL-storage-idc is of two bits, which have relative importance, when the picture is buffered. The NAL-unit-type is of one-ten bits according to need. Usually they are referred as reserved bits. Particularly three operations are performed during the mapping phase those are

1) Byte Alignment: NAL unit adds byte header, that defines the categorization of the NAL unit. Synchronization is performed in the NAL unit through byte sequence.

2) Emulation Prevention: Emulation prevention bytes are used to interleave NAL unit payload data [3]. Those bytes prevent accidental data generation within the payload. These bytes are termed as start code prefix, which are inserted in the data pattern. NAL unit structure are used in both bitstream-oriented and packet-oriented transport systems.

3) Framing: H.264 slices frames according to five types, which are listed below.

- I(Intra) Slice: This type reference only itself. Usually it is the first received image of the video sequence or a still image. All the first frames from video sequences needed to be built from I-slices.
- P(Predictive) slice: It takes reference from decoded or predicted slices for construction of video sample/image. Prediction is mostly not accurate therefore some residual images may be added.
- B(Bi-Directional) Slice: It takes reference from future and former P or I slices, except of that Bi slices are very similar to P slices. For the reason, I an dP slices are decoded after B slices.
- SI and SP (Switching) Slices:Usually they are used for transition between video samples of different natures.

1.2 H.264 Main Modules

Spatial dependency layer in H.264 have the essential requirement of its own prediction module for performing intra and motion-compensated predictions within the layer. Another module that manages the scalability of quality is termed as SNR refinement module. In the inter-layer prediction module, the dependency is managed between spatial layers by reusing residual signal and motion vectors so as to improve compression

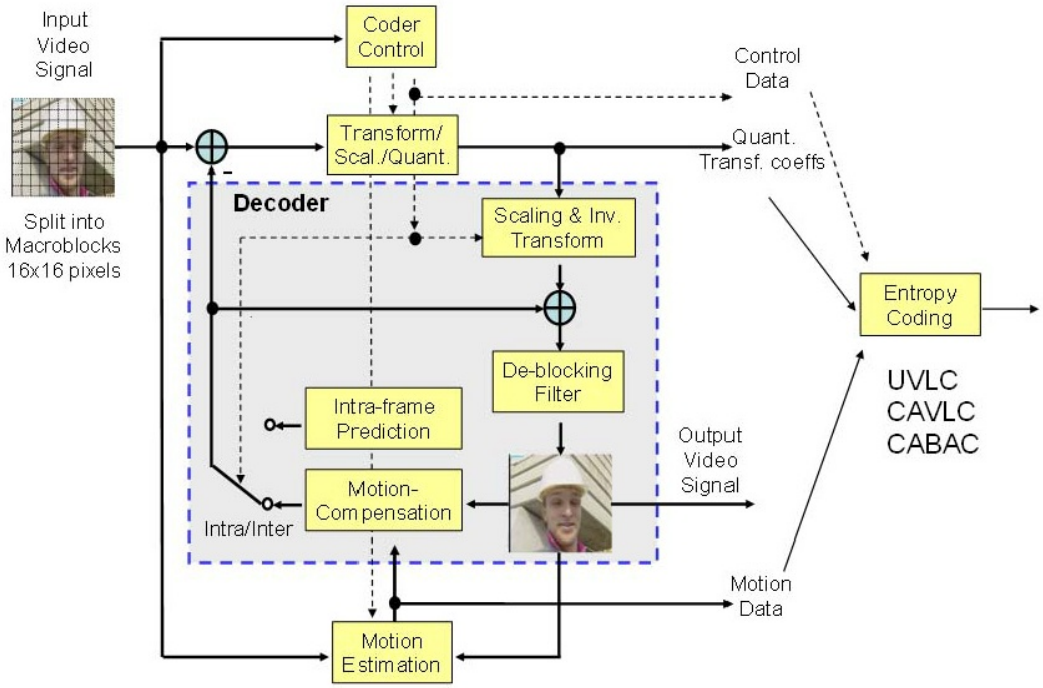

Figure 1.1: H.264 Codec Block Diagram.

efficiency. All the modules are finally merged in a multiplex, a single integrated scalable bitstream having different spatial, temporal and SNR levels.

