Flexi-WVSNP-DASH: A Wireless Video Sensor Network Platform for the Internet of Things

by

Adolph Seema

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2016 by the
Graduate Supervisory Committee:

Martin Reisslein, Chair
Jennifer Kitchen
Patrick Seeling
Yanchao Zhang

ARIZONA STATE UNIVERSITY

May 2017

ABSTRACT

Video capture, storage, and distribution in wireless video sensor networks (WVSNs) critically depends on the resources of the nodes forming the sensor networks. In the era of big data, Internet of Things (IoT), and distributed demand and solutions, there is a need for multi-dimensional data to be part of the Sensor Network data that is easily accessible and consumable by humanity as well as machinery. Images and video are expected to become as ubiquitous as is the scalar data in traditional sensor networks. The inception of video-streaming over the Internet, heralded a relentless research for effective ways of distributing video in a scalable and cost effective way. There has been novel implementation attempts across several network layers. Due to the inherent complications of backward compatibility and need for standardization across network layers, there has been a refocused attention to address most of the video distribution over the application layer. As a result, a few video streaming solutions over the Hypertext Transfer Protocol (HTTP) have been proposed. Most notable are Apples HTTP Live Streaming (HLS) and the Motion Picture Experts Groups Dynamic Adaptive Streaming over HTTP (MPEG-DASH). These frameworks, do not address the typical and future WVSN use cases. A highly flexible Wireless Video Sensor Network Platform and compatible DASH (WVSNP-DASH) are introduced. The platform's goal is to usher video as a data element that can be integrated into traditional and non-Internet networks. A low cost, scalable node is built from the ground up to be fully compatible with the Internet of Things Machine to Machine (M2M) concept, as well as the ability to be easily re-targeted to new applications in a short time. Flexi-WVSNP design includes a multi-radio node, a middle-ware for sensor operation and communication, a cross platform client facing data retriever/-player framework, scalable security as well as a cohesive but decoupled hardware and software design.

To Mirembe, Tidimalo, both Sisimogangs and Kgosiemang.

ACKNOWLEDGEMENTS

I am especially thankful to my advisor, Professor Martin Reisslein, for his patience, guidance and most of all trusting me enough with the independence to pursue my passion in this research area. Thank you members of my Thesis Committee for your time, direct and indirect support and your advice during my oral examinations. Just by knowing who has to approve this, makes me focused on improving the quality of my work.

I am grateful to the four women in my life, my wife for her unconditional support and professional advice, my mother, my grandmother and my daughter for living with a distracted dad for the good part of her early life. To my daughter and son, thanks for helping me straighten out my priorities.

This dissertation is also a reflection of a lot of work and curiosity by many participants who tested, questioned and used the discoveries and theories resulting from this project. I am especially proud and thankful to the more than sixty five Electrical, Energy and Computer Engineering (EECE) and Computer Science and Engineering (CSE) students who used this work to complete their Capstone Senior Design Projects, Fulton Undergraduate Research Initiative (FURI) projects and Master of Science theses. Your work and feedback has improved this platform from a concept to a real world prototype.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.0.1 WVSN Motivation

The world of sensor network research has grown tremendously in the past few years [229, 113, 168, 202], which makes it difficult to treat all sensor nodes along traditional generalized challenges [93]. Wireless Video Sensor Nodes/Platforms, (WVSNP: pronounced WaveSnap), are quickly emerging as a field of research on their own and the challenges they pose are heavily documented in the literature [220, 94, 13]. It is therefore fitting to review what has been attempted so far and identify a pragmatic way forward. The wireless sensor node literature generally agrees that the main hurdles facing real world wireless sensor networks are: power, size, adaptability, security, communication, computation, synchronization, robustness, and cost. Each of these challenges affects a particular class of wireless sensors more than others. The contradicting demands of wireless sensors are even more pronounced in the world of multimedia sensors. We define multimedia sensor nodes as those platforms that have a significant component of audio, image, and/or video processing and transmission [13]. These can be further divided into real-time WVSNPs (RT-WVSNP) and non-real-time WVSNPs (NRT-WVSNP). A RT-WVSNP is capable of acquiring a live audio, video and/or image, process it with/without compression and wirelessly transmit it to a compatible receiver. The matching receiver should be able to play it back to a human observer intelligibly without showing a significant time or quality difference from the live feed. A NRT-WVSNP should perform identically to an RT-WVSNP except that the time synchronization with the live feed is unnecessary.

The existence of a live feed/acquisition itself is irrelevant as stored content can be used as a source to be transmitted. Both real and non-real-time nodes should enable temporal accuracy for a network to be able to construct and maintain a time order. This is needed to chronologically order acquisition, relay, or playback data. Since it is assumed that video is the most demanding of the above multimedia elements, it will be used as a yardstick throughout this thesis.

Popular applications for WVSNPs are computer vision [30, 52], video tracking [27, 41] and locating [133], video surveillance [124, 239, 90], remote live video and control [118, 74] and assisted living [200, 221]. An ideal WVSNP should be robust and flexible enough to perform well on any of the application groups above. In Chapter 2 we summarize and critique existing video sensors in the literature that are the closest to achieving the requirements of a WVNSP outlined in Section 2.2. All the thirteen (13) platforms are objectively reviewed and their pros and cons detailed as an inspiration for this work. We also define and introduce the fundamental categories under which existing image capable sensors fall. From the categorization we clearly identify the missing parts in the WVSNP research area.

Most of the research and proposed solutions to the challenges of WVSNPs involve multi-tier systems [160, 35, 122], new protocols and enhancements [159], streaming [142], compression techniques [111], distributed algorithms [200], light-weight operating systems, middleware [132, 35], and some resource allocation strategies [88]. Most solutions so far are heavily software biased. Several toolkits [188, 189, 89] have been developed to help in software based solution research and often treat hardware as an afterthought. There are other efforts to study video traffic characteristics in order to enable better video transmission profile [82, 163, 217, 172]. To address this shortcoming, we focus on a specialized segment of the wireless video sensor networks, that is, the hardware architecture of the node/platform itself. In Chapter 3, we try to setup a foundation of what is needed to solve most of the problems we highlighted in Chapter 2. We discuss the blue print of the Flexi-WVSNP architecture, Flexi-WVSNP, and discuss its expected functionality and performance metrics. The platform consist of five major contributions, the flexible hardware, the maintainable and easily adaatable software image, the user interface as defined by the WVSNP-DASH framework [181] and reconfigurability for different WVSN use cases.

The latter happens to be the major impediment in making WVSN accssible for daily user activities and is the major part of the contribution beyond the hardware and the software. Without an easy and accesible user interface, no one would use the hardware nor the software that controls it, no matter hos good those components are. This work introduces a cross platform video retriever and distributor that implements a user interface to a WVSNP-DASH framework. There has been a growing interest in the Internet of Things (IoT) and the presumption that they are finally becoming useful to ordinary consumers. Many hubs and platforms are being introduced to make the household sensors and actuators easily accessible and controllable from smart phones and other consumer devices. What seems to be ignored in most of the platforms is video as part of the IoT nor how to easily make Videos Sensor Networks (VSNs) part of the IoT platforms. The WVSNP-DASH framework is described in more detail in a separate paper. A brief and sufficient summary of the framework to aide the reader in evaluating the WVSNP-DASH player against other players is provided. WVSNP-DASH Player (WDP) is referred to as a player for the purpose of comparing its capabilities and benefits to existing DASH type players from popular DASH frameworks. WDP is capable of much more features crucial to integrating Wireless Video Sensor Networks (WVSNs) to IoT, and the Machine to Machine interface (M2M).

To appreciate the reason video is seen as a major data element in Sensor Networks and the reason VSNs should be easily accessible to consumer devices, the concept of video has to be understood as multi-dimensional data (2D/3D). It is understood that consumer devices for entertainment, residential and industrial use want to be able to view video from a wide range of video sensors within Internet Protocol (IP) networks and from across traditional WVSNs that do no necessarily have IP addresses. Additionally, consumers now want to also see output from infrared sensors, heat maps, Light Emitting Diode (LED) pixel sensor maps, x-rays, and many other wirelessly linked sensor nodes and remote acquisition devices that are becoming ubiquitous and expected by consumers. It is also expected that no specialized software or protocols be needed for each type of data. Consumers just want to request data and view it on their devices and be able to see the same data as they switch around devices and operating systems during their typical day in between home, work and fun.

An effort is under way to address the cross platform use case on consumer devices for video data via the emerging Dynamic Adaptive Streaming over HTTP (DASH) specifications and their commercial and open standards variants. The DASH research activity, as well as work on version five (5) of the Hypertext Markup Language (HTML5) provides an excellent starting point to simplify and improve ease of consumer access to WVSN data. The WDP draws on the current activity in this area to extend these promising technologies into wireless video sensor node/platforms (WVSNP). The WDP implementation exposes the novelty of how a similar and compatible streaming architecture could enhance and improve accessibility of sensor networks' video data and its delivery.

1.0.2 True Streaming vs HTTP Streaming

Downloading a file over HTTP is normally referred to as progressive download or HTTP streaming. This is not streaming at all, but a bulk download of a video file to the viewer's computer. This stores a temporary copy of the video file on the client's local computer. This enables repeated viewing of the file if need be without having to download the file each time. Assuming a video file is encoded at 200kbps. The originating server just uploads data to the viewer's machine with no knowledge of the encoding rate as quickly as it can. Playback can begin as soon as enough of the file is available on the client computer, giving an illusion that the file is being streamed. The user cannot not skip to parts of the file that have not yet been downloaded. If the client is using a 56 kbps dial-up modem, they will have to wait a long time to play the 200 kbps video. The quality will still be great (exactly the same) once you start watching it. If using a 200 kbps broadband line or faster, playback should start almost immediately and the file should download faster than it will play.

What is observed above is called "Chunked transfer encoding". It is a data transfer mechanism in HTTP. This allows HTTP data to be delivered reliably, without knowing in advance of transmission, the size of the entire message body. This is possible only in version 1.1 of HTTP (HTTP/1.1). HTTP splits the data payload of the message into small parts (chunks). Each chunk together with its size are then transmitted one after the other. The client knows it received the last chunk if it received a final chunk of length zero. This enables transmitting dynamically generated content in web pages. Without chunked transfer encoding, the size of data bytes delivered in HTTP responses must follow the colons after the *Content-Length* header field. This allows clients to determine the end of transmission. A typical chunked response looks like this:

Listing 1.1: HTTP Chunked transfer encoding example.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