The process is combination of forward and inverse (encoding and decoding) path as depicted in Fig. 1.1. The video frames are structured in macroblocks upon which prediction is performed using either Inter or Intra predictions. After the transformation and quantization process, the resultant product is forwarded to Entropy Encoder Module. The output packets are finally formed in the Network Abstraction Layer (NAL) module. Decoding or Inverse path involves reconstruct of Macroblock data from previously transformed module which consist of Deblocking Filter [4] and Transform and Quantization(ITQ).

1.2.1 Encoding Modes

Generally there are six encoding modes presented in academic and commercial bases, which are listed below

- Single Pass-Constant Quantizer
- Single Pass-Constant Rate Factor
- Single Pass-Bitrate
- Two Pass- Average Bitrate

- Two Pass-File Size
- Lossless Mode

1.2.2 Prediction Model

This section individually explains the characterization of all types of scalability [5]. It is generally suitable if the encoding and decoding complexity tallies itself proportionally with accordance to temporal and spatial resolution. A hierarchy should be established for video compression tools which involves scalability complexity.

1.2.3 Temporal Scalability

In this category video stream is transferred as subset of bitstream. Video frames are classified as three distinct types: I (intra), P (predictive) and B (Bi-predictive) as already known. In this procedure, three types of simplified motion compensation, termed as Temporal Prediction.

2 CBS Video Coding Technique

The aim of this work is to reduce video encoder complexity(both space and time), by the use of compressed sensing, while maintaining fairly low bit-rates (for transmission). In this deliverable we have implemented the CBS technique on the Raspberry Pi 2 with, 700 MHz Quad Core and Raspbian Debian (Linux) OS running on it. For performance comparison we are using the H.264 library in the ffmpeg software installed on the same Pi 2. The key idea behind this implementation was to make a hybrid codex, based on the theory of compressed sensing, while maintaining some useful features of existing codex, such as H.264.

Internet of Multimedia (IoM) devices has accelerated the development of low complexity video coding techniques.However, the IoM devices have strict constraints on the cost, power, bandwidth and computational capability which limits the usage of current video coding techniques (H.265, VP9, etc.) due to their high computational complexity on the encoder side. However, emerging techniques from the area of Compressive Sensing(CS) have given rise to a new generation of low complexity video encoders. In CS based techniques, compression is inherent in the acquisition stage whereby a small number of linear measurements of the scene are taken at sub-Nyquist rates before data being passed on to the encoder. To match IoM device constraints, we have implemented a new video compression technique, namely Change-based Block Suppression (CBS), which shifts the complexity from encoder to decoder.

An adaptive video encoding/decoding scheme is designed in which compressed sensed video data is further compressed within the CS domain. A nested approach is introduced in which CS frames are compressed further (during and) after acquisition on the encoder side, and on the decoder side the full CS frames are reconstructed and then recovery algorithms are used for the reconstruction of video frames. Our CBS technique is independent (if the CS technique is block based) of the type of measurement matrix and the recovery technique being used. This is followed by pre-acquisition block suppression based on block analysis at the start of each GoP and post-acquisition block suppression based on block analysis on a frame-by-frame basis. Using our block suppression technique, instead of labeling individual MB's, a two dimensional binary array (with one element representing one MB) is sent with each frame.