23
```

```
This is the data in the first chunk


16
here is the second one


7
con dah
E
data ends here
0
```

If a *Transfer-Encoding* field has a *chunked* value, it is either a request sent by a client or the response from the server. The number of bytes of each chunk are in hexadecimal.

On the other side, a true streaming (TS) server is a piece of software which opens a conversation with the client computer. One side transfers the video and the other side is for control messages between the media player and the server. These control messages or commands can be play, pause, stop, and so forth. This expects that network bandwidth between the server and client to at least be the bit rate of the video requested. If not, the video will not be delivered. If the server offers a lower resolution video you can begin another session for that lower bit rate. The advantages of (TS) are that: (*i*) You can play video at any point of the video, or fast forward or rewind. (*ii*) It efficiently uses bandwidth as one uses bandwidth only for part of the video they are actively watching as opposed to HTTP delivery where the whole file gets delivered. (*iii*) The video file is not stored on the viewer's computer. It is discarded by the media player. This feature is liked by video content creators.

Let us assume you are using a 56 kbps dial-up-modem. Assume the server then transmits to you, the client, the same video, but instead, this video is encoded at 36 kbps. This means you can still see the video immediately though it is not high quality resolution. This means that there is some adaptability in the video sent over HTTP.

This is one way to make progressive download adaptive. Scalable video codecs already exist. Can they generate different resolutions of the video for the server to pick from? What if you could still fast forward, live stream, skip and pause over HTTP? DASH tries to bring you the best of both worlds and even more. Other proposals beyond just DASH's intelligent (adaptive) client, try to encode the content itself with the goal of reducing data rates by using more DASH aware frame placement on the server side during compression. This type of content optimization is still valid as DASH encapsulates content encoding options. So, WVSNPs would still benefit from these and similar future research. The same shared benefits can be derived from works that focus on buffer management algorithms as well as segment scheduling fairness and adaptive switching algorithms on the client side [244, 107].

A study of the various DASH media presentations, their video segmentation formats, their delivery frameworks, and their required hardware/software (HW/SW) integration, revealed how power efficiency, scale ability, cost reduction, and improved ease of integration can be derived even for WVSNP's that don't quite fall into the latest "Internet of Things" classification. The WDP implementation uncovers a direct mapping of DASH into what WVSNPs are already expected to meet. That is, service modes (Live, On-Demand, Time-Shift Viewing), Quality of Service (adaptive bit rate switching, scalable video, receiver controlled transmission) and efficient flexible configurations like duty cycled delivery, video indexing capability and random access to video.

The next chapters will show the core contribution of WVSNP-DASH via the WDP consumer interface, implementation architecture, test-bed setup for evaluation versus other popular cross platform HTML5 players, results analysis, recommendations and future work discussion to conclude.

Chapter 2

A CRITIQUE OF EXISTING PLATFORMS

2.1 Introduction

Wireless sensor networks capable of capturing video at distributed video sensor nodes and transmitting the video via multiple wireless hops to sink nodes have received significant interest in recent years [13, 94, 142, 220]. Wireless video sensor networks have been explored for a wide range of applications, including computer vision [30, 52], video tracking [27, 41] and locating [133], video surveillance [124, 239, 90], remote live video and control [118, 74], and assisted living [200, 221]. Many aspects of wireless video sensor networks have been extensively researched, including multi-tier network structures, e.g., [160, 35, 122], multisensor image fusion, image and video compression techniques, wireless communication protocols, e.g., [159], distributed algorithms, e.g., [200], light-weight operating systems and middleware, e.g., [132, 62, 35], and resource allocation strategies [88, 162]. Generally, a large portion of the research has focused on software-based mechanisms. Several toolkits, e.g., [34, 89, 188, 189], have been developed to facilitate software based video sensor network research. Other techniques in the research community involve flexible transmission, including cognitive radio transmission[12, 112].

In this section, we focus on the wireless video sensor nodes forming the sensor network. We comprehensively survey the existing wireless video sensor node platforms (WVSNPs) considering the hardware and software components required for implementing the wireless video sensor node functionalities. All functional aspects of a wireless video sensor network ranging from the video capture and compression to the wireless transmission and forwarding to the sink node depend critically on the hardware and software capabilities of the sensor node platforms. Moreover, the sensor node platform designs govern to a large extent sensor network performance parameters, such as power consumption (which governs network lifetime), sensor size, adaptability, data security, robustness, and cost [93]. Also, computation capabilities, which are important for video compression, and wireless communication capabilities, which are important for the wireless transport from the source node over possibly multiple intermediate nodes to the sink node, are determined by the node platforms. An in-depth understanding of the state-of-the-art in WVSNPs is therefore important for essentially all aspects of wireless video sensor network research and operation. To the best of our knowledge, there is no prior work of the field of wireless video sensor node platforms. Closest related to our survey are the general review articles on the components of general wireless (data) sensor networks, e.g., [14, 93, 220, 237], which do not consider video sensing or transmission, and the general works on multimedia sensor networks, e.g., [13, 142], which include only very brief overviews of sensor platforms.

Toward providing communications and networking generalists with an in-depth understanding of wireless video sensor node platforms (WVSNPs) and their implications for network design and operation, we first briefly review the requirements for WVSNPs in Section 2.2. In Section 2.2 we also define ideal requirements for the power consumption, throughput of video frames, and cost of WVSNPs suitable for practical networks. Our exhaustive literature review revealed that currently no existing platform meets the ideal practical requirements. We therefore relax our requirements in Section 2.3 and according to the relaxed requirements select about a dozen platforms for detailed review. We introduce a classification structure of WVSNPs consisting of the categories: general purpose architectures, heavily coupled architectures, and externally dependent architectures. In Sections 2.4 through 2.6 we critique the existing WVSNPs following our classification structure. For each critiqued WVSNP we examine overall structure and resulting advantages and disadvantages for the wireless video sensor node functionalities, including video capture and encoding as well as wireless video transmission. In Section 2.7 we summarize the insights gained from our detailed survey, including the key shortcomings that cause existing WVSNPs to fail the ideal practical requirements. Building on these insights, we propose in Section 3.1 a novel Flexi-WVSNP design that addresses the shortcomings of existing WVSNPs through a number of innovative architectural features, including a cohesive integration of hardware and software and a dual-radio.

2.2 Requirements for Wireless Video Sensor Node Platforms

In this section we review the sensor node requirements and define our ideal, yet reasonable practical requirements for a wireless video sensor node platform (WVSNP). From detailed reviews of the requirements for WVSNPs, e.g., [13, 94, 179], we identified three core requirements, namely power consumption, throughput, and cost and summarize these core requirements as follows. The power requirements are influenced by a wide range of design choices, including power source type, component selection, power management hardware and software, and importantly sensor node and network management algorithms, such as implemented by a real time operating system (RTOS) [62] or sensor network duty cycling schedules [16].

We define the desirable power consumption of an entire sensor node platform to be less than 100 mW when idle (also referred to in the literature as standby or deep sleep mode). We also require that a WVSNP has an instantaneous power consumption of less than 500 mW. These requirements are based on rule of thumb calculations that a node running on two AA batteries lasts a year if it consumes on average less than

0.2 mA [94, 33]. Compare this to a cell phone which typically consumes more than 4 mA.

To satisfy these stringent power consumption requirements, a sensor node has to provide most, if not all, of the following power modes. (For general background on microprocessor design and their power-efficient design and operation we refer to [39, 154, 165, 187, 236].)

On: At this fully functional state the main processor, e.g., microcontroller unit (MCU) chip/integrated circuit (IC), uses most power as all of its parts are in use. Power can only be conserved by dynamically changing the core frequency or operating voltage.

Ready: This mode saves power by shutting down a chip's core clock when not needed (also referred to as clock gating). The chip's core clock resumes when an interrupt is issued, for instance to process some input/output (I/O).

Doze: As in Ready mode, the chip's core clock is gated. Additionally, the clocks for pre-configured peripherals can be switched off. An interrupt can quickly reactivate the chip's normal functions.

Sleep: This mode switches off all clocks and reduces supply voltage to a minimum. External memory runs at a self-refreshing low-power state. Data is preserved during Sleep and hence there is no need to recover it on wake-up.

Idle: Unlike Sleep mode, data in the chip's registers is lost in Idle mode. The chip's core is turned off. An interrupt resumes the chip's normal functionality.

Hibernate: The entire chip's power supply is shut down and the chip loses all internal data. This requires a full initialize (cold-boot) resumption,

The node design and control need to generally trade off the power savings achieved by duty cycling through these power modes with the frequency of checking the radio channels and the cost of waking up for channel checking [34, 16, 198, 153].

The throughput of a node is generally defined as the number of video frames per second received by the sink node from the source node [223]. More specifically, a frame cycle consists typically of five stages:

1. The source sensor node loads a raw frame from the attached imager into the node's memory;
2. The source node compresses the raw frame and loads the result to its output buffer;
3. The source node's radio transmits the compressed frame from the buffer to the sink node;
4. The sink node uncompresses the received frame; and
5. The sink node displays/stores the raw uncompressed frame.

We define the required throughput as a frame rate of at least fifteen common inter-frame format (CIF, 352×288 pixels) frames per second (fps). We choose the 15 fps as it is widely documented as an acceptable frame rate for human perception of natural motion [120, 17, 32, 101, 123].

The throughput is primarily limited by the MCU chosen as the master component of the sensor node. The choice of an MCU has implications for peripheral components and bit-width as well as the availability of power modes, multimedia processing, and memory interfaces. 32-bit MCUs are typically significantly faster and computationally more capable than the 16- or 8-bit MCUs for video; moreover, a 32-bit MCU consumes typically two orders of magnitude less power than an 8-bit MCUs for the same work load [13, 89, 63]. Therefore, we require the master processing unit to be 32-bit capable. Other main throughput limiting components are typically the radio communication and the image acquisition.

The cost of a node depends primarily on the technology chosen for the architecture, the type and maintenance cost of the selected components, the intellectual accessibility of the SW/HW components, and the scalability and upgradeability of the architecture. A low-cost platform generally has very few, if any, proprietary components. It should be possible to substitute components based on competitive pricing in a modular manner. Such substitutions require in-depth knowledge of the functions and limitations of each HW/SW component which is rarely possible for proprietary platforms. Therefore, standardized HW/SW components and well architected open source software and open hardware cores that benefit from economies of scale are important for meeting the low cost objective.

We require a fully functional sensor node platform that meets the above power and throughput requirements to cost less than \$100 USD with the cost expected to decrease as standardized components get cheaper. We choose this cost requirement, as we envision a sensor node as a semi-disposable component that is widely deployed.

A sensor node can be designed to incorporate low-level input, that is, physical-layer and middleware-level input from the environment. For instance, the node can use input from other physical sensors (e.g., motion sensors) to decide when to capture a frame. We refer to a node with this capability as a *smart mote*. Smart motes can further reduce power consumption and improve effective throughput beyond the manufacturer's stated hardware capabilities for a specific application.

2.3 Classification of Wireless Video Sensor Node Platforms

In the preceding section, we reviewed the main requirements for wireless video sensor node platforms (WVSNPs) and defined ideal performance requirements. We comprehensively reviewed the existing literature and found that none of the existing nodes meets the ideal requirements. In an effort to conduct an insightful survey that uncovers the underlying structural shortcomings that cause the existing nodes to fail our ideal (yet practically reasonable) requirements, we relaxed our requirements. We selected WVSNPs for our survey that meet at least three of the following rules based on the test scenarios considered in the literature about the sensor node.

1. The node has most of the power modes defined in Section 2.2 and its average power consumption is less than 2 W;
2. The node's throughput is at least two CIF fps;
3. The estimated cost of the node using current off-the-shelf technology and accounting for economies of scale projection is at most $50 USD;
4. The sensor node platform is capable of wireless transmission; and
5. The architecture implementation and major HW/SW building blocks are open to researchers without proprietary legal restrictions to educational experimentation.

Many platforms, e.g., [19, 29, 36, 38, 44, 57, 64, 72, 86, 87, 96, 127, 131, 136, 152, 156, 175, 184, 204, 225, 231, 238, 93, 124, 30, 200, 160, 27, 220, 118, 239, 133, 74, 52, 111], do not meet these relaxed requirements. For example, the platform [44] employs advanced techniques for detecting changes in brightness to achieve ultra-low-power wireless image transmission. However, the platform employs a coarse 90×90 pixel imager as well as non-standard compression that is customized for the node and test application.

Any design approach based on field programmable gate arrays (FPGA) [127] likely fails the cost rule as FPGAs have very limited off-the-shelf economies of scale; further, FPGAs have low computation performance relative to power consumption and exploit limited standardized intellectual property (IP) [1, 66]. The ScoutNode [184] embraces modularity and power mode flexibility. However, it is focused on military proprietary communication protocols, has a high cost, and a high power consumption.

From our exhaustive literature review, we found that only the platforms noted in Table 2.1 satisfy our selection criteria. As summarized in Table 2.1, we classify the selected platforms into three main categories, namely *General Purpose Architectures*, *Heavily Coupled Architectures*, and *Externally Dependent Architectures*.

Summary of classification categories for existing

In each of the Sections 2.4 through 2.6 we first give an overview of a category and then individually critique each of the existing platforms in the category.

Before delving into the different node platform categories, we note a common characteristic of most existing nodes, namely the use of a IEEE 802.15.4 radio, in particular the Chipcon/Texas Instruments CC2420 2.4 GHz IEEE 802.15.4 RF transceiver. (Following the common Zigbee terminology, we use the term “radio” to refer to the physical and medium access control layers.) Most nodes implement only the PHY and MAC layers of IEEE 802.15.4 and use custom protocols or Zigbee-compliant protocols for the higher protocol layers. Nevertheless, all nodes using the CC2420 or other IEEE 802.15.4 radios are “Zigbee-ready”, meaning that they can be easily made Zigbee compliant by a software update of the relevant Zigbee protocol stack. The IEEE 802.15.4 radio is readily availability, has low cost, is easy to implement, and facilitates benchmarking among nodes using the same radio. However, the IEEE 802.15.4 radio has shortcomings that can significantly weaken a node platform if the node fails to leverage the IEEE 802.15.4 advantages and does not complement its weaknesses. For instance, IEEE 802.15.4 radio transmission is limited to a 250 kbps data rate, which makes real-time video transmitting almost impossible, unless efficient supplemental architectural techniques are employed. We will comment on the specific implications of the IEEE 802.15.4 radio on each sensor node’s architecture in the individual critiques.

2.4 General Purpose Architectures

General purpose platforms are designed similarly to a personal computer (PC), following a "catch-all" functionality approach. They attempt to cover all possible peripherals and printed circuit board (PCB) modules that an application may need. This strategy results in designs that include as many building blocks as prescribed cost limits permit. General purpose architectures are useful for fast prototyping of applications. Generally, they consist of a node (MCU) PCB to which many MCU peripherals and PCB modules are attached that highlight the capabilities of the MCU.

General purpose platforms typically suffer from high power consumption and dollar cost, as well as underutilized functional blocks despite not meeting basic WVSNP requirements. Furthermore, general purpose platforms often overuse standard interfaces, such as universal serial bus (USB), personal computer memory card international (PCMCIA), universal asynchronous receiver/transmitter (UART), and general purpose input/output (GPIO) interfaces. The disadvantage of having many I/O pins and peripherals is that the I/O subsystem can consume a disproportionately large amount of power. Powering down GPIO interfaces is not always an option as in most cases the wakeup cost negates the advantages gained from periodic shutdowns.

In Table 2.3 we summarize and contrast the considered general purpose architectures. In the first row of the table we rate the platform's flexibility from 0 to 10 (0 being functionally and architecturally inflexible and 10 being highly robust, adaptable, and extensible).

2.4.1 Stanford's MeshEye [90] and WiSN Mote [63]

Overview

MeshEye is a smart camera mote architecture designed for in-node processing. It selects among several available imagers based on changes in the environment. The architecture follows the philosophy that, as the level of intelligence (a priori decision making before acquiring and compressing an image) increases, bandwidth requirements on the underlying data transmission network decrease proportionally. The host processor is a 32-bit 55 MHz Atmel AT91SAM7S family MCU with an ARM7TDMI ARM Thumb RISC core. The MCU internally has up to 64 KB SRAM and 256 KB of flash memory as well as a built-in power management controller. The mote is designed to host up to eight KiloPixel imagers (Agilent Technologies ADNS-3060 high-performance optical mouse). The ADNS-3060 is a 30x30 pixel, 6-bit grayscale camera also referred to as image sensor or optical mouse sensor (due to its use in a computer mouse). The sensor node also has one programmable VGA camera module (Agilent Technologies ADCM-2700 landscape VGA CMOS module, 640x480 pixel, grayscale or 24-bit color). The dynamic use of a variety of mouse sensors and a VGA camera makes this mote "smart". The mote has a serial peripheral interface (SPI) bus attached multimedia card (MMC)/secure digital (SD) flash memory card for temporary frame buffering or archiving of images. As illustrated in the top right part of Figure 2.1, a single SPI interface connects an IEEE 802.15.4 radio, up to eight KiloPixel imagers, and a flash card (on the left) to the MCU.

As shown in the bottom right part of Figure 2.1, the VGA camera module is controlled via a two wire interface (TWI also denoted as I^2C). The VGA camera module captures and encodes the video into CCIR (ITU-R BT.601). The encoded video data is read from the camera through general-purpose I/O pins.

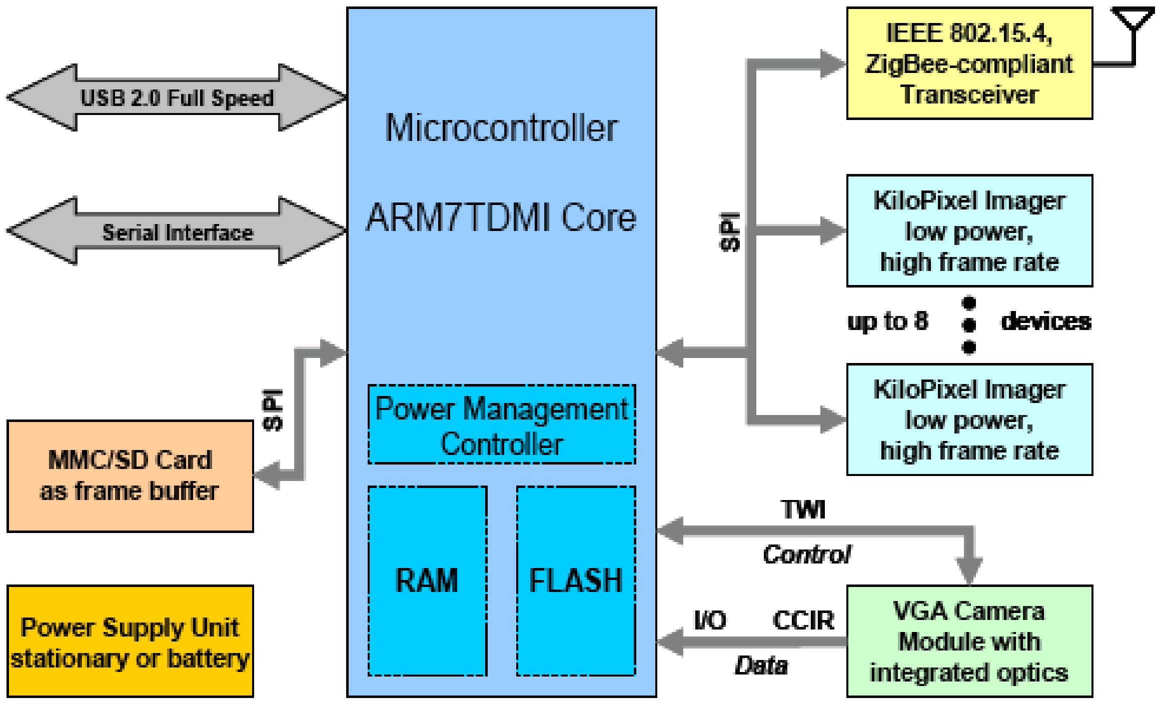

USB 2.0 Full Speed
Serial Interface
Microcontroller
ARM7TDMI Core
SPI
MMC/SD Card as frame buffer
Power Supply Unit stationary or battery
Power Management Controller
RAM
FLASH
SPI
IEEE 802.15.4, ZigBee-compliant Transceiver
KiloPixel Imager low power, high frame rate
up to 8
devices
KiloPixel Imager low power, high frame rate
TWI
Control
I/O
CCIR
Data
VGA Camera Module with integrated optics

Figure 2.1: Block diagram of Stanford's MeshEye architecture [90].

The Stanford WiSN node, illustrated in Figure 2.2, has many similarities with MeshEye with more focus on implementing networked image sensing where multiple image sensors observe the same object from different view points. This enables collaborative data processing techniques and applications. For its higher resolution imaging, WiSN uses two ADCM-1670 CIF (352x288 pixel) CMOS imagers, instead of MeshEye's one VGA camera. As shown in Figure 2.2, the node also adds a flexible expansion interface that connects to a variety of sensors, though some are not necessarily critical for a video sensor requirement. The WiSN also introduces a Linear Technology LTC3400 synchronous boost converter for regulating voltage levels (1.8 V and 3.0 to 3.6 V). The converter has a 19 μA quiescent current draw and can supply up to about 3 mA.

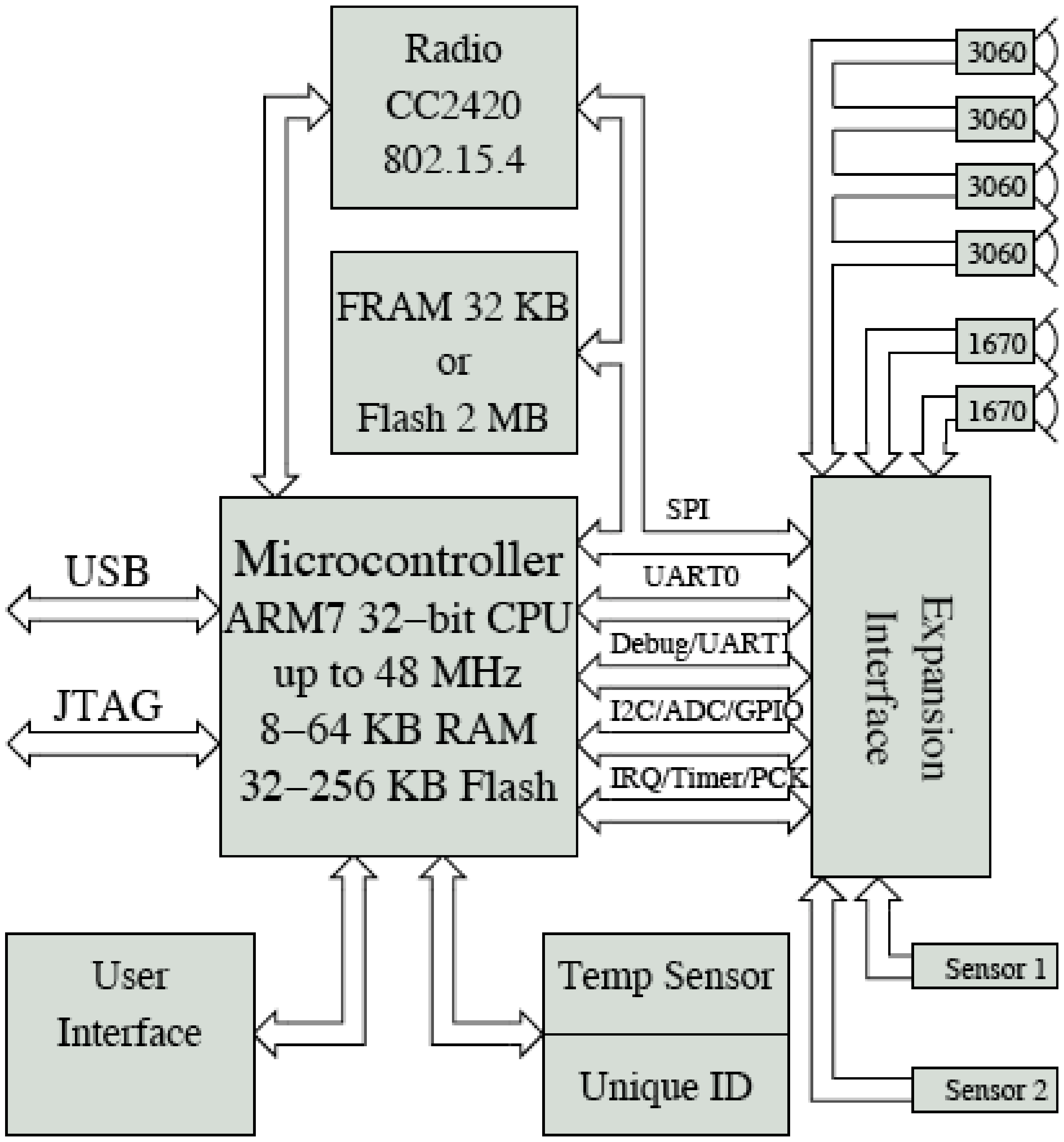


Figure 2.2: System diagram of the Stanford WiSN mote board [63].

Advantages

Processing the video stream locally at the camera is advantageous as it can reduce bandwidth requirements and hence save power or improve frame rate as only necessary information is processed or transmitted. The use of more than one image sensor seems suited for distributed vision-enabled applications. The smaller imagers are used to detect some events, which removes the need to unnecessarily trigger the VGA imager for image acquisition. This saves power as the KiloPixel imagers do most of the vision monitoring whereas the slower and more power-hungry VGA imager is idle most of the time.

The external MMC/SD Flash card/Flash memory gives the motes a persistent, scalable, and non-volatile memory. The ability to store files locally is helpful for debugging, logging, and data sharing.

The platforms have an option of either mains power supply or battery based supply. This makes the motes flexible for both mobile and fixed applications. The MCUs' built-in power management hardware is an efficient way of putting the MCU and its peripherals into different power-saving modes instead of depending on software managed algorithms. A programmable phase locked loop (PLL) in the MCUs allows for dynamically setting the core's clock rate to lower rates when less processing is required, which saves power.

Using a single SPI interface for several modules is an efficient use of the MCU interfaces and conserves I/O pin use. The choice of directly reading CCIR encoded video in MeshEye reduces component count, power, and cost.

WiSN's use of the expansion interface simplifies design and supports other traditional sensors. The interface also enables it to use two CIF cameras which are more useful in collaborative/stereoscopic imaging compared to having only one VGA imager. Additionally, the expansion port exposes timer inputs/outputs, and programmable clock outputs. Further, the interrupt request (IRQ) lines and standard GPIO pins are multiplexed using the remaining pins, making this platform easily expandable. Some of the GPIO pins have enough current drive (16 mA) to power attached sensors. This reduces the need to route many power lines on the board. The choice of the AT91SAM7S MCU allows an easy upgrade path as the AT91SAM7 MCU family has the same in-chip peripheral set, except for the amount of RAM and Flash memory.

Another WiSN advantage is that its LTC3400 linear regulator, which operates at low I/O voltages, protects the battery by presenting the entire circuitry as a single current sink. It also helps reduce the sleep current draw. The LTC3400 can start up and operate from a single cell and can achieve more than 90 % efficiency over a 30 to 110 mA current draw range.

Disadvantages

MeshEye's capture-and-save frame rate of 3 fps is quite low. The CC2420 radio module, which is limited to 250 kbps, is the only transmission module. This requires a very high video compression ratio to be able to transmit video and limits real-time video streaming.

KiloPixel imagers are not necessarily the least energy consuming and cheapest event detectors. Events within the field of view of the VGA imager can, for instance, be sensed with infrared (IR) or ultrasound sensors, which are cheaper and consume less energy than the KiloPixel imagers.

WiSN's video capture is limited to the CIF resolution. In an attempt to support both the mouse (30x30 pixel) sensor and the CIF sensor the designers opted for a serial interface connection to the MCU. This serial connection is robust, but limits the data rate and hence the frame rate of the video.

External memory access via the serial peripheral interface (SPI) bus, due to its serial nature and its master/slave coordination, is significantly slower than on-chip memory or parallel external memory. The Ferroelectric RAM (FRAM) is currently limited to 32 KB. The off-chip Flash memory is not a direct substitute for RAM as it offers limited write/erase cycles and has slow write speeds and wait states when writing. If flash memory is used as a frame buffer, it can limit the node's lifetime depending on the frequency of data writes. For example, a 2 MB flash device designed for 100,000 write/erase cycles will last only 230 days if a 100 KB frame is written to it every 10 seconds.

2.4.2 Portland State's Panoptes [71]

Overview

The Panoptes video sensor captures, compresses, and transmits video at low-power levels below 5 W [71]. The tested 5 W consumption does not meet our 2 W power threshold, but the node meets most of our five criteria in Section 2.3. The sensor node can be fine-tuned to meet the 2 W for some applications. Panoptes uses a personal digital assistant (PDA) platform called Bitsy. The platform runs Linux kernel 2.4.19 on a 206 MHz Intel StrongARM MCU and 64 MB of memory. A Logitech 3000 webcam is used to capture high-quality video and attaches to the PCB via a USB 1.0 interface. Panoptes uses spatial compression (not temporal), distributed filtering, buffering, and adaptive priorities in processing the video stream. A stand-alone third party 802.11 card attached via PCMCIA is used for wireless transmission.

Advantages

Panoptes is one of the few platforms with the architectural components capable of real-time video capture. It uses special multimedia instructions that are custom to this MCU for most of the video compression. These special MCU primitives enable high frame rates as they speed up multimedia processing, such as JPEG and differential JPEG compression. The Panoptes board supports network wake-up as well as optimized ”wake-up-from-suspend” energy saving mechanisms. In addition to compression, Panoptes uses priority mapping mechanisms, including raw video filtering, buffering, and adaptation to locally pre-process the video stream which can be strategically used to conserve power.

The third party stand-alone 802.11 module makes the platform flexible as the module can be easily exchanged for more power efficient and faster modules as they become available or affordable. The use of Python scripting to connect software module objects is good for supporting a modularized system that is easily adaptable as each object can be associated with its exchangeable hardware component.

Disadvantages

The drawback for Panoptes is that it requires several watts of power, which is relatively high, compared to the similar Stargate platform, see Section2.6.2.

Similar to many other platforms, the StrongARM does not have a floating point unit. Connecting the board to the camera via a 1.0 USB interface creates a data bandwidth bottleneck, especially for 352x288 (common intermediate format, CIF) and 640x480 (video graphics array, VGA) pixel frame sizes, and increases power consumption. This is because the image data from the camera coming in over the USB needs to be decompressed from a camera-specific compression format to generic raw image data in the kernel before being sent to the host's user space and then recompressed with JPEG.

The use of polling to check whether a frame is ready is an inefficient way of acquiring video. Although using specialized built-in MCU multimedia primitives to speed frame processing is helpful, reliance on MCU specific features limits the portability of the code and complicates platform upgrades.

The 802.11 networking on Panoptes consumes about a third of the total platform power [71] and therefore needs to be optimized. The Python scripting employed for adaptability suffers from the common drawbacks of scripting engines, namely large memory space requirements and execution inefficiencies. Also, the Python interconnects result in a 5 % frame overhead [71].

2.4.3 Yale's XYZ plus OV7649 and ALOHA modules [205, 49, 134, 50]

Overview

The XYZ is a motion-enabled and power-aware sensor platform targeting distributed sensor network applications. As illustrated in Figure 2.3, the platform consists of several subsystems, including subsystems for sensing (light, temperature, and accelerometer), communication (TI CC2420 Zigbee radio), mobility (geared motor), power (voltage regulator, power tracker, supervisor, and three AA 1.2 V Ni-MH rechargeable battery pack) and a camera. The capacities of the batteries range from 1200 to 2000 mAh.

The XYZ node is designed around the 57.6 MHz 32-bit OKI Semiconductor ML67Q500x ARM THUMB (ARM7TDMI MCU core). The MCU has an internal 256 KB of Flash, 32 KB of RAM, and 4 KB of boot ROM as well as external SRAM. The Omnivision off-the-shelf OV7649 camera module and the 32x32 pixel event-based

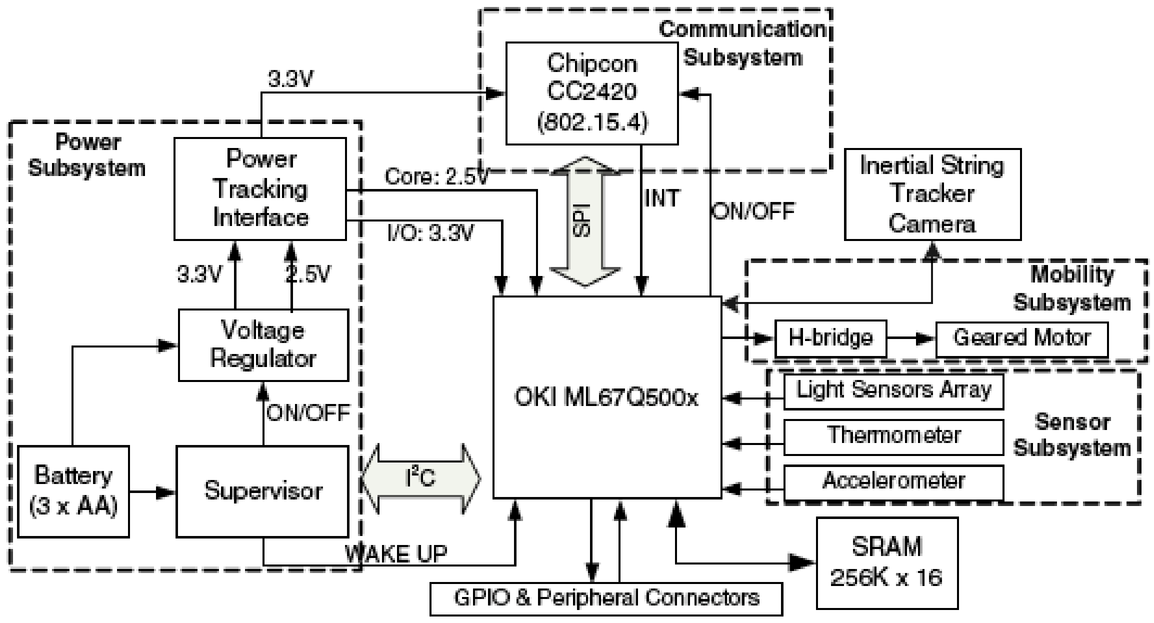


Figure 2.3: The XYZ node architecture [134].

ALOHA CMOS imager have been connected to the XYZ node in separate research efforts [49, 204]. The OV7649 can capture VGA (640x480) and quarter VGA (QVGA, 320x240) images. The image data is transferred from the camera to the on-board SRAM with an 8-bit parallel port using direct memory access (DMA), which does not involve the MCU.

Advantages

The MCU provides numerous peripherals which can be turned on and off as required by the application. The on and off switching is accomplished through software enabling/disabling of clock lines to MCU peripherals. The node is therefore capable of a myriad of power management algorithms. The node provides halt and standby power saving sleep modes in addition to the internal software controlled clock divider that can halve a range of MCU speeds from 57.6 MHz down to a minimum of 1.8 MHz. During standby mode the oscillation of the MCU clock is completely stopped while the MCU still receives some power. The halt mode, on the other hand, does not stop clock oscillation, but blocks the clock from the CPU bus and several MCU peripherals.

The custom supervisor circuit supports a long-term deep sleep mode that puts the entire node into an ultra-low power mode (consumes around 30 μA) by using a real-time clock (RTC) with two interrupts. This setup adds to power management options as transitioning the node into a deep-sleep mode can be done through software control by disabling its main power supply regulator. The RTC can be scheduled to wake the node from every 1 minute up to once every 200 years.

Disadvantages

The XYZ uses the CC2420 radio with its limited transmission rate. The node implements the Zigbee protocol stack on the host MCU, which increases power consumption. Operating the OS and the Zigbee protocol stack on the host MCU at the maximum clock frequency is estimated to require 20 mA [205, 49, 134]. An independent stand-alone radio module with its own in-built protocol stack would relieve the MCU from the network management tasks and improve power savings management. Another challenge for power management is that the MCU I/O subsystem consumes between 11 and 14 mA (i.e., 35.75 to 45.5 mW) due to the high number of I/O pins and peripherals.

The node uses the SOS RTOS, which is an open-source operating system with a relatively small user base and therefore has only a small pool of available re-usable software modules.

Using the OV7649, the XYZ achieves a frame capture-and-save rate of 4.1 QVGA fps. Additionally, only 1.7 16-bit color frames, 3.4 8-bit color, or 27.3 1-bit (black and white) QVGA frames can be stored in the off-chip SRAM. The number of frames

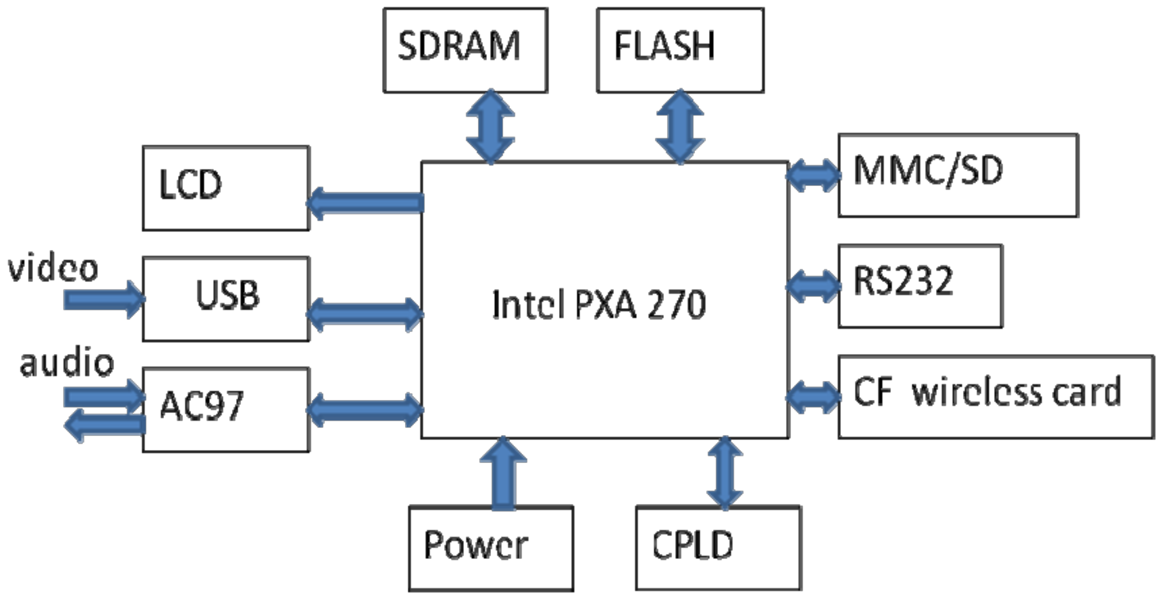


Figure 2.4: NIT-Hohai node hardware architecture [70].

that can be stored increases 4.6 times if a platform optimized 256x64 resolution is used [205, 204]. These limited frame storage capacities can potentially reduce frame rates as application processing may require holding frames in memory, blocking the next frames.

2.4.4 The NIT-Hohai Node [70]

Overview

This sensor node, designed jointly by Nanchang Institute of Technology (NIT) and Hohai University, is centered around the Intel 500 MHz 32-bit PXA270 RISC core SoC, as illustrated in Figure 2.4, and runs a modified Linux 2.4.19 core. Multithreading is used to multitask custom application-level streaming protocols that are layered on top of TCP/IP. The node uses IEEE 802.11 for wireless streaming, with a throughput of 10 to 15 QCIF fps. The node has external SDRAM and FLASH storage as well as a Liquid Crystal Display (LCD).

Advantages

The PXA27x family of processors, which is also used in IMote2 [147], has a rich set of peripheral interfaces and I/O ports, see Figure 2.4. The standardized ports permit use of a wide range of peripheral I/O modules, facilitating the selection of low-cost modules. The architecture is a simple plug-and-play attachment to the core SoC via standard bus protocols and has the benefits of Linux. The design uses run-time loadable module drivers to make the system flexible and scalable. The node uses an optimized H.263 video compression library and is able to transmit in real time.

Disadvantages

The board uses a PCMCIA compatible Compact Flash (CF) based 2.4GHz WiFi card which functions in stand-alone mode, but lacks options for independent direct power management through applications running on the attached PXA270 SoC. Significant design efforts went into the touch-capable 16-bit color 640 x 480 LTM04C380 LCD and related Graphical User Interface (GUI) components, which are not a requirement for a WVSNP. Building on the basic Linux drivers, the design is almost exclusively focused on software functionalities and lacks cohesive HW/SW optimization. All major processing, such as frame capturing, compressing, and networking management, is performed by the SoC, which limits opportunities for power saving through duty cycling. Overall, the node suffers from the disadvantages of general purpose architectures in that it is a rather general design (similar in philosophy to a personal computer) and lacks the mechanisms to achieve the low power consumption and cost required for a WVSNP.

2.5 Heavily Coupled Architectures

2.5.1 Overview

While general purpose platforms are designed for a wide range of applications, heavily coupled platforms are designed for a specific application and are typically over-customized and lack flexibility. The advantage of these highly customized nodes is that they can be optimized to achieve good performance for the original application that the platform has been specifically designed for (often referred to as the parent application).

On the down side, the optimization for a specific parent application often leads to over-customized architectures. For instance, in order to meet prescribed timing or cost constraints of the parent application, the hardware modules are designed to be highly dependent on each other, i.e., they are heavily coupled. The hardware is often so inflexible that any change in application load or on-site specification requires a complete hardware re-design. Similarly, the software modules are typically heavily coupled with each other and with the specific hardware such that the software modules are not reusable if some other software module or the hardware changes.

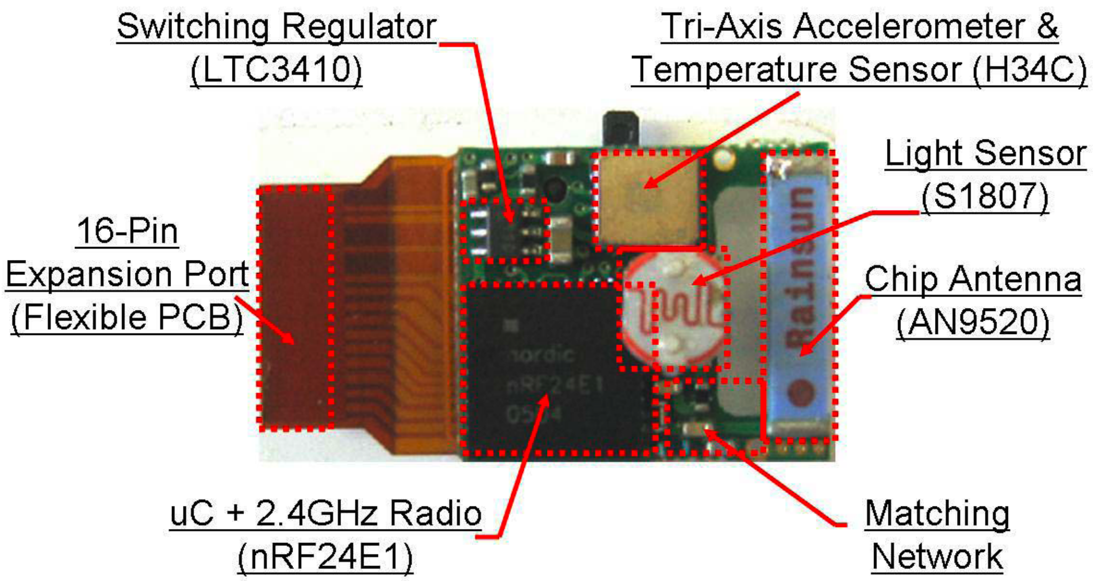
CMUcam3, for example, uses an MCU with very few GPIO pins, so that there is no extra pin to add basic next-step functionality, such as adding a second serial peripheral interface (SPI) slave. This leads to underutilization of the SPI module which is dedicated to only the MMC module, even though it is capable of supporting tens of slaves. An attempt to use SPI for any other purpose requires removing the MMC module.

In eCAM, the radio and the MCU have been merged into one module. This merged radio/MCU module speeds up data processing since the software instructions and data are co-located in the module. Thus, instructions and data do not need to be fetched from external memory or over serial buses and the module synchronization overhead is reduced. As a result, eCAM can implement a simple medium access control (MAC) protocol with increased data rate. However, this optimization prevents future expandability and compatibility with other radio standards. Moreover, in eCAM, the compression stage has been merged with the imager. Should the need for a new compression scheme or imager frame capture arise, the entire camera module will need to be replaced and re-designed.

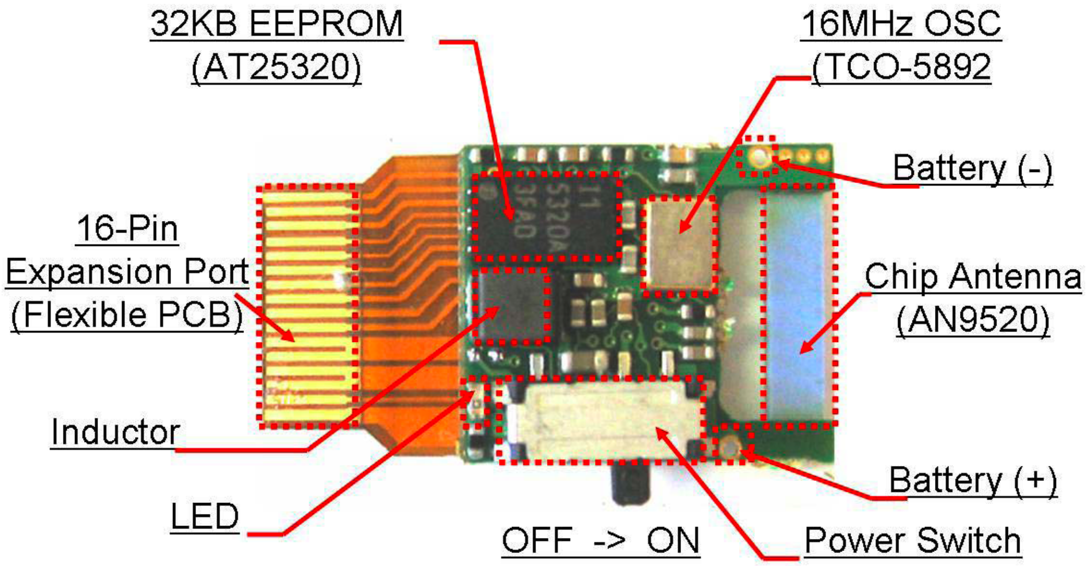
2.5.2 UC Irvine's eCAM and WiSNAP [155]

Overview

The eCAM is constructed by attaching a camera module (with up to VGA video quality) to an Eco mote. As illustrated in Fig. 2.5, the 1 cm^3 sized Eco mote consists of a Nordic VLSI nRF24E1 System on a Chip (SoC), a chip antenna, a 32 KB external EEPROM, an Hitachi-Metal H34C 3-axial accelerometer, a CR1225 Lithium Coin battery, an LTC3459 step-up switching regulator, an FDC6901 load switch, a power path switch, a temperature sensor, and an infrared sensor. The nRF24E1 SoC contains a 2.4 GHz RF transceiver and an 8051-compatible DW8051 MCU. The MCU has a 512 Byte ROM for a bootstrap loader and a 4 KB RAM to run user programs loaded by the bootstrap from the SPI attached EEPROM. The camera module consists of the Omnivision OV7640 CMOS image sensor and OV528 compression/serial-bridge chip. The camera can function as either a video camera or a JPEG still camera. The OV528 is used as a JPEG compression engine as well as a RS-232 interface to the Eco node. The imager supports a variety of size and color formats, including VGA, CIF, and QCIF. It can capture up to 30 fps. The platform radio's transmission consumes less than 10 mA (0 dBm) whereas receiving consumes around 22 mA.


Switching Regulator (LTC3410)
Tri-Axis Accelerometer & Temperature Sensor (H34C)
Light Sensor (S1807)
16-Pin Expansion Port (Flexible PCB)
Chip Antenna (AN9520)
uC + 2.4GHz Radio (nRF24E1)
Matching Network

a) Front side



b) Back side

Figure 2.5: Main architecture components of the Eco board [155].

Advantages

The eCAM platform has a customized radio, which achieves high-speed and low-power due to a simple MAC protocol, instead of a generalized complex MAC which would consume more power. The eCAM bandwidth can theoretical peak at 1 Mbps, which is four times the theoretical peak of the 250 kbps of Zigbee. This makes the eCAM a good candidate for real-time VGA resolution video transmission. The radio's transmission output power can be configured through software to -20 dBm, -10 dBm, -5 dBm, or 0 dBm levels. The eCAM is more power efficient than Bluetooth and 802.11b/g modules, which are typically 20 dBm and 15 dBm respectively, for a 100 m range[194, 138]. The eCAM in-camera hardware JPEG compression is significantly more power efficient than software implementations [155, 134]. The camera compression engine's JPEG codec supports variable quality settings. The imager's ability to capture up to 30 fps enables considerable control of the video quality.

A shown in Figure 2.5, the Eco node has a 16 pin expansion port, which has been designed to use the flexible parallel male connector instead of the typical rigid PCB headers. This choice of "Flexible PCB" makes the Eco node flexible and suitable for different types of packaging, which makes it easy to customize to a variety of applications.

Additionally, the Eco node has an OPTEK OP591 optical sensor, which helps with low resolution and low power vision event processing. When major sensing events are detected, the VGA camera is triggered.

Disadvantages

The customized MAC and radio reduce the networking adaptability and compatibility with other motes. Moreover, the MAC and radio customization misses the low-cost benefit of standardized networking protocols and radio hardware, such as Zigbee compliant radios.

A further drawback of the radio is that it has a range of only about 10 m. Under a demonstration [155, 134], eCAM could only transmit relatively low resolution 320x240 (at 1.5 fps) or 160x128 video streams to the base station. This low performance suggests that the platform has a bottleneck in the video acquisition path and can not exploit its theoretical radio transmission rate of 1 Mbps. The base station then aggregates the data and transmits it to a host computer, which displays the videos in real-time. Reliance on a base station is a limitation as WVSNPs are expected to function in adhoc mode and have access to popular networks, such as WiFi, cellular, or 3G networks.

The platform is a highly optimized board-level system design that achieves a very compact form factor. However, merging MCU and radio as well as JPEG compression and the imager module makes the platform inflexible and fails to take advantage of future improvements in critical components of a mote, such as radio, MCU, compression engine, or encoder. Another concern is that the camera module attaches to the Eco via an RS232 interface, which limits the data transfer rates.

2.5.3 UCLA's Cyclops and Mica [166]

Overview

A typical Cyclops platform is a two-board connection between a CMOS camera module illustrated in Fig 2.6 with an FPGA and a wireless mote, such as a Berkeley MICA2 or MICAz mote. The camera board consists of an Agilent ADCM-1700 CIF CMOS imager with a maximum 352x288 pixel resolution. The camera has an F2.8 lens, image sensor and digitizer, image processing units and data communication units. The camera supports 8-bit monochrome, 24-bit RGB color, and 16-bit YCbCr color image formats.

The Cyclops camera module contains a Complex Programable Logic Device (Xilinx XC2C256 CoolRunner CPLD), a 512 KB external Flash, and a 64 KB external SRAM for high-speed data communication. The CPLD provides the high speed clock, synchronization, and memory control that is required for image capture. The MCU and CPLD and both memories share a common address and data bus. The 7.3728 MHz 8-bit ATMEL ATmega128L MCU controls the imager and performs local image processing, e.g., for inference and parameter configuration. The MCU can map 60 KB of external memory into its memory space. The combination of the internal and external memory presents a contiguous and cohesive memory space of 64 KB to the node's applications.

The Cyclops design isolates the camera module's requirement for high-speed data transfer from the speed ability of the host MCU. It can optionally provide still image frames at low rates if the connecting modules are slow. The camera module is programmable through a synchronous serial I^2C port. Image data is output via an 8-bit parallel bus and three synchronization lines.

Advantages

The modularity of Cyclops, that is, its use of a separate host mote enables "hardware polymorphism", which abstracts the complexity of the imaging device from the host mote. Moreover, the standardized interface makes the Cyclops camera module adaptable to a variety of host motes.

The dedicated image processor enables global serialization of image processing operations by offloading these image processing operations from the host MCU. The global serialization loosens the need for tight synchronization in the "acquire-process-

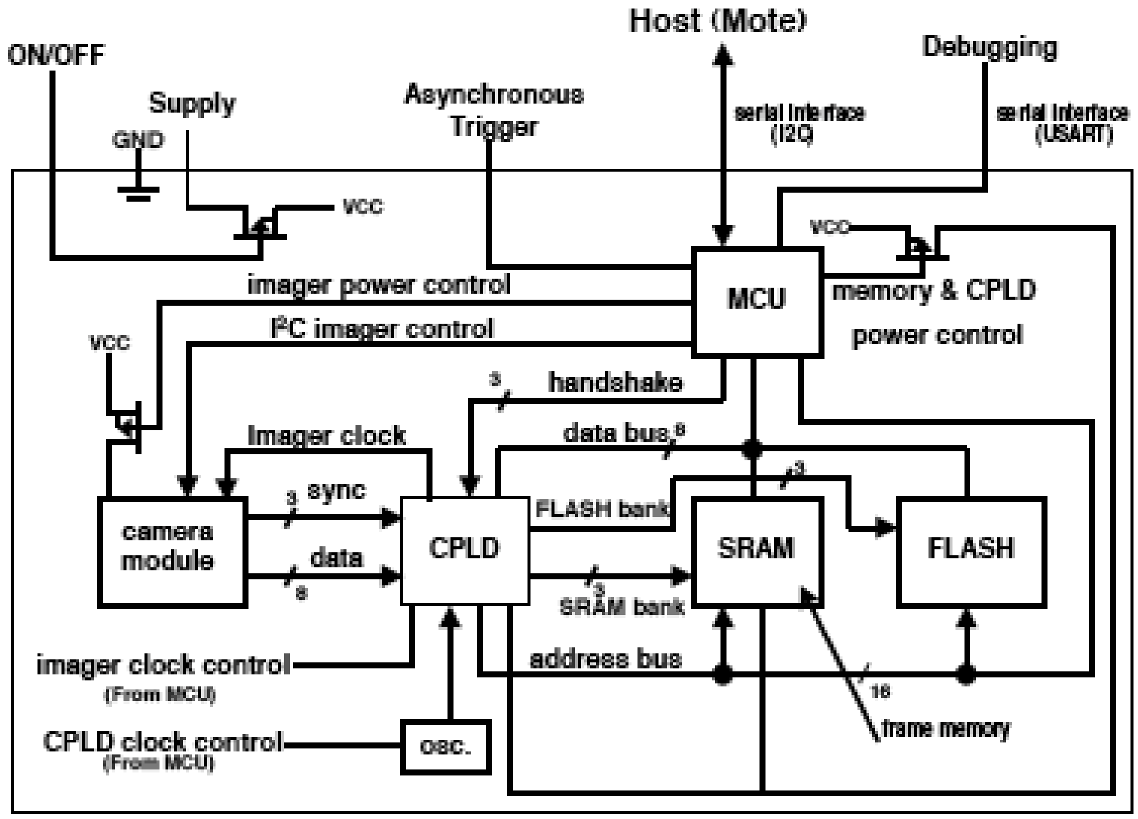


Figure 2.6: Hardware architecture of Cyclops camera module [166].

play" path so that interrupts or handshaking signals can indicate when the dedicated image processing MCU is ready.

The dedicated image processor provides computational parallelism, such that prolonged sensing computations can be isolated to the image processor. This helps with duty cycling idle modules and saves power.

The power consumption of Cyclops is very low and enables large-scale long-term deployment. Cyclopes uses on-demand clock control of components to decrease power consumption. Moreover, to save power an external SRAM is used for storing image frames and is kept in sleep state when not needed. The camera node can automatically drive other subsystems to their lower power state. Cyclops has an asynchronous trigger input paging channel that can be connected to sensors of other modalities for event triggering. A study [122] has shown that object detection operations with Cyclops are 5.7 times more energy efficient than with CMUCam3 under the same settings and functionality.

The CPLD used by Cyclops can perform basic operations during frame capture, such as on-demand access to high speed clocking at capture time and possibly computation. In particular, the fast CPLD clock enables the camera module to carry out calculations and pixel image storage to memory while the imager is capturing. A CPLD also consumes less power than an FPGA during initial configuration reducing the overall cost of the power-down state.

Disadvantages

The slow 4 MHz MCU in Cyclops is not fast enough for data transfer and address generation during image capture. Therefore, the Cyclops design uses a CPLD, an additional component, to provide a high-speed clock. This design choice increases cost, power consumption, and PCB area. Also, as noted in Section 2.2, an 8 bit processor consumes often more power for image related algorithms than a 32 bit processor.

This platform was not intended for repeated image acquisition. Instead, the Cyclops architecture targets applications that occasionally require capture of one (or a few) images. As evaluated in [122], the PCB Header-MCU architecture in Cyclops is six times slower than the FIFO-MCU architecture in CMUCam3. Cyclops also pales CMUCam3 with its 2 fps maximum capture-and-save image processing speed. It also has a low image resolution of 128x128 pixel due to its limited internal Atmega128L MCU memory (128 KB of Flash program and 4 KB of SRAM data memory). The performance analysis in [166], reveals that improving the CPLD's synchronization with the imager would significantly improve the timing (and energy cost) of the image capture. Using more parallelism in the CPLD logic could also reduce the number of CPLD clock cycles needed to perform pixel transfer to SRAM. This could also allow higher imager clock speed and facilitate faster image capture.

Another shortcoming of Cyclops is its firmware's use of the nesC language which is based on TinyOS libraries. This limits its code reusability and refinements often enjoyed by Linux targeted firmware. TinyOS does not provide a preemptive mechanism in its synchronous execution model, i.e., tasks cannot preempt other tasks.

Other key weaknesses are that the Cyclops platform does not include a radio and does not perform any on-board compression. Though Cyclops provides the ability decouple some image processing functions, it does not provide mechanisms for guaranteeing data access or modification integrity, such as semaphores or spin locks.

The Cyclops camera module relies on third-party boards to function as a complete wireless sensor node. Given the need to manage power via duty cycling, the power-aware hardware and algorithms on the camera module may need frequent adjustments to interface with a variety of third-party daughter boards with different power definitions.

2.5.4 Philips' Smart Camera Mote [116, 115]

Overview

The Smart Camera mote focuses mostly on reducing power consumption through low-power local image processing. Local image processing filters out unnecessary data and compresses data before transmission. As illustrated in Figure 2.7, the camera consists of one or two VGA image sensors, an Xetal IC3D single instruction multiple data (SIMD) processor for low-level image processing, and the ATMEL's 8051 host

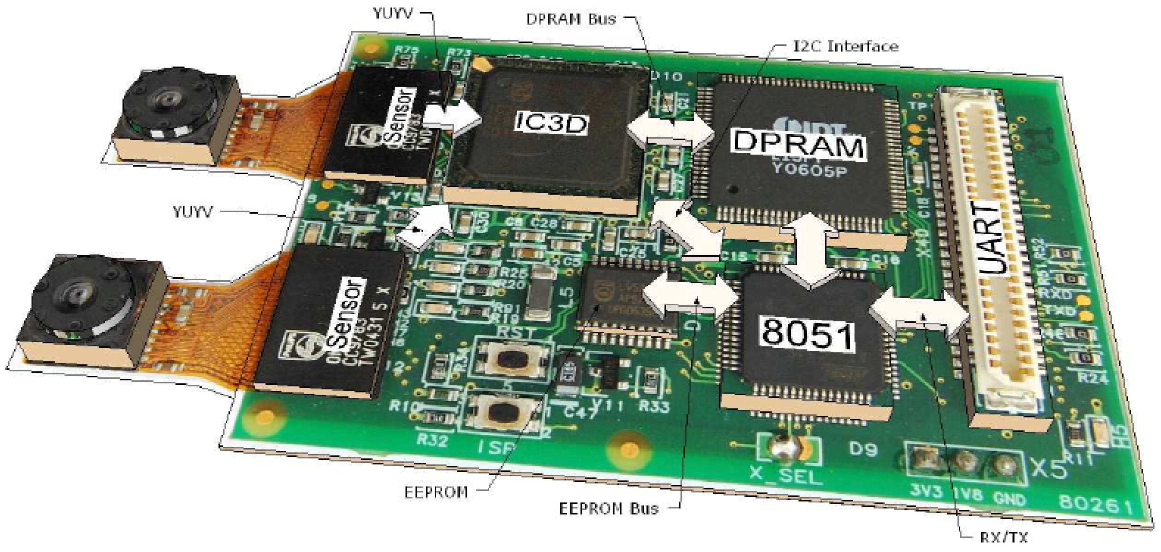


Figure 2.7: Architecture of the Philips Camera Mote [116].

MCU for intermediate and high-level processing, control, and communication. The host 8051 and the IC3D share a dual port RAM (DPRAM). The platform uses a customized Aquis Grain ZigBee module made of an 8051 MCU and Chipcon CC2420 radio. The radio's software control is reprogrammable on the 8051.

A global control processor (GCP) within the IC3D system-on-chip (SoC) is used to control most of the IC3D as well as performing global digital signal processing (DSP) operations, video synchronization, program flow, and external communication. The 8051 host MCU has direct access to the DPRAM and has its own internal 1792 Byte RAM, 64 KB FLASH, and 2 KB EEPROM. It uses its large number of I/O pins to control the camera and its surroundings. The host has its own tiny task-switching RTOS. The radio module attaches to the platform via the 8051 host's UART.

Advantages

The IC3D is designed for video processing and has dedicated internal architecture blocks for video, such as linear processor arrays, line memories, and video input and output processor blocks. The video processor blocks can simultaneously handle one pixel at a time for CIF (320x240) or two at a time for VGA (640x480). Pixels of the image lines are interlaced on the memory lines. Sharing the DPRAM enables the main processors to work in a shared workspace on their own processing pace. This enables asynchronous connection between the GCP and IC3D and simple shared memory based software synchronization schemes. The DPRAM can store two images of up to 256×256 pixels and enables the IC3D to process frames at camera speed [116, 115], while a detailed evaluation of the frame capture-and-save and transmission rates remain for future research.

The SIMD based architecture of the IC3D decodes fewer instructions for more computational work and hence requires less memory access, which reduces energy consumption. In contrast, each 30x30 pixel imager of MeshEye [90], captures its own small image, loads it into memory and process the duplicate instructions on each image only to detect an event. In [116], on the other hand, a large frame is loaded to the same memory and the same "detect event" instruction is issued for each MCU core to process part of the image for an event, sequentially or in parallel. The first core to detect an event can signal the other core to stop, hence reducing not only processing time but also memory paging which conserves power.

The IC3D has a peak pixel performance of around 50 giga operations per second (GOPS). The GCP is powerful enough to perform computer vision tasks, such as face detection at power consumption levels below 100 mW.

The 8051 host's UART has its own baud rate generator which leaves the 8-bit and two 16-bit timers available for RTOS switching and user applications. The radio module's peer-to-peer structure enables point-to-point camera-to-camera communication. The camera can be remotely programmed via the radio and the in-system programmability feature of the 8051.

Disadvantages

The employed Zigbee module has a range of only five meters. Further, its maximum data rate of around 10 kbps makes the Zigbee module poorly suited for real-time image transmission. This low transmission rate limits the module to transmitting only meta-data of the scene's details or events.

The module has numerous major components that altogether are expensive. The power efficiency of the SIMD approach is not yet well understood and requires more research to evaluate whether the dual imagers and the parallel processing of the subsets of the VGA image for frame differencing are beneficial in typical video sensor application scenarios. Overall, the node suffers from a mismatch between the extensive image and video capture capabilities and the limited wireless transmission capability.

2.5.5 Carnegie Mellon's CMUcam3 [173] and DSPCam [109, 110]

Overview

Carnegie Mellon's CMUcam3 sensor node is probably the most open of the heavily coupled platforms in that all hardware schematics, software, and PCB files are freely available online for the research community. Many commercial vendors are also allowed to copy, manufacture, and sell the platform with or without design modifications. CMUcam3 is capable of RGB color CIF resolution (352x288 pixels). At its core is an NXP LPC2106, which is a 32-bit 60 MHz ARM7TDMI MCU with built-in 64 KB of RAM and 128 KB of flash memory. It uses either an Omnivision OV6620 or OV7620 CMOS camera-on-a-chip/image sensor, which can load images at 26 fps. As shown in Figure 2.8, CMUcam3 also uses Averlogic's AL4V8M440 (1 MB, 50 MHz) video FIFO buffer as a dedicated frame buffer between the camera and the host MCU.

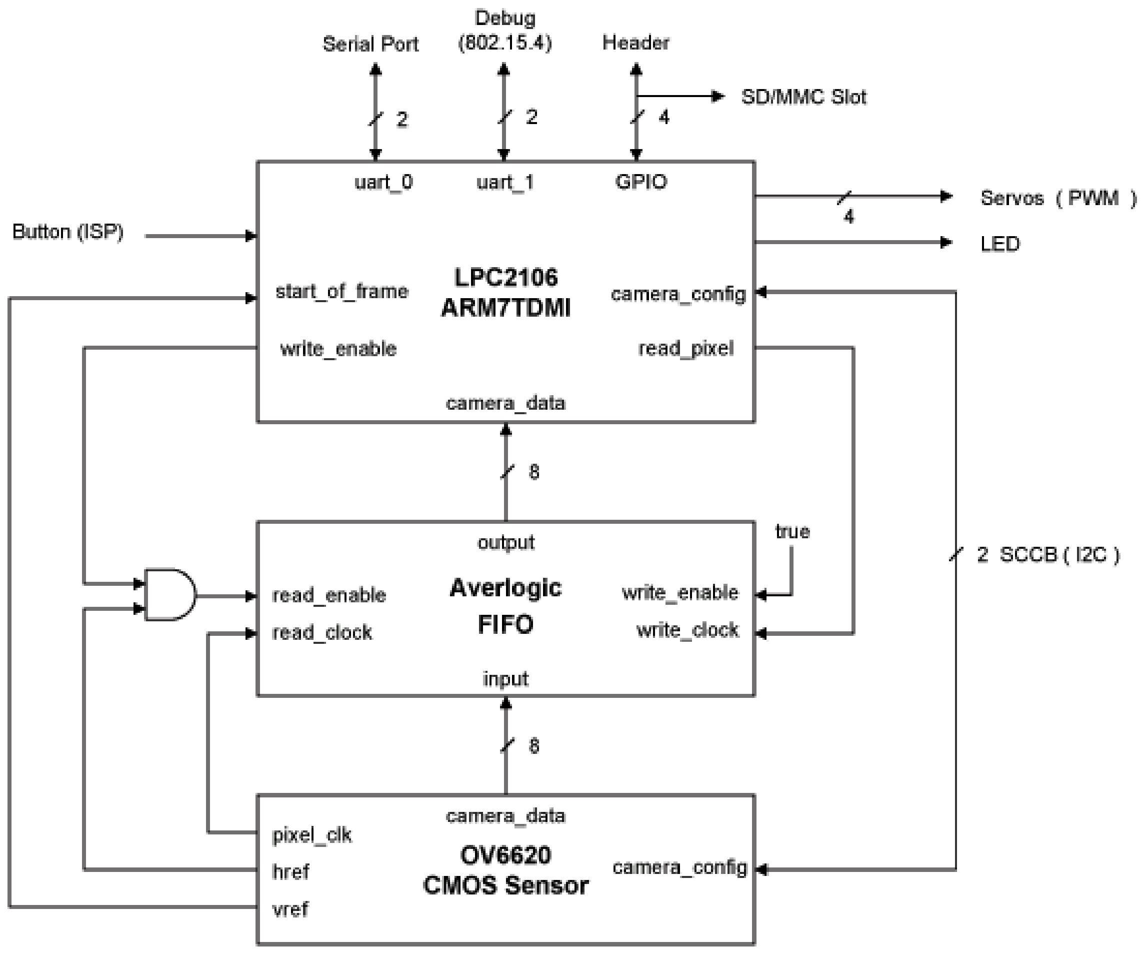


Figure 2.8: CMUCam3's major block architecture [173].

Hence, the actual capture-and-save frame rate is limited by the hardware FIFO buffer between the imager and the MCU. Clocking the frames out of the FIFO buffer to the MCU memory gives the actual overall capture-and-save frame rate. CMUcam3 has software JPEG compression and has a basic image manipulation library. CMUCam3 uses an MMC card attached via SPI for mass data storage. The card uses a FAT16 file system type, which is compatible to almost all other flash card readers.

An improved follow-up to CMUCam3 is the DSPCam [109], which has the characteristics of an externally dependent architecture and is therefore included in Table 2.7.

Nevertheless, since DSPCam grew from CMUCam3, we discuss both in this section. As illustrated in Figure 2.9, DSPcam uses the 32-bit RISC Blackfin DSP-MCU SoC from Analog Devices and a SXGA (1280x1024), VGA (640x480), QVGA (320x240), and CIF capable OmniVision CMOS image sensor. A stand-alone WiPort 802.11b/g module is integrated on the board. DSPCam provides an interface for third party modules for possible 802.15.4 based radios as well as other low data rate sensors. The image array's throughput can be as high as 30 VGA fps and 15 SXGA fps. The imager consumes 50 mW for 15 SXGA fps with a standby power of 30 μW. DSPCam is a smart mote, which creates metadata and tags for video to enable efficient video retrieval and transmission. DSPCam runs a uCLinux OS and a custom Time Synchronized Application level MAC (TSAM) protocol which provides quality of service (QoS) through a priority-based dynamic bandwidth allocation for the video streams. TSAM bypasses standard Linux network API calls. Depending on the power states of the three major modules, the power consumptions of the DSPCam ranges from above 0.330 W (all idle) to 2.574 W (all active).

Advantages

The CMUCam3 hardware can carry out two modes of frame differencing. In low resolution mode, the current image of 88x143 or 176x255 pixels is converted to an 8x8 grid for differencing. In the high resolution mode, the current CIF image is converted to a 16x16 grid for differencing.

The single board FIFO-MCU architecture of CMUCam3 is faster than the PCB Header-MCU setup used in Cyclops. In particular, the FIFO buffer decouples the processing of the host MCU from the camera's pixel clock, which increases frame rates. Decoupling the MCU processing from the individual pixel access times allows the pixel clock on the camera to be set to a smaller value than the worst case per

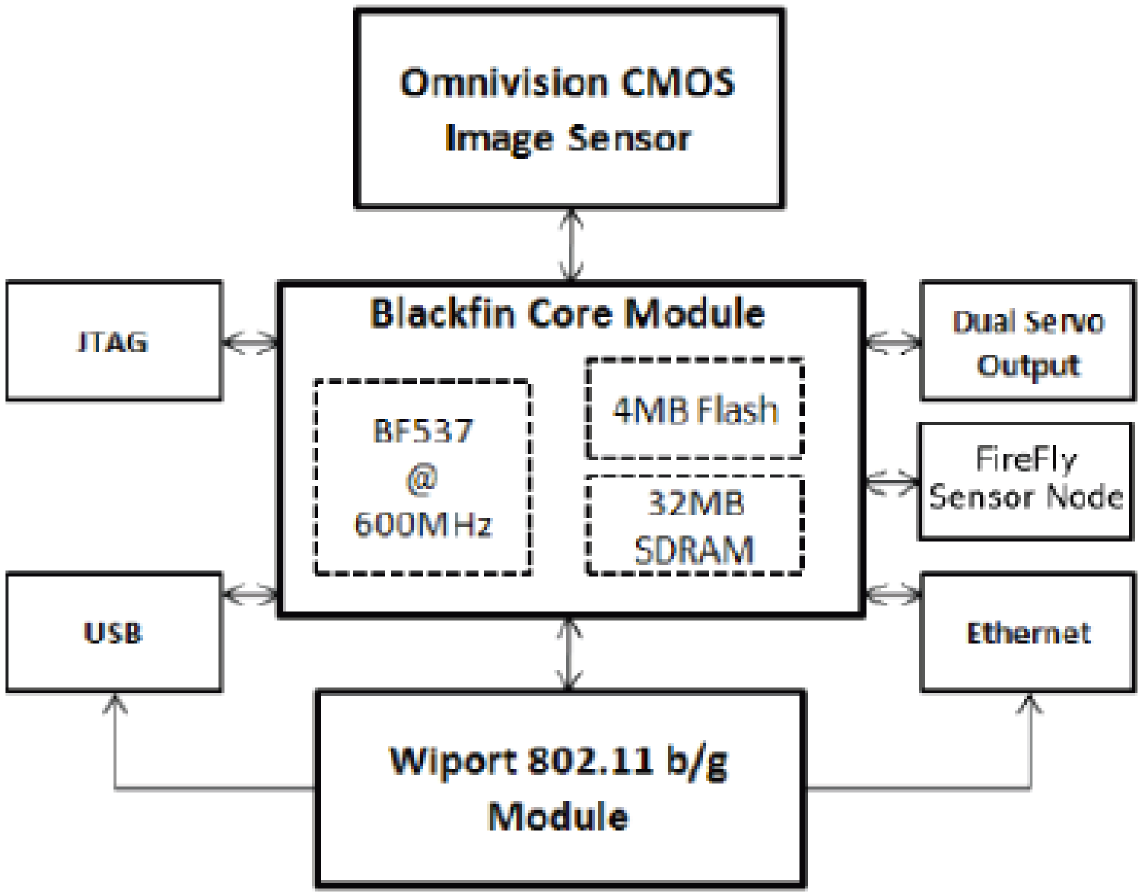


Figure 2.9: DSPCam's major block architecture [109].

pixel processing period. As evaluated in [122], the Cyclops design is six times slower than CMUCam3. Compared to the 2 fps of Cyclops, CMUCam3 can capture and save between 2 and 5 fps. An additional advantage of the FIFO buffer is its ability to reset the read pointer, which enables basic multiple pass image processing, such as down sampling, rewinding, and windowing.

The CMUCam3's OV6620 camera supports a maximum resolution of 352x288 at 50 fps. CMUCam3 is capable of software based compression only and supports other optimized vision algorithms. The sensor node software provides the JPEG, portable network graphics (PNG), and ZIP compression libraries, which are useful for low data rate streaming.

The MCU of the CMUCam3 platform uses software controlled frequency scaling for power management. CMUCam3 has three power modes (active, idle, and power down). The camera module, for example, can be powered down separately without affecting the other two main CMUCam3 blocks.

The CMUCam3 MCU core has a memory acceleration module (MAM) for fetching data from flash memory in a single MCU cycle. The MMC option in CMUCam3 provides easy external access to its data as the data are readable by standard flash readers. The availability of serial in-system programming (ISP) provides for inexpensive built-in firmware loading and programming as compared to many MCUs that require extra joint test action group (JTAG, IEEE 1149.1) hardware. The MCU provides a co-processor interface which can be useful for offloading some heavy computation from the host MCU. CMUcam3 provides an expansion port that is compatible with a variety of wireless sensor nodes, including the popular Berkeley sensor platforms.

DSPCam has considerably more memory than CMUCam3 with 32MB of fast SDRAM, clocked up to 133MHz, and 4MB of Flash. A new high-performance feature is the Direct Memory Access (DMA), which enables low overhead block transmission of video frames from the camera to the SoC's internal memory. This frees up the CPU core for other critical tasks. In addition to standard MCU interfaces, the Blackfin SoC provides a Parallel Peripheral Interface (PPI) which enables a direct connection to the CMOS image sensor. DSPCam accelerates video and image processing through its special video instruction architecture that is SIMD compliant. The USB-UART bridge provides useful external mass storage options for the DSPCam.

Disadvantages

The CMUCam3 design avoids high-cost components and hence lacks efficient storage and memory structures, such as L1 cache, memory management unit (MMU) and external direct memory access (DMA), as well as adequate random access memory (RAM) and flash memory. This shortcoming as well as the relatively slow I/O can be a throughput bottleneck. For example, reading one pixel value can take up to 14 clock cycles, of which 12 are wasted on waiting for input/output (I/O) transactions. The small memory of the "MMU less" ARM7TDMI core prohibits the use of even the tiniest Linux RTOS, such as uCLinux, which has been tested to work on other "MMU less" MCUs [61].

The coarse frame differencing leads to high object location error rates and is hence unsuitable for estimating object locations. Further, as used in [122], CMUCam3's processing and object detection algorithm (frame capture and frame differencing) were 5.67 times less energy efficient than Cyclops. The CMOS camera lacks a monochrome output mode, and hence color information must be clocked out of the FIFO. Also, the FIFO structure prevents random access to pixels.

The CMUCam3's MCU has very few I/O ports to enable extensible direct access to the MCU. That is, only a few I/O ports are configurable to be used for other I/O purposes and some bus protocols are underutilized. For example, the SPI bus has only one chip select pin, which is connected directly to the MMC card. This means that no other module can be connected to the SPI bus without first disconnecting the MMC card. This inflexibility may force designers to use alternate connectors, such as UART, which are slower and limit the throughput of the sensor node.

The optimization of the hardware architecture has focused on the video acquisition but neglected the wireless transmission and memory components critical to a WVSNP. Although a dedicated frame buffer speeds up and simplifies the camera image acquisition it is not accessible to other components when not in use. A DMA system would be more efficient and cheaper.

The CMUCam3 MCU, similar to many other low-cost systems, lacks the floating point hardware, RAM, and computation speed required for many complex computer vision algorithms. Further, CMUCam3 lacks a real time clock (RTC) which could be critical in duty cycling of attached modules, global packet tracking, and time stamping of real time video.

During board power down, the RAM is not maintained. Therefore, the camera parameters must be restored by the firmware at startup. CMUCam3 takes relatively long (sometimes close to a second) to switch between power modes or to transition from off to on. These long switch times limit applications that require fast duty cycling and short startup times, for example, when alerted to capture a frame.

DSPCam depends on an external node or module, such as a Firefly sensor node, to provide access to low data rate nodes using IEEE 802.15.4-based radios. The DSPCam board includes an Ethernet module, which is operated in a bridge configuration for wireless transmissions with the attached Wiport module. The TCP/IP networking drivers thus continue sending data to the Ethernet module, which is then forwarded to the Wiport module for wireless transmission. At the same time, the core module directly controls the Wiport module via a serial port. This setup introduces inefficiencies as there is duplication in the wireless transmission path.

The DSPcam architecture does not provide mechanisms for the host SoC to control the power modes of the camera, the WiFi module, and other external nodes. This is a critical functionality for a low-power WVSNP. Future research needs to evaluate in detail the impact of the TSAM protocol and other in-node processing on the QVGA/-CIF frame rate. Although DSPCam is a significant improvement over CMUCam3, it traded the highly coupled architecture of CMUCam3 for an externally dependent architectures that relies on third-party modules with no power management control.

2.6 Externally Dependent Architectures

2.6.1 Overview

Externally dependent architectures depend on a mosaic of external "daughter boards" to achieve basic functionality. The justification for this designs approach is that nodes operating at different tiers in a multi-tier network have different functionality requirements. As a result, the externally dependent architectures depend heavily on the designer's view of the sensor network and hence suffer from similar target application limitations as the heavily coupled architectures.

Nodes that depend on external PCB modules often lack a cross-platform standard interface, limiting interoperability with daughter boards. In particular, a given base platform can usually interoperate only with the daughter boards specifically designed for the base platform, limiting flexibility. This design model often hides the real cost of a node and results in cumbersome designs that are inefficient. For example, the use of basic interfaces, such as RS-232, Ethernet, USB, and JTAG on Stargate requires a daughter board. Similarly, a special daughter board is required to supply the Imote2 with battery power. Assembly of an image capable platform based ScatterWeb requires at least four different boards.

The need of externally dependent architectures for daughter boards for a basic application result often in excess power consumption. This is because each stand-alone daughter board needs some basic circuitry, which consumes power. This circuitry is usually duplicated on other daughter boards and hence consumes more power than reusing the same circuitry on one PCB.

2.6.2 UC Berkeley's Stargate [221, 135, 43, 189, 122]

Overview

Stargate is a relatively popular platform and is commercialized by Crossbow Technology Inc. The Stargate platform is capable of real-time video compression. The platform offers a wide range of interfaces, such as Ethernet, USB, Serial, compact flash (CF), and PCMCIA, making the platform suitable for residential gateways and backbone nodes in multi-tier sensor networks.

As illustrated in Figure 2.10, Stargate consists of an XScale PXA255 processor whose speed ranges from 100 to 400 MHz and consumes between 170 and 400 mW. The Stargate processor can be configured to have 32 to 64 MB of RAM and/or 32 MB of Flash. Energy profiling [135] shows that Stargate consumes more energy during intensive processing (e.g., FFT operations) and flash accesses than through transmissions and receptions. Interestingly, the energy consumption for data transmission was found to be 5 % less than that for data reception. This is a reversal of the typical characteristics of wireless devices and can be attributed to the specific employed duty cycling mechanisms. On average, Stargate uses about 1600 mW in active mode and around 107 mW in sleep mode.

Advantages

The Stargate platform is extensible enough that it can attach to other modules as needed to communicate with other wireless sensors and third-party application-specific modules. The platform has sufficient RAM and Flash memory to run a complete Embedded Linux OS. As a result, Stargate has extensive software capabilities, including support for web cams attached via USB or PCMCIA, and compact flash (CF) based 802.11b radios to communicate with higher data rate sensors.

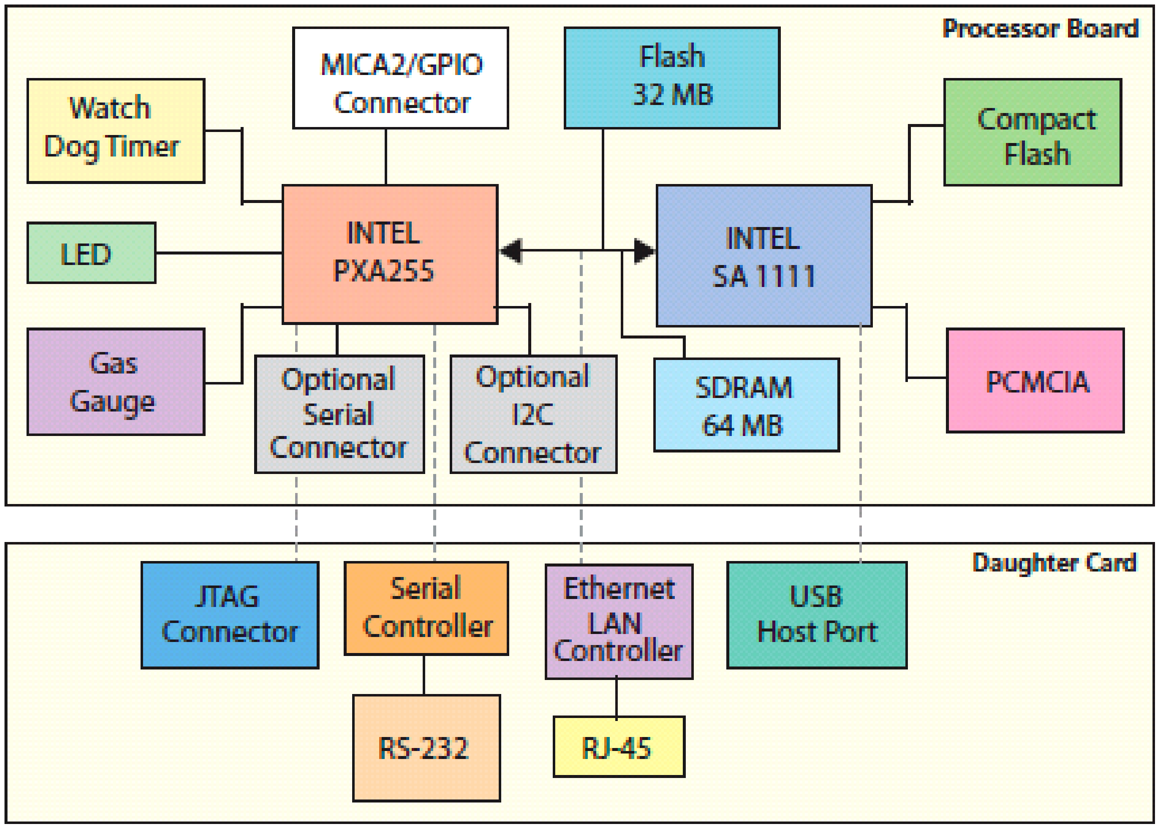


Figure 2.10: Stargate architecture block diagram showing the main board and the daughter board.

The processor is sufficiently powerful to locally run object recognition algorithms. Studies have shown that Stargate is more energy efficient than Panoptes. It consumes 25 % less energy for some applications in spite of having twice Panoptes' processing power [135, 189, 122]. Increasing the clock speed of the Stargate MCU by 300 % results only in a small increase of 24 % in power consumption [43], which is a desirable characteristic for a video processing MCU.

Disadvantages

As used in [189], Stargate operates akin to a computer networking gateway interface and is architecturally too general and not optimized for low power consumption. It uses power-inefficient interfaces, such as a personal Computer memory card international association (PCMCIA) interface based card for the 802.11b module. The PCMCIA standard is a general computer standard and not readily optimized for a low power sensor.

The webcam attached to Stargate is not suitable for a resource-constrained standalone video sensor. Stargate does not have hardware support for being woken up by other motes. Special mechanisms have to be implemented on the other connected motes to mimic the wake-up functionality. This makes Stargate dependent on the Mica-type motes for the wake-up functionality. Stargate is also dependent on the Mica-type motes for simultaneous 900 MHz low-data rate transmissions. The extra wakeup overhead adds to wakeup latency costs. The latency and power consumption further increase due to the architecture's inefficient reliance on the daughter board for Ethernet, USB, and serial connectors, see Figure 2.10. Though both the main and daughter boards have battery input, only the daughter board has a direct current (DC) input, which increases the main board's reliance on the daughter board.

Regarding the multimedia functionalities, the XScale MCU lacks floating-point hardware support. Floating-point operations may be needed to efficiently perform multimedia processing algorithms. Images acquired through USB are typically transmitted to the processor in a USB compressed format. This adds to decompression overhead prior to local processing as well as loss of some image data. The employed version 1.0 USB is slow and limits image bandwidth.

2.6.3 Crossbow's Imote2/Stargate 2 [147, 13, 135] and UC's CITRIC [40].

Overview

Imote2 is the latest in a series of attempts to create a powerful and general sensor node by Intel and Crossbow. Its predecessors, the original trial Imotes, lacked many elements expected of a WVSNP. The first trial Imote used a slow 8-bit 12 MHz ARM7 MCU with 64 KB RAM and 32 KB Flash memory. Its successor used an ARM7TDMI MCU with 64 KB SRAM, 512 KB Flash, and speed ranging from 12 to 48 MHz. The first two Imotes had an on-board Bluetooth radio and support for the TinyOS RTOS.

Compared to its predecessors, Imote2 has substantially increased computation power and capabilities. It features a PXA271 XScale SoC. The SoC's 32-bit ARM11 core is configurable between 13 and 416 MHz clock speeds. The ARM core contains 256 KB SRAM, and is attached to a 32 MB Flash, and 32 MB SDRAM storage within the SoC. Imote2 has a Zigbee compliant IEEE 802.15.4 CC2420 radio and a surface-mount antenna, but has no default Bluetooth radio. Supported RTOSs for Imote2 are TinyOS, Linux, Microsoft's .NET Micro, and SOS. Imote2 is intended to replace the original Stargate platform and is therefore also referred to as Stargate 2.

A similar recent platform, CITRIC [40], Figure 2.12, by the Universities of California at Berkeley and Merced as well as the Taiwanese ITR Institute is a follow-up design to Imote2. CITRIC consists of a 624 MHz frequency-scalable XScale MCU, 256KB of internal SRAM, 16MB FLASH, and 64MB external low-power RAM running at 1.8 V. Compared to Imote2, CITRIC is more modular in its design in that it separates the image processing unit from the networking unit. CITRIC also uses a faster Omnivision 1.3 megapixel camera, OV9655, capable of 15 SXGA (1280x1024) fps, 30 (640x480) VGA fps, and a scale-down from CIF to 40x30 pixels. CITRIC runs embedded Linux. The imager has an active current consumption of 90 mW for 15

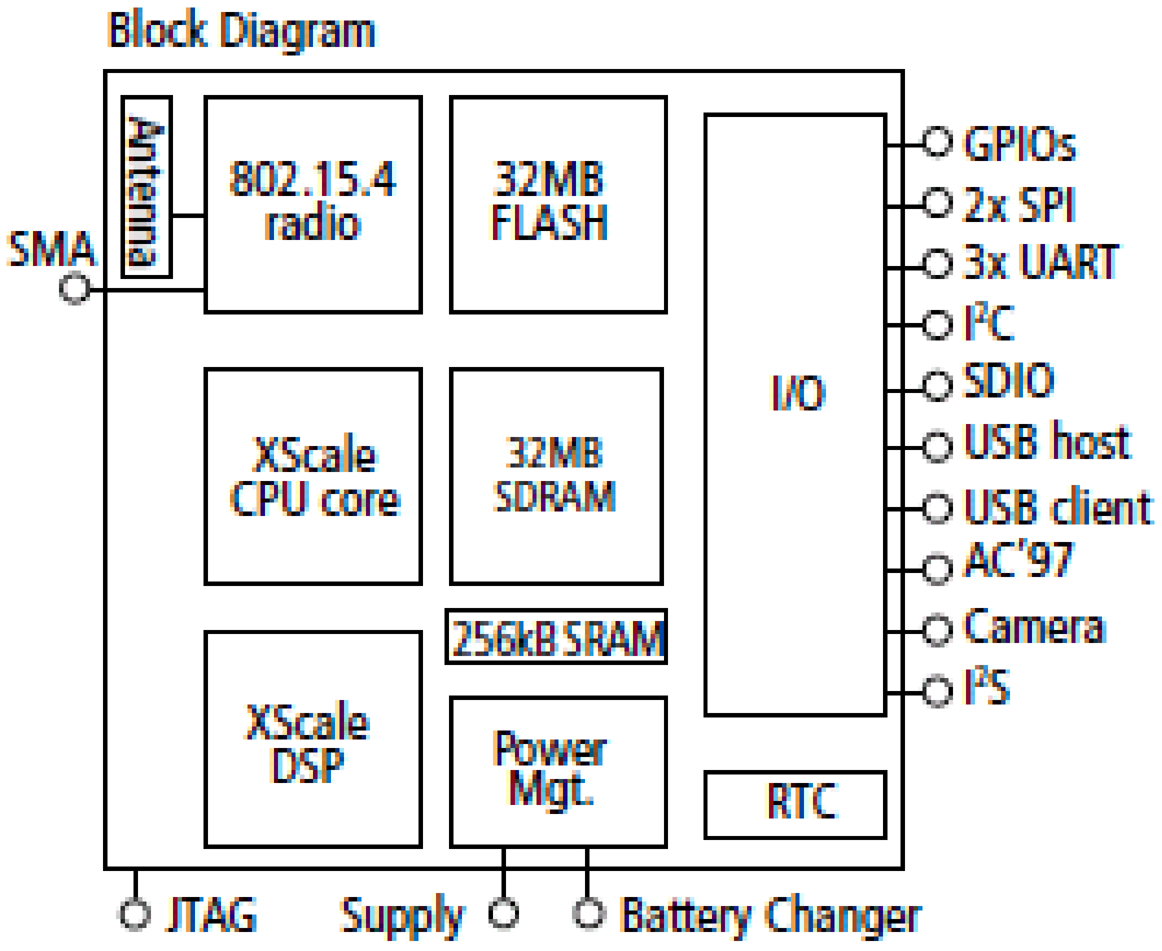


Figure 2.11: Imote2 block architecture [147].

SXGA fps and a standby current of less than 20 μA. CITRIC has an overall power consumption from 428 mW (idle) to 970 mW (active at 520 MHz). This means that CITRIC can last for slightly over 16 hours with four AA batteries with a power rating of 2700 mAh.

Advantages

The PXA271 XScale in Imote2 is a very powerful SoC platform, combining an ARM11 Core, a DSP core, as well as Flash and RAM memories. This compact design improves data access and execution speeds and facilitates power management algorithms that

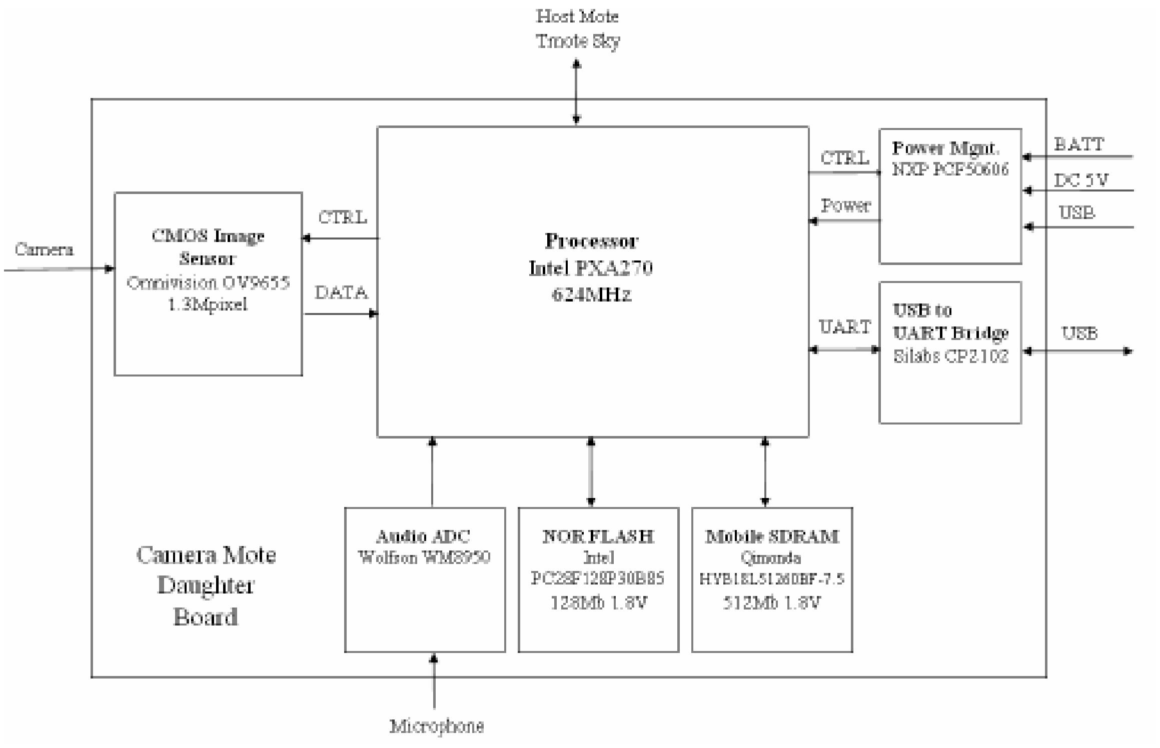


Figure 2.12: CITRIC block architecture [40].

use the SoC's power modes. Specifically, the clock speed of the Imote2 MCU (PXA271 XScale) has a very wide range of power applications through its use of Dynamic Voltage Scaling. It can be set to as low as 13 MHz and can operate as low as 0.85 V, which enables very low power operation.

The Imote2 on-chip DSP coprocessor can be used for wireless operations and multimedia operation acceleration. This co-processor improves the parallelism of the node, especially for storage and compression operations.

The nodes have large on-board RAM and Flash memories. Imote2 provides an interface to support a variety of additional or alternate radios. Further, Imote2 has a variety of targeted high-speed standard interface modules, such as I2S and AC97 for audio, a camera chip interface, and a fast infrared port, in addition to the usual MCU interfaces, such as UART and SPI.

The latest Imote2 board is quite compact, measuring 36 mmx48 mmx9 mm, enabling its inclusion in many sensor node applications. Further, the support for many RTOSs, especially Linux, makes it a good choice.

CITRIC's modular separation of the image processing unit from the networking unit makes it more adaptable to applications than Imote2. CITRIC's 16 MB external Flash is a NOR type memory with faster access times than the typical NAND based memories. It also is capable of the latest Linux supported eXecution-In-Place (XIP), which provides the capability to boot-up and execute code directly from non-volatile memory. The USB-UART bridge provides useful external mass storage options for CITRIC.

The very low standby current consumption of 20 μA makes CITRIC a good candidate for power conservation with duty cycling. Further, the choice of low-power memory is significant as memory typically consumes about the same power as the processor, that is, approximately 20 % of the node's power. The CITRIC cluster of boards can be powered with four AA batteries, a USB cable, or a 5 V DC power adapter.

Disadvantages

Though Imote2's PXA271 provides many peripheral interfaces suitable for multimedia acquisition and processing it depends heavily on external boards for basic operations. These external boards include daughter boards for battery power supply as well as JTAG and USB interfaces. The many attachments required for core functionalities make the platform eventually expensive. Also, the hierarchy of hardware PCBs required for core functionalities introduces latency and power drawbacks similar to those arising with Stargate.

Any high-throughput wireless transmission of multimedia will also need an external board attachment. The surface mount antenna for the on-board Zigbee radio has only a range of 30 m line of sight, requiring an external antenna. Moreover, CITRIC depends on an external Tmote Sky mote with a Zigbee-ready radio for low data rate wireless transmissions. Additionally, CITRIC's camera is attached to a separate camera daughter card. Both the main processor board and the camera daughter board depend on the Tmote Sky mote for battery operated power. This introduces power inefficiencies due to the high number of passive components on each board.

The PXA270 CITRIC core does not support NAND type memories, which limits the designer's choices. Although CITRIC has a power management IC, it is located on the camera daughter board which means the main processor board is dependent on the camera to manage its power. Similar to Imote2 this architecture is heavily externally dependent and despite its higher computational power, it does not have the radio hardware resources for faster video streaming.

2.6.4 Freie Universitt ScatterWeb's ESB430-, ECR430-COMedia C328x modules [176]

Overview

This is a platform designed for research and education. To accommodate diverse research and educational needs it consists of a mosaic of function-specific PCB modules that can be assembled for a desired application area. A sensor node built with these function-specific PCB modules may form an ad-hoc network with other nodes. Some nodes can act as data sources, some as relays, and some as data collectors. A node can simultaneously perform all three functionalities. There are many translator gateway boards to interface ScatterWeb-type boards with standard interfaces, such as RS485, Bluetooth, Ethernet, and USB.

A camera node can be assembled from the ScatterWeb boards by combining an embedded sensor board (ESB, i.e., ESB430), an ECR430 board, and a COMedia C328-7640 VGA (640x480 16-bit pixel) camera module. The camera's resolution can be configured to 80x64, 160x128, 320x240, and 640x480 pixels. The ESB430 can be programmed via UART or USB. The ESB typically has a TI MSP430 MCU, a transceiver, a luminosity sensor, a noise detector, a vibration sensor, an IR movement detector and IR transceiver, a microphone/speaker, and a timer.

The radios are usually 868 MHz RFM TR1001 transceivers and lately the longer range 434 MHz CC1021 transceiver from Chipcon. For energy harvesting, the nodes store solar cell energy in gold-cap capacitors. Piezo crystals and other thermo-elements are also used.

The camera modules have a VGA camera chip and a JPEG compression block. They draw 50 mA while operating at 3.3 V. They are about 2x3 cm^2 in area. The camera module takes commands via the serial interface, processes/compresses the image, and feeds back the resulting image through the same serial port. The VGA frames can be compressed to 20–30 KB sizes. Images are first transferred from the camera module to the built-in 64 KB EEPROM and then transmitted over the air.

Advantages

The PCB module based architecture provides flexibility of reconfiguring the platform for different uses. A cascade of an embedded sensor board (ESB) with compatible GSM/GPRS modules and embedded web server modules (EWS) provides a gateway to receive configuration commands and send node data from/to the Internet and cellular networks.

One of ScatterWeb's PCB modules, the so-called ScatterFlasher, can be attached to a PC for over-the-air programming (flashing) of all sensors, debugging, and remote sensor data collection. Other boards, such as embedded web server (EWS) use power over Ethernet (PoE) to power the host MCU and other PCB components. This is a good way to reduce cost. The EWS can be used to setup ad-hoc Ethernet networks.

The MCU requires about 2 μA in deep-sleep mode, which is power efficient for duty cycle applications. The entire camera module uses about 100 μA in power down mode. The ESB can switch off the camera module's power supply for additional energy savings. The energy scavenging options provided by the nodes make them candidates for long-term outdoor deployment. The employed 1 F capacitors last for about ten hours for typical monitoring, which is enough energy for over 420 sensing and sending cycles.

Disadvantages

While the PCB module based architecture of ScatterWeb provides flexibility, this design strategy suffers from extensive component repetition and underutilization since the modules are expected to be stand alone. Also, the ESB lacks the interfaces and power management infrastructure to control power modes of the individual components on the attached boards.

The serial interface has a maximum data rate of 115 kbps, which is low for image transfer. The module can only wirelessly stream a 160x128 8-bit preview video at 0.75–6 fps. Downloading a compressed image from the camera module to the ESB takes about 2 s. Transmitting an image can draw 7 mA and take about 9.6 s. Overall, this consumes about 0.058 mAh per transmitted image. This translates into about 27,000 images for a rechargeable AA battery with a 2000 mAh capacity and a usable capacity of 80%. As evaluated in [176], a 20–30 kB image takes 12 to 17 s to send, which allows capturing and transmitting only 3 to 5 compressed images per minute.

2.6.5 CSIRO ICT Centre's FleckTM-3 [111, 224]

Overview

Fleck-3 is made up of an 8 MHz Atmega128 MCU running a TinyOS RTOS. The platform consists of a 76.8 kbps Nordic NRF905 radio transceiver and two daughter boards: one for the camera and one for all image processing operations, as illustrated in Figure 2.13. The daughter boards interface and communicate with Fleck-3 via SPI and GPIO interfaces and relevant interrupts.

The DSP daughter board consists of the TI TMS320F2812, a 32-bit, 150 MHz DSP with 128 KB of on-chip program FLASH and 1 MB of external SRAM. The camera board is made up of an Omnivision OV7640 VGA (640x480) or QVGA (320x240) color CMOS sensor with Bayer pattern filter [23]. The progressive scan sensor supports windowed and sub-sampled images. The DSP on the daughter board can control and set camera parameters via an I2C bus. Frames are moved from the sensors into external SRAM using the circuitry implemented in an FPGA on the DSP daughter board. Reference frames are also stored on the DSP board's external memory.

Advantages

The choice of a 32-bit DSP chip satisfies the 32-bit energy advantage over 16- or lower-bit MCUs, see Section 2.2. The 32-bit DSP achieves 0.9 MIPS/mA compared to 2.1 MIPS8/mA for the 8-bit Atmega 128L. Also, the acquire, compress, and transmit strategy has been shown to be eight times more energy efficient than the acquire, store, and transmit strategy [111, 224], justifying the compression stage in the architecture.

The daughter cards can be turned on and off by the Fleck baseboard. This board-to-board power mode flexibility and the use of interrupts for communication with the Fleck-3 can be used by power management algorithms. The separation of function-

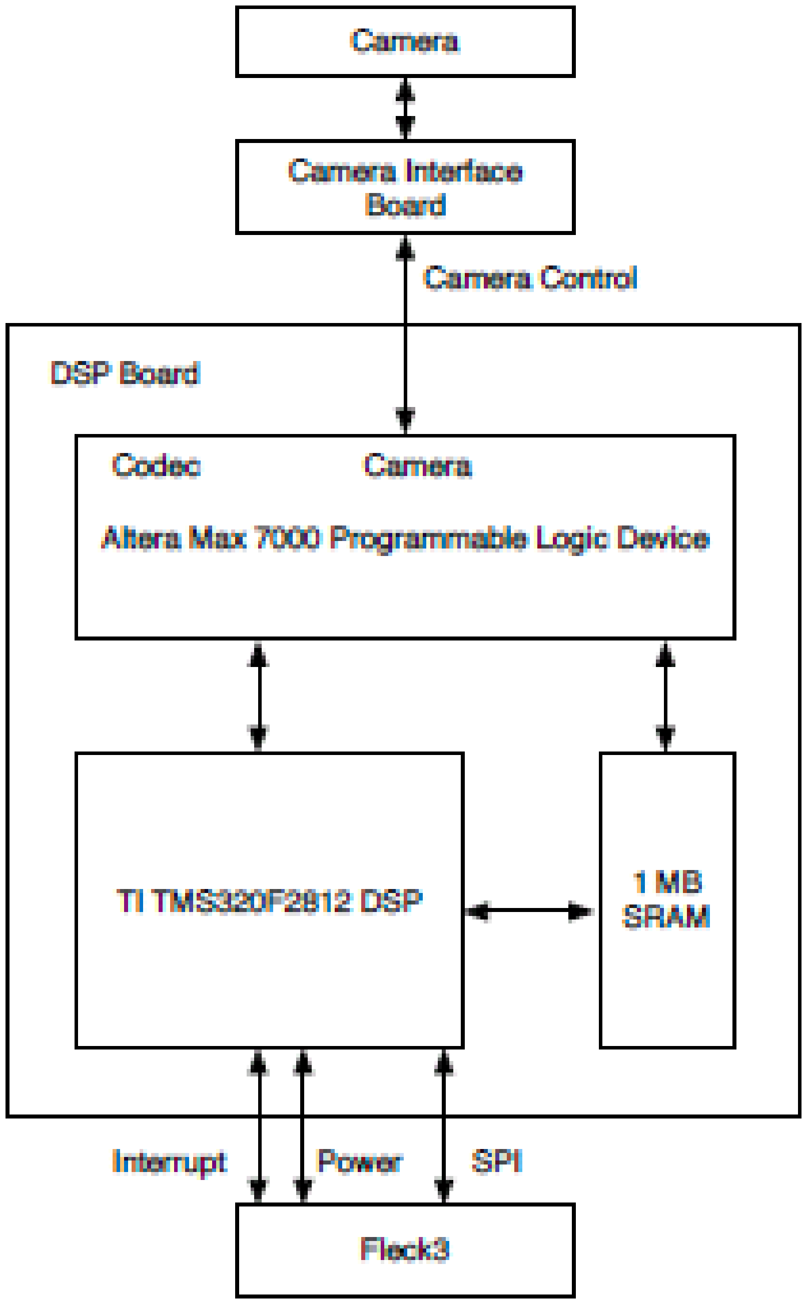


Figure 2.13: Hardware architecture of a camera node formed by a Fleck sensor node, a DSP board, and an image sensor [224].

ality into daughter boards also provides flexibility and expandability for the Fleck-3 platform. The DSP chip is programmable in C which is widely supported.