2.0.4 Compressed Sensing and Further Compression

We are performing CS on a MB by MB basis (as described in our Q5 & Q7 reports in detail). Therefore in this process we have rearranged the data into columns where each column is one MB. This is done by reading the whole Y component from a file, placing chunks into columns and then writing the columnized data back to another

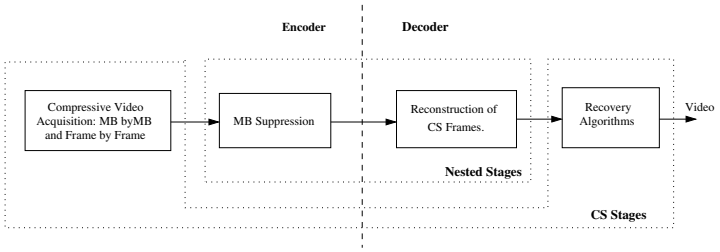

Figure 2.1: Nested video encoding-decoding.

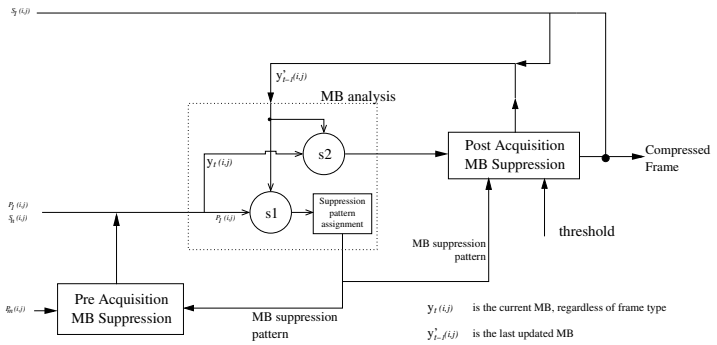

Figure 2.2: The Change based block Suppression Architecture(CBS).

file in memory. At this stage, we have 30 frames each having 15 columnized MB's. Now we apply CS on each MB one-by-one and write the CS MB's back to a third file using f_write function. The measurement matrix (i.i.d Gaussian Random matrix) is stored in yet another file. For each measurement we read one row of the measurement matrix from the SD Card (using the f_read function), perform the floating point matrix multiplication , and store the measurement in a CS-MB column-matrix. . In this way, each row of the measurement matrix is read one-by-one and the same procedure is repeated until the desired number of measurements are taken, and consequently saved to a file.

The same procedure is repeated for all the MB's. Now that we have 15 columnized CS-MB's (for each frame), we retrieve CS-MB's (from the file in which they were stored) of neighboring frames to calculate SAD values, which are subsequently used for block suppression. Finally compressed frames are stored back into the SD card, ready to be passed down to the transmission module.

2.0.5 Reconstruction and Recovery

Recovery of these frames is performed in the MATLaB on any WINDOWS OS. Reconstruction at the decoder is in two stages. Initially, the full CS frames are recovered followed by algorithms for recovery of video frames. The compressed frame received by the decoder has two parts. First, all the unsuppressed columnized MB's are bundled together in the frame $\hat{F}_t$ such that each column, $\hat{F}_t(:, i)$, is a MB. The second part is a

bit-array, I_t which has the compression information. So for reconstruction of CS frames we traverse through I_t and for every zero entry, we place a copy of the corresponding MB from the previous reconstructed CS frame, F_{t-1}, into the current reconstructed CS frame, F_t. Similarly for every non-zero entry in I_t we place the next-up MB from $\hat{F}_t$ into the current reconstructed CS frame, F_t. Once the current CS frame is reconstructed, the video frame can be recovered using any recovery algorithm in the bases in which the signal of interest is sparse. The elements of the vectorized measurement vector are quantized individually by an 8-bit uniform scalar quantizer. At the decoder, total variation minimization(min-TV) [6, 7] is used for the recovery.

2.1 Performance Analysis

In this section, we evaluate the performance of our adaptive compression technique on three standard test sequences, container, coastguard and hall. Of the three sequences, coastguard has the most abrupt temporal variations. All sequences are in special resolution of 352 $\times$ 258 pixels with a frame rate of 25f/s (only the luminance component is processed). The sample size for each test is 30 frames (2.8 GoP's). For the general evaluation we have done rate-performance studies, where performance is the ratio of distortion (PSNR) over execution time, T_E (sec). This performance matrix indicates the overall gain in using our low complexity CBS technique.