Fleck-3 has a large 1 MB flash memory, which is sufficient for a real time operating system (RTOS). The combination of a real time clock (RTC), an integrated solar battery charging circuit, and regulator facilitates intelligent power management schemes. For example, RTC interrupts can be used based on the time of day to choose energy sources and even schedule storage or recharging. The circuit can also monitor battery and solar current and voltage. This makes the platform ideal for long-term outdoor use.

The camera board (in addition to the image sensor chip and lens holder) has two ultra-bright LEDs for capture illumination. The camera by itself is capable of acquiring a maximum of 30 VGA fps or 60 QVGA fps.

Disadvantages

Fleck-3 camera functionality requires both the camera and the DSP daughter boards. The DSP board alone draws a current of more than 290 mA when active. Taken together, this platform architecture is relatively power inefficient and has a costly component count.

Support and reusable software for the TinyOS are not as readily available as for other open source RTOSs, such as embedded Linux, uCos, and FreeRTOS. While the image sensor is capable of acquiring up to 60 QVGA fps, the camera can only stream compressed QVGA images at up to 2 fps [224], limiting its usefulness for a WVSNP.

Another Fleck-3 limitation is its use of a serial interface to a gateway computer to perform as a base node for network management. This is single point of failure, a bandwidth bottleneck for the network, and limits flexibility of the Fleck-3 network. The radio is very low data-rate and uses custom network access and radio management protocols. Taking advantage of open radio standards would likely reduce cost and improve compatibility with other WVSNPs.

2.6.6 University of Franche-Comte's ACME Fox based node [37]

While this node is very similar in its design as well as advantages and shortcomings to the preceding externally dependent architectures (and is therefore not included in Table 2.7), we briefly note its distinguishing features. This sensor node relies exclusively on a Bluetooth radio. This radio choice is an interesting attempt to strike the balance between a high-power 802.11 (WiFi) radio and a limited data rate 802.15.4 (Zigbee ready) radio with very low energy consumption. The node has also an energy analyzer module that reports current consumption. The energy analyzer helps in revealing an application's power consumption characteristics and enables designers to fine-tune operational algorithms.

2.7 Critique Summary

Of all the sensor node platforms reviewed in Sections 2.4 through 2.6, only few node platforms approach the architectural requirements required for WVSNP functionality. For instance, Imote2 and CITRIC approach WVSNP functionality provided the daughter boards are judiciously selected and the HW/SW is efficiently integrated. Unfortunately, Imote2 and CITRIC still suffer from the limitations of the externally dependent architecture category. The externally dependent platforms have architectures that are extensible and general enough as WVSNP candidates. However, they lack critical features, such as compression modules, high bandwidth wireless transmission, power mode flexibility, memory resources, and RTOS capability.

As noted in Tables 2.3 through 2.7, the wireless video capture and transmission capabilities of many implemented platforms have not been quantitatively evaluated and reported. None of the existing sensor node platforms has demonstrated the wireless transmission of more than 4 fps of CIF video.

The prevailing shortcoming of the existing platforms is that they have some image acquisition capability but lack the necessary HW/SW integration to achieve commensurate processing and wireless transmission speeds. In other words, the HW/SW integration and performance considerations have not been consistently examined across all major stages of the video acquisition, processing, and delivery path. Further, consistent attention to power management has been lacking. Other capable platforms are close sourced and lack the openness to be used for research and further modification for re-targetting. The open hardware movement is one factor that will lead to lowering the cost and barrier of entry for this exciting research of do-it-yourself (DIY) era of the Internet of Thigns (IoT).

Table 2.1: Part 1: Summary of classification categories for existing wireless video sensor node platforms (WVSNPs).

Architecture Categories	**General Purpose**	**Heavily Coupled**	**Externally Dependent**
Example Platforms	Stanford's *MeshEye* [90] and *WiSN Mote* [63], Portland State's *Panoptes* [71], Yale's *XYZ* [205, 49, 134], NIT-Hohai Node [70]	UC Irvine's *eCAM* and *WiSNAP* [155], UCLA's *Cyclops* [166], Philips' *Smart Camera Mote* [116, 115], CMU's *CMUcam3* [173]	CMU's *DSPCam* [109, 110] UC Berkeley's *Stargate* [221, 135, 43, 189, 122], Crossbow's *Imote2/Stargate 2* [147, 13, 135], UC's *CITRIC* [40], FU's *ScatterWeb* [176], CSIRO ICT's *FleckTM-3* [111, 224], UFranche's *Fox node* [37],
Identifying Features			
Objective	Catch-all approach. MCU centered. Many peripherals highlighting host MCU capabilities. High GPIO count.	Hardware designed to fit specific application. Highly customized. Special optimization of one of the acquisition, processing, or transmission stages but not the entire path.	Typically targeted for multi-tier networks. Modularized PCB approach. Main PCB with host MCU. Main PCB depends on external daughter boards for interfacing, power, and peripherals.
Flexibility	Flexible support for wide application range. Many interface options due to high peripheral count and GPIO count.	Very limited. Changing application requires hardware re-design. Few GPIO options.	Limited flexibility within its ecosystem of compatible daughter boards.
Extensibility	Most extensible. Standardized interfaces enable extensibility.	Very limited. Rarely accommodates a new application. Customized block to block interfacing.	Moderately extensible. Predetermined application options supported by the daughter boards.
Architecture	Similar to a PC. Medium to high MCU speed. Occasionally Co-processors. High memory and mass storage capability. Support for RTOS. Interface compatible with many imagers and radios. Assumes third-party functionality for acquisition and transmission.	Specialized hardware modules with sequential dependencies. High throughput modules offload processing from host MCU. Custom software required for external hardware block coordination. The stage-by-stage optimizations typically ignore integration of other sensor stages. Customized radio modules typical.	Medium to high speed MCU. Major application building blocks spread over daughter boards. Typical co-processor in a separate daughter board. Daughter boards customized to the host board's interfaces. High memory and mass storage options. Support for RTOS.
Performance	High performance depends on application's software design and use of available hardware.	High throughput hardware accelerator blocks. Emphasis on module image processing, filtering, and inference. Optimized custom radio protocols	Similar performance characteristics as general purpose platforms. Performance depends on the assembled parts and interboard communication.
Advantages	Most flexible. Most extensible. Potentially high performance. Enables quick application prototyping. Accepts many standardized peripheral interfaces.	Usually optimized for the target application. Saves power as there are few idle modules. Custom hardware usually faster than standard hardware.	Potentially many configurations with different daughter boards for desired functionality. Each daughter board can be separately optimized. Enables modularity of important sub-modules.

Table 2.2: PART 2: Summary of classification categories for existing wireless video sensor node platforms (WVSNPs).

Architecture Categories	**General Purpose**	**Heavily Coupled**	**Externally Dependent**
Example Platforms	Stanford's *MeshEye* [90] and *WiSN Mote* [63], Portland State's *Panoptes* [71], Yale's *XYZ* [205, 49, 134], NIT-Hohai Node [70]	UC Irvine's *eCAM* and *WiSNAP* [155], UCLA's *Cyclops* [166], Philips' *Smart Camera Mote* [116, 115], CMU's *CMUcam3* [173]	CMU's *DSPCam* [109, 110] UC Berkeley's *Stargate* [221, 135, 43, 189, 122], Crossbow's *Imote2/Stargate 2* [147, 13, 135], UC's *CITRIC* [40], FU's *ScatterWeb* [176], CSIRO ICT's *FleckTM-3* [111, 224], UFranche's *Fox node* [37],
Identifying Features			
Advantages	Most flexible. Most extensible. Potentially high performance. Enables quick application prototyping. Accepts many standardized peripheral interfaces.	Usually optimized for the target application. Saves power as there are few idle modules. Custom hardware usually faster than standard hardware.	Potentially many configurations with different daughter boards for desired functionality. Each daughter board can be separately optimized. Enables modularity of important sub-modules.
Limitations	No HW/SW integration codesign. Idle module functionality. Most functionality unused by most sensor applications. No multimedia optimization modules. Most expensive. Not necessarily suited for video processing. Transmission not accounted for in HW design. Over-reliance on standard interfaces	Not flexible. Not extensible. Costly re-designs needed for changes in application. All modules need to be active and coordinated for each task pipeline. Little opportunity for duty-cycle based power management. Few standardized modules lead to incompatibility with other sensors.	Main PCB board can rarely function stand-alone. Redundant basic PCB components on multiple daughter boards for power reliability. Overhead in coordinating daughter boards. Many idle modules within the daughter boards. Usually many boards needed for simple functionality.
Cost	Most expensive. Dollar cost proportional to System on Chip peripheral count and external interface module count.	Expensive. Hardware accelerators and hardware blocks add to the cost. Custom hardware is generally expensive.	Expensive. Daughter boards introduce hidden costs. Prices often quoted for the host MCU board only.
Power	Idle GPIOs consume high power. High clock rates proportionally costly. Unoptimized data access and transmission wasteful.	Low idle power loss. Limited power management options.	Power wasted on board to board overhead. Inter-board power management hard to implement and wasteful.

Table 2.3: Part 1: Summary comparison of general purpose platforms. Distinct characteristics of the WiSN mote with respect to the related MeshEye mote are given in brackets.

	Stanford's MeshEye and [WiSN] Motes	**Portland State's Panoptes**	**Yale's XYZ**	**NIT-Hohai Node**
Flexibility Rating	5.5/10 [6/10]	6/10	7/10	6.5/10
Processor(s), Core, Speed	Atmel AT91SAM7S (ARM7TDMI), 55 MHz, 32-bit	(PDA Platform) Intel StrongARM, 32-bit, 206 MHz, No Floating Point	OKI Semiconductor ML67Q500x, ARM7TDMI, 57.6 MHz, 32-bit	Intel PXA270 RISC core, 500 MHz, 32-bit
Node Power and Supply *(mW)*	DC input or AA cells [LTC3400 voltage reg. (1.8 V and 3.0 to 3.6 V)]	¡ 5000 mW, DC input	7 to 160 mW, 3×AA 1.2 V Ni-MH rechargeable cells, multiple voltage regulator	DC input
Supported Power Modes	Unknown	Suspend, Active	Halt, standby, deep sleep (30 μA)	Unknown
Node and Peripheral Power Manage-ment	In-built MCU power management (PM) controller, Software controlled phase locked loop (PLL)	Support for network wakeup/power mode	Power tracker, supervisor, SW controlled clock divider (57.6 to 1.8 MHz), most peripherals switch on/off	Unknown
Memory/ Storage	64 KB on-chip SRAM, 256 KB on-chip Flash, MMC/SD [2 MB off-chip Flash/32 KB FRAM]	64 MB	256 KB on-chip Flash, 32 KB on-chip RAM and 4 KB boot ROM, 2 MB off-chip SRAM	External SDRAM plus Flash (size undocumented)
I/O, Interface	USB2, UART, SPI, I2C	UART, SDLC, USB, Serial CCODEC, PCMCIA, IrDA, JTAG	SPI, I2C, 8-bit parallel port, and a DMA	USB2, UART, SPI, I2C, AC97, PCMCIA
Radio	TI CC2420 2.4 GHz Zigbee Ready	PCMCIA based 2.4 GHz (802.11b)	TI CC2420 2.4 GHz Zigbee Ready	Stand-alone 802.11
Wireless Trans. Rate	¡ 250 kbps	802.11b (¡ 11 Mbps),	¡ 250 kbps	802.11g (¡ 54 Mbps)
Imager, Max Imager Resolution, Max Frame Rate	8×ADNS-3060 (30 × 30/6-bit grayscale) and 1×ADCM-2700, VGA (640 × 480/24-bit) [2×ADCM-1670, CIF (640 × 480/24-bit) and 4×ADNS-3060 (30 × 30/6-bit)]	Logitech 3000 USB based video camera, VGA (15 fps),	Omnivision OV7649, VGA	USB based Webcam

Table 2.4: Part 2: Summary comparison of general purpose platforms. Distinct characteristics of the WiSN mote with respect to the related MeshEye mote are given in brackets.

	Stanford's MeshEye and [WiSN] Motes	**Portland State's Panoptes**	**Yale's XYZ**	**NIT-Hohai Node**
Capture-Save Frame Rate	3 fps [Not evaluated]	¡ 13 CIF fps	4.1 QVGA fps	¿ 15 QCIF fps
HW Image Processing	None	MCU Multimedia performance primitives	None	None
SW Image Processing	None	JPEG, Differential JPEG	None	H.263
Frame Trans. Rate	Not evaluated	Not evaluated	Not evaluated	10 to 15 QCIF fps
OS / RTOS	None	Linux (kernel 2.4.19)	SOS	modified Linux 2.4.19 core.
Cost	Unknown	Unknown	Unknown	Unknown

Table 2.5: Summary comparison of heavily coupled platforms.

	UC Irvine's eCAM and WiSNAP	**UCLA's Cyclops**	**Philips Smart Camera Mote**	**CMU's CMUCam3**
Flexibility Rating	6.5/10	5.5/10	4/10	3/10
Processor(s), Core, Speed	Nordic VLSI nRF24E1 (Eco mote)	Atmel 4 MHz 8-bit Atmega128, Xilinx XC2C256 CoolRunner	Xetal IC3D SIMD and Atmel 8051	NXP LPC2106 ARM7TDMI, 32.bit, 60 MHz, No Floating Point
Node Power and Supply *(mW)*	CR1225 Lithium battery, LTC3459 switching regul.	33 mW, 2×AA cells	100 mW (typical ICD3 only)	100mW, 4×AA, DC power
Supported Power Modes	None	active, power-save, or powerdown	None	Idle (125 mW), Active (650 mW)
Node and Peripheral Power Management	FDC6901 load switch, a power path switch	External block power mode control from host	None	Software controlled frequency scaling
Memory / Storage	In-MCU 512 byte RAM and 4 KB RAM, 32 KB external EEPROM	512 KB external Flash, 64 KB external SRAM	DPRAM, 1792 bytes (inside 8051), 64 KB RAM, 2 KB EEPROM	64 KB RAM, 128 KB Flash, Up to 2 GB MMC mass storage
I/O, Interface	UART, SPI	I2C, UART, SPI, PWM	UART, OTA 8051 programming	Very few GPIO, SPI, 2×UART, I2S
Radio	2.4 GHz RF transceiver, chip antenna, 10 m range	None (depends on attached Mica Mote)	Aquis Grain ZigBee (8051 and CC2420), 5 m range	None
Wireless Trans. Rate	¡ 1 Mbps	38.4 kbps	¡ 10 kbps	None
Imager, Max Imager Resolution, Max Frame Rate	Omnivision OV7640, 30 VGA fps, 60 QVGA fps; OPTEK OP591 optic sensor	Agilent ADCM-1700, CIF	2 VGA imagers	Omnivision VGA, OV6620 (26 fps) or OV7620 (50 fps), CIF (352x288)
Capture-Save Frame Rate	Not evaluated	2 fps	Not evaluated	¡ 5 fps (CIF)
HW Image Processing	On camera OV528 compression/serial-bridge chip	Xilinx XC2C256 CoolRunner CPLD	ICD3 Image Processor Arrays, Line Memories and Video I/O processor blocks	Averlogic AL4V8M440 (1 MB, 50 MHz) video FIFO
SW Image Processing	None	None	None	Frame differencing, JPEG, and PNG
Frame Trans. Rate	1.5 CIF fps	Not evaluated	Not evaluated	Not evaluated
OS/RTOS	None	TinyOS	Custom RTOS on 8051	None
Cost	Unknown	Unknown	Unknown	$250

Table 2.6: Part 1: Summary comparison of externally dependent platforms. Distinct features of CITRIC from the similar Imote2 are in brackets.

	UC Berkeley's Stargate	**Crossbow's Imote2/Stargate 2, [UC's CITRIC]**	**Freie Universitt's ScatterWeb**	**CSIRO ICT 's FleckTM-3**	**CMU's DSPCam**
Flexibility Rating	5/10	6/10	5/10	5/10	6/10
Processor(s), Core, Speed	XScale PXA255, 32-bit, 100 to 400 MHz, No Floating Point	XScale PXA271 SoC, 32-bit, 13 to 416 MHz, Intel Wireless MMX DSP Co-pr., No Float. Point [624 MHz, PXA270]	TI MSP430 (ESB430)	8 MHz 8-bit Atmega128, TI 32-bit DSP daughter b.	ADI 8 ADSP-BF537 Blackfin Processor, 600 MHz, DSP-MCU SoC
Node Power and Supply *(mW)*	170 to 400 mW, AA cells, DC input only via daughter board	231 mW, Liion / Li-Poly / 3×AAA NiMH / standard cells (via daughter board), via USB mini-B [428 mW (idle) to 970mW (active at 520 MHz), 4x AA, or USB, or 5V DC]	165 mW, 1 F gold-cap capacitors for energy harvesting (10 hours)	DSP daughter board (290 mA), AA cells Integrated solar charger	0.330 W (all idle) to 2.574 W (all active), 0.8 to 1.32 V, 3.3 V DC
Supported Power Modes	Sleep (107 mW), Active (1600 mW)	Deep Sleep (1.365 mW), Active Low Voltage (26.35 mW, 13 MHz), Active (231 mW, 416 MHz) [428 mW (idle) to 970mW (active at 520 MHz)]	Sleep (100 mW), Active (165 mW), Off	None	Active, idle, standby
Node and Peripheral Power Management	No support for network wake, battery monitoring utility	On PCB power management chip, frequency control from 13 MHz to 416 MHz with Dynamic Voltage Scaling [CPU speeds 208, 312, 416, and 520 MHz, External NXP-PCF50606 PMIC]	None	Board to daughter mode flexibility	Dynamic clock up to 600 MHz.
Memory / Storage	64 MB SDRAM, 32 MB Flash	256 KB in core SRAM, 32 MB in-SoC SDRAM, and in-SoC 32 MB Flash [16MB NOR FLASH eXecution-In-Place (XIP) and 64MB RAM external running at 1.8 V]	64 KB EEPROM (within camera)	128 KB on-chip, 1 MB external SRAM	32MB of SDRAM clocked up to 133MHz, and 4MB of Flash.
I/O, Interface	Ethernet, USB, UART, JTAG (on daughter board), PCMCIA, I2C	3×UART, 2×SPI, I2C, SDIO, I2S, AC97, Camera Chip Interface, JTAG, USB, Tmote Sky	UART, USB, I2C, OTA programming (via ScatterFlash board)	I2C, UART, SPI	USB, JTAG, Ethernet, PWM, UART, I2C, TWI, FireFly, SPI
Radio	PCMCIA or CF based 2.4 GHz (802.11b)	CC2420 802.15.4 radio and 2.4GHz antenna, 30 m range [Tmote Sky mote with 801.11.15 radio]	RFM TR1001 868 MHz, CC1021 434 MHz	Nordic NRF905	Stand alone 802.11b/g module, FireFly mote with 802.11.15.
Wireless Trans. Rate	802.11b (¡ 11 Mbps),	¡ 250 kbps	None	76.8 kbps	802.11g (¡ 54 Mbps),

Table 2.7: Part 2: Summary comparison of externally dependent platforms. Distinct features of CITRIC from the similar Imote2 are in brackets.

	UC Berkeley's Stargate	**Crossbow's Imote2/Stargate 2, [UC's CITRIC]**	**Freie Universitt's ScatterWeb**	**CSIRO ICT 's FleckTM-3**	**CMU's DSPCam**
Imager, Max Imager Resol., Max Frame Rate	Logitech Pro 4000 USB Webcam, VGA (640 × 480)	None [Omnivision 1.3 megapixel camera, OV9655, 15 SXGA (1280x1024) fps, 30 (640x480) VGA fps, CIF to 4x30]	COMedia C328-7640, VGA (640 × 480/16-bit)	Omnivision OV7640, VGA, 30 VGA fps, 60 QVGA fps	OmniVision OV9653 CMOS image sensor, VGA (640x480), SXGA (1280x1024), CIF.
Capture-Save Frame Rate	15 fps (CIF)	Not evaluated [OV9655, 15 SXGA (1280x1024) fps, 30 (640x480) VGA fps]	Not evaluated	Not evaluated	30 VGA fps and 15 SXGA fps.
HW Image Processing	None	MMX DSP with 30 media instructions for video [separate camera module]	In camera JPEG block	TMS320F2812 32-bit DSP	Parallel Peripheral Interface (PPI), DSP assisted SIMD, DMA
SW Image Processing	None	None. [JPEG, OpenCV]	None	None	JPEG
Frame Trans. Rate	Not evaluated	Not evaluated	0.75 to 6 fps (160 × 128/8-bit), 3–5 fr./min. (VGA)	2 QVGA fps	approx. 5 QVGA/CIF fps
OS / RTOS	Linux OS (kernel 2.4.19)	Linux, TinyOS, SOS or Microsoft .NET Micro	None	TinyOS	uCLinux
Cost	$500	$300 [Unknown, medium]	Unknown	Unknown	Unknown, medium

Chapter 3

AN IDEAL DESIGN OF A WVSNP

3.1 Flexi-WVSNP Design

3.1.1 Overview

As mentioned in introduction, a WVSNP that can be useful and still be relevant in the mordern era of cell phones, big data and the ever changing Internet of Things interfaces needs to be highly flexible, low power, lowly coupled low cost, highly cohesive and yet scalable to a large number of applications. This work focuses on a platform that consist of five major contributions: flexible hardware, maintainable and easily adaptable software image, user interface as defined by the WVSNP-DASH framework and configurability for different WVSN use cases. The preceding survey of the state of the art in video/image capable node platforms for wireless sensor networks revealed the need for a platform that is designed to incorporate acquisition, processing, and wireless transmission of multimedia signals. The sensor node should operate in practical application scenarios and with practically useful image resolution while satisfying the cost and resource constraints of a sensor node. In this section we outline a novel Flexi-WVSNP design to achieve these goals. We first provide the rationale for our major system design choices and then describe the hardware and software architecture.

3.1.2 Overall Flexi-WVSNP Design Concept and Architecture

We design *Flexi-WVSNP* as a video sensor node capable of wireless video streaming via both Zigbee and WiFi. Such a dual-radio system (*i*) integrates well with other

Zigbee sensors, and (*ii*) provides gateway access for the sensors to the Internet via WiFi.

As analyzed in the preceding work sections, most existing designs have the shortcoming of either attempting to incorporate too many components to cover an overly wide application range resulting in general purpose architectures, or attempting to be too specialized for a very narrow specific application resulting in heavily coupled architectures. In contrast, our design strives for high cohesion by meshing hardware and software architecture, while at the same time avoiding the tight binding (coupling) of components to each other as in the heavily coupled and externally dependent architectures. Our design strives to be highly adaptable and cost flexible; such that in its barest form, it may consist of only a processor. We believe that a WVSNP design needs to be application-targetable within a few days if it is to cover a wide array of cost-sensitive applications ranging from low-cost surveillance to remote instrument monitoring and conventional web camera.

Our generic WVSNP architecture follows a design concept that (*i*) eliminates the hard choices of anticipating a specific application scenario, and (*ii*) initially bypasses the tedious process of designing a comprehensive WVSNP. Our design concept is motivated by the basic fact that hardware and semiconductor processes will continue to improve and hence power savings will depend on the main components added for the specific application. This means that the major initial decision is the processor selection. The processor should be a powerful yet efficient System on a Chip (SoC) that satisfies essentially all requirements for a WVSNP in Section 2.2. Almost each module within the SoC should be able to be independently controlled from active power state all the way to off. The SoC needs direct hardware supported co-processor module capability and accelerators useful for video capture, encoding, and streaming. The remaining functionalities can be achieved by flexible connectors, e.g., the open general purpose input output (GPIO) ports of the main processor, to the video sensor and wireless modules.

Another major motivating factor for our design concept is that software is improving continuously and open source software, in particular, is evolving at an astonishing rate. This means that tying the design to existing software and hardware limits the system and violates the requirements for application adaptability, as well as low power and cost. A real time operating system (RTOS) is definitely necessary and it should only serve the purpose of booting up the initial master controller and allow loading modules as needed by the configuration. Depending on the configuration, modules should be able to control themselves if the master module is unable to (e.g., if the master crashes) or if application design prescribes that they should bypass the master under certain conditions, e.g. independent real time operation.

3.1.3 Middleware and the Multiple Radio Approach

The Flexi-WVSNP design strives to achieve cost effectiveness and flexibility through a robust middleware that delivers two major capabilities. First, the middleware introduces an operating system (OS) independent abstraction layer for inter-chip communication. This provides a semi-high level application programming interface (API) that enables the processing modules to communicate with each other regardless of the OS or underlying hardware interconnect.

Second, we employ the middleware for seamless and dynamic interchange of radios as required by data rates or data type. For example, if a small volume of temperature data is sent by a temperature sensor, the small data amount should automatically go out via the Zigbee radio and not the WiFi radio. If a large volume of video data is sent to a remote site via the Internet through the home router, the data should automatically go out through WiFi. If the video data is requested by some low resolution display that is Zigbee capable and is within the limit of the Zigbee data rates the video data should go out via the Zigbee radio. More generally, the middleware should switch between the two radios and control their rates such that the dual-radio appears as a single radio to the application. This transparent parallel WLAN-Zigbee radio software design enables a seamless operation and handover between small sensors and WLAN, Wi-Max, and/or Cellular devices. Thus, the dual-radio design enables the Flexi-WVSNP to function as a primary sensor, a relay within a Zigbee or WiFi network, or a gateway between Zigbee and WiFi networks.

Our design avoids a software-based solution of the radio control, which would demand memory space, execution time, and power from the host MCU. Instead, we exploit the increasing processing power and decreasing cost of radio SoCs. These radio SoCs can operate as separate and stand-alone wireless components. Our design only requires a simple middleware that allows the MCU to interface with and transparently use both radios. Each radio SoC operates its own network stack, QoS, and power saving mechanisms. This approach offloads the radio communication tasks (such as channel monitoring) and networking tasks (such as routing) from the host MCU and RTOS. Furthermore, the radios within the wireless modules can operate statemachines for channel monitoring without using their built-in firmware, which saves more power. The host MCU would still control the power modes of the attached radios.

Instead of a customized optimal radio, we prefer to employ proven standardized radio technologies that take advantage of multi-channel and spread-spectrum technologies. We intend to rely heavily on the Zigbee half of the dual radio for the main power and network management and coordination functions. This choice is motivated by the energy efficiency of Zigbee which can last years on an AA battery [10, 69], as well as a wide range of useful Zigbee mechanisms, including built-in scanning and reporting, which automatically selects the least-interference channel at initial network

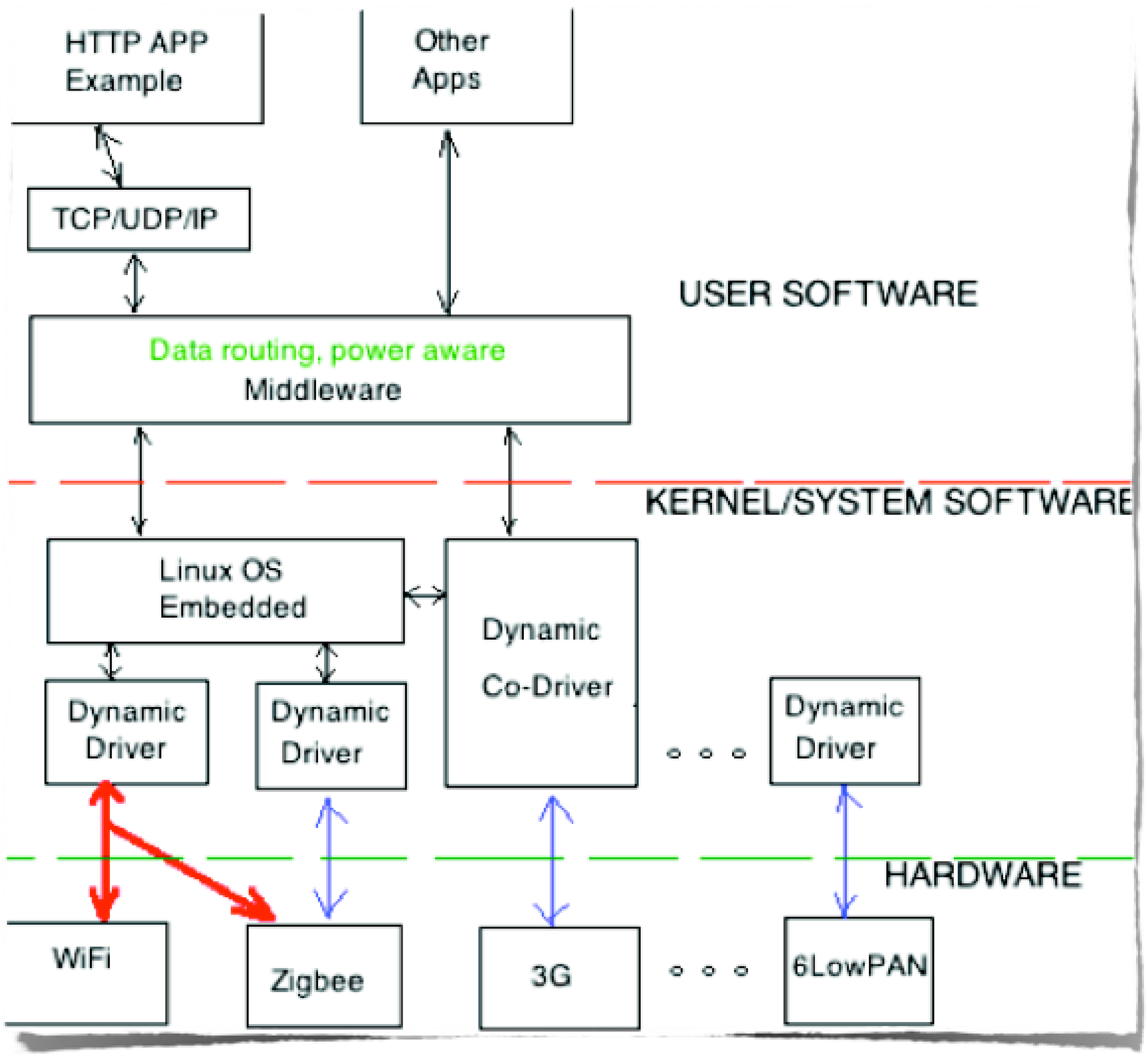


Figure 3.1: Flexi-WVSNP middleware architecture block diagram.

formation, as well as Zigbee mesh networking and path diversity features. In this context, we also envision to exploit recent miniature antenna techniques, e.g., [97], for efficient video sensor platform design.

A number of studies, e.g., [186, 209, 31, 246], have examined Zigbee-WiFi co-existence issues and concluded that typical WiFi usage patterns do not severely disrupt Zigbee. Even under very severe interference conditions, such as overlapping frequency channels and real-time video traffic, the transmission of Zigbee packets is not crippled, but may experience increased latency. The studies [246, 186, 209] have shown improved coexistence properties in later generations of WiFi, such as the 802.11g and 802.11n. This is explained by short on-air packet durations in and hence fewer interference and collision opportunities.

Zigbee is just being used as an available low cost off the shelf option. The node is designed to use whichever radio is preferred by the application as shown is Figure 3.1. With the proliferation of many radio technologies like Bluetooth Low Energy(BLE), 6LowPAN, Bluetooth Smart, CSR Mesh, Thread, we believe Flexi-WVSNP would be one of the only platforms which can easily adapt to whatever the industry gravitates toward as times change.

3.1.4 Powering the Flexi-WVSNP

Our architecture is designed for low power usage as well as to allow power management algorithms for a variety of applications. We therefore use the cheapest and readily available off-the-shelf battery technology, such as lithium-ion batteries. We propose to employ a multiple-output voltage regulator to disperse different voltage levels, as needed, to the various PCB components. As the WVSNP can also be used as a gateway node or continuous surveillance camera, we include an 110 V mains power supply. The node is also designed to be able to use an alternative power supply module for environmental harvesting like a solar module. See section on the node's current state.

3.1.5 Flexi-WVSNP Hardware Block Design

As shown in Figure 3.2, every component of the Flexi-WVSNP is connected to all other components. All functional pins of all components are exposed to the outside of the PCB via a flexible "mega" expansion port. This design is inspired by CPU

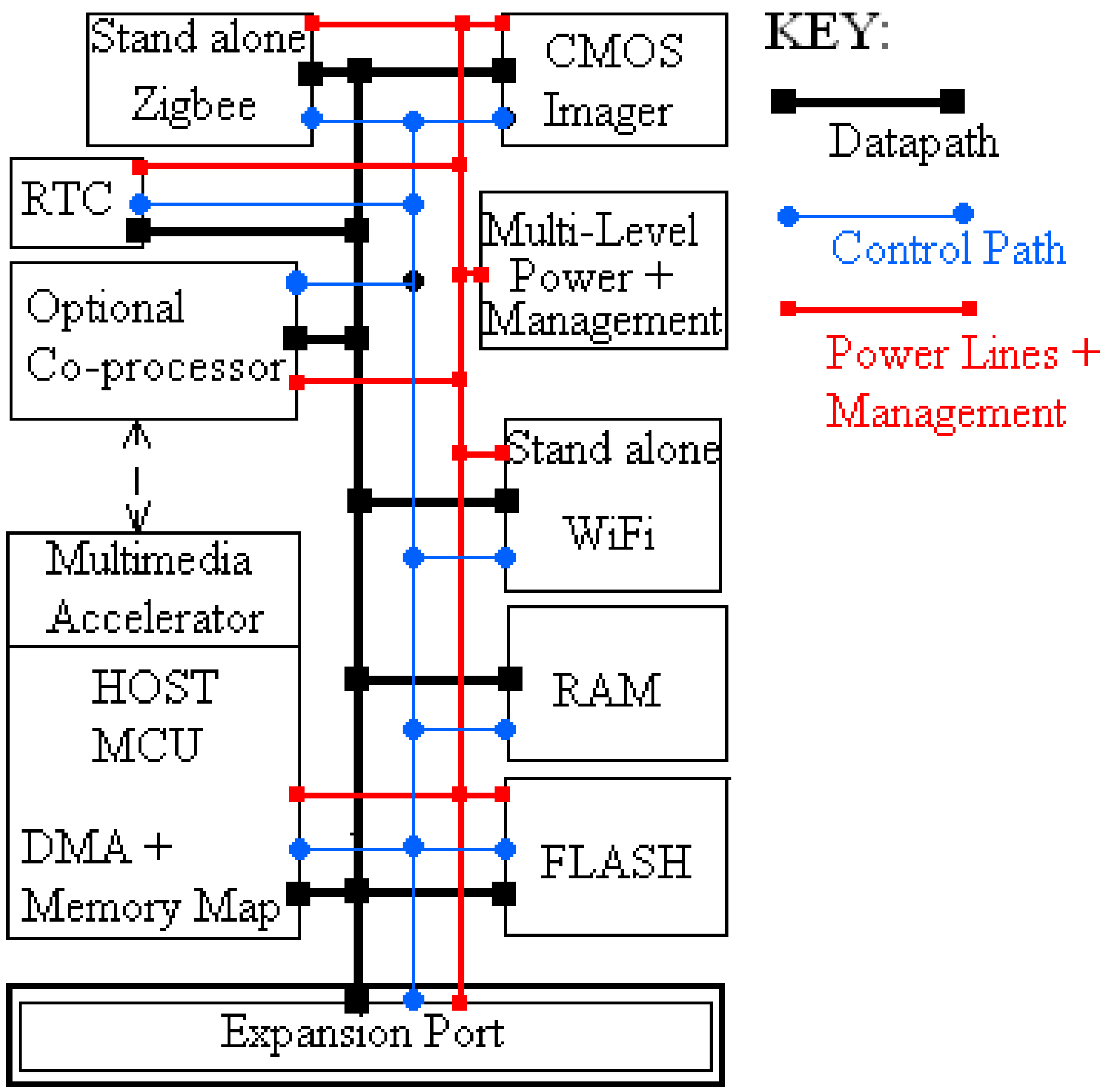


Figure 3.2: Flexi-WVSNP hardware architecture block diagram.

integrated circuits (IC) architectures and VLSI design concepts, which until now were only used within an IC, not outside it.

The Flexi-WVSNP requires a SoC with multimedia capabilities. Multimedia capabilities can be an MCU specific extended multimedia instruction set, or hardware compression and processing engines. Additionally, the SoC requires a built-in co-processor interface for high throughput cooperation with an attached module, such as a DSP or other dedicated video processing module. A Flexi-WVSNP SoC requires a direct memory access (DMA) sub-module and/or a memory management unit (MMU) sub-module to enable transparent data accesses and exchanges between the PCB modules without continuous MCU coordination.

Importantly the Flexi-WVSNP SoC needs to be a modern power managed chip with almost total control of the power modes of its sub-modules. In addition, dynamic voltage scaling can be employed to save power. The overall clock speed of the MCU should be tunable over a wide range of sub speeds via software control. The focus on power variability is critical as large power savings in an application is achieved through power aware algorithms [16].

The visibility of the control path signals to all major modules of the PCB enable all modules to participate in power management control, which can be software initiated or externally managed by a power management module. The individual modules should be selected based on their ability to support a wide range of the power modes supported by the SoC.

Unlike most existing designs, the Flexi-WVSNP wireless modules are stand-alone modules that incorporate the necessary protocol stacks and power management options. The SoC and other modules on the PCB view the wireless modules as available data channels. The control path still exposes the wireless modules to further control or interventions by the SoC if needed, especially by power management algorithms or configuration commands.

CMOS imagers have become very popular due to their low cost and increased in-chip processing of the acquired image [53, 75]. The Flexi-WVSNP imager module is intended to be mainly stand alone in capturing a VGA image with possible resolution, zoom, sampling, and color space selection commands issued by the SoC. Although an advanced module may have a built-in compression module, we believe that compression is more efficient on computationally advanced modules, such as the SoC and/or coprocessor. This is because high level inference algorithms can be used to decide when it is necessary to compress frames. Compression on the SoC or coprocessor also provides for flexibility of compression schemes as better algorithms are developed.

The RTC maintains temporal accuracy of the system and in conjunction with the power module (see Section 3.1.4) can be used to implement time triggered duty cycles or power states for any module in the control path.

The direct access of the modules to memory resources through the use of DMA allows the imager to operate at camera frame speeds. As observed in Sections 2.4 through 2.6 most existing platforms have high frame rate imagers but are limited by a "capture to pre processing-storage" bottleneck.

3.1.6 Flexi-WVSNP Software Architecture

Figure 3.3, shows the Flexi-WVSNP software architecture. The modular design of the architecture enhances power efficiency by enabling each sub-component of the platform to be powered and controlled individually as well as allowing applications direct and fine-grained access and control of the hardware. The architecture satisfies the WVSNP expectation of a decoupled but highly cohesive platform. This is an advantage over traditionally stacked or layered architectures whose components suffer from layer dependencies and power inefficiency.

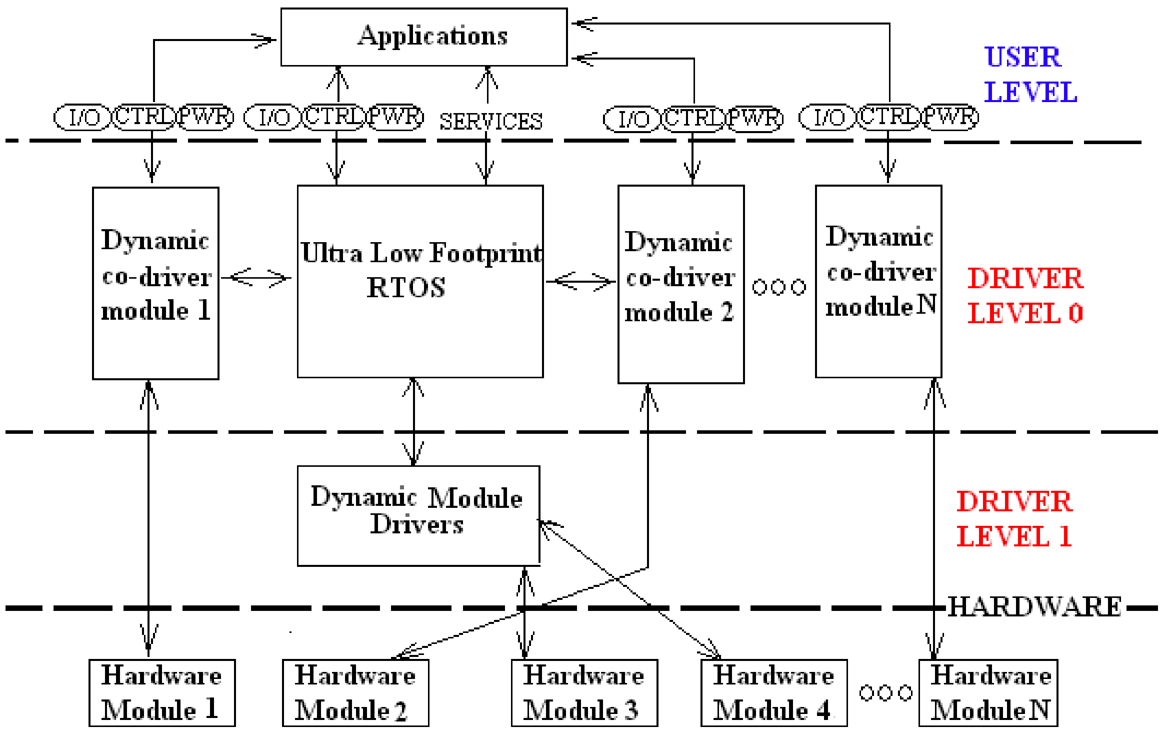


Figure 3.3: Flexi-WVSNP software architecture.

As shown in Figure 3.3, applications are aware of the hardware modules and treat them as I/O data channels with control parameters (CTRL) and power states (PWR). This ensures that the relationship between the hardware modules and applications is data centric and enables application algorithms to be power aware. The real time operating system (RTOS) is the main scheduler for applications and driver modules.

An important feature introduced in Flexi-WVSNP is the ability for some trusted drivers to be at the same priority and capability level as the RTOS. As shown in Figure 3.3, the dynamic co-driver modules (DyCoMs) are those special drivers that at initialization load as normal drivers but then acquire full control of hardware and can run independently of the RTOS. Middleware, such as the transparent dual radio (see Section 3.1.3), is implemented as DyCoMs. Thus, when the RTOS crashes, some critical applications continue to function for a graceful exit or intervention.

Each hardware driver provides a three part bidirectional generic interface to an application, that is, I/O, control, and power states. This enables uniform use of the hardware architecture. Traditional RTOS drivers can still be used as shown for example for Driver Level 1 controlling hardware modules 3 and 4 in Figure 3.3. This facilitates the low cost reusability advantage of popular RTOSs, such as the Linux based RTOSs.

Ensuring that the DyCoMs and the rest of the drivers are dynamic and have a direct relationship with the hardware modules enables the Flexi-WVSNP software architecture to closely match the hardware architecture. Adding or removing a hardware module is directly related to adding or removing a software module. For example, exchanging the imager only requires unloading the old imager driver module, loading the new driver module together with the new hardware. The "software module" to "hardware module" match in the Flexi-WVSNP design further enables design time PCB software simulation, which enables high flexibility in component choices and hence low system cost. Moreover, forcing modules to follow the three part (I/O, CTRL, and PWR) bidirectional generic interface with an application reduces the maintenance cost, improves upgradeability, and enables power sensitive operation.

We expect the Flexi-WVSNP to deliver real-time frame rates via WiFi of between 15 and 30 VGA fps. This assumes an average WiFi data rate of around 1.5 Mbps using H.264 SVC compression [58, 230]. We expect to deliver between 15 and 30 CIF fps via Zigbee transmission. This assumes an average Zigbee data rate of 250 kbps with H.264 SVC compression. Since the management of the Flexi-WVSNP network is done primarily with Zigbee, we expect Flexi-WVSNP to last months to a year with 4 AA batteries for an application with a low frequency of events requiring video streaming.

Chapter 4

AN INTERFACE TO THE WORLD VIA WVSNP-DASH

4.1 Framework Introduction

As mentioned before. The hardware you have and the software you have on the node is relatively irrelevant if you have no framework of getting that data from the physical world to the eyes and displays of the consumer or data analyst. It is therefore, very important that a major portion of the platform be invested in how data moves through from capture to storage to transmission and eventually application usage.

4.1.1 Motivation

Wireless sensor nodes collect data that can support a wide range of services on today's consumer electronic devices. For instance, wireless sensor nodes can support surveillance and security applications [15, 22, 45, 242]. A number of platforms and gateways have been introduced to make the sensor data readily accessible over the Internet and to enable interactions between networked sensors and consumer devices [100, 193, 208].

Video has been emerging as a particularly important type of sensor data as consumer electronics with video displays, such as smart phones, are becoming ubiquitous. Video data from a wide range of video sensors has the potential to enhance a variety of entertainment, residential, and industrial use cases of wireless sensor networks. Additionally, general multi-dimensional (2D and 3D) data, e.g., from infrared sensors, heat maps, Light Emitting Diode (LED) pixel sensor maps, x-rays, and many other wirelessly linked sensor nodes and remote acquisition devices are becoming ubiquitous. A wide range of consumer devices with web browsers should ideally support the smooth continuous playback of sensed video and general multi-dimensional data without specialized software or protocols for the different data types. Consumers want to flexibly request data and view time periods of interest in the video or data on their devices. Preferably, the video and data should be viewable in a seamless manner, as consumers switch around device platforms and operating systems during their typical day in between home, work, and leisure.

The cross-platform video use on consumer devices is addressed by version five of the Hypertext Markup Language (HTML5) and the emerging Dynamic Adaptive Streaming over HTTP (DASH) specification [11, 206]. However, as reviewed in detail in Section 4.2, HTML5 does not support adaptive real-time video playback. Also, all existing DASH players can only interact with video server nodes operating the TCP/IP networking protocol stack. That is, none of the existing players are designed to interact with the popular non-TCP/IP protocol stacks on resource-constrained sensor (server) nodes, such as Zigbee [13, 8, 180]. Moreover, existing DASH players require complex plug-ins that invite security vulnerabilities or have very limited cross-platform support.

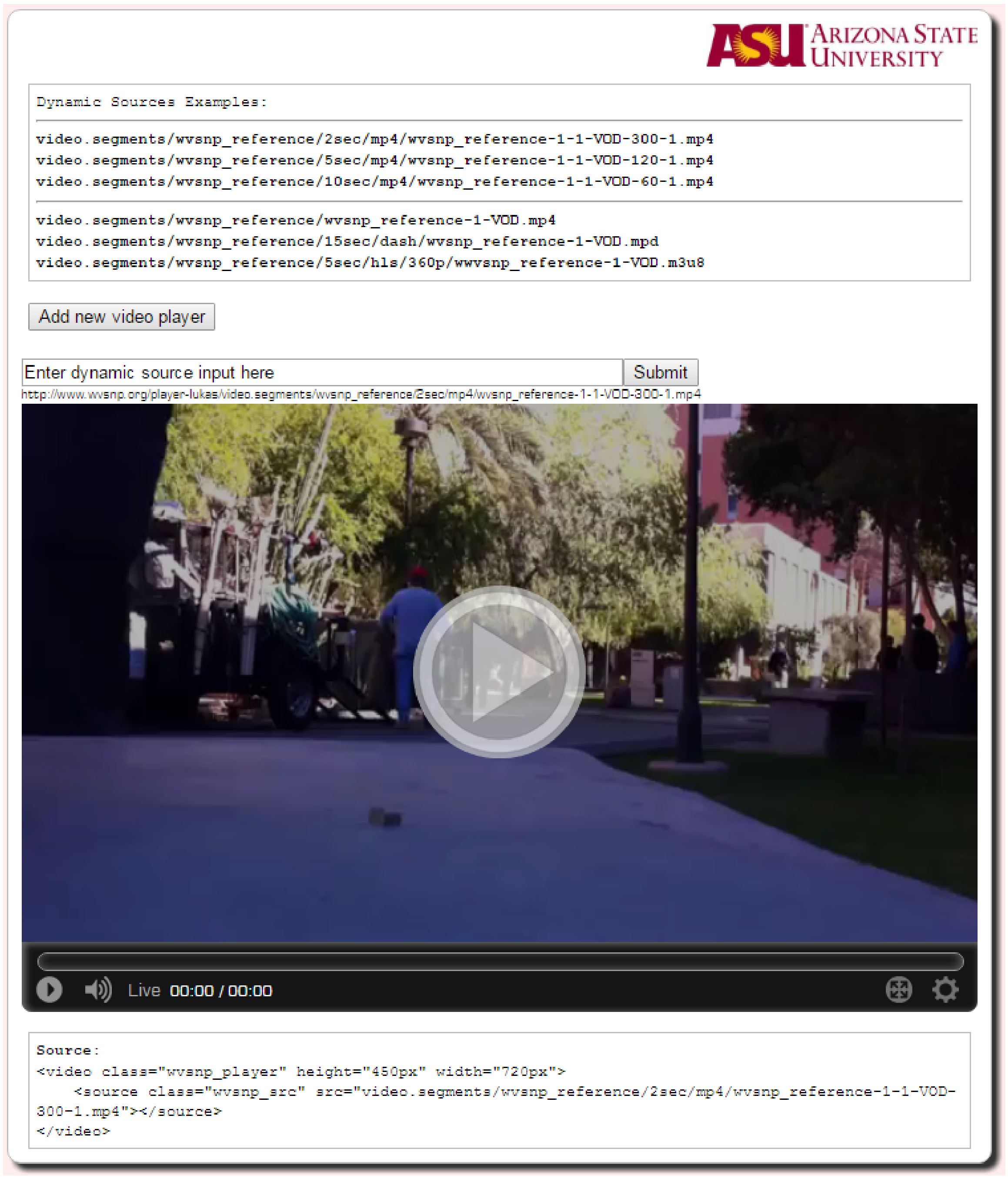
An important structural limitation of existing web-based video streaming frameworks, such as the HyperText Transfer Protocol Live Streaming (HLS) and the Motion Picture Experts Group Dynamic Adaptive Streaming over HTTP (MPEG-DASH), is that individual video segments cannot be independently distributed and played. Instead, a video segment can only be interpreted and played with reference to a manifest file (and typically a special initialization video segment). These dependencies complicate the video data management on resource-constrained video sensor nodes as well as the distribution of video segment files through sensor networks, e.g., video segment files cannot be independently cached and distributed by storage-constrained sensor nodes.

4.1.2 Contributions

This chapter introduces the Wireless Video Sensor Node Platform (WVSNP, may be pronounced “WaveSnap”) compatible DASH framework, abbreviated as WVSNP-DASH. Existing web-based video streaming frameworks rely on special manifest files and initialization segments to convey the video meta information. These manifest files require special segment generation tools and create dependencies between the segments of a given video stream, complicating video data management and distribution and leading to compatibility issues. In contrast, each video segment in the proposed WVSNP-DASH framework is an *independently playable* file carrying its essential meta data in its name. The proposed WVSNP-DASH framework includes a specific name syntax for video segments. The name syntax conveys essential meta information about the video segment; thus eliminating dependencies to manifest (and initialization) files. The proposed WVSNP-DASH framework is highly backward compatible and interfaces with wireless sensor networks (WSNs) without special re-design of video file formats, video containers, sensor nodes, or networks. WVSNP-DASH is video container agnostic and encapsulates any container, codec, or Digital Rights Management (DRM), as long as the web browser supports it.

This chapter also presents the design of a WVSNP-DASH Player (WDP) that is based on core elements of HTML5. See Figure 4.1below.

The WDP provides a user interface to the WVSNP-DASH framework by allowing consumers to retrieve and play video from sensor nodes. WDP does not rely on any plug-in or back-end engines. Instead, WDP employs elementary downloading through HTTP as well as the HTML5 Filesystem (FS) together with the HTML5 video tag for managing video segment retrieval, transmission, and playback. The



```
Dynamic Sources Examples:

video.segments/wvsnp_reference/2sec/mp4/wvsnp_reference-1-1-VOD-300-1.mp4
video.segments/wvsnp_reference/5sec/mp4/wvsnp_reference-1-1-VOD-120-1.mp4
video.segments/wvsnp_reference/10sec/mp4/wvsnp_reference-1-1-VOD-60-1.mp4

video.segments/wvsnp_reference/wvsnp_reference-1-VOD.mp4
video.segments/wvsnp_reference/15sec/dash/wvsnp_reference-1-VOD.mpd
video.segments/wvsnp_reference/5sec/hls/360p/wwvsnp_reference-1-VOD.m3u8
```

http://www.wvsnp.org/player-lukas/video.segments/wvsnp_reference/2sec/mp4/wvsnp_reference-1-1-VOD-300-1.mp4

```
Source:
<video class="wvsnp_player" height="450px" width="720px">
    <source class="wvsnp_src" src="video.segments/wvsnp_reference/2sec/mp4/wvsnp_reference-1-1-VOD-
300-1.mp4"></source>
</video>
```

Figure 4.1: The WVSNP-DASH Player.

video segment fetching into the HTML5 FS enables video segment delivery from non-TCP/IP networks, such as Zigbee networks. The video segments are displayed on the HTML5 canvas element.

WDP includes a module for compatibility checking so as to function on a wide range of platforms feature improves backward compatibility as legacy video can still be played via the existing basic HTML5 video tag.

The WVSNP-DASH framework is evaluated with a WDP prototype that is compared with optimized HLS and MPEG-DASH framework players. To the best of the authors' knowledge, this is the first evaluation of DASH streaming of video from sensor nodes. This study thus provides empirical baseline performance data for the streaming of sensor video with different frameworks.

One of the biggest obstacles is that the HTML5 `<video>` tag is not uniformly nor completely implemented by browsers. Even if the video tag accepts the same video container extension, the codecs within the container might no be accepted. The Safari browser, for example, accepts the HLS manifest file name and extension in its `<video>` tag. No other browser accepts any manifest file name as a video source. This limitation means that JavaScript and Flash based players need to implement work arounds and fall backs if the video tag fails to open, play or render the `<video>` source. To implement DASH playback, it is therefore necessary to execute JavaScript or other means of detecting and pre-processing the video or the input manifest file before exposing the raw video data to a specific browser's `<video>` tag. Additionally the player needs to detect and react to other HTML5 features that might not be implemented by the browser, such as the Canvas, FileSystem (FS), Blob objects and others.

4.2 Background and Related Work

4.2.1 HTML5 Video Playback

Playback of video is supported by most modern web browsers as a feature of HTML5 [215, 219]. Generally, HTML5 based video players utilize the HTML5 video tag to download and play full-length video files. However, the HTML5 `<video>` tag is not uniformly nor completely implemented by common web browsers, requiring players to implement work-arounds if the video tag fails to open, play, or render the `<video>` source.

Also, with HTML5, adaptation of the video bit rate or presentation quality would require the re-download of the entire video file. Media Source Extensions (MSEs), which have been developed by the World Wide Web Consortium (W3C) [227], could serve as a basis for adaptive streaming in HTML5 based video players. Early in the platform development MSE support was inconsistent in popular web browsers, limiting the cross-browser compatibility of an MSE based player design. Fortunately this has changed and as will be detailed later in this document, an MSE version of the player client has been developed to replace the now faltering cross platform support for the HTML5 FS across browsers.

Alternatively, web browser video plug-ins, e.g., [146, 67, 234], could adapt the video streaming through slicing the full-length video file and monitoring the download of the different presentation qualities (versions). However, such plug-ins can give rise to a multitude of security and incompatibility issues as well as the burden on the user to update and maintain the plug-ins [157, 171].

Zhu et al. [245] recently designed a pure HTML5 based video player that uses the canvas element to display video in different web browsers. The player circumvents the problem of accommodating different video codecs by decoding the received video chunks using JavaScript in the browser into an intermediate format. The intermediate format is then passed to the browser's video tag for decoding with the native video decoder of the browser. However, performing both the video decoding to an intermediate format and the video drawing on the canvas in the browser leads to very high CPU load, which is prohibitive for mobile devices.

Though the proof of concept WDP designs use either HTML5 or MSE, these are not required. It is relatively easy to add a filter or create retriever client applications using popular libraries such as VLC [161], ffmpeg [6] or gstreamer [80]. This could be just an extra switch in a VLC application, for example, to fetch files based on WVSNP-DASH framework syntax. More importantly, within browsers, the WVSNP-DASH framework requires no plug-ins; instead, WDP relies only on the legacy single video tag reference. Also, WDP does not use any extra decoder module; instead, WDP uses the decoder engine native to the browser, resulting in no extra CPU load for the video decoding.

4.2.2 DASH Video Streaming and Playback

According to the Dynamic Adaptive Streaming over HTTP (DASH) specification [5, 195], a web-based DASH video player must be able to adapt the video presentation quality level during playback by switching among different quality versions of the video stream. A plug-in free player must support the playback of chunks (segments) derived from a video stream via an HTML5 video element. DASH players support interactions, such as jump backward (rewind, RW) or forward (fast forward, FF) by fetching the video segment for the desired playback point. However, existing DASH players must first process the manifest file and play the initialization segment before such playback jumps. The WDP does not require an initialization segment and thus provides truly random segment playback on demand.

Quacchio et al. [164] proposed a DASH player that utilizes a custom-built Web-kit based browser to achieve customized handling of the HTML5 video elements. The proposed player design relies heavily on plug-ins to achieve adaptive streaming.

In order to facilitate the adoption of the MPEG-DASH standard [5], the DASH Industry Forum developed a reference player, namely dash.js [54]. The dash.js player employs media source extensions (MSEs). Several derivatives of the dash.js player, e.g. [126, 146], have recently been proposed. These derivatives require specific web browsers and extensive plug-in support. Similarly, there are a variety of DASH video players available that work only in conjunction with specific web browsers and require plug-ins or rely on the Real Time Messaging Protocol running over TCP/IP.

Importantly, a thorough literature search and examination of a wide range of available proprietary player solutions revealed that none of the existing video players is designed to integrate with non-TCP/IP networks, such as resource-constrained WSN employing the Zigbee protocol. Overall, these recent DASH player developments do not comply with the cross-platform design goal of HTML5, instead they are limited to specific web browsers with their respective plug-ins. In contrast to the recent DASH player developments, the goal of the proposed WVSNP-DASH framework is to make video data from sensor networks as widely accessible as possible with little effort, if any, from the consumer device user.

We note for completeness that recent research has sought to provide additional features, such as subtitles [47], in DASH video streaming as well as examined the implications of the adaptive DASH streaming on network resource requirements [121, 222]. Other complementary related research has sought to optimize the video encoding for DASH streaming [11] and improve buffer management algorithms and segment scheduling [244, 107].

4.2.3 Video Sensor Networking

There has been a recent focus on the lower layers of the Internet protocol stack for integrating sensor networks into the overall Internet of Things (IoT) [48]. A widely considered approach is IP version 6 based Low power Wireless Personal Area Networks (6LoWPAN), which covers radios and firmware that compress IPv6 packet headers into smaller 6LoWPAN headers suitable for low-data-rate IEEE 802.15.4 personal area network standards, such as Zigbee, operating in sensor networks [143, 99, 185, 125, 98, 201, 233, 76, 148]. The proposed WVSNP-DASH framework and WDP are designed to directly work within these low-data-rate mesh networks.

Sensor focused services may be viewed as components in service based frameworks [218] to serve as a cloud-based repository directory of sensor data using service oriented architectures (SOA), infrastructures, and protocols [51, 145, 216, 199]. However, video sensors are typically not designed to take advantage of SOA data exchange structures. Additionally, video as a data element is rarely if ever mentioned in the IoT consumer literature. The proposed WVSNP-DASH framework enables video sensors to seamlessly form part of the IoT by exploiting the mechanics and design of DASH enabling architectures. This chapter shows that, there are sensor specific benefits to adopting DASH to a WVSNP design. These include video storage, play back simplification (random network wide seek), low power, video as needed and high adaptability of segmented video to wireless delivery and to the necessary duty cycling algorithms that are the staple of sensor networks. For example, the storage format of segmented video can even be used beyond just for streaming but as part of data points for a service like Pachube and others [3, 2]. In particular, sensor data cloud repositories presently have no concept of video as search-able or addressable sensor data. The name syntax of the proposed WVSNP-DASH framework enables cloud repositories to offer video segments as part of data sets and services.

A multimedia playback framework based on MPEG-DASH within an information centric network has recently been proposed in [59, 60]. The framework in [60] encompasses named addressing and routing within the context of TCP/IP networks. The name-based structure of the proposed WVSNP-DASH framework is complementary to the framework in [59, 60] and compatible with information centric networking. At the same time, the proposed WVSNP-DASH framework is designed to work beyond TCP/IP networks so that it makes the video data on sensor nodes employing other (non-TCP/IP) network protocols readily available to consumer electronics.

4.3 WVSNP-DASH Framework

4.3.1 Independent Video Segments

The WVSNP-DASH framework facilitates multimedia acquisition, storage, distribution, and playback through assigning a unique name to a given independently playable multimedia object addressable by the WVSNP-DASH node. If a node has a WiFi, Zigbee, or Bluetooth radio, a video object source can be uniquely named for WDP to be able to fetch the object. This makes the video object accessibility limited only by the radios available to the WDP client. This WVSNP-DASH design of independently playable media objects with a specific naming syntax enables video data object distribution across networks, including cross-network data transfers between traditional IP network and sensor networks.

All WVSNP-DASH video segments have the same type and format; in particular, each segment is a complete video file. The sensor (server) node only processes the video file data when creating the video file. A created video file is then always ready to be fetched and transmitted without any further pre-processing on the sensor (server) node. This approach reduces sensor (server) node power consumption compared to the existing HLS and MPEG-DASH frameworks, see Section 4.5.4.

The WVSNP-DASH framework is conceptually similar to the MSE-based MPEG-DASH [54] in that it avoids requiring the browser to support DASH directly via the video element. MSE only exposes the HTML5 media element to its interface, which then allows flexible appending of media segments to this exposed element. The browser continues to perform the traditional decoding and rendering. The intelligence required to parse manifest files, request segments, and switch adaptively is left to the player client script. This is somewhat of an improvement over HLS in that HLS requires browsers to be re-written and the video tag to be modified to support manifest files; MSE avoids these modifications by preprocessing the stream and passing video chunks to the video tag as if each chunk was a traditional HTML5 supported video file. WDP further simplifies this concept by not even expecting multimedia data from the server to have been specially formatted, as MPEG-DASH and HLS do. WVSNP-DASH does not reinvent multimedia files nor require extra manifest files to handle the new file formats.

4.3.2 WVSNP-DASH Segment Name Syntax

The WVSNP-DASH framework prescribes a video segment naming syntax that uniquely names each video segment. The syntax of the segment name has complete information for a WVSNP-DASH player (WDP) to be able to play back the video files stored in a remote network node. The client should be able to playback an entire video-on-demand (VOD) set or live video based solely on the meta information gleaned from parsing the segment name.

The WVSNP-DASH segment naming syntax follows the simplified Backus-Naur Form [117] `<filename>-<maxpresentation>-<presentation>`

`-<mode>-<maxindex>-<index>.<ext>`

The components of the name syntax are defined as follows:

- `<filename>`: This is a unique string for each video stream (set of video segments). The ¡filename¿ represents the path, e.g., through an IP address or a URL, to the video stream, or represents a unique prefix describing the stream.

- `<maxpresentation>`: This integer defines the index of the highest presentation quality (e.g., quality version) available for the stream.

- `<presentation>`: The actual presentation quality of this video-segment file. A lower index defines a lower quality of the stream, whereby 0 is defined as the lowest index denoting the lowest available video quality.

- `<mode>`: This string indicates the playback mode of this segment, e.g., video on demand (`VOD`) or live playback (`LIVE`).

- `<maxindex>`: This integer gives the total number of segments available for playback for the current video stream.

- `<index>`: This integer gives the index of this segment within a finite set of segments of this stream.

- `<ext>`: This string indicates the video container format of this segment, e.g., `.mp4` or `.webm`. The player decides if the container format and encoded video can be played back and informs the user accordingly.

This simple WVSNP-DASH video segment name syntax contains the complete information needed by the WVSNP-DASH client to retrieve and play the video segments in a sensor network.

4.3.3 Implications

The simplicity of this segment name syntax has far-reaching implications as it transfers most of the video playback and retrieval complexity to the player. The player is the consumer of the video data, and thus knows what it wants to do with the video data and how to interpret the video data. Player specific details are also in [178, 182].

The WDP introduced in this chapter demonstrates the concept of the uniqueness of the video segment's name. Since WSNs are resource constrained in terms of power, computing resources, and storage space, WVSNP-DASH enforces that video files within a sensor network are captured and stored as complete, individually playable video files of short duration, preferably no longer than ten seconds (the impact of the segments length is examined in Section 4.5.4).

Recall that with HLS and MPEG-DASH, the player requires the manifest file in conjunction with the individual video segment files for playback. In contrast, the WVSNP-DASH video segment name syntax ensures that the name of the segment, or any other future network video object, conveys sufficient information for the player to decide how to fetch and play the video segments. The manifest files required by HLS and MPEG-DASH introduce incompatibility issues as they require browsers to support manifest files as well as live playback maintenance of the manifest files. By communicating all player pertinent meta data through the segment file name, WVSNP-DASH avoids these incompatibility issues and is thus highly backward compatible.

Another important feature of the WVSNP-DASH framework is random playback of any segment, as needed by the player. In particular, WVSNP-DASH has no special initialization segment, which is necessary for HLS and MPEG-DASH players to understand how to play a video. Each WVSNP-DASH video segment has enough meta data (in its name) to start playing the video. This feature enables unlimited "crawler type" network search algorithms. These can be used by future information centric network (ICN) routers and content distribution networks (CDNs). Future active players may start playing the video as soon as a segment is discovered.

For instance, as detailed in Section 4.4.2, a segment name:

`src=filename-1-1-VOD-45-7.mp4` can be easily passed to the player. Based on the meta information contained in this name, the player can initiate the streaming process of segments 7 to 45 by fetching all segments and playing them one after another. The random playback feature of WVSNP-DASH enables the player to perform arbitrary fast forward (FF) and rewind (RW) of the video. The naming syntax of WVSNP-DASH also enables the player to take advantage of all DASH features, such as adaptive switching.

4.4 WVSNP-DASH Player (WDP)

4.4.1 Design and Implementation

The WDP design goal is to rely only on widely supported HTLM5 features, so as to achieve broad cross-platform support. Selecting HTML5 features that are reliably supported by most browsers is generally difficult. Therefore, the first prototype version of WDP is constrained to using only official core HTML5 features. WDP combines the advantages of common support for HTTP downloading via Asynchronous JavaScript and XML (AJAX) (through the XMLHttpRequest functionality) [157] as well as the HTML5 File System (FS) Application Programmers Interface (API) [24]. The HTML5 FS is run-time memory allocated to a process running on a browser. Data objects in this protected (sandbox) space can be cached for future use by the same URL on the same browser. Other URLs in separate tabs or other browser windows cannot access this memory space.

Both AJAX and the FS API are used as they are broadly supported HTML5 features [157, 24]. Where the FS API is not yet fully supported, third party wrappers such as IndexedDB [46] are used to mimic the FS API. This approach helps to keep the design logic intact across browsers as well as to maintain compatibility as official support improves. For example, Safari in iOS and Mac OSX support neither the FS API nor IndexedDB, however, another wrapper can utilize IndexedDB using WebSQL [21], which is supported by Safari for Mac OSX and iOS.

4.4.2 Flow

Compatibility Test

The high level segment playback by the client is depicted in Figure 4.3.

The entry point of the player is the browser detection and compatibility test module illustrated Figure 4.2. This module tests if the browser supports the core HTML5 features used by WDP. AJAX tools check support for downloading, saving, and playing back requested video via FS API or a suitable wrapper. If an MPEG-DASH manifest file is provided, and the browser supports MSE, the DASH-JS player is launched for playback as an independent module within the WDP modular architecture. Alternatively, for an HLS manifest file, the native video player of the web browser is set up with HLS. If the URL/name of a requested video does not match the WVSNP-DASH video segment name syntax, and neither the HLS or MPEG-DASH manifest syntax, WDP assumes a traditional HTML5 compatible video file. The HTML5 video file is then played back in the browser's native HTML5 video element as legacy full-length video. Otherwise, the default is to play WVSNP-DASH video.

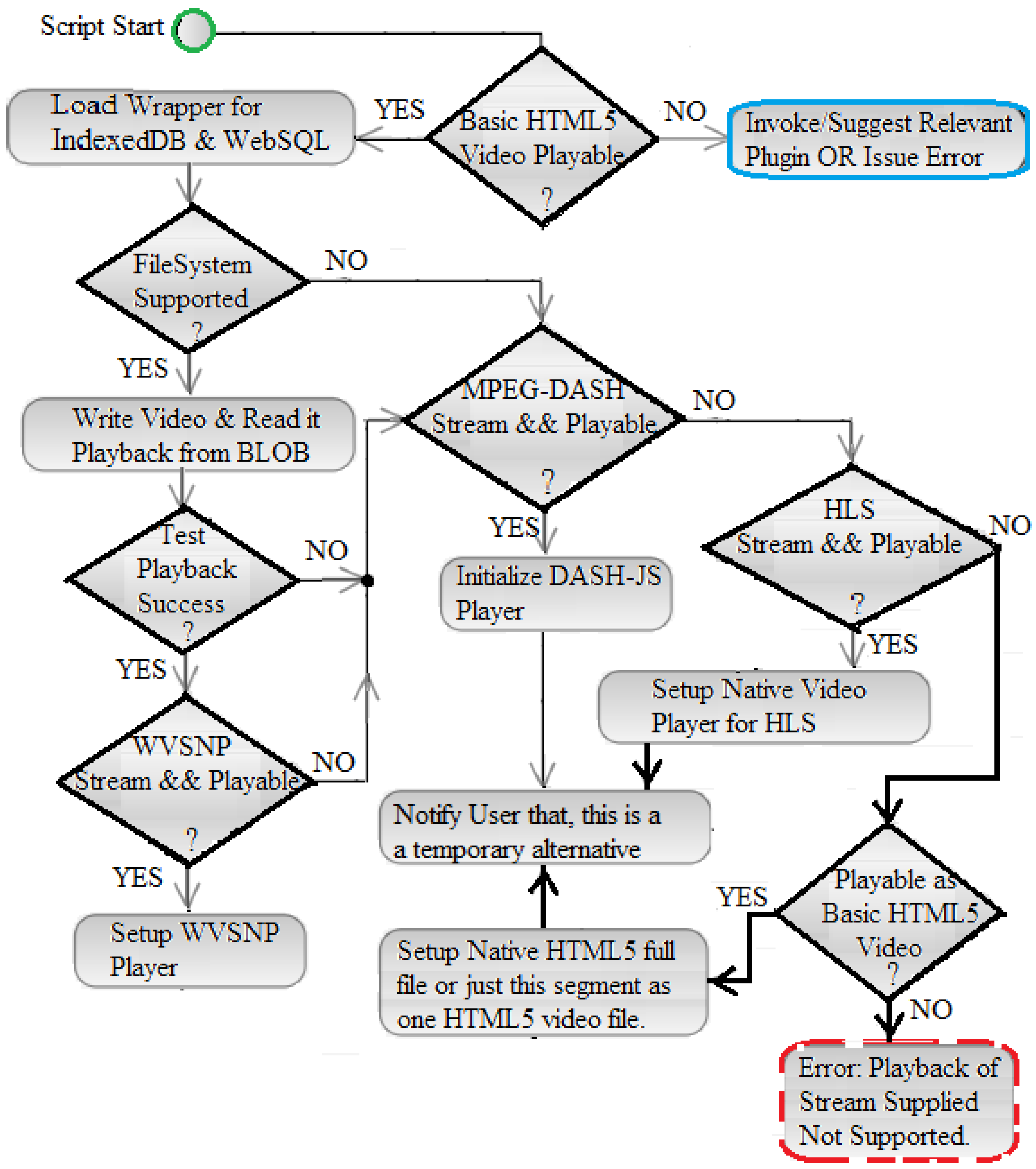

Script Start
Basic HTML5 Video Playable ?
YES
NO
Load Wrapper for IndexedDB & WebSQL
Invoke/Suggest Relevant Plugin OR Issue Error
FileSystem Supported ?
NO
YES
MPEG-DASH Stream && Playable ?
NO
Write Video & Read it Playback from BLOB
YES
HLS Stream && Playable ?
NO
Test Playback Success ?
NO
Initialize DASH-JS Player
YES
YES
Setup Native Video Player for HLS
WVSNP Stream && Playable ?
NO
Notify User that, this is a a temporary alternative
YES
Playable as Basic HTML5 Video ?
YES
Setup WVSNP Player
Setup Native HTML5 full file or just this segment as one HTML5 video file.
NO
Error: Playback of Stream Supplied Not Supported.

Figure 4.2: Illustration of compatibility test flow.

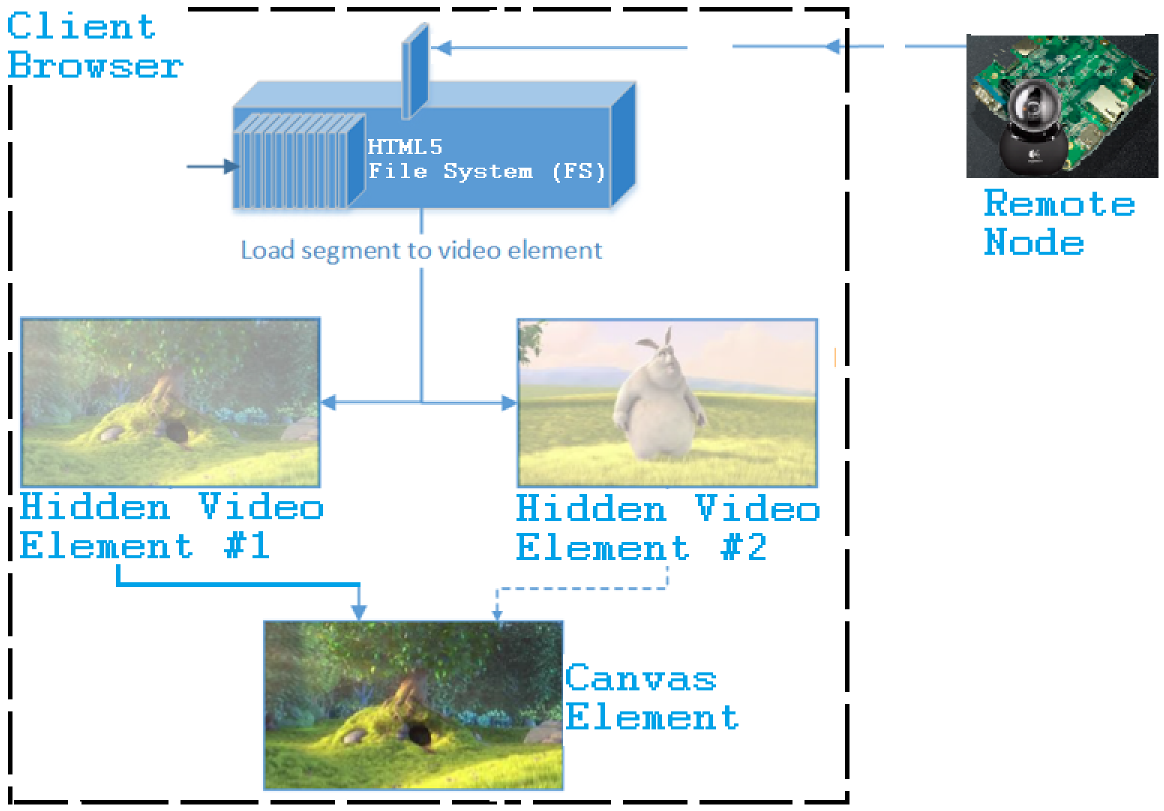


Figure 4.3: Overview of WVSNP-DASH Player (WDP): Video segments are fetched from the remote sensor (server) node into the HTML5 File System (FS) by the buffering process. The playback process plays the current segment from one hidden video element to the canvas element, while the other hidden video element loads the next segment from the HTML5 FS.

The WDP design is relatively future proof in that WDP works on any browser with support for the HTML5 core elements. Also, any future new video codec, such as H.265/HEVC or VP9, or any future video container format, that can be played as a whole video via an HTML5 `<video>` element, is supported by WDP.

Buffering and Playback Processes

The player consists of two main processes running in parallel: a buffering process and a playback process. Each process has its own writing (buffer) and reading (player) counters. The buffering process runs continuously in a loop to fetch segments from

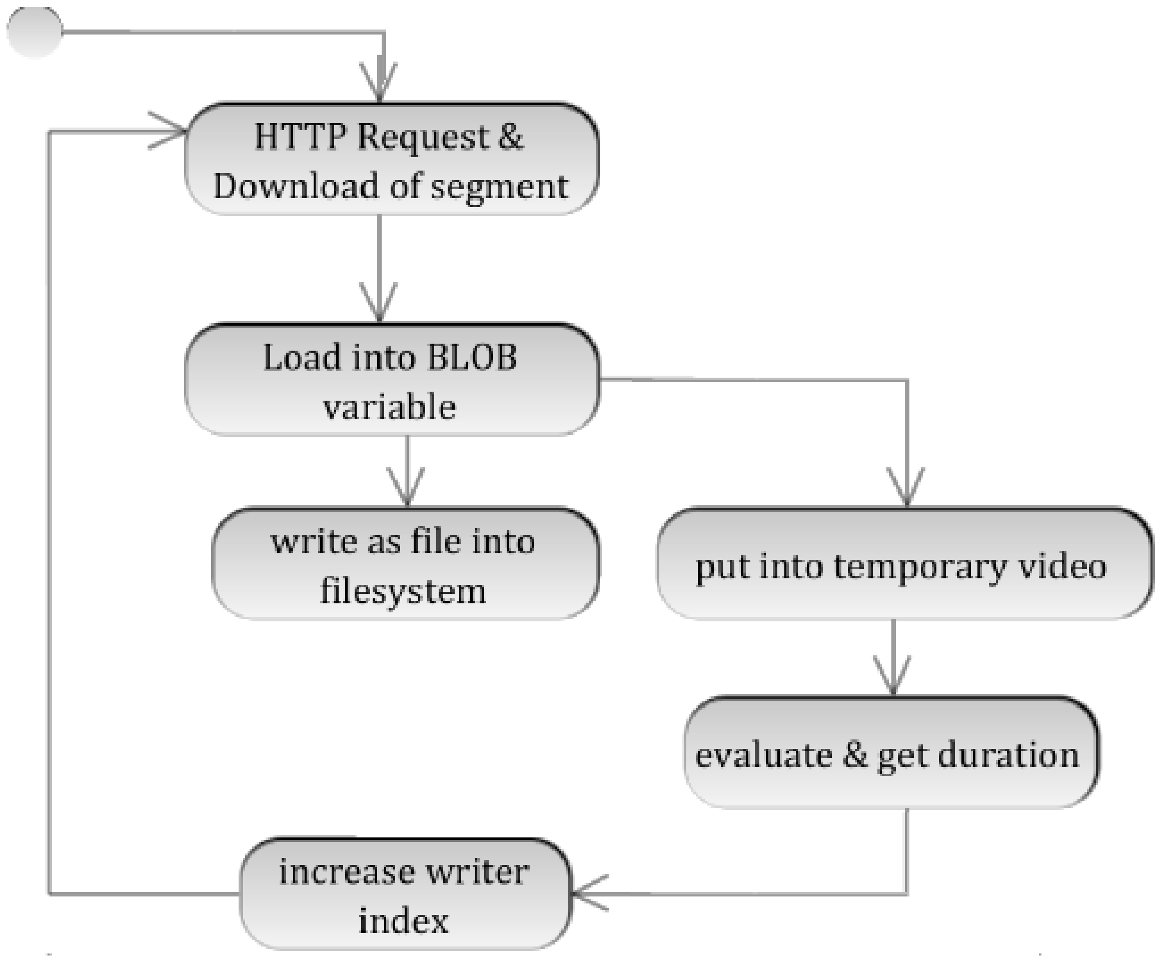


Figure 4.4: The WDP buffering loop.

the remote server node into the HTML5 FS, see top part in Fig. 4.3. Each loop iteration handles one video segment (file) for the currently selected video stream and presentation quality. The file is represented as a binary large object (Blob) [149]. A Blob is a data structure that encapsulates raw binary data and can be fed directly to an HTML5 `<video>` element. For VOD, the buffering process runs continuously until the last segment has been downloaded. However, the user may restart the process with an arbitrary segment by changing the presentation quality or fast-forwarding to a point beyond the buffer line.

The playback process is at least three segments behind the buffering process. As shown in Figure 4.3, in order to stich the segments seamlessly together, there are two hidden `<video>` elements which contain consecutive video segments. When one segment ends, the corresponding `<video>` element triggers an event, which in turn triggers the playback of the next video segment from the other `<video>` element. The now unused `<video>` element is linked to the subsequent segment by loading the segment from the HTML5 FS. Similar to the buffering process, the file is loaded as raw binary data in a Blob variable and fed directly to the `<video>` element.

Glitches in the form of resizing `<video>` elements or black screens during the transitions from one `<video>` element to the next are avoided with an HTML5 ¡canvas¿ element. Each frame of the currently active video element is drawn on the ¡canvas¿ element. During a transition, the last active frame is displayed. With the canvas, the transition glitches caused by the slow JavaScript are not visible to the viewer. For streams with continuous audio, longer video segments reduce glitches in the audio track playback due to fewer transitions. Audio fading techniques can be employed to seamlessly morph the audio tracks of the video segments; such fading techniques are beyond the video-specific scope of this work.

The use of the HTML5 FS in conjunction with the HTML5 canvas for rendering video in WDP enables the rendering of video that arrives to the player via other network mechanisms, aside from basic HTTP/AJAX mechanisms [157]. For instance, the WDP design allows the rendering of images and data that arrive to the client via Common Gateway Interface (CGI) frameworks [81], which are very important for cross-network exchanges, e.g., between Zigbee and Bluetooth networks. The WDP architecture can also readily take advantage of peer-to-peer features of the client and local hardware access features, e.g., via the emerging Web Real Time Communication (WebRTC) interface. WDP's networking flexibility, combined with WVSNP-DASH's independently playable video segments (i.e., no need for manifest file or initialization segment), enables video segment streaming from different sensors. This enables simultaneous streaming and interweaving of segments from e.g., Bluetooth, Zigbee, and WiFi networks.

4.5 WDP Evaluation

4.5.1 Evaluation Set-up

A prototype of WDP was extensively compared with popular DASH players considering the following metrics:

- *CPU load*: Especially for mobile consumer devices, the central processing unit (CPU) load during video playback should be low; also, the CPU load directly influences the power consumption and resulting battery drainage.

- *Memory consumption*: The consumption of working memory, which is limited in mobile devices, gives an indication of the efficiency of resource handling. Inefficient resource usage leads to inefficient power usage. We are still not close to advanced image coding such as this memoryless coder[203].

- *Power consumption*: The main objective of the WVSNP-DASH framework is to work with sensor nodes. Low power consumption at the server side, i.e., the sensor node, which captures, stores, and serves the video files, is critical.

- *Cross Platform Support*: A key factor for consumer acceptance of a player, is the range of supported OS/browser platforms.

- *Supported Codecs*: The player should support a wide range of major codecs (as well as media containers). Wide codec and media container support ensures that the player can easily be used for legacy video playback.

For the prototype evaluation, a typical surveillance video was captured on the campus of Arizona State University (ASU). The 10 minute ASU video (without audio) contains timelapse captures of everyday outdoor activity and scenery around the ASU campus. The ASU video was used for both VOD and LIVE video scenarios. For the LIVE video scenario, a node camera was pointed at the full-screen display of the pre-recorded video while measurements were captured in real time. Focusing and positioning the node camera to see only the full-screen video display made the LIVE video essentially identical to the pre-recoded video. The H.264 encoded .mp4 video was generated in two quality versions, namely a "SMALL" version with 320×180 pixel resolution, 150 kb/s bit rate, and 15 frames/s, and a "BIG" version with 640×360 pixel resolution, 500 kb/s, and 25 frames/s. Each quality version was segmented into MPEG2 Transport Stream (TS) segments for HLS playback, independent MP4 segments for WVSNP-DASH playback, and ISO Base Media File Format (BMFF) segments for MPEG-DASH. The segments were created for 2, 5, 10, and 15 second segment lengths.

The WDP prototype was compared with HLS using the JWPlayer 6 with the HLSProvider plug-in and with MPEG-DASH using the DASH-JS player. The presented evaluations were performed with the Google Chrome (version 32) web browser running on a client operating on a *Ubuntu 13.10* 64 bit, *Dell OptiPlex 360* with *Intel Core2 Duo E7300* 2.66GHz processor and 2 GB RAM. The evaluations were also run with a client operating on *Macbook Air Mid 2012* with *i5-3427U* 1.8 GHz processor and 4 GB RAM as well as a *Windows 7* 64 bit client booted on the same *Macbook* and gave similar results, which are not included due to space constraints.

The server node power consumption was measured from an *i.MX6 ARM Cortex-A9*, 1.2 GHz Quad core, 2 GB node development board. The server node captured video with a USB webcam and generated and served video segments via WiFi and

Table 4.1: Average and standard deviation (SD) of CPU load and memory consumption on WiFi client node as well as power consumption (measured through drawn current) on server node.

LIVE 5 second segments

	CPU load (%)		Memory (MB)		Current (mA)	
Framework	Avg.	SD	Avg.	SD	Avg.	SD
HLS	69.5	8.8	136	9	95.8	29.6
WVSNP	68.4	26.5	122	16	100	26.5
WVSNP-no-c	31.2	12.1	126	15	100	26.5

VOD 5 second segments

	CPU load (%)		Memory (MB)		Current (mA)	
Framework	Avg.	SD	Avg.	SD	Avg.	SD
HLS	61.2	6.3	145	7	37.5	9.92
WVSNP	79.6	12.1	182	18	49.5	8.27
WVSNP-no-c	36.9	13.4	189	21	49.5	8.27
DASH	30	7	143	18	N/A	N/A

VOD 2 second segments

	CPU load (%)		Memory (MB)		Current (mA)	
Framework	Avg.	SD	Avg.	SD	Avg.	SD
HLS	62.8	7.4	145	9	48.8	8.04
WVSNP	79.2	10.2	187	26	51.1	8.43
WVSNP-no-c	39.7	9	180	22	51.1	8.43

a streamlined low-footprint mongoose HTTP server. More detail on the power consumption measurements in the next Chapter.

4.5.2 WVSNP-DASH Results Analysis

Table 4.2 presents summary evaluation results for the prototype WDP in comparison with HLS and MPEG-DASH.

4.5.3 Client Node CPU Load and Memory Consumption

We observe from Table 4.2 that for the 5 s LIVE scenario, WDP has a similar CPU load as HLS (JWPlayer 6) while the WDP memory consumption is somewhat lower

Table 4.2: Average and standard deviation (SD) of CPU load and memory consumption on WiFi client node as well as power consumption (measured through drawn current) on server node. For 10s segments and full 10 minute video.

VOD 10 second segments.

	CPU load (%)		Memory (MB)		Current (mA)	
Framework	Avg.	SD	Avg.	SD	Avg.	SD
HLS	61.1	6.7	146	7	36.7	8.57
WVSNP	79.3	16	180	29	48.0	8.25
WVSNP-no-c	37.3	14.4	200	25	48.0	8.25
DASH	29.5	6.5	168	21	39.1	9.13

Progressive Download of Full Video, No segmentation.

	CPU load (%)		Memory (MB)		Current (mA)	
Framework	Avg.	SD	Avg.	SD	Avg.	SD
Full Video	28.7	4.5	91	2	43.1	9.57

than for HLS. On the other hand, for the 5 s VOD scenario, WDP has higher CPU load and memory consumption than HLS and MPEG-DASH (DASH-JS). The higher CPU load and memory consumption of WDP for the VOD scenario are mainly due to the WDP buffering algorithm for VOD, which fetches all segments as fast as the network bandwidth and the client FS space allocation allow. In contrast, HLS buffers only approximately three segments and discards them after playback. WDP stores all segments in the FS space leading to high memory usage. This WDP approach facilitates power savings on the server node by avoiding re-fetches during quality switches or rewind, or actions that reuse previously fetched VOD segments.

The WDP LIVE buffering algorithm behaves similar to HLS. As Table 4.2 indicates, WDP LIVE playback has lower CPU load and memory consumption than HLS. This is remarkable in that the WDP prototype uses a resource-heavy canvas element as well as two video elements concurrently to render the video, while HLS uses a highly optimized HLSProvider Flash helper plug-in to the browser with only one video element. MPEG-DASH (DASH-JS) playback, included for the 5 and 10 s

VOD scenarios in Table 4.2, has the lowest CPU load and memory consumption. This MPEG-DASH result indicates that using pure Javascript and MSE with one video element can be more efficient than using a plug-in.

In order to further examine the high CPU load and memory consumption of WDP for VOD, the WDP prototype was slightly modified to play segments directly in the video elements without multiplexing two video elements nor rendering on the canvas. This modified WDP, which is denoted by "WVSNP-no-c" in Table 4.2, has significantly lower CPU load and memory consumption than HLS as well as similar CPU load and memory consumption as MPEG-DASH. This result for WSNP-no-c, i.e., WDP without using the canvas element, indicates that the high WDP CPU load and memory consumption for VOD are mainly due to the canvas element. This validates that the use of full (independently playable) video segments does not increase client resource usage.

4.5.4 Server Node Power Consumption

Next, the WVSNP-DASH implications for the power consumed by the server node are examined. This is derived by comparing power consumed by the node server while serving each of the different types of the player frameworks. Since the voltage readings were generally consistent at 5 V, only the current drawn by the server node during each scenario is reported as a relative measure of power consumption in Table 4.2. This is a relative measure of the network power implications of the different frameworks. The results for current in Table 4.2 indicate that the LIVE scenarios have significantly higher currents and thus higher power consumption than the VOD scenarios. This is because the server node captures, transcodes, stores, and serves the video segments at the same time. Importantly, for the LIVE scenario, the WDP prototype has only slightly higher power consumption than HLS. In interpreting these power results, it is important to note the differences in using ffmpeg for segment capture in the compared frameworks. HLS captured and transcoded LIVE video in an optimized built-in (native) HLS function of ffmpeg, which achieves highly efficient software-based capture and transcoding. On the other hand, WVSNP-DASH capture used a prototype-level bash script that newly invoked ffmpeg for each LIVE segment capture. This means that the WVSNP-DASH prototype incurred extra power, resources, and inefficiencies for each context switch of launching ffmpeg, transcoding, storing, and then shutting down the ffmpeg process for each video segment capture. Moreover, WVSNP-DASH interpreted a script at run time for every segment, adding to the resource usage. In contrast, HLS capture invoked ffmpeg only once at the start of the video stream capture and captured the remaining segments with the same optimized ffmpeg (from the original invocation context). These conceptual differences imply that an optimized native WVSNP-DASH capture application has considerable power savings potential for LIVE video in the WVSNP-DASH framework compared to the already optimized HLS framework.

A further potential for power savings arises from the WVSNP-DASH operation without a manifest file. HLS and MPEG-DASH require a manifest file that needs to be managed and re-read and updated during the capture and/or playback. For VOD, the manifest files and segments are typically static and the indices do not need to be continuously updated. However, for synchronization of LIVE video, the manifest files have to be typically re-fetched regularly for LIVE video synchronization. Additional processing to create special subsequent segments different from the initialization segment adds to the power consumption of HLS and MPEG-DASH for LIVE video.

For VOD, the results for current in Table 4.2 indicate that longer segments generally reduce the power consumption. This is consistent for both HLS and WDP. Again, noting the inefficiencies in the WVSNP-DASH prototype due to script interpretation and ffmpeg invocation for each video segment, there are power saving potentials for WVSNP-DASH compared to HLS and MPEG-DASH.

Note that, since there are really no browser MPEG-DASH players there to use for all scenarios "DASH-JS" player had issues playing 2 second segments and LIVE video. Even the industry supported dash.js only recently started putting out a LIVE stream example to test and help implement LIVE functionality.

In order to further examine the impact of the segmented video streaming, the full 10 minute ASU video was streamed via progressive download and the results are reported in the bottom line of Table 4.2. Table 4.2 indicates that 10 s segment HLS and MPEG-DASH streaming consumed less power compared to full-video streaming. These results indicate that segmented video streaming does not lead to higher server node power consumption than streaming a full (unsegmented) video to an HTML5 element. The 36.7 mA measured for HLS for 10 s VOD segments can be considered as the worst-case (maximum) current consumption expected of a WVSNP-DASH using a specialized native capture. More generally, the comparison of currents for 10 s VOD segments and full video download indicates that segmented streaming can result in about 15 % power savings compared to streaming progressively downloaded full videos. From further test evaluations, that are not included due to space constraints, it was noted that there is no benefit of using 15 s segments compared to 10 s segments.

In WVSNP-DASH, each video segment file has its own file header. The file header contains all the video file properties and internal video container meta data, such as duration, compression type, size, and bit rate, that are needed for decoding by any player. In HLS and MPEG-DASH, most of this file header meta data is moved to the first segment and the remaining segments are merely fixed data elements (blocks) that cannot be decoded independently. e.g. you can do a seek (FF/RW) within a

10 second WVSNP segment but you cannot do that for an HLS nor MPEG-DASH segment. Thus, it might appear that WVSNP-DASH introduces some overhead by including the file header in each video segment file, whereas HLS and MPEG-DASH are continuous streams with random access points due to pre-set intracoded (I) frames. However, due to the self-contained file headers in each WVSNP-DASH segment, there is no look-up data that needs to be maintained and referenced at the server node every time from the initial segment when there are quality switches for dynamic adaptation. This reduction of server node processing for managing the video segments has the potential to reduce power in video sensor nodes. WVSNP-DASH segments can still be captured with specific I-frame positioning to match efficient transcoding practices.

The smaller multiple files as opposed to one big file for storage in the WVSNP-DASH framework also enables flexible use of the node storage as well as sharing among storage deprived nodes in a sensor network. Heterogeneous WSN bandwidth, congestion, and diverse radio capabilities across or in between networks, can be accommodated by storing segments with a range of encoding parameters, such as different resolutions, bit rates, encoding qualities, and encoder types. and even source type.

Compared to a big movie file which other HLS and MPEG-DASH accepts, WVSNP-DASH recommends that all video files be stored in sizes not longer than 10 seconds. This allows for distributed storage of the same movie stream across multiple nodes with limited storage. Note that HLS and MPEG-DASH having small segments as well is not the same since their segments have to be tied to one node logically as well as the initialization file that precedes it and the manifest file.

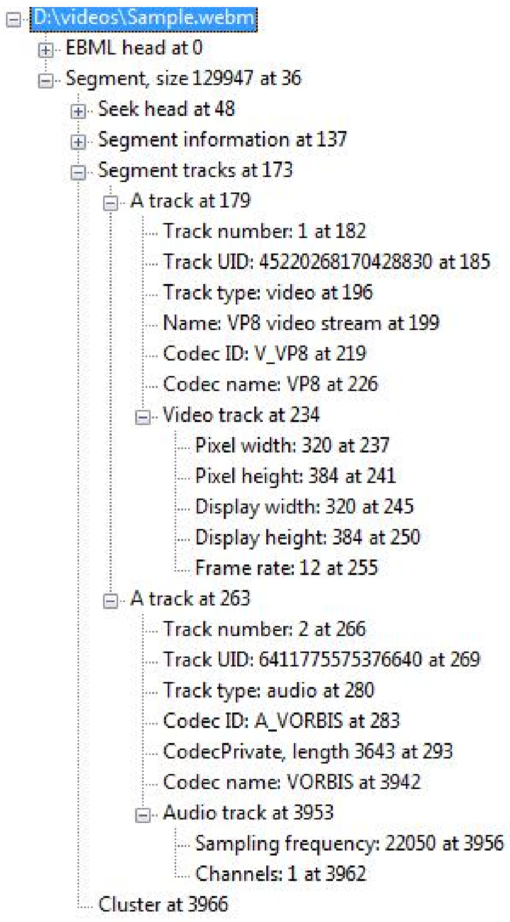
4.6 MSE Based Player Option

As mentioned in section 4.2.1 most browsers now seem to be moving in the direction of supporting Media Source Extensions (MSE). The WDP above relied on the

HTML5 File System. This works well with some disadvantages. It does not seem to have uniform support across all key browsers. It also relied a lot on the HTML5 canvas. MSE as adds buffer-based source options to HTML5 media for streaming support. This is different from progressive download which eventually downloads a complete video file to play. If also differs from WDP in that it does not require local browser file system storage of the segments nor the canvas. As explained at the end of section 4.3.1 above the MSE based WDP gets rid of the disadvantages above and behaves like MPEG-DASH while still avoiding initialization files and manifest files. To be able to achieve this a few discoveries had to be made. Non licensed video containers and codecs like webm/vp8-9/opus seem to follow consistent video structure that enables easy segmenting and fragmented streaming without any modification. The default recording of a WebM video will yield a structure as below.

It always has two level 0 elements there: EBML and Segment. The EBML element tells that the file is actually a valid EBML file, e.g. what version? The parser will not attempt to read EBML file that it does not support. Within the segment element, there are four basic level 1 elements that a well-formed WebM file should contain. They are Seek Head element, Segment Information, Tracks Information and Clusters. This means you can parse it easily as everything is always in hierarchical order and every internal data box can be dereferenced to know complete information for MSE buffering. Unfortunately good containers like WebM are not well supported in browsers, we have to deal with working with mp4/h.264/aac type files.

MP4 on the other hand can follow different file formats. MPEG-DASH itself has different file specifications. So, for MSE, mp4 files should not only be structured so that meta data is fragmented across pieces of the container, it must be also across the actual audio/video streams being fragmented. Not clustered together. See the specification from ISO BMFF Byte Stream Format, section 3 [105]. This specifies an



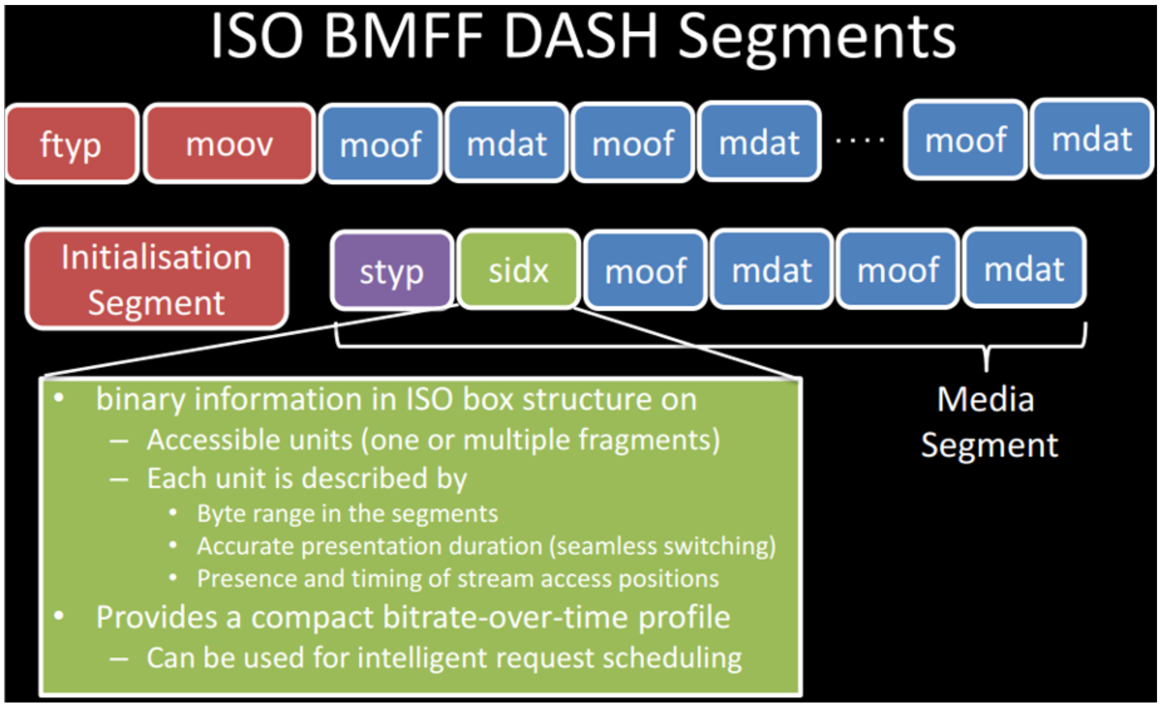
```
D:\videos\Sample.webm
  EBML head at 0
  Segment, size 129947 at 36
    Seek head at 48
    Segment information at 137
    Segment tracks at 173
      A track at 179
        Track number: 1 at 182
        Track UID: 45220268170428830 at 185
        Track type: video at 196
        Name: VP8 video stream at 199
        Codec ID: V_VP8 at 219
        Codec name: VP8 at 226
        Video track at 234
          Pixel width: 320 at 237
          Pixel height: 384 at 241
          Display width: 320 at 245
          Display height: 384 at 250
          Frame rate: 12 at 255
      A track at 263
        Track number: 2 at 266
        Track UID: 6411775575376640 at 269
        Track type: audio at 280
        Codec ID: A_VORBIS at 283
        CodecPrivate, length 3643 at 293
        Codec name: VORBIS at 3942
        Audio track at 3953
          Sampling frequency: 22050 at 3956
          Channels: 1 at 3962
    Cluster at 3966
```

Figure 4.5: The structure of a WebM file.

initialization segment file as a single File Type Box (ftyp) followed by a single Movie Header Box (moov). See Figure 4.6 or 4.7 below.

Our simple Ffmpeg transcoded WVSNP segments do not have the expected format and thus fail when trying to play back in a browser with MSE. Investigating our structure shows Figure below whereas the video files used by Google's Shaka reference MSE player show

As you can see, the WVSNP one lacked sidx box so that might be the cause. This means we have to capture video files in a way that they will play for cases where MSE is used and when it is not. Just for background, File Type Box (ftyp) specifies file type and compatibility. All the meta data is defined in Movie Box (moov). Media



- binary information in ISO box structure on
  - Accessible units (one or multiple fragments)
  - Each unit is described by
    - Byte range in the segments
    - Accurate presentation duration (seamless switching)
    - Presence and timing of stream access positions
- Provides a compact bitrate-over-time profile
  - Can be used for intelligent request scheduling

Figure 4.6: The structure of an MP4 file as expected by MSE.

Data Box (mdat) specifies all the data (audio and video samples). The index of one media stream within the media segment is defined by Segment Index box (sidx). Its the same level as Movie Fragment Box. There can be more than one Movie Fragment Box per one Segment Index Box.

It turns out that what actually needs to happen is that the basic mp4 container does not have [ftyp] followed immediately by a [moov]. This must be the case. So we re-engineered so that the order is [ftyp] –¿ [moov] and –¿ [moof]/[mdat] pairs. Fortunately Ffmpeg has flag `-movflags frag_keyframe+empty_moov` flag that can move these around when trans-coding or at capture. `frag_keyframe` forces starting a new fragment at each video keyframe. `empty_moov` was originally used by Microsoft smooth streaming files to write an initial moov atom directly at the start of the file. It does not describe any samples in it. Unlike the basic MOV/MP4 files that have

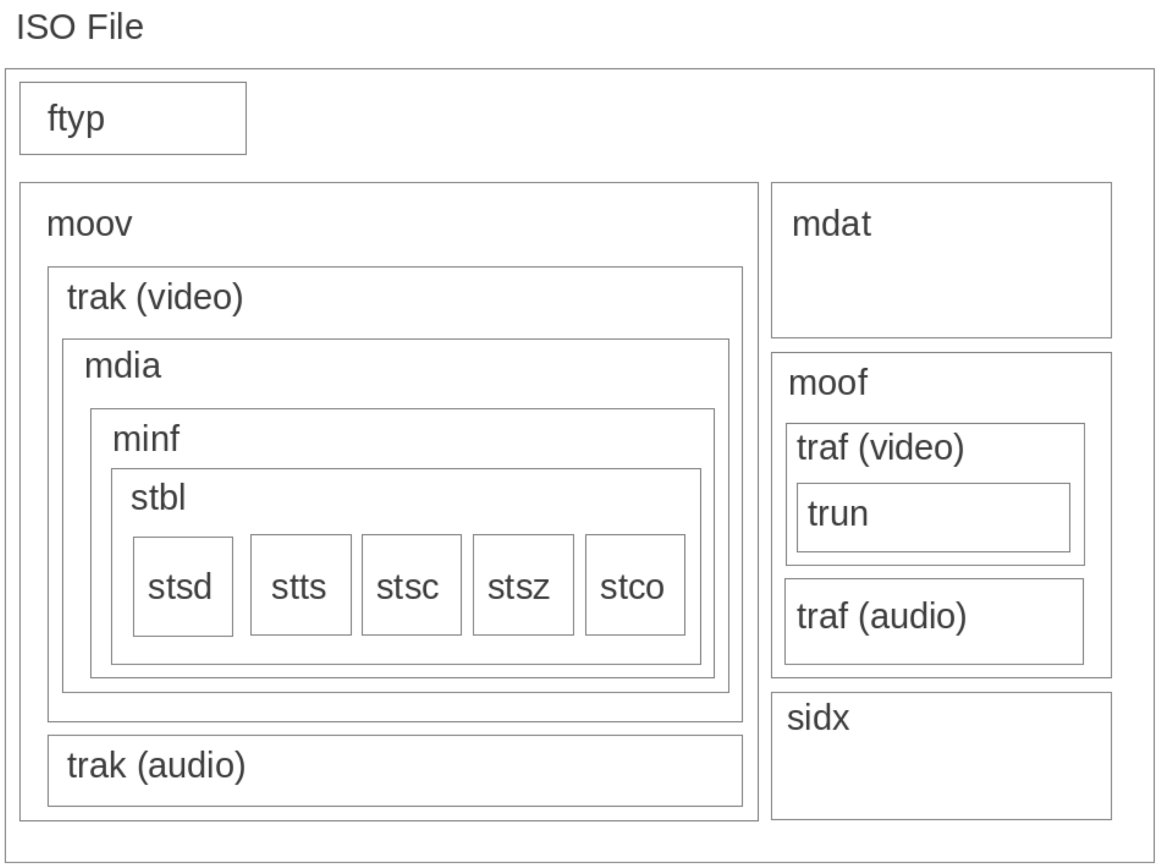
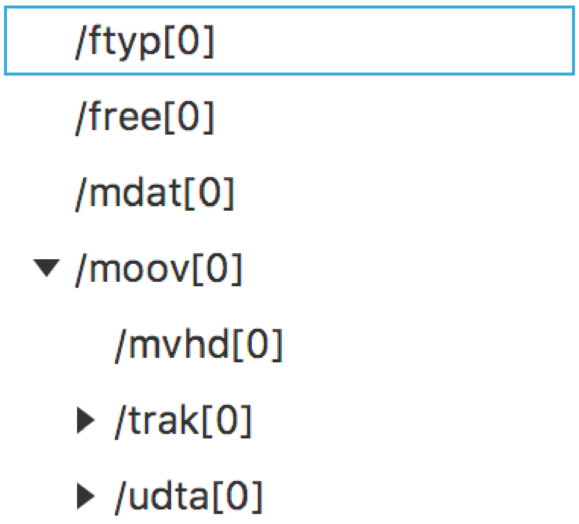
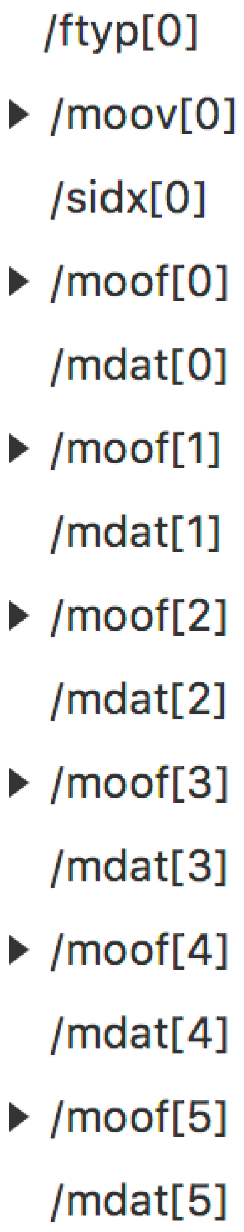


Figure 4.7: The structure of an MP4 file as expected by MSE.