For experimentation with our CBS technique, we kept a MB size of 32 $\times$ 32 in the CS encoding (CS acquisition) framework. Therefore the number of pixels, N, in each MB is 1024. The overall compression achieved depends on the number of measurements, M, so we have taken results for different number of measurements, (M): 256, 170 and 128. And for the H.264 performance, we have used the H.264 library of ffmpeg, to obtain a single set of results: rate, distortion, execution time.
For each of these we obtained the Peak Signal to Noise Ratio(PSNR) values along with the execution time(T_E). The elements of the vectorized measurement vector are quantized individually by an 8-bit uniform scalar quantizer. The comparison is fair because both the techniques are being run on the same platform, Raspberry Pi 2 with, 700 MHz Quad Core and Raspbian Debian (Linux) OS running on it At the decoder we have chosen to use total variation minimization (min-TV) [6, 7], for the recovery, due to its exceptional recovery performance.

The rate-distortion data (in Fig.2.3) shows that our CBS technique is able to bring down bit-rate (more compression), to the extent that we are able to compare our encoder to the industry standard video codex H.264 instead of other CS based techniques which fail to bring down bit-rates (more compression) to such low ranges. The bar-chart representation of the results gives a clear comparison of H.264 and our CBS technique.

As discussed earlier in this section; the rate-performance studies for Container, Hall, Coastguard, (going from top to bottom) have been presented in Fig.2.3. It is clear from the charts that using our CBS-minTV technique, we have significantly reduced the encoder complexity, which has resulted in far superior, overall performance when compared to H.264, for all the test sequences. Also it can be observed that for the

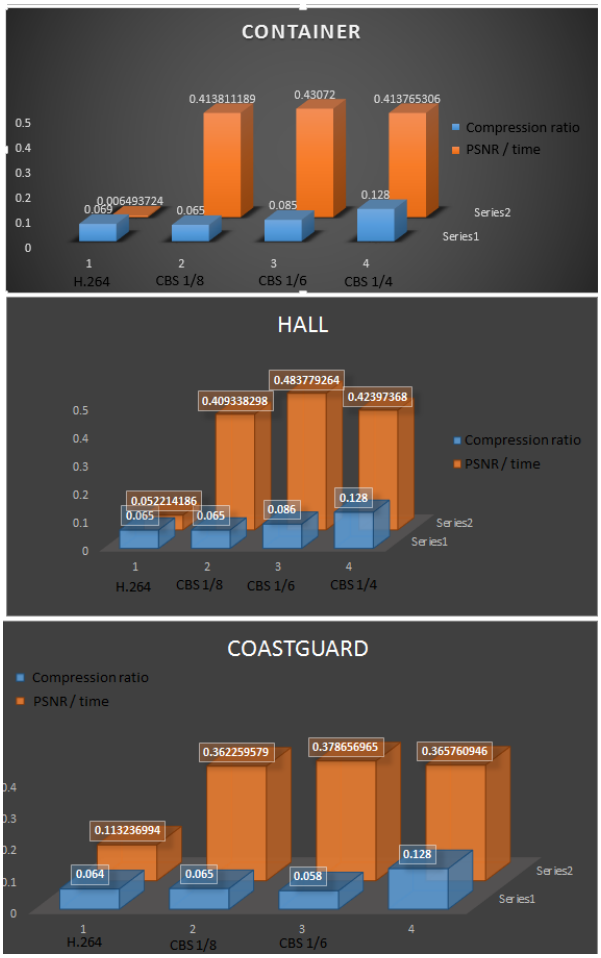

Figure 2.3: Rate-performance studies on the Container, Hall Monitor, Coastguard, from top to bottom.

sequences with lower temporal variation, i.e. Container and Hall, our CBS-minTV technique brings down the bit-rates (more compression) to the 100 kbps range. While for the Coastguard sequence, our technique shows a more modest attempt, primarily due to greater temporal change.

2.2 Conclusion

We proposed a Change based Block Suppression technique, nested inside the CS domain. The complexity is kept significantly low both in the design and implementation of the CBS technique. Our encoder outperformed H.264 in the overall distortion/execution-time study, thanks to it's very low execution-times. However H.264 performance remains superior when distortion alone is considered. Since our technique is universal to all block based CS techniques, future work can be done to improve PSNR using existing CS techniques as well as devising new techniques to improve performance of CBS. Considering

resource constraints, we have developed expertise in near-optimal utilization of embedded system resources based upon which, one can build a more robust CS based camera node. A lot of analytical research work can be done to model the relation between resources and compression threshold.

3 Green camera node

The camera node has four basic components, a CMOS camera, a microcontroller board for encoding, a microcontroller board running OS, and a Wi-Fi module. In this section, we will discuss each of these components in the following sections. The block diagram depicting each of these components of the green camera node is shown in Figure **??**.

3.1 CMOS camera

As the paradigm of IoT develops, it is evident that 'things' will be predominantly simple, and in large numbers. For this reason and for ease of development we have selected the standard CIF resolution of 288x352. We have to perform compressed sensing inside our controller (in digital domain) so the frame data from the camera (CMOS sensor) needs to be in a raw format (compressed sensing can only be performed on raw data). There are many raw image formats such as, YUV/YCbCr4:2:2 RGB565/555/444 GRB4:2:2 Raw RGB. So we need a camera which gives some of these formats. Another requirement is that the camera interface needs to be compatible with the camera interface on the Base Board. We have selected OV7670 camera to fit these requirements. The camera accept for a few variations in the output video formats. The salient features of this camera are listed below:

Photosensitive Array: 640 x 480 IO
Large capacity with 380KB FIFO Buffer IC AL422B, easy to apply with
MCU/ARM system via I/O port
24MHZ active crystal
Voltage: 3.3V
Operating Power: 60mW/15fpsVGAYUV
Sleeping Mode: ¡ 20 uA
Operating Temperature: -30 to 70 deg C.
Output Format: YUV/YCbCr4:2:2 RGB565/555/444 GRB4:2:2 Raw RGB
Data (8 digit)
Lens Size: 1/6 inches, Vision Angle: 25 degree
Max. Frame Rate: 30fps VGA
Sensitivity: 1.3V / (Lux-sec)
Signal to Noise Ratio: 46 dB
Dynamic Range: 52 dB
Browse Mode: By row
Electronic Exposure: 1 to 510 row
Pixel Coverage: 3.6um x 3.6um

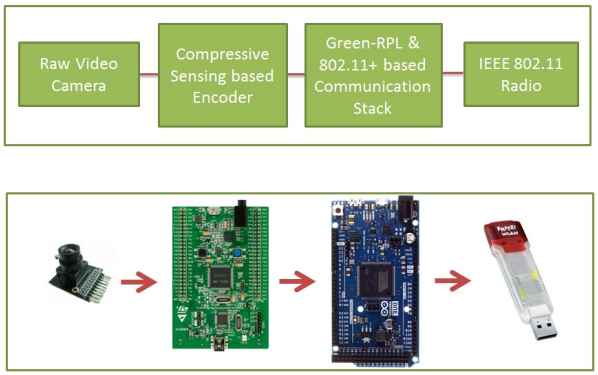

Figure 3.1: Green camera node

Duck Current: 12 mV/s at 6'C

3.2 Microcontroller board for encoding

Once the raw video is provided by the camera module, the raw video is now required to be encoded for compression. At the heart of the camera node is the 15 Dollars ARM 32-bit CortexTM- M4-based STM32 F407 MCU. The STM32F407 has a maximum CPU clock of 168MHz, with current consumption at about 180 microA per MHz. The MCU is rated at 210 DMIPS, with up to 1Mbyte of Flash. It has USB OTG HS-FS, 3ADCs, Ethernet MAC and camera interfaces.