```
/ftyp[0]
/free[0]
/mdat[0]
▼ /moov[0]
    /mvhd[0]
  ▶ /trak[0]
  ▶ /udta[0]
```

Figure 4.8: Basic WVSNP video file with no sidx.



```
  /ftyp[0]
▶ /moov[0]
  /sidx[0]
▶ /moof[0]
  /mdat[0]
▶ /moof[1]
  /mdat[1]
▶ /moof[2]
  /mdat[2]
▶ /moof[3]
  /mdat[3]
▶ /moof[4]
  /mdat[4]
▶ /moof[5]
  /mdat[5]
```

Figure 4.9: Google reference MSE MPE-DASH videos.

mdat/moov pair written at the start of the file containing only a short portion of the file. `empty_moov` results in no initial mdat atom, and with moov atom only describing the tracks but with a zero duration.

Once that was resolved it was important to make sure the the meta-data inside the video is set properly to what MSE expects as (avc1.XXXXXX) format. Some browsers' rendering engines require that MediaSource.isTypeSupported is true only when "Codecs String" is avc1.42E01E.

Figure 5.16 shows sidx and a series of moof+mdat boxes. The precise byte range locations of each of the moof+mdat segments is stored in sidx, the segment index.

With this analysis and rework of the how we capture MP4 files, the next task was to create an algorithm that follows WVSNP framework rules, uses MSE buffering and does not need one initial segment nor manifest file. This make WVSNP and even more robust framework that becomes even more cross platform. This enhancement to the player was added for case where HTML5 File System is not desirable or not supported. More analysis will lead to whether this should be our default mode of the WDP. None-the-less this has major beneficial implications on the platform and will be discussed later in the document.

Chapter 5

DESIGN CHOICES VIA PROFILING

5.1 Introduction

Attempts to include video streaming from sensors via miniaturized devices has been attractive for a wide range of web-based applications but mostly surveillance. Very low power video/frame data structures are needed to expand Multi-Dimensional (MD) streaming into non surveillance applications. MD data perception and analytics will be key for a more complete IoT future. There has been other work to study energy-efficient video transmission over a wireless link[130]. Some work involved adapting the video codec itself to make the compressed data suitable for multi-rate streaming[177]. None of these is addressing the data format itself. To create helpful video data formats for sensor type storage and distribution, it is imperative that all the components of power consumption within a sensor network be understood at the capturing node level. This section profiles power consumption in a wireless video sensor node. This includes power implications of popular web-based video streaming frameworks, such as the Hypertext Transfer Protocol (HTTP) Live Streaming (HLS) and the Motion Picture Experts Group's Dynamic Adaptive Streaming over HTTP (MPEG-DASH) are analyzed as well as basic Progressive Download. These are also evaluated against a new Wireless Video Sensor Network Platform compatible DASH (WVSNP-DASH) framework. Additionally, power aspects of the MD capture, storage and streaming pipeline are evaluated. This work therefore provides real world empirical data on architectural decisions necessary for a design of an IoT and Machine to Machine (M2M) compatible MD sensor node. The empirical data and architectural recommendations contributed by this work further validates the new (WVSNP-DASH) framework as more suitable for flexible web-based access of video due to the power implications of its independently playable video segments and a unique non-manifest based naming syntax that conveys elementary metadata that facilitates low power flexible search, transfer, distribution, and playback.

There have been many attempts to analyze sensor node power consumption, throughput and ease of use after a prototype has been built. There has been a heavy focus on just the video compression itself as a large sink of power. Proposed alleviations therefore try to address this without empirical data that is the course[170, 169, 167, 114]. Usually this is with the hope that designers have resources and time to iterate over several prototypes until an optimal or acceptable trade-off has been found. This ends up in unaccounted for time cost that includes engineering salaries. Inter-dependent components are harder to decouple in the later stages of prototype refining. Prototype refining is often followed by software optimizations that try to make the best or most efficient use of the locked-in late stage prototype components. This is another cost. Too much software optimization design to make up for hardware deficiencies, usually results in very high HW/SW coupling which is the opposite of the ideals of a node design elaborated on in previous publications [180]. Printed Circuit Board (PCB) designers know that it is hard to decide to take everything apart and start from scratch even if it is a prototype. There are marketing, software and time pressures that end up with compromises. Additionally, in software, it is pointless to optimize program code without knowing where the bottleneck is. This requires profiling. This section additionally highlights the benefits of profiling, even before prototyping (pre-profiling). Pre-profiling helps node architects spend time and resources only on important areas of a future node. In hardware, it is particularly a wasteful exercise to attempt to build a power efficient node without knowing which parts of the application actually consume the most power nor which are most critical to the throughput of the node and its applications.

As mentioned in the WVSN literature, video data acquisition and transmission has a history of being power hungry and therefore not useful for the typically battery operated sensor nodes. In [182], existing web-based video streaming frameworks, such as the Hyper Text Transfer Protocol (HTTP) Live Streaming (HLS) and the Motion Picture Experts Group's Dynamic Adaptive Streaming over HTTP (MPEG-DASH), are evaluated as possible video data acquisition and transmission solution to the many mentioned WVSN shortcomings. Additionally a similar Wireless Video Sensor Network Platform compatible DASH (WVSNP-DASH) framework was described together with its WVSNP-DASH Player (WDP) as a suitable and more efficient framework for WVSN than HLS and MPEG-DASH. In this section, a comprehensive empirical analysis of the major components of video capture, storage and transmission are evaluated in terms of power consumption at the node. This reveals many useful and convincing design parameters for a WVSN Node (WVSn). Evaluations performed with a WDP prototype against optimized HLS and MPEG-DASH players indicate that WVSNP-DASH provided significant potential for power savings on the sensor node serving the video streams. Empirically derived conclusions are summarized with tabulated data showing power consumption patterns of critical modules that make up WVSn's video data path flow. This work, therefore, serves as a quick reference for WVSn architects, designers, algorithm and application developers.

This work evaluates most elements of a WVSn that are likely major consumers of power. These range from the hardware (HW) acceleration versus software (SW) only; interface used in the data movement pipeline; wireless versus wired transmission; data type captured; compression types; client type and most importantly the framework or protocol used for storage, distribution, transmission and client playback. Because

IoT users expect seamless cross platform user experience, the evaluation skews toward use cases that compare video streaming frameworks that are easily comparable across different OS and web client environments. Version five of the Hypertext Markup Language (HTML5) and the emerging Dynamic Adaptive Streaming over HTTP (DASH) specification [11, 206] address these cross-platform video use cases. These use cases can be easily extended to streaming applications and other yet to be defined IoT use cases that rely on MD data (MDD). There are a many power consumption implications on the frameworks used. For example, as outlined later in Section 5.2, HTML5 does not support adaptive real-time video playback. The power implications of this fact and that existing DASH players are designed to work with video server nodes limited to the TCP/IP networking protocol stack is explored. The evaluations give insight on component power consumption since the node is designed to also use popular non-TCP/IP protocol stacks on resource-constrained sensor (server) nodes, such as Zigbee [13, 8, 180] and Bluetooth. Existing DASH players for example, require complex plug-ins that invite security vulnerabilities and/or have very limited cross-platform support. The power cost of these use cases is also evaluated.

Each major video capture-to-playback pipeline component contributes differently depending on the capture format, storage and transmission choices. Evaluation results show the negative power consumption side effects as the main structural limitation of existing DASH frameworks for sensor nodes. That is, HLS and MPEG-DASH have individual video segments that cannot be independently distributed and played back. Each video segment requires an up-to-date manifest file to be played back in conjunction with a special initialization video segment. This dependency introduces complications of video data management to the resource-constrained video sensor nodes. Additionally, these video segment files cannot be independently cached and distributed by storage-constrained sensor nodes. Any work around to centrally manage the manifest file further creates local network coupling and will result in a net increase in power consumption in the node or network. This is because unique video segment files cannot be independently distributed across nodes nor across networks. Empirical data provided by this work also shows the type of components affected, and how they are affected, to contribute to this overall conclusion.

5.2 Background and Related Work

5.2.1 Sensor Node Power Monitoring and Measurement

There is a good amount of general power measurement literature. There are a lot of references to the critical part power plays in a sensor network design and implementation. There is, however, very little if any literature about power measurement and management in sensor networks/nodes. Just to review the basics, energy consumption of an electrical device is calculated by the product of current (I), voltage (V) and time (t). To calculate energy, voltage and time can be calculated directly, but there is no direct way to calculate current. There are in-system power monitoring tools especially in the Linux environment like ***PowerTop*** [9], ***powerstat*** [78], SW library options like ***PowerAPI*** [28, 150, 151] for process specific monitoring, kernel specific ones like ***powerman*** [73] and ***powerscripts*** [4]. All the above methods suffer from the problem of affecting the device under test (DUT). Since they are processes within the node, they are also consuming power and not measuring ideal operating scenario for the node. Additionally they need power to be on and the operating system to have booted before they can be useful. This misses the boot loader stages and even the power-up part of the node. Below are a few methods reviewed for performing power measurement which leads to this work's focus on power profiling of a wireless video sensor node and resulting architectural implications on the design of a WVSn.

A similar approach to a subset of this work's testbed evaluates the energy efficiency of HW accelerated cryptography modules on sensor nodes [84]. Using the SANDbed testbed [92] equipped with Sensor Node Management Devices (SNMDs) [92], [84] revealed about seventy six percent energy savings were possible using a VaultIC420 HW module compared to using only SW. [84] also concludes that HW-based mechanisms improve the energy-efficiency of the overall application only for specific algorithms within the application. The savings were especially realized when HW modules were duty-cycled. In [91], different approaches to measure energy in wireless communication devices and analysis of each approach are discussed as well as their pros and cons. The [91] survey, therefore, outlines different current measuring techniques and makes recommendations on techniques to use based on targeted requirements. One popular approach is to place a ***shunt resistor*** in series with the total load circuit. The current draw is the same across the whole circuit which implies a current draw across the resistor of (V/R). Though the shunt resistor is the easiest method to use, if the voltage over the resistor is too large, the device might malfunction. This is avoided by using a very low resistor value to keep the voltage across the resistor low. The low shunt resistor value solution comes at the expense of not being able to measure highly dynamic signals. The higher the dynamic range of the current, the lower the accuracy of the low currents measured. The dynamic range of the shunt resistor can be improved by using the ***Voltage to Frequency Conversion*** method which can also improve the accuracy for low current. This is done by connecting the shunt resistor to a *"voltage to frequency conversion"* block [129]. This way, highly dynamic low as well as high currents can then be measured from the block with some high accuracy.

Another current measurement technique is the ***inductor method***, usually used in current clamps for heavy engineering tools. Current is determined by sampling voltage induced in the clamp inductor by the electric field around the wire supplying the load circuit. The inductor technique supports high sampling rates. The required calibration after each measurement, as well as susceptibility to noise are some of the drawbacks of this method. The ***Coulomb Counter*** method uses two capacitors, which are charged and discharged in turn. The capacity of the capacitor is used to calculate the time required to discharge. The temporal resolution for coulomb counter method depends on current draw. This in turn implies the amount of the current drawn during a capacitor discharge. Since the current drawn is not constant, the temporal resolution is also not constant. Since low current draw results in low frequency this method has low temporal resolution.

A low-cost power measurement experiment for wireless sensor networks is discussed in [141]. Some calibration and validation procedures use a clamp-on current probe to collect current measurements and some, the shunt resistor technique. As mentioned above, the current probe is placed around the power supply wire and the output voltage through the current probe is sampled to linearly calculate current through the clamp. In [92], an SNMD is used in a WSN testbed to measure the power of the sensor nodes. SNMD is a wire-based infrastructure which is only available in testbed setups. SNMD is targeted to protocol and network evaluation to estimate and enhance the network and node's lifetime. It uses a shunt resistor method to collect current measurement and calculate energy consumption. A Scalable Power Observation Tool (SPOT) is introduced in [108]. It is designed to measure the current consumption of a sensor node to a microsecond resolution and exceed four decades of dynamic range. SPOT also uses the shunt resistor method for in situ current measurement where the SPOT block is connected in the path of the supply to the node. In [85], Avrora simulation tool [212], is used to validate energy measurement in wireless sensor networks. The simulation results are compared with the results obtained from a SANDbed testbed.

This work used an inductor method with a 10 uA resolution current clamp probe attached to a digital oscilloscope with a 10 MHz bandwidth, 100 MS/s sampling rate and an 8-bit (12-bit enhanced resolution). The voltage across the entire node load is taken with the total load current measured from the supply line as shown in Figure 5.1 below.

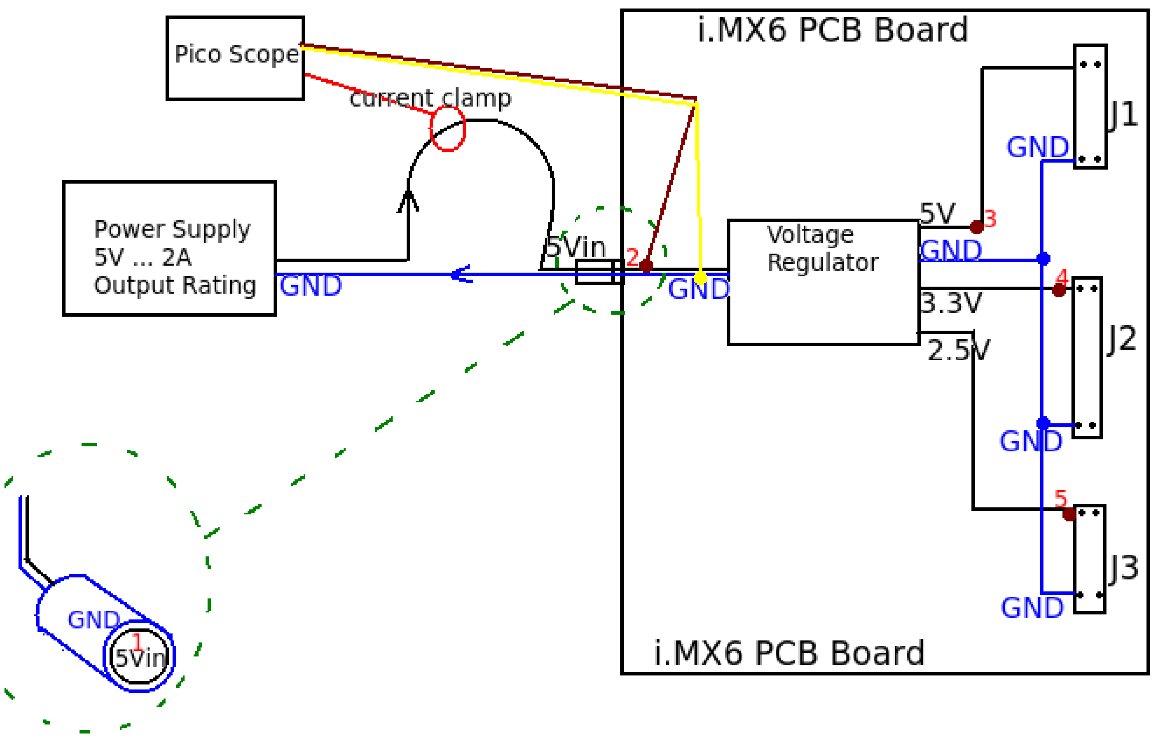


Figure 5.1: DUT power measurement setup and data logging.

More expensive and higher accuracy tools can be used to follow the same setup. These tools are low cost and show adequate data to draw conclusions on the data. Another interesting tool we use to double check the measurements done in Figure 5.1 is a 24-bit ADC open meter tool called the Mooshimeter [144]. This is a very low cost higher accuracy meter that can simultaneously log current and voltage. This has higher than 0.5% accuracy DC, more than 1.0% for AC with harmonic less than

1kHz. Though it can measure up to 10 A, it has a resolution of less than 5 uA. It has more than 10 Megaohm input impedance and more than 1% resistance accuracy over 20 Ohms to 20 MOhms. The Voltage precision of 15 nV per count for up to 100 mV or 200 nV per count up to 1.2V. This comes with 8kHz dual simultaneous sampling capability. Its 24-bit maximum resolution has greater than eighteen (18) effective bits at 125 samples per second. The advantage of using compact power monitors and loggers like Mooshimeter, is the ability to do stand alone power monitoring on the field as opposed to attaching the nodes to lab equipment like in [241, 79]. Laboratory power monitors are not only expensive, but hard to move to the field under normal usage environments. This means usage scenarios of the device under test might reflect only laboratory environments.

5.2.2 Power Efficiency Implications

One of the benefits of the simplicity of the WVSNP-DASH name syntax is that it transfers most of the video playback and retrieval complexity to the player. The player is the consumer of the video data, and thus knows what it wants to do with the video data and how to interpret the video data. This has far- reaching implications for power budget reasons. A client can request lower quality levels of the same data or even can request the next segment from another node within the network with better power or bandwidth parameters. For example, quality index zero (0) can be still-frames encoded at one frame per second or other ways used by application streaming methods. The consumer has the fetching algorithm intelligence on the client tailored to their need or application.

There are two past papers in particular [241, 79], whose work can be expanded to demonstrate the power efficiency benefits of the WVSNP-DASH framework versus MPEG-DASH and versus HLS. For example, [241] focuses on an MPEG-DASH client's network scheduling and measuring client power consumption over forth generation Long Term Evolution networks (4G LTE). The paper concludes that longer segments and larger buffers are best for the MPEG-DASH clients' power consumption. Their power measurements on the client revealed that the important variable in managing power consumed was the radio resource control (RRC) state in LTE. In [79] an energy-efficient HTTP adaptive streaming algorithm is proposed. The algorithm requires that the MPEG-DASH client be on multiple networks like LTE and WiFi so it can exploit multiple networks simultaneously to find opportunities for low power during streaming. They conclude that by dynamically changing wireless network environments during streaming, energy consumption can be reduced. Both focus only on MPEG-DASH. Neither mentioned HLS, interestingly. Both [241] and [79] are essentially an attempt to improve the client's adaptive algorithm in fetching segments to save power on the client itself. They just observe other network parameters to add to the adaptive fetching decisions by the HTTP client. This section focuses on the node server/network side. While it reveals general WVSN power conservation architectural parameters to consider in designing a node, it introduces and highlights a new framework which removes a lot of issues [241] and [79] are trying to improve on the client side. This paper also shows how these choices are applicable beyond HTTP. The work in [241] and [79] is, therefore, a subset of the use cases where WVSNP-DASH can be tested following exactly what they are doing on the clients side and further contrasting WVSNP-DASH with MPEG-DASH and HLS. Again, these papers did not even contrast with nor mention the most popular DASH framework, HLS.

WVSNP-DASH [182] demonstrates the concept of the uniqueness of the video segment's name. Since WSNs are resource constrained in terms of power, computing resources, and storage space, WVSNP-DASH enforces that video files within a sensor network are captured and stored as complete, individually playable video files of short duration, preferably no longer than ten seconds (the impact of the segment length is examined in section 5.3).

Recall that with HLS and MPEG-DASH, the player requires the manifest file in conjunction with the individual video segment files for playback. In contrast, the WVSNP-DASH video segment name syntax ensures that the name of the segment, or any other future network video object, conveys sufficient information for the player to decide how to fetch and play the video segments. The manifest files required by HLS and MPEG-DASH introduce incompatibility issues as they require browsers to support manifest files as well as live playback maintenance of the manifest files. By communicating all player pertinent meta data through the segment file name, WVSNP-DASH avoids these incompatibility issues and is thus highly backward compatible.

Another important feature of the WVSNP-DASH framework is random playback of any segment, as needed by the player. In particular, WVSNP-DASH has no special initialization segment, which is necessary for HLS and MPEG-DASH players to understand how to play a video. Each WVSNP-DASH video segment has enough meta data (in its name) to start playing the video. This feature enables unlimited "crawler type" network search algorithms. These can be used by future information centric network (ICN) routers and content distribution networks (CDNs). Future active players may start playing the video as soon as a segment is discovered. Most importantly, there is no need to download prior segments, initialization files nor manifest files, to play just the portion of video needed by consumer. This is a power saver.

For instance, as detailed in [182], a WVSNP-DASH segment name:

src=filename-1-1-VOD-45-7.mp4 can be easily passed to the player. Based on the meta information contained in this name, the player can initiate the streaming process of segments 7 to 45 by fetching all segments and playing them one after another. The random playback feature of WVSNP-DASH enables the player to perform arbitrary fast forward (FF) and rewind (RW) of the video. The naming syntax of WVSNP-DASH also enables the player to take advantage of all DASH features, such as adaptive switching, where quality switching index can be changed for the next segment. For example based on the power budget of a source node, a player can decide that the next segments must be of the lowest quality, (lowest resolution, lowest bit-rate) in order to extend the life of the source. This decision can even be based on the clients' computer vision feedback indicating dangerous activity to be monitored longer, therefore needing to preserve power or high resolution capture switch based on activity seen in the previous segment. Or based on previous segment, the client might decide to not use too muck power and just fetch very low quality/power segments since there is little activity to see. Note that this is another level above compression types used within the video file segment itself. The details of the player operations and options are in the framework section of [182].

5.3 Evaluation Criteria And Setup

5.3.1 Evaluation Set-up

A WDP prototype is used to playback WVSNP-DASH segment files and other DASH video frameworks where possible. Where not possible, other popular DASH players were used to play video segments and compared to other DASH players' effects on power consumption. Specifically, comparisons with HLS used the JWPlayer 6 with the HLSProvider plug-in. For MPEG-DASH the DASH-JS player module was used from within the prototype WDP. Google Chrome (version 32) web browser was used as the default client display outlet. The clients ran on a *Ubuntu 13.10* 64 bit, *Dell OptiPlex 360* with *Intel Core2 Duo E7300* 2.66GHz CPU and 2 GB RAM. The evaluations were also run with a client operating on a *Macbook Air Mid 2012* with *i5-3427U* 1.8 GHz CPU and a 4 GB RAM. For redundancy checks, the same Macbook Air was rebooted into a *Windows 7* 64 bit client. These gave similar results, which are not included due to redundancy and space constraints.

For the prototype evaluation, a typical surveillance video was captured on the campus of Arizona State University (ASU). The 10 minute ASU video (without audio) contains timelapse captures of everyday outdoor activity and scenery around the ASU campus. The ASU video was used for both VOD and LIVE video scenarios. For the LIVE video scenario, a node camera was pointed at the full-screen display of the pre-recorded video while measurements were captured in real time. Focusing and positioning the node camera to see only the full-screen video display made the LIVE video essentially identical to the pre-recoded video. The H.264 encoded .mp4 video was generated in two quality versions, namely a "SMALL" version with 320×180 pixel resolution, 150 kb/s bit rate, and 15 frames/s, and a "BIG" version with 640×360 pixel resolution, 500 kb/s, and 25 frames/s. Each quality version was segmented into MPEG2 Transport Stream (TS) segments for HLS playback, independent MP4 segments for WVSNP-DASH playback, and ISO Base Media File Format (BMFF) segments for MPEG-DASH. The segments were created for 2, 5, 10, and 15 second segment lengths.

The server node power consumption was measured from an *NXP i.MX6 ARM Cortex-A9*, 1.2 GHz Quad core, 2 GB node development board. The server node captured video with a USB webcam for some cases and a Camera Sensor Interface (CSI) attached Wandcam [68]. The captured video segments are served via WiFi and a streamlined low-footprint mongoose HTTP server. Where the Quad-core board was not available nor necessary some data was collected using an almost identical Dual-Core board. This will be noted where needed including how the data was reconciled to still be a useful and valid trend. Other comparison parameters were focused on the critical parts of the WVSn's "capture-store-stream" pipeline.

For example, power consumption results were performed while capturing the video with different camera interfaces, (USB and CSI) facing a monitor running BIG reference video using a dual board of different segment sizse (2sec, 5sec and 10sec) and encoding the segments with different codecs. A comparison was then made between HLS and WVSNP-DASH. HLS used an MPEG2 container versus WVSNP-DASH's MP4 container. Since ffmpeg currently doesn't support HW encoder for i.MX6 processors, results were only collected using a SW encoder. A comparison between USB camera and CSI camera were done by capturing WVSNP-DASH segments with ffmpeg with each of the cameras using an H.264 SW encoder (libx264). These were repeated using the Gstreamer tool and encoded with both a HW encoder and the same H.264 SW encoder. Using this data, different comparisons were made between CSI and USB camera interfaces, HW and SW H.264 encoders and other perspectives on the data. The same setup was repeated but this time using HW and SW MPEG4 encoders. Another iteration uses HW and SW MJPEG encoders.

Other special setups will be described below along with the data as were used to tabulate and graph different comparisons plots. These include comparison between SW and HW encoders on same camera, a comparison between same encoders on different cameras and so forth.

This section tabulates all the data from the criteria above and will be referred to in detail on the analysis of results that follow. The measurements were done on at least five separate occasions and locations to validate consistency. The averages shown are typically the average of more than 14000 or more data points sampled by the oscilloscope SW tool for the entire ten (10) minute run in one of the measurements. Observations showed the measurements were consistent.

Figure 5.1 above shows the partial sketch of the actual tools used in Figure 5.2 below. It shows the oscilloscope, current clamp and oscilloscope probe. One side of each probe and clamp are connected to the oscilloscope to measure current and voltage and in turn, oscilloscope is connected to laptop to view the current and voltage measurements and save it on the laptop. To measure current, clamp is placed around the wire connected to the board for powering it up, while voltage is measured by connecting probe to the 5V jack on the board.

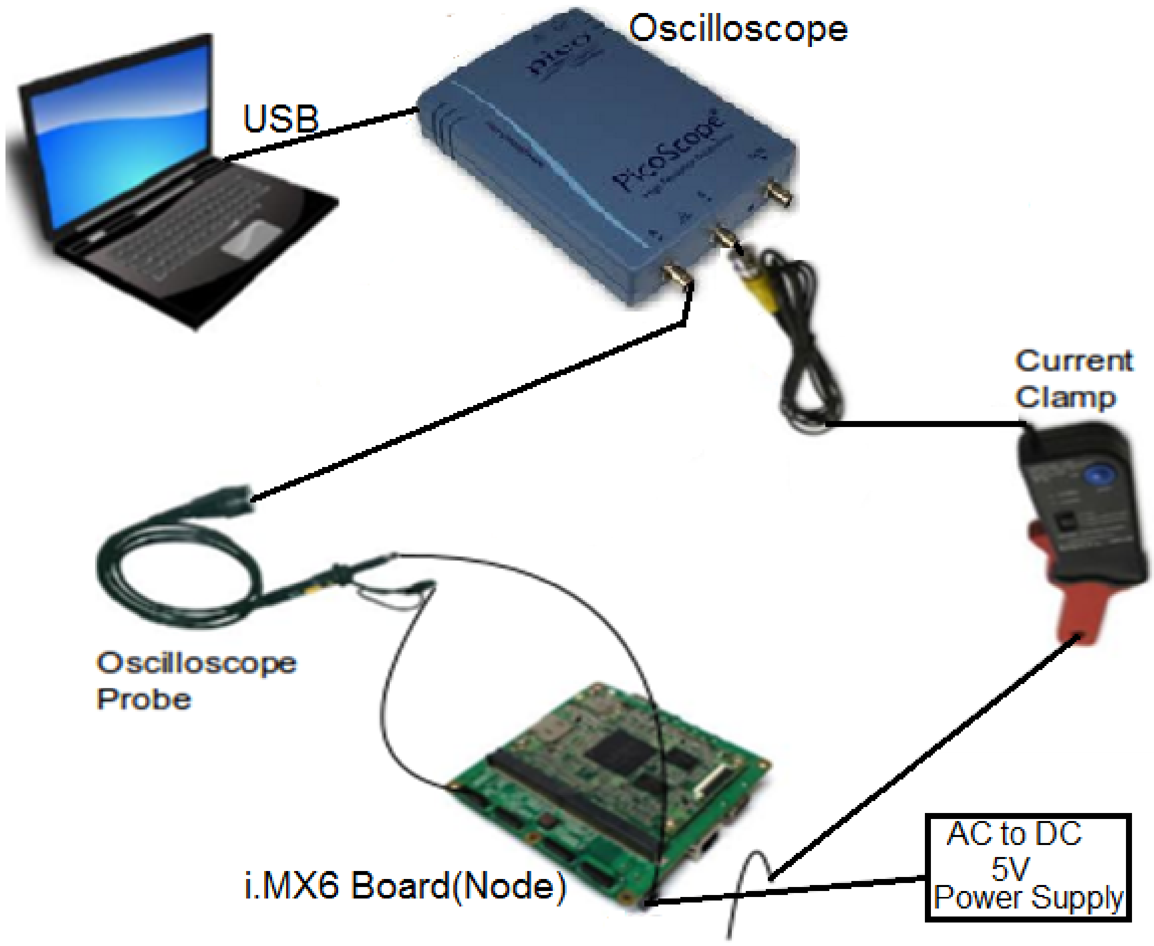


Figure 5.2: Tools for power measurement and data logging.

To normalize the measurements, several measurements were taken from boot time till end of video capture as shown in Figure 5.3. Major four stages from pre-boot,

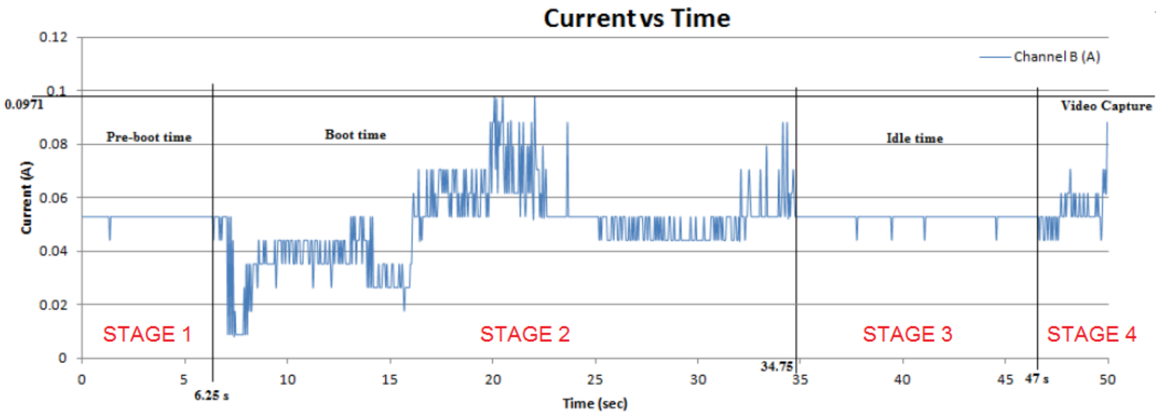


Figure 5.3: Four stages for node power measurement and data logging.

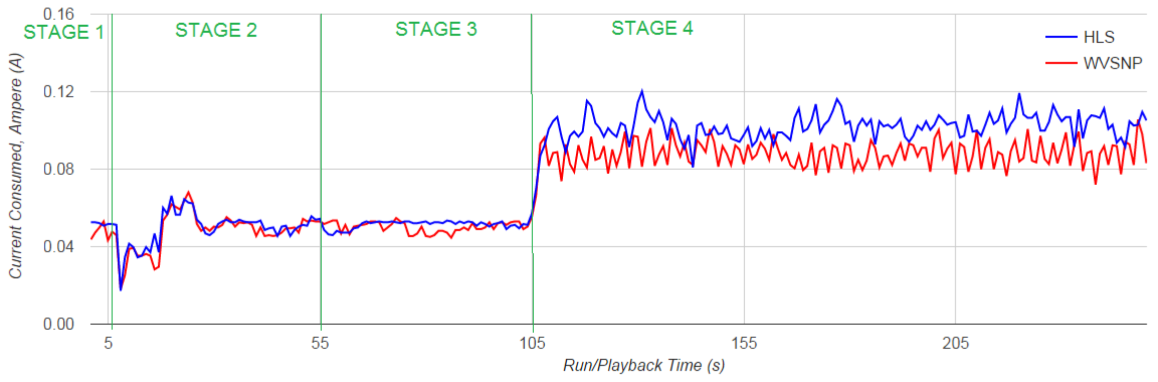


Figure 5.4: Four stages for node power measurement and data logging. 2s segment LIVE capture example.

boot, idle and video capture were observed for consistency.

After verifying that the graph/data looks the same for the first three stages, the rest of all other collections start at the second idle stage just before capture/stream. All the data presented in this section uses the these four stages as a reference point. But the averages were calculated only within the fourth stage. That is, during video capture or stream. Figure 5.4 shows the first 250 seconds of 2 second segment boot and capture stages.

The tables in section 5 present summary evaluation results for how the framework chosen affects power at the node. Comparison across HLS and WVSNP-DASH frameworks shows some interesting patterns.

5.3.2 Framework Power Profiling

The main objective of the WVSNP-DASH framework is to work with sensor nodes. Low power consumption at the server side, i.e., the sensor node, which captures, stores, and serves the video files, is critical. Data collected to evaluate these is in milli-Amperes (mA). We believe this is representative enough to reveal the trends as voltage was pretty much a constant at 5V.

Table 5.1 below shows comparison of current consumed by the node while streaming WVSNP-DASH vs HLS. It compares 2s, 5s, and 10s segments for Video On Demand (VOD). Since most of the Linux OS and internals are open and accessible, most of the detailed analysis focuses on the frameworks being evaluated from the clients running on Ubuntu. The WVSNP-DASH implications for the power consumed by the server node are examined. Again, this is derived by comparing power consumed by the node server while serving each of the different types of the player frameworks.

As mentioned in the previous section, HLS in this experiment was implemented by the JW Player and HLSProvider. It buffered segments differently from WVSNP-DASH. HLS had a buffer window maximum of thirty seconds to a minute. It also starts streaming only after it has buffered at least one segment. This is useful for VOD server power because the node is not being used nor triggered into network and file IO activity unless the segment is really needed. WVSNP-DASH on the other hand blindly buffered three segments before it could start playing. Once playing resumed there was no maximum buffer window. In VOD mode, the WVSNP-DASH client continues fetching segments until there are none available or stream has finished. This means, while playing back the buffered three segments, a separate process continues

Table 5.1: Averages and standard deviations (SD) of current (mA) consumed by the node while streaming WVSNP-DASH versus HLS. It compares 2s, 5s, and 10s VOD segment lengths.

WVSNP-DASH vs HLS, Video On Demand (VOD) segments.

	2 seconds				5 seconds				10 seconds			
	WVSNP		HLS		WVSNP		HLS		WVSNP		HLS	
Link, OS	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD
WiFi, Mac, BIG	38.23	9.77	46.90	8.79	36.30	9.24	45.57	8.98	36.36	8.88	34.80	8.86
WiFi, Ubuntu, BIG	51.10	8.43	48.83	8.04	51.51	6.95	37.46	9.92	50.05	7.00	36.68	8.57
WiFi, Ubuntu, SMALL	42.00	9.51	39.70	8.64	40.39	9.09	43.31	8.58	37.83	8.76	38.95	8.17
WiFi, Windows, BIG	40.23	9.84	37.42	9.36	39.71	9.42	35.14	8.79	37.81	9.01	40.04	9.34
Ethernet, Windows, BIG	41.26	9.98	40.32	9.28	39.89	9.63	39.81	8.86	38.32	9.55	38.03	8.92
Ethernet, Mac, BIG	42.23	9.84	38.70	9.49	41.12	8.69	37.64	8.74	39.88	9.13	36.82	8.88

to fetch files in the background.

HLS also uses Transport Stream packet that has extra information to lessen latency and increase greater error resilience. Additionally the HLS packet stream interleaves the frame data across packets which means HLS segments have information about each other and are dependent on each other. While this dependency is what WVSNP-DASH avoids it is helpful for HLS under VOD mode as that packet information is used to schedule segment fetching and buffering in a more intelligent way with transport information feedback. For example, the Forward Error Correction (FEC) embedded in the packet can be used to recover lost or damaged packets instead of re-fetching the segments again (as WVSNP-DASH would). This can save power on the node, especially in VOD mode.

Theoretically the above make HLS more efficient due to its stream type and its in-built error resilience. But we can't just rely on what makes common sense. We have to measure and get surprised or confirmed. For example, we cannot guarantee if duty cycling consumes less power in total or on average than the peak short burst of continuous fetches, then staying mostly idle. For example a more refined test would be to match the fetching and buffering of HLS and then compare the case "with the match" against with te current continuous fetch for WVSNP-DASH only. During experiments there were some slight stalls during playback now and then. How WVSNP-DASH or HLS reacts to these stalls can be hard to show but can be measured.

The above playback behavior by WVSNP-DASH also means a quality switch would waste power as all pre-fetched segments will be ignored if not discarded to get all the new representation quality segments. This can happen with fast forwarding and rewinding as well. The buffering might not check if the segments already exist in the file system when it re-initiates the 3-segment window sequence. The prototype blind buffering of WVSNP can be improved to match the packet structure feedback enhanced streaming of HLS. So, for cases where HLS in Table 5.1 seems to consume a little bit less power, it is likely due to its polished buffering scheme which can be matched by WVSNP-DASH with more experimentation.

Another theoretical negative for WVSNP-DASH is that each of its segments have complete self-initializing header data. That is, each WVSNP-DASH segment contains a file header with the video file properties and video container metadata, e.g., compression type, bit rate, and size, needed for decoding at the client (player) side [182].

This would imply that each WVSNP-DASH segment is slightly larger than HLS media segments that follow an HLS initialization segment. Empirical data however indicated that for the same video resolution and quality, WVSNP-DASH segment files are on average smaller than the corresponding HLS segment files. The sizes vary widely within a range from segment to segment due to motion based video compression. As can be seen in Table 5.2, HLS segments were found to be larger than WVSNP-DASH segments.

Table 5.2 shows on average, for 2 s segments, the HLS BIG segments are 21 % larger than the corresponding WVSNP-DASH segments for same video segmented. The SMALL HLS segments are 28 % larger than the SMALL WVSNP-DASH segments. For 5 s segments, the HLS BIG segments are 14 % larger than the corresponding WVSNP-DASH segments, while the HLS SMALL segments are 9.7 % larger. For 10 second segments the HLS BIG segments are on average 11.5 % larger, while the HLS SMALL segments are 9.7 percent

From a review of the container structures used by the frameworks. The larger HLS segments might be due to the HLS use of a 188-byte MPEG 2 Transport Stream (M2TS) packet size. This was originally chosen for compatibility with ATM systems. The packet size can get larger with additional headers, e.g., for synchronization, time code, adaptation, broadcasting meta-data. On the other hand, WVSNP-DASH is container agnostic, i.e., one can select whichever container is most efficient for WVSNP-DASH streaming. The MP4 container used by WVSNP-DASH in the evaluations follows an atom/box structure in a hierarchical form with four bytes for the atom length, four bytes for the atom name, and optional bytes for any data the segment holds. The length of the box is determined by its own size plus all atoms in the level immediately below it. A basic WVSNP-DASH MP4 segment has three boxes: `ftyp, moov` and `mdat`. This is one other reason just using WVSNP-DASH

Table 5.2: Averages and standard deviations (SD) of video file segment sizes (kilobytes) between HLS and WVSNP-DASH. Comparison for both SMALL and BIG video quality representations.

BIG quality representation segment sizes (kB).

	2 seconds		5 seconds		10 seconds	
	WVSNP	HLS	WVSNP	HLS	WVSNP	HLS
Average	113.18	137.17	299.85	342.79	613.96	683.57
Std. Dev.	206.77	12.67	73.21	25.15	118.82	35.94
Lowest	329.00	77.70	39.70	227.30	130.40	520.90
Median	682.30	110.45	113.90	295.90	322.65	616.00
Largest	1200.00	191.30	410.50	386.40	713.60	714.90

SMALL quality representation segment sizes (kB).

Average	212.08	38.24	49.07	97.39	106.83	193.31
Std. Dev.	79.09	4.18	20.01	7.82	44.15	9.29
Lowest	19.30	26.80	14.00	73.90	19.30	170.00
Median	202.30	37.35	46.10	96.85	102.35	194.65
Largest	465.40	60.20	128.90	117.20	315.20	214.50

Table 5.3: Averages and standard deviations (SD) of current (mA) consumed by the node while streaming WVSNP-DASH vs HLS segments in Real Time Streaming (LIVE).

WVSNP-DASH vs HLS, Real Time Streaming (LIVE) segments.								
	5 seconds				2 seconds			
Video Type	WVSNP		HLS		WVSNP		HLS	
Link, OS	Avg	SD	Avg	SD	Avg	SD	Avg	SD
WiFi, Ubuntu, BIG	98.30	28.96	95.80	29.56	85.05	24.50	106.24	30.82
WiFi, Ubuntu, SMALL	87.58	23.33	92.84	25.34	80.52	20.66	90.04	24.80
Ethernet, Windows, BIG	100.17	27.98	104.23	29.98	.	.	.	.
Ethernet, Mac, BIG	91.59	26.04	101.75	29.73	.	.	.	.
WVSNP-DASH and HLS: LIVE minus VOD cost difference between Ubuntu rows in this Table 5.3 above and Ubuntu ones in Table 5.1								
WiFi, Ubuntu, BIG (Delta, mA)	46.79	29.78	58.34	31.18	33.95	25.91	57.41	31.85
WiFi, Ubuntu, SMALL (Delta, mA)	47.19	25.04	49.53	26.75	38.52	22.74	50.34	26.26

can save power. The segment file size measurement results indicate that the MP4 container utilized in WVSNP-DASH requires on average less overhead than the default HLS M2TS container. Generally, smaller segment files consume less power than large segment files; thus, the more efficient MP4 containers that become possible with WVSNP-DASH are generally preferable for low-power streaming.

Table 5.3 below shows comparison of current consumed by the node while streaming WVSNP-DASH vs HLS. It compares 2s, 5s, and 10s segments for LIVE video. For Ethernet the trend confirmed WiFi trends, so only the critical 2s and 5s segments are noted for network emphasis.

A quick comparison between Table 5.1 and Table 5.3 indicate that the LIVE scenarios have significantly higher current draw and thus higher power consumption than the VOD scenarios. This is because the server node captures, transcodes, stores, and serves the video segments at the same time. Importantly, for the LIVE scenario, the WVSNP-DASH results in less power consumption at the node than HLS as shown by Table 5.3.

Table 5.1 reveals a lot of interesting new topics to explore depending on how you read the rows and cross comparisons that are shown by the data. For example the same experiment was performed on SMALL size quality segments versus BIG quality segments. As expected, SMALL quality segments in general perform better than BIG quality segments for all segments and both frameworks due to their low data rate and size. We can also add, the higher the speed of transmission which reduces the temporal component of power loss as well. This is confirmed for both VOD and LIVE cases in spite of one or two outliers for HLS's 5s and 10s which we attribute HLS player's advanced and mature buffering techniques which are not employed yet in the WDP prototype used in the testbed. For both WiFi and Ethernet transmission and across all operating systems, larger WVSNP-DASH segments consume less power on the node compared to smaller ones. This appears to be the case as well for HLS except for a couple of outliers. Another surprising trend is that for a WVSNP-DASH client playing back over WiFi there is less power being consumed than if the streaming is over Ethernet. This is the case for all VOD segments. This trend seems to hold for HLS as well except for two outliers when the client is playing back 2s and 5s segments on a Mac and once case on Windows for 10s segments. These outliers based on confirmatory tests do not affect the trend conclusions.

Another interesting data trend from Table 5.1 is that at first, one might be tempted to conclude that WVSNP-DASH is under-performing HLS for all VOD cases except when playing back on a Mac client over WiFi. When interpreting these power results, it is important to note the differences in how we are using ffmpeg for segment capture in the compared frameworks. HLS captured and transcoded LIVE video in an optimized built-in (native) HLS function of ffmpeg, which achieves highly efficient SW-based capture and transcoding. On the other hand, a WVSNP-DASH capture used a prototype-level bash script that invoked ffmpeg for each LIVE segment capture. This means that the WVSNP-DASH prototype incurred extra power, resources, and inefficiencies for each context switch of launching ffmpeg, transcoding, storing, and then shutting down the ffmpeg process for each video segment capture. Moreover, WVSNP-DASH interpreted a script at run time for every segment, adding to the resource usage. In contrast, HLS capture invoked ffmpeg only once at the start of the video stream capture and captured the remaining segments with the same optimized ffmpeg (from the original invocation context). These conceptual differences imply that an optimized native WVSNP-DASH capture application has considerable power savings potential for LIVE video in the WVSNP-DASH framework compared to the already optimized HLS framework. This will become apparent when we discuss LIVE streaming results shown in Table 5.3.

Since Table 5.1 is VOD all the video is already available at the start of the stream and all initialization files and manifest files have been precomputed and final for HLS. WVSNP-DASH segment sizes and fetch structure are the same for VOD and LIVE. Figure 5.5 shows the simplified structure of WVSNP-DASH capture-store-stream flow.

HLS on the other hand as shown in Figure 5.6, has a structure which behaves differently when streaming VOD than when streaming LIVE.

So the slight HLS advantage on VOD can only be attributed to structural file organization and VOD mechanics. Note in Figure 5.6 that HLS has a *Stream Segmentor* stage which adds to the workload process during capture. This does not exist in WVSN-DASH as shown in Figure 5.5. During VOD this stage is not active in HLS, therefore saving power and making HLS look like it is more efficient. Another VOD HLS relative power savings comes from the fact that WVSNP-DASH segments for the same video data have self initializing segments which implies that each segment is slightly bigger than media segments of HLS that follow an initialization segment. Initialization segments have extra file header information and other metadata that

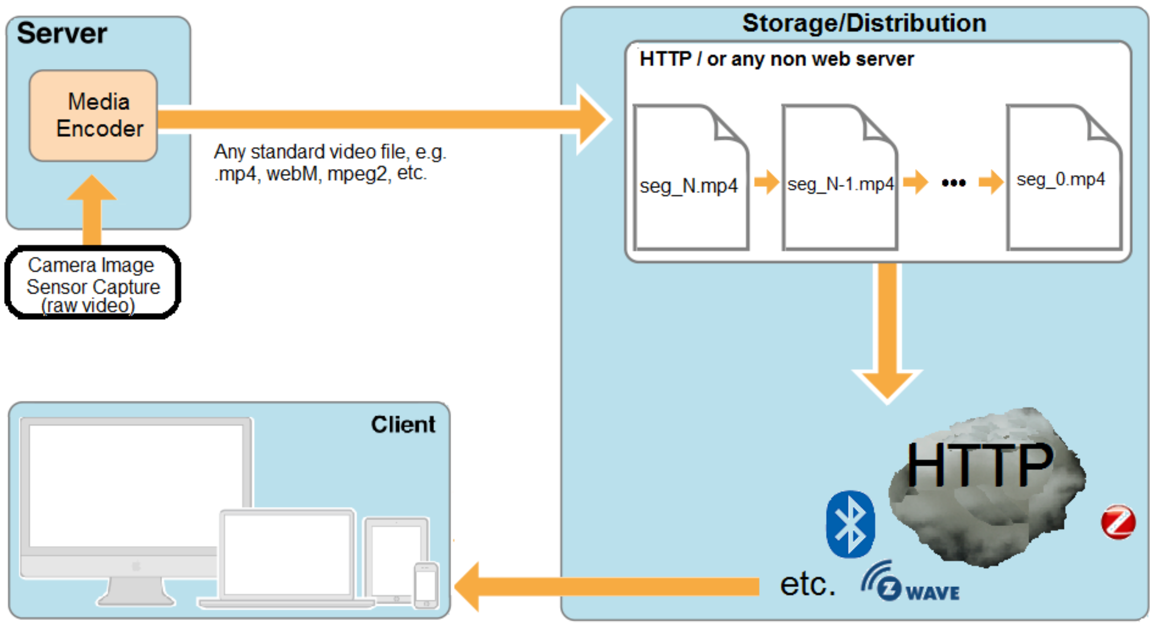

Server
Media Encoder
Camera Image Sensor Capture (raw video)
Any standard video file, e.g. .mp4, webM, mpeg2, etc.
Storage/Distribution
HTTP / or any non web server
seg_N.mp4
seg_N-1.mp4
•••
seg_0.mp4
HTTP
etc.
WAVE
Client

Figure 5.5: The simplified structure of WVSNP-DASH capture-store-stream flow.

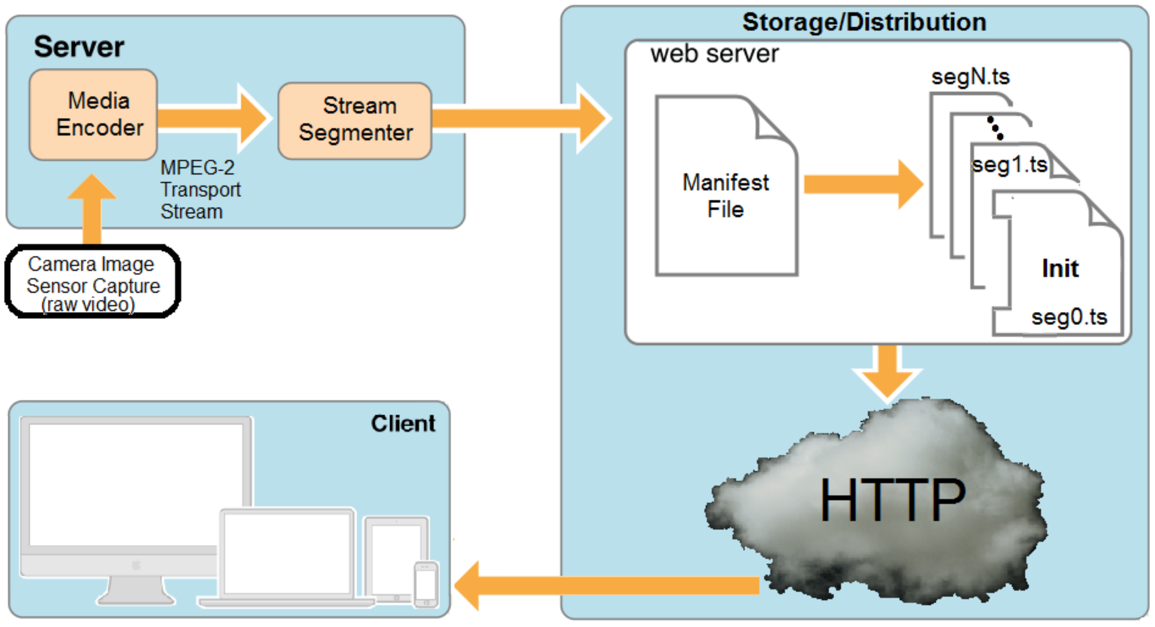


Figure 5.6: The simplified structure of HLS capture-store-stream flow.

Table 5.4: Averages and standard deviations (SD) of current (mA) consumed by the node while capturing WVSNP-DASH versus HLS segments. It compares 2s, 5s, and 10s segment lengths.

WVSNP-DASH vs HLS, cost of capturing segments. Current Consumed (mA).

	2 seconds				5 seconds				10 seconds			
	WVSNP		HLS		WVSNP		HLS		WVSNP		HLS	
	Avg	SD	Avg	SD	Avg	SD	Avg	SD	Avg	SD	Avg	SD
Node Capture Cost	77.07	17.00	79.84	17.00	80.66	16.41	79.77	16.41	81.56	16.38	83.79	16.48

help player or browsers decode the complete video stream. See media stream technical reports at [192, 226, 191]. As discussed already above, smaller segments consume less power than big ones. Another part of HLS that is not being exercised during VOD streaming is the precomputed manifest files. Under normal LIVE streaming tis would continuously need to be updated and re-transmitted with every new segment generated. This saves HLS some power. The capture stages for HLS and WVSNP-DASH were isolated to demonstrate that HLS is less efficient than WVSNP-DASH at the capture stage as reported in Table 5.4. this demonstrates just the cost of capture at the node without any networking nor transmission costs. Note that even though an inefficient script loop that opens and closes ffmpeg for every WVSNP-DASH and is interpreted at runtime, WVSNP-DASH still used less power in most cases. This clearly shows that if both frameworks used optimized native executions, HLS would be much worse at this stage. With that isolation experiment, it can be concluded that in spite of some misleading trend from Table 5.1, WVSNP-DASH actually is more efficient than HLS.

Table 5.1 also reveals another interesting point. That is, capturing two (2) seconds segments does not necessarily result in higher cost as one might expect. Logically this would be due to the more frequent file operations inherent in handling more files for the same length of video captured. The data seems to imply this is not generally the case for both WVSNP-DASH and HLS. So, the trend in Table 5.4 is due to the streaming component. The data shows a slight edge for WVNP-DASH which is has a much simpler: camera open, capture and store as opposed to HLS's need to repack segments to be relative to initialization segment and additions to the manifest files which costs time and energy.

To focus only on the LIVE effects on both frameworks, only 2s and 5s segments were studied since segment length effects are known from Tables 5.4 and 5.1. As discussed for VOD results, Table 5.3 further shows more clearly the power savings arising from the WVSNP-DASH framework's non use of a manifest file. Again HLS and MPEG-DASH require a manifest file that needs to be managed and re-read and updated during the capture and/or playback [18, 192, 226]. For VOD, the manifest files and segments are static and the indices do not need to be continuously updated. However, for synchronization of LIVE video, the manifest files have to be typically re-fetched regularly for LIVE video synchronization. Additional processing to create special subsequent segments different from the initialization segment adds to the power consumption of HLS and MPEG-DASH for LIVE video. Another reason is as mentioned above that LIVE HLS requires an additional stream segmentation stage in te capture stage, which needs to be fully active during LIVE playback consuming more power than WVSNP-DASH which does not need this as shown in Figure 5.5 and Figure 5.6.

Please refer to a previously published paper [182] which examined the impact of the segmented video streaming on HLS versus MPEG-DASH versus WVSNP-DASH. In [182] a full 10 minute ASU video was streamed via progressive download and the results indicated that *10s* segment HLS and MPEG-DASH streaming consumed less

Table 5.5: Averages and standard deviations (SD) of current (mA) consumed by the node while streaming WVSNP-DASH versus normal un-segmented ten (10) minute long video (progressive).

Progressive (Un-segmented 10 minutes video) versus WVSNP-DASH. Current Consumed (mA).

Video Type	Progressive		WVSNP (5 seconds)		WVSNP(10 seconds)	
	Avg	SD	Avg	SD	Avg	SD
WiFi, Mac,	43.25	9.44	41.11	8.68	39.90	9.13
WiFi, Windows,	43.90	8.72	39.71	9.42	37.81	9.01

power compared to full-video streaming. These results indicated that segmented video streaming does not lead to higher server node power consumption than streaming a full (unsegmented) video to an HTML5 element. The 36.7 mA measured for HLS for *10s* VOD segments was considered as the worst-case (maximum) current consumption expected of a WVSNP-DASH using a specialized native capture. More generally, the comparison of currents for *10s* VOD segments and full video download indicated that segmented streaming can result in about 15 % power savings compared to streaming progressively downloaded full videos. The paper also concluded that there is no benefit of using *15s* segments compared to *10s* segments. These results are confirmed in Table 5.5. As can be seen, both *5s* and *10s* WVSNP-DASH segment streams perform better than an un-segmented video stream. Table 5.5 also demonstrates how WVSNP-DASH compares with progressive video. Note that progressive video also represents the way HLS and MPEG-DASH other modes of streaming using byte ranges of an opened full un-segmented often large video file. Using both the Mac and Windows based clients WVSNP-DASH proves to consume less power. This is important as the node gets more benefits of fine tuned storage and distribution options while not losing more power. This shows that WVSNP-DASH actually saves power regardless of client.

It is worth stating that the World Wide Web Consortium (W3C) [232] in its efforts to standardize DASH technologies [192, 226, 191, 227] prefers initialization files since its focus is solely on the web and not power. Again as shown by Table 5.3, this is inefficient for power, especially for LIVE playback.

5.3.3 Node Component Data Path Power Consumption

This result section focuses on the main components of the node that are critical to the video capture, store and transmission data path.

Table 5.6 empirically shows the contribution of major video capture and transmission data path components. This data was used to influence the design of the node. For standardization, Ffmpeg [6] and Gstreamer (Gst) [80] video library tools were used as they are standard tools used in many embedded video systems. Unless if Ffmpeg was compared directly with Gstreamer, Ffmpeg was used where Gstreamer lacked capability, and vice versa.

The empirical data from Table 5.6 reveals a lot of information about the video capture and transmission options available and used in the node design. For example, the bottom row shows that contrary to popular use, Gstreamer tool, whether using USB attached camera or the specialized Camera Serial Interface (CSI) camera, performs better than Ffmpeg in terms of power used on the node. This result narrows down other evaluations above this row using other capabilities of the SoC used in this node. Gstreamer therefore becomes the most used tool to do most of the architectural analysis. The bottom row is actually not necessary to arrive at this conclusion as that has already been been shown on the CSI versus USB analysis using both libraries in the top two rows. This favors Gstreamer across all video file segments and sizes.

The first two rows analyze cases where the Logitech “webcam” was attached to the node via USB camera whilst a ”wandcam” camera was attached via the camera

Table 5.6: Averages and standard deviations (SD) of current (mA) consumed by the node while capturing WVSNP-DASH segments video. Comparison of major data path capture elements.

Comparison of major WVSNP-DASH video capture data path components.

Video Types		2s segments		5s segments		10s segments		Full Video	
Activity	Compared	Avg	SD	Avg	SD	Avg	SD	Avg	SD
SW Ffmpeg H264 Capture	USB	78.40	14.60	77.98	14.00	77.68	14.39	78.38	14.78
	CSI	69.32	15.42	72.88	15.19	71.58	16.20	69.80	15.77
SW Gst H264 Capture	USB	59.49	11.89	60.13	12.07	58.70	12.32	62.84	11.81
	CSI	47.96	12.79	48.79	12.51	48.25	12.82	54.79	11.17
HW Gst H264 Capture	USB	57.97	13.36	59.07	12.80	58.80	13.54	58.82	13.62
	CSI	40.66	9.38	40.23	9.53	39.17	9.80	37.87	9.42
Gst H264 CSI Capture	HW-ENC	40.66	9.38	40.23	9.53	39.05	9.90	38.15	9.17
	SW-ENC	48.02	12.73	48.83	12.47	48.67	12.39	54.79	11.17
Gst H264 USB Capture	HW-ENC	57.97	13.37	58.46	13.46	58.80	13.54	59.11	13.29
	SW-ENC	59.49	11.89	59.60	12.65	59.30	11.64	62.69	11.93
Gst JPEG USB Capture	HW-ENC	67.53	16.19	67.28	16.96	72.84	16.50	.	.
	SW-ENC	61.94	13.41	62.24	13.37	62.50	12.65	.	.
Gst JPEG CSI Capture	HW-ENC	43.55	9.67	.	.	.	.	.	.
	SW-ENC	56.01	13.49	.	.	.	.	.	.
Gst MPEG4 USB Capture	HW-ENC	65.30	15.96	67.26	15.73	66.55	15.64	.	.
	SW-ENC	61.00	13.37	61.56	13.31	61.73	13.84	.	.
Gst MPEG4 CSI Capture	HW-ENC	44.21	8.19	.	.	.	.	.	.
	SW-ENC	49.01	11.07	.	.	.	.	.	.
SW H264 CSI Capture	Gst	47.96	12.79	.	.	.	.	.	.
	Ffmpeg	67.57	17.34	.	.	.	.	.	.
SW H264 USB Capture	Gst	59.49	11.89	.	.	.	.	.	.
	Ffmpeg	77.92	14.87	.	.	.	.	.	.

serial interface (CSI). As expected the CSI camera should consume less power than a USB camera. Figures 5.7 to 5.9 shows an eample of the comparison graphs that produced the data in Table 5.6.

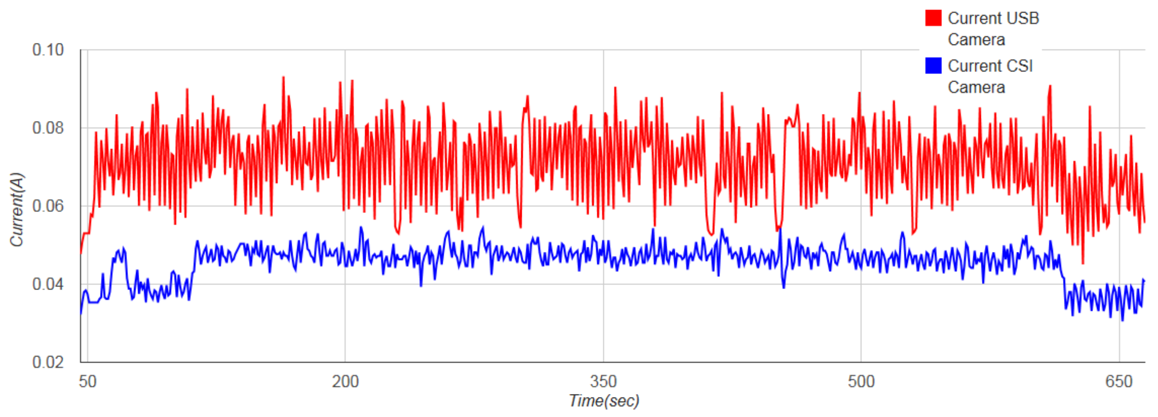


Figure 5.7: The HW accelerated video capture current consumption. Comparison between using a USB interface versus Camera Serial Interface (CSI) for 2s WVSNP-DASH video segments.

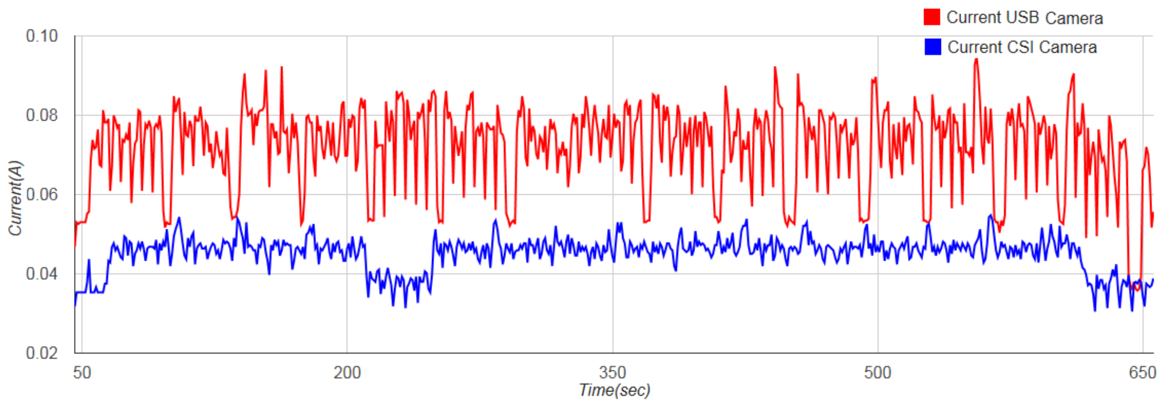


Figure 5.8: The HW accelerated video capture current consumption. Comparison between using a USB interface versus Camera Serial Interface (CSI) for 5s WVSNP-DASH video segments.

The USB camera processes its frames through the additional USB Video Class (UVC). Since the raw data collected by USB camera has to pass through more layered components, we can quickly conclude this increases the processing resources.

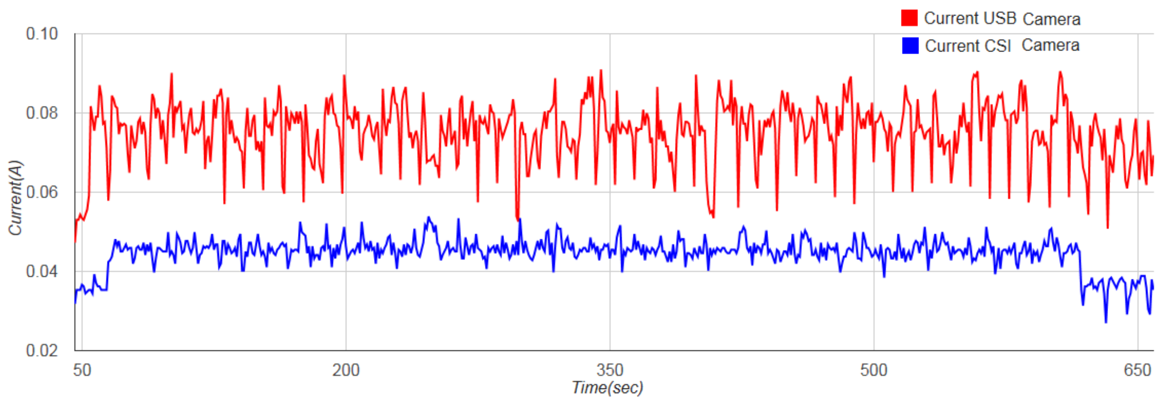


Figure 5.9: The HW accelerated video capture current consumption. Comparison between using a USB interface versus Camera Serial Interface (CSI) for 10s WVSNP-DASH video segments.

Additionally there is a SW hand-off of raw data on the board for video encoding to SW/HW encoder which increases the current consumption. The CSI stack is smaller and much more optimized for the board's (System-On-Chip) SoC which has a direct path from the Wandcam CMOS imager [68] driver to the encoder [190]. See Figure 5.10 below.

Top eight rows in Table 5.6, show the comparison of the full 10 minute video capture and the 2, 5 and 10 second segment cases. There is no evident difference between current consumption comparison of segmented video and full video in this case because ffmpeg and Gstreamer have a capability of segmenting the videos while the capturing is happening without turning off the camera. Ability to capture multiple smaller files instead of one big file without adding more power is noted as one of the features that make simple WVSNP-DASH segment capture more efficient than the HLS or MPEG-DASH post capture processing. As elaborated in section 5.3.2 above, the HLS post processing cost is shown by a slight increase in power consumed in the "capturing only" case for WVSNP-DASH versus HLS in Table 5.4. This is attributed to the manifest file updates that are needed after every capture and finalization of

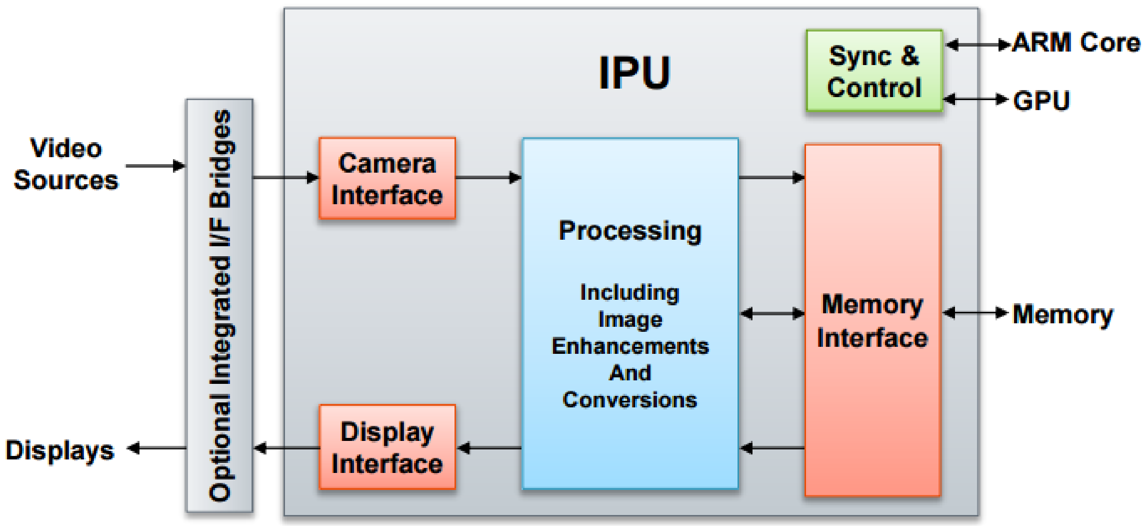


Figure 5.10: The i.MX SoC HW accelerated path from Camera to VPU to direct memory access. Note the decoupling of ARM Core from the entire capture to memory path [190].

the stream files.

Table 5.6 shows that video captured using HW encoder is more efficient than the SW encoder in most of the cases, except for JPEG and MPEG4, (and only when using a USB attached camera). HW encoding is performed by dedicated SoC components such as VPU for processing video data. A SW encoder uses board's CPU most of the time which increases load on CPU, which results in higher current consumption. Dedicated processors are more efficient if they rely on accelerated, optimized instructions which are job-specific. As a result significant savings in current consumption can be realized when using HW accelerated encoders. For the i.MX SoC used for this node, the video and image processing unit (VPU) in Figure 5.11 below is the HW accelerator.

As mentioned above, and as shown in Table 5.6's rows five and six and more, even for HW encoding, USB cameras consume more power. This is because of additional USB SW and HW that adds to the data path shown in Figure 5.10 above. This is

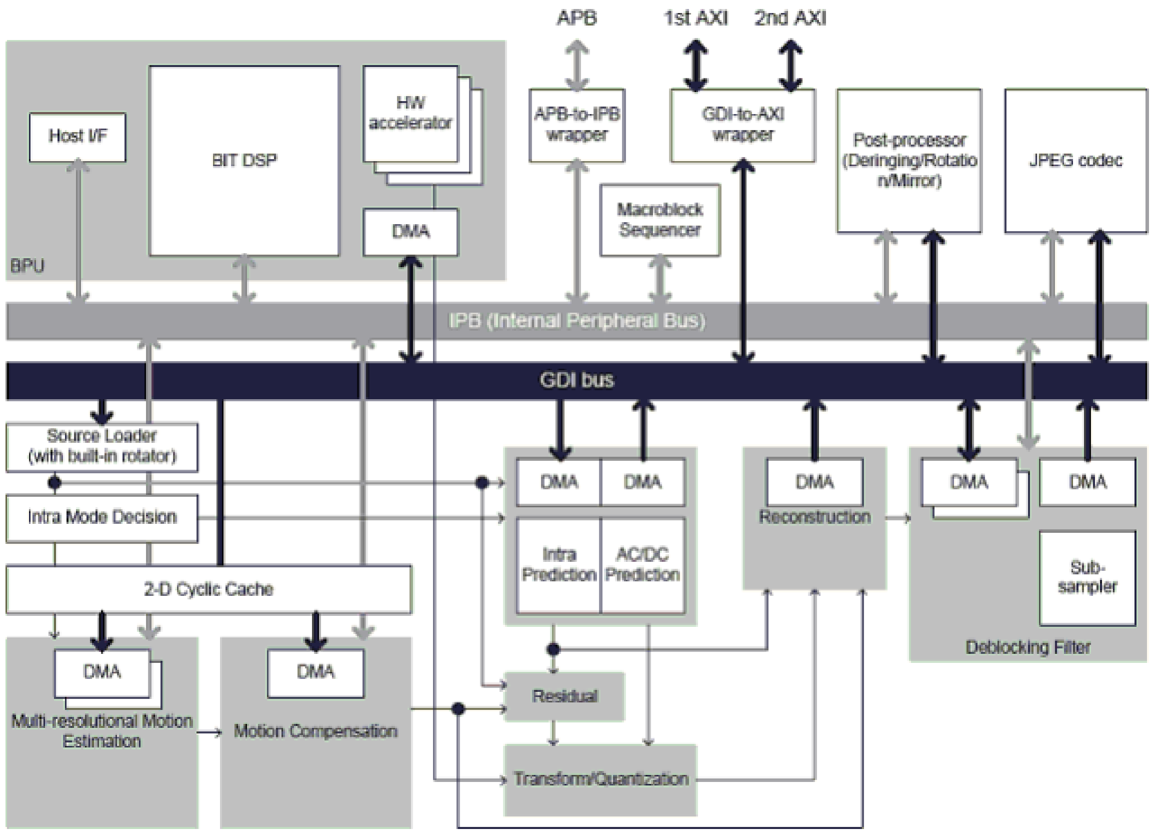


Figure 5.11: The i.MX VPU, this node's HW accelerator [103].

a synchronous serialized operation unlike the parallel asynchronous HW accelerated control shown in Figure 5.12 below. Additionally, since the USB stack does some kind of compression to its raw data and its many layers, there is too much energy drained within the USB pipeline that skews the difference seen between the current consumed by hardware or software encoding. Rows three and five of Table 5.6 still shows HW assisted USB camera is still more efficient than USB connected software encoder. In general a HW encoder considerably consumes less energy than the SW encoder for CSI camera, that is, more than 15% across the board.

We can conclude therefore, that HW encoded video captured using CSI camera is more efficient than using USB camera with the same HW codec. A trend to

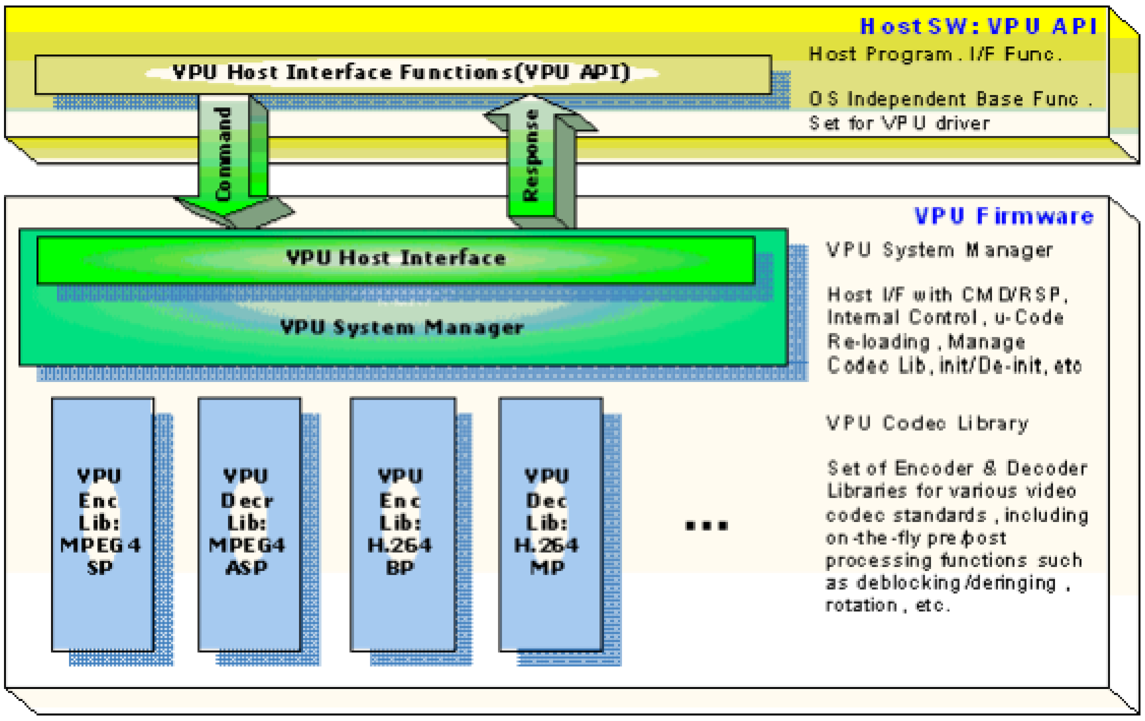

Host SW: VPU API
Host Program. I/F Func.
OS Independent Base Func. Set for VPU driver
VPU Host Interface Functions(VPU API)
Command
Response
VPU Firmware
VPU System Manager
Host I/F with CMD/RSP, Internal Control, u-Code Re-loading, Manage Codec Lib, init/De-init, etc
VPU Host Interface
VPU System Manager
VPU Codec Library
Set of Encoder & Decoder Libraries for various video codec standards, including on-the-fly pre/post processing functions such as deblocking/deringing, rotation, etc.
VPU Enc Lib: MPEG4 SP
VPU Decr Lib: MPEG4 ASP
VPU Enc Lib: H.264 BP
VPU Dec Lib: H.264 MP
...

Figure 5.12: The i.MX VPU host interface for the node's programs [103].

note is that, this efficiency increases with the segment size. We attribute this to application layer software hand-off of raw data to hardware encoder more often, for smaller segments therefore under utilizing the VPU to some extent.

More encoders were evaluated for the USB and CSI cameras beside H.264 (vpuenc, codec=6). For example, Gstreamer is used to compare SW jpegenc encoder and AVI (vpuenc, codec=0) HW encoder. That is, the Audio Video Interleaved (AVI) encoder container. It is interesting to note that for the AVI encoder in rows 12 and 13, the HW encoded segments appear to be less efficient than the software encoder. This is an unexpected result that needs more follow-up on the implementation of the encoder or even if the HW compressor is actually equivalent to Gstreamer's jpegenc SW encoder. But since the CSI capture seems to follow the expected trend, we attribute this to the extra USB pipeline and processing explained above. The same story applies to comparing mpeg4 SW encoder with the (vpuenc,codec=12) HW encoder.

5.3.4 Results Conclusions and Architecture Recommendations

Based on the results and experiences while collecting the empirical data, many conclusions have been reached and most expectations have been confirmed. It it, therefore useful to list some of the observations and points that could be used, for literature reference, architectural and node design decisions.

- Currently, gstreamer is capable of capturing video encoded with both software encoder and hardware encoder, while ffmpeg is only capable of capturing video encoded with software encoder. This is mainly because hardware acceleration wrappers for ffmpeg are not yet implemented for the i.MX VPU. This is a worthy research implementation effort to undertake for the near future. Ffmpeg has a capability of segmenting the videos while it is being captured without any loops nor scripts. Gstreamer requires scripts to capture in segments. Gstreamer wrappers so far appear to be more power efficient than ffmpeg while capturing the videos. A good effort for further research would be to implement a native WVSNP capturing application for both Ffmpeg and Gstreamer. These can both be compared natively for throughput and power efficiency. The node is capable of using either of the libraries as both are part of the WVSNP OS image (WOS).

- The results for hardware and software encoder comparison for avi and mpeg4 with video captured using USB camera were not as expected. The power consumption of encoding video using software jpeg and mpeg4 encoders through USB camera is somehow performing better than power consumption of encoding video using hardware jpeg and mpeg4 encoders. From analysis above, it is definitely recommended to use a CSI/MIPI based camera for the node instead of USB attached camera.

- During the live capture and real-time playback experiments, it was evident that HLS completed playback of the 10 minute "LIVE" video before WVSNP completed. In fact, it appeared that it took 15 minutes to capture and playback a 10 minute LIVE video. This was due to what was discussed earlier. That is, HLS live playback used a natively executing and optimized ffmpeg capture and segmenter whereas WVSNP capture invoked ffmpeg from script every time for each segment in a loop. The other reason was the respective players' buffering strategies. The HLS player buffered only one segment and started playback. If there was any live-capture interruption, the HLS player only needed to re-buffer one (1) segment and resume playback. The WDP on the other hand buffered three (3) segments before resuming. This meant that every LIVE playback interruption resulted in a three (3) segment additional delay before resuming playback. Therefore there is a need for a standalone native capture application for the node to approach real world LIVE playback timing. This will speed up video capture file preparation. Time lost due to buffering can then be used for live capture parallelization; threaded capturing and file preparation; and other asynchronous HW acceleration activities.

- It is recommended that WVSNs use WVSNP-DASH for video or multi-dimensional data streaming instead of HLS because it consumes lower power for all cases during live capture and real-time playback. An added WVSNP-DASH advantage is that, HLS is limited to only one or two containers and codecs (primarily MPEG2-TS), while WVSNP-DASH video segments can be encapsulated into MP4, AVI, MPEG2-TS, WebM and others. WVSNP-DASH is easier to implement for a server node and different OSes as it is officially playable on different browsers (Chrome, Windows Internet Explorer, Firefox), while HLS is only officially playable on Safari or Chrome only (at the time of these experiments). WVSNP-DASH is backward compatible, while HLS might not work with older version of browsers. And lastly, WVSNP-DASH does not require any type of browser modification to understand the video or manifest file format.

- Results showed that the power consumption cost of a USB camera interface is higher compared to cameras attached via CSI. This observation was true for either hardware or software encoders. The USB camera data path goes through multiple USB, peripheral and other bus stacks. This leads to data flowing through many blocks as shown by the long red arrowed line (1) in Figure 5.13. This contributes to increased power loss and data latency. USB interface also has an extra software hand-off of the data on the board which overwhelms any advantage HW acceleration had on the SW encoder. The CSI camera's SW stack is smaller/shallow and has a direct path from the CSI port to the VPU's encoders. See blue line, (2), in Figure 5.13 below.

  This work, therefore, highly recommends a CSI/MIPI attached camera for the node instead of USB attached cameras.

- There are two types of CSI (camera serial interface) cameras available (Parallel and Serial). Here a serial camera was used for measurements and is clearly recommended. There is a new MIPI standard camera. Parallel camera interface take lot of valuable pins around an SoC. but it is worth it to review power consumption results for a parallel camera to draw a comparison between them. This comparison will make it clear, which camera is efficient among CSI camera's.

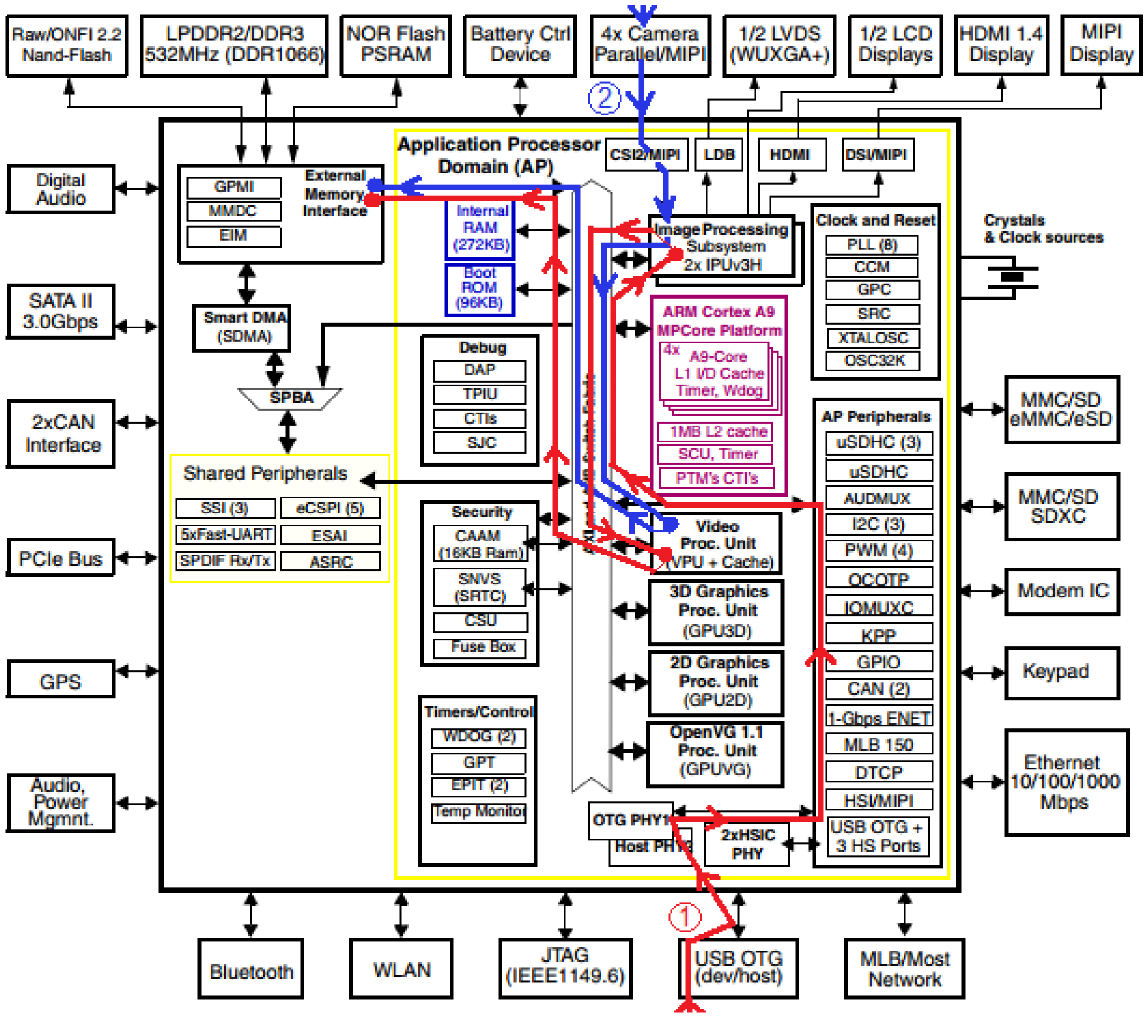


Figure 5.13: The i.MX block diagram showing video and image processing unit (VPU), vs USB pipeline. Note USB path (1), red, is much longer than CSI path (2), blue. From [103]

- The measurements were performed with three different segment length (2s, 5s and 10s). Based on the results, 10s is the recommended segment length. There are lots of fluctuations in the bandwidth. Depending on the buffering algorithm of the client if small segment sizes are used, they might be lots of flickering seen in the playback. Flickering occurs because adaptation of increase or decrease in video quality is never smooth due to the small segment size. With bigger segment size, if the video quality or segment length changes, adaptation will be smooth before the next segment is played. And, with bigger segment size, the coding efficiency also increases. Therefore 10s is a recommended segment size for both smoother playback and low power. In cases where the wireless link is more error prone or LIVE responsiveness is higher priority 2s segments would reduce the stall effects relative to the live event. Testing showed that for slow or high link error scenarios, retransmissions and buffering of 10s segments might cancel out the power consumption advantage and, therefore, favoring 2s segments. This relegates these last mile architectural decisions to the buffering algorithms depending on the event type and link environment.

- This work has confirmed that progressive download consumes more power than segmented video playback. Therefore segmented videos are recommended to be used than using a large video file. This is ideal for the duty cycle nature of WVSN applications and their limited storage space. A progressive download implies a big file in a node. This big file needs more energy to just open it before reading. While DASH clients request HTTP server to send them short segments. These segments are not required to be opened, they just need to be sent. With progressive download, the whole file is opened and a small portion of a file, (byte-by-byte) is sent. Now, if more portions (bytes) are required, the big file has to be opened over and over again and has to seek the past last position it was at some known point in the file. Therefore this opening and closing of large files and remembering the position of known point in the file consumes power.

- There are some implications with using BIG and SMALL video option. The difference between BIG and SMALL option is in the resolution, framerate and bitrate of the video. It is recommended to use BIG video option for better quality if the bandwidth is high, while SMALL video option is better for use if the bandwdith is lower.

- There were some segment length effects on LIVE and VOD sessions. For VOD, graphs showed that with an increase in the segment length, the number of segments also decrease and due to this the current consumption peaks are reduced. With smaller segment size, the number of segments are higher which dominates the segment to segment file transmission costs hence favoring longer segments. For LIVE video the opposite effects were observed. There was almost always a simultaneous segment being fetched and another being played during LIVE playback. This keeps the node in peak consumption state for a prolonged time. For both WVSNP-DASH and HLS, more power is consumed as the segment length increases. As the bigger segments gets packaged, saved and streamed there is more simultaneous file open/close or input/out (IO) and progressive chunked transfer activity. This cost is similar to how large progressive video was shown to consume more power in Table 5.5. The cost becomes more dominant in LIVE scenarios as the segment length increases. Since LIVE video also has a critical temporal component, it is actually good news that smaller segments consume less power as DASH algorithms require as little segment buffering as possible for realistic real time playback. This segmentation and file IO cost is very apparent in HLS which has a segmenter stage as shown by the delta calculations at the bottom of Table 5.3 and as contrasted in Figures 5.5 and 5.6. Note that the LIVE cost component jumps from around 50 mA to around 60 mA for SMALL and BIG segments respectively in HLS. Again WVSNP-DASH does not have the segmenter stage so, the cost of the, LIVE component only, actually reduces significantly for shorter WVSNP-DASH segment lengths and a relatively smaller reduction for bigger size segments. This is another empirical evidence of the advantage of WVSNP-DASH over HLS.

- There are other effects on BIG and SMALL video options during LIVE or VOD sessions. For VOD case with BIG option the current consumption of WVSNP-DASH is consistently higher than HLS. With the SMALL video option, both WVSNP-DASH and HLS consume less current but they are much closer to each other with WVSNP-DASH consuming less for most SMALL cases. In case of LIVE, with BIG video option WVSNP-DASH is consistently consuming less power then HLS as further shown in Table 5.3 across three operating systems and on both Ethernet and WiFi.

The data in this work was played back using the Application Programming Interface (API) for HTML5 File System (HTML5 FS), HTML5 Canvas and the HTML5 Video Element. The HTML5 FS at the time was promising to be the cross platform standard that can be used to seamlessly and uniformly across all platforms. This meant that the buffering algorithms, playback and capturing strategies took this into account together with the inherent power implications. As detailed in a prior publication [182], there are power efficiency costs to using the trio of APIs above, especially on the client device. As time passed, it became apparent that the adoption of HTML5

FS was not going to be as uniform across browsers as expected. Additionally there are yet un-studied power implications versus other web browser/mobile API. One such obvious concern is that a video segment file encapsulation means that the entire video file has to be re-downloaded to switch video bit rate or presentation quality. This is a waste of power even if the client has limitless buffer resources to pre-load all possible representations. An alternative emerging API is the Media Source Extensions (MSEs), which have been developed by the World Wide Web Consortium (W3C) [227] with clearly defined byte streams for the mobile era [192, 226, 191].

5.3.5 MSE Architecture Recommendations

MSE seems to be getting more broad support across all popular browsers. This work did some trial experiments on MSE whose observations serve as a basis for an even more efficient adaptive streaming framework.

MSE adds a buffer-based source options to HTML5 media for streaming support. Like WVSNP-DASH it does not expect browsers to change to support whatever the new streaming framework is. It relies on client side programming to use just one HTML5 video element. In [182] it was detailed that WVSNP-DASH uses two video elements to improve appearance of smoothness as it renders video on the HTML5 canvas. This has been shown to result in high memory, CPU and power consumption on the client. MSE pretends that just one media data buffer is feeding one video element. See Figure 5.14. The data appended to the *SourceBuffer* is playedback by the *MediaElement* as track buffers for audio, video and text data that is decoded and played.

The problem with the [227] definition/prescription is that it assumes that the media data is in the form used by the popular fragmented DASH formats as defined in [192, 226, 191]. This as mentioned before forces HLS and MPEG-DASH to have

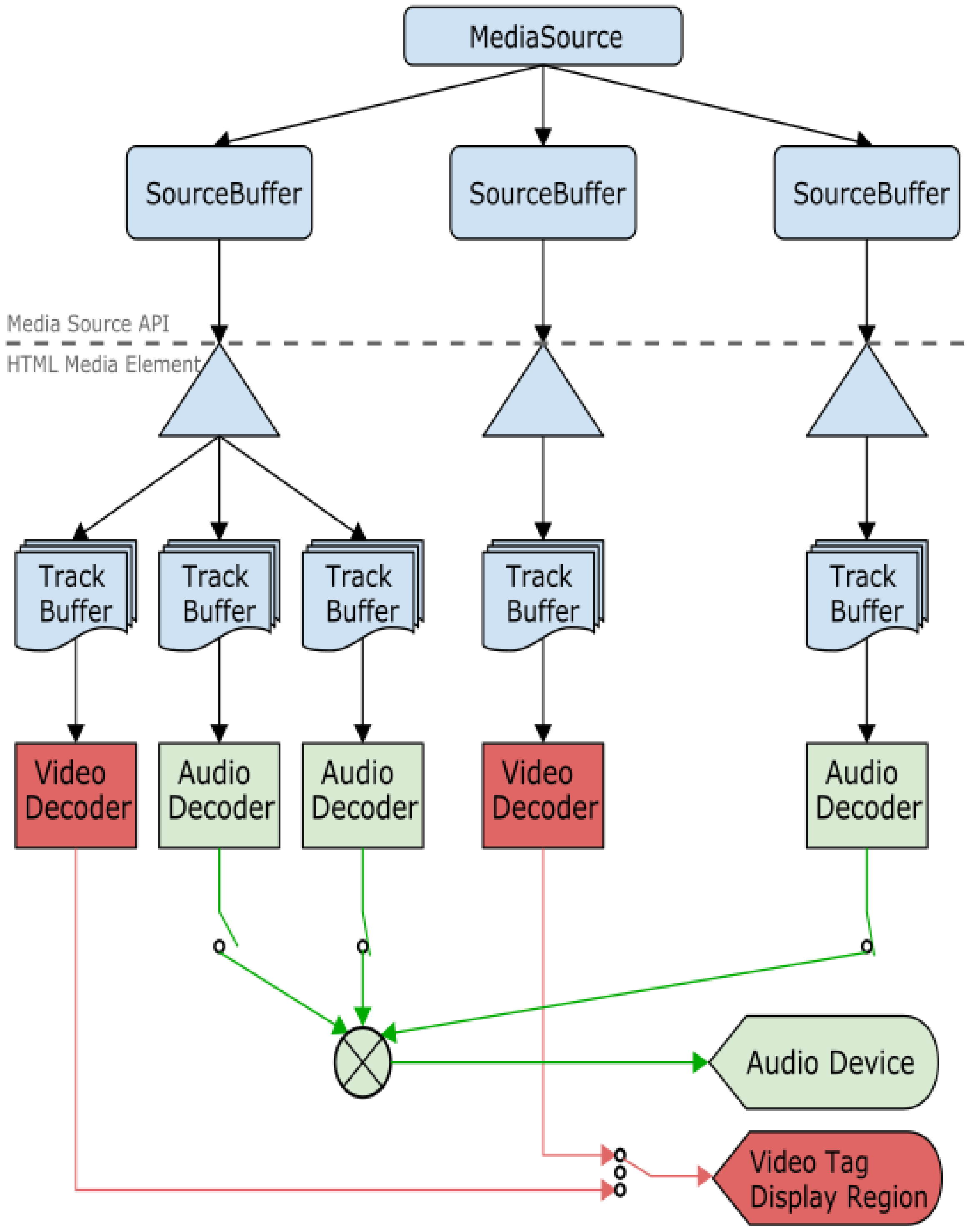


Figure 5.14: The Media Source Extensions pipeline definition in the W3C Recommendation [227].

initialization files for the subsequent segments to work. Though it is not a requirement in the specification for the rest of the stream segments to depend on one initialization segment, the DASH-IF [207] enforces this. This work takes advantage of this loose requirement to make WVSNP-DASH work with MSE in spite of all its segments being independent self initializing video files. There are other creative segment data manipulations that were done to enable pure MSE playback to conserve power. MSE allows a large video file to be streamed by fetching only parts of its data independently as described typically in manifest files as byte range chunks. WVSNP-DASH does not have manifest files, so it takes advantage of this by assuming every segment's data starts at data byte chunk zero. Then all subsequent segment chucks are expected to be the same size except for the last chunk. This enables the client to playback longer segments without downloading the entire ten (10) second segment for example.

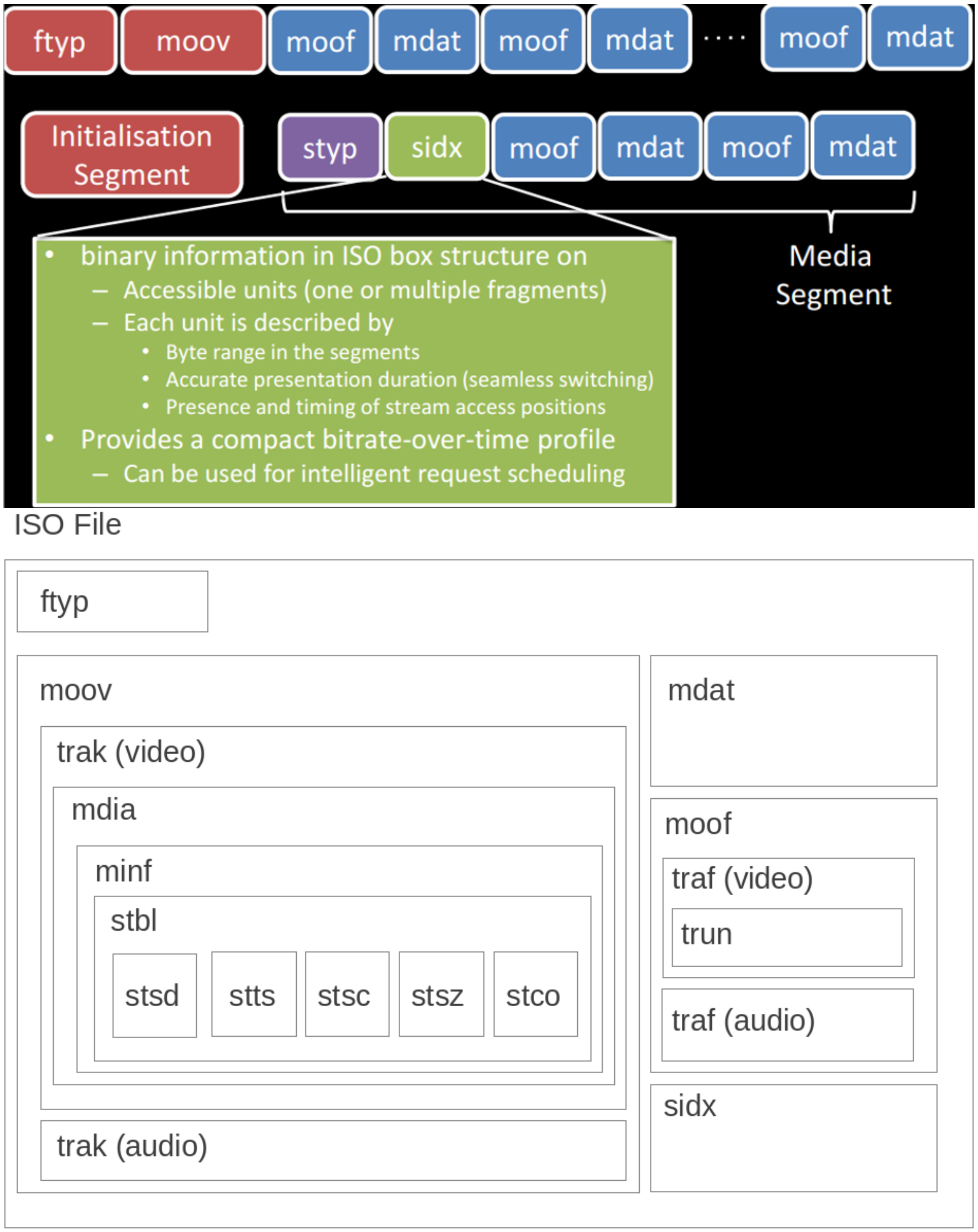
One thing that is often missing but not prohibited in the basic WVSNP-DASH MP4 container is the (ISO/IEC 14496-12) Segment Index, *(sidx)* box [106]. See Figure 5.15 for the ISO BFF MP4 structure [226].

The *sidx* box is an index table of all accessible video data units in the stream. That is, one or more fragments of the entire multimedia tracks. Most MPEG-DASH segments show the box structure in Figure 5.16.

Figure 5.17 shows how the *sidx* box actually has all the information about multimedia data fragments available for playback.

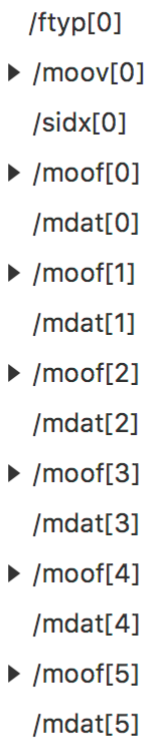
This means a WVSNP-DASH segment can be setup like Figure 5.18 or 5.17 and be capable of transmitting only the fragments of the segment needed.

So, in case of quality switches a client can save power fetching only the next chunk offset in the next segment of different quality based solely on the name of the previous segment. Basically, as segments are being fed to the browser, the client scripts use the current segment's header to fetch the next data chunk as long as a prior segment



- binary information in ISO box structure on
  - Accessible units (one or multiple fragments)
  - Each unit is described by
    - Byte range in the segments
    - Accurate presentation duration (seamless switching)
    - Presence and timing of stream access positions
- Provides a compact bitrate-over-time profile
  - Can be used for intelligent request scheduling

Figure 5.15: Structure of .mp4 video containers expected by MSE [227]. Top image from [196].



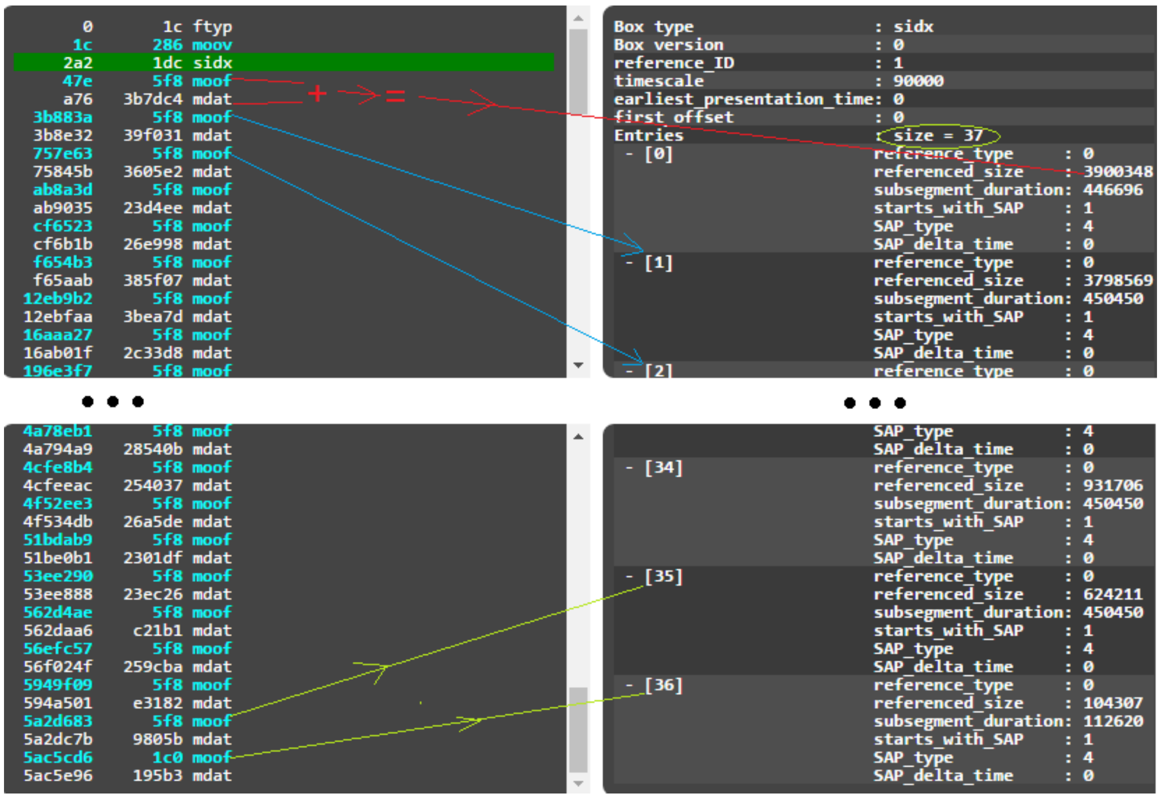
```
  /ftyp[0]
▸ /moov[0]
  /sidx[0]
▸ /moof[0]
  /mdat[0]
▸ /moof[1]
  /mdat[1]
▸ /moof[2]
  /mdat[2]
▸ /moof[3]
  /mdat[3]
▸ /moof[4]
  /mdat[4]
▸ /moof[5]
  /mdat[5]
```

Figure 5.16: Structure of a fragmented .mp4 video container.

with the same name pattern was initialized.

To be able to make WVSNP-DASH work on MSE, it was important to understand the MP4 video container structure further. Other video containers were easy to parse as they are organized well. For example, the WebM [228] container showed in Figure 5.19 is a much stricter subset of Matroska multimedia container format [7]. This makes it easier to parse and playback. It has only two key Level Zero parts, EBML header and the Segment block, which contains all the information needed by the decoder for the rest of the stream. The Extensible Binary Meta Language (EBML) is a generalized file format for any kind of data. It is designed to be a binary equivalent of XML.

So when using WebM for WVSNP-DASH, it is easier to find and playback only chunks that actually are media tracks as the Segment header contains only four (4) next level structures with the fourth one being Segment Tracks. These are easier