The camera node is to be a low power, low cost device, capable of handling video frames of CIF resolution 352 x 288. Therefore higher end processor architectures such as the Cortex A and Cortex R series simply don't fall in 10 dollar range, and with GHz clocks and OS overheads, they don't fall in the low power range when the subject is a node. On the other hand, low to midrange microcontrollers such as ATMEL's ATMEGA8, Microchip's PIC16F and PIC18F series, with maximum clocks of up to 40MHz and 8-bit, 16-bit architectures with no DSP capability, simply can't handle the 30fps frame rates and fairly large matrix multiplications in real time. Therefore we need a high end microcontroller with DSP capabilities i.e. a Digital Signal

Controller (DSC, capable of performing single instruction multiplications). In multimedia processing, inter-frame sparsity needs to be exploited in order to achieve greater compression, therefore neighboring frames (2-3) need to be quickly saved and accessed to and from the memory. When memory operations are large (as in this case) they put a huge burden on the processor core unless the architecture is memory mapped with a dedicated Memory Protection Unit (MPU). Furthermore, our green camera node for IoT involves interfaces such as CMOS camera interface, interface with wiFi module, and a USB port for tasks such as software up-gradation. The Cortex M4 architecture from ARM is undoubtedly the best fit to all these requirements, as it is DSC, memory mapped, multi-stage pipelined and manufacturers such as Texas Instruments, ST Electronics and NXP are manufacturing their own product ranges based on the Cortex

M4 ARM architecture. Another important thing about the cortex M4 based microcontrollers is that support material such as development boards, extension boards and a range of open source supporting middleware (such as RTOSs) are easily available at very nominal prices.

Our proposed CBS-Video Coding Technique is implemented in this controller which encodes the raw video. Once the video is encoded its is required to be passed on to the main controller running OS. For this reason a Discovery based board is used which provides interface to camera and communication controller module running OS.

3.3 Microcontroller board running OS

The encoded video is received by the ATMEL microcontroller on Aruduino Due board over SPI interface. The arduino module is operated by the Contiki operating system. The Contiki-OS provides the functionality of a complete light weight communication stack. This communication stack has been proposed for IoT. Since, the current IoT protocols are not designed to support multimedia communication. Therefore, we proposed new protocol designs and enhancements in order to support multimedia communication. These protocols were implemented and tested in Contiki-OS in previous deliverables. In this deliverable, both of the proposed protocols routing layer protocol, MAC layer protocol were implemented in the green camera node. This enabled multimedia communication over IoM. This Arduino board requires a radio module to transmit that compressed video received from the encoder module for this we used Wi-Fi dongle to provide wireless link for transportation of multimedia traffic.

3.4 Wi-Fi module

The Wi-Fi dongle we used is the Fritz! USB Wi-Fi dongle, whose radio drivers are open source. This enabled us to modify its operation. We modified these drivers to enhance Wi-Fi communication performance. The encoded video is transmitted using this dongle device over the wireless channel. This transmitted multimedia traffic is received by a similar Wi-Fi dongle which then saves this data, to be later used for decoding the video.

Bibliography

[1] I. E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley, 2nd ed edition, 2010.

[2] Linux news. *http://linuxfr.org/news/h-265-est-finalise*. 2013.

[3] Gary J Sullivan and Stephen J Estrop. *Methods and systems for start code emulation prevention and data stuffing*. US Patent 7,505,485, 2009.

[4] Ntu university. *http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/jpeg/jpeg/encoder.2013*. 2013.

[5] Motion-Prediction. *http://wiki.multimedia.cx/index.php?title=MotionPrediction*. 2007.

[6] Z. Liu, H. Vicky Zhao, and A. Y. Elezzabi. "BLOCK-BASED ADAPTIVE COMPRESSED SENSING FOR VIDEO". In *Proceedings of 2010 IEEE 17th International Conference on Image Processing*, September 2010.

[7] Z. Liu, A. Y. Elezzabi, and V. Zhao. "Maximum Frame Rate Video Acquisition Using Adaptive Compressed Sensing". *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 21(11), November 2011.