```
       0      1c ftyp
      1c     286 moov
     2a2     1dc sidx
     47e     5f8 moof
     a76  3b7dc4 mdat
  3b883a     5f8 moof
  3b8e32  39f031 mdat
  757e63     5f8 moof
  75845b  3605e2 mdat
  ab8a3d     5f8 moof
  ab9035  23d4ee mdat
  cf6523     5f8 moof
  cf6b1b  26e998 mdat
  f654b3     5f8 moof
  f65aab  385f07 mdat
 12eb9b2     5f8 moof
 12ebfaa  3bea7d mdat
 16aaa27     5f8 moof
 16ab01f  2c33d8 mdat
 196e3f7     5f8 moof
```

```
Box type                 : sidx
Box version              : 0
reference_ID             : 1
timescale                : 90000
earliest_presentation_time: 0
first_offset             : 0
Entries                   size = 37
 - [0]                    reference_type     : 0
                          referenced_size    : 3900348
                          subsegment_duration: 446696
                          starts_with_SAP    : 1
                          SAP_type           : 4
                          SAP_delta_time     : 0
 - [1]                    reference_type     : 0
                          referenced_size    : 3798569
                          subsegment_duration: 450450
                          starts_with_SAP    : 1
                          SAP_type           : 4
                          SAP_delta_time     : 0
 - [2]                    reference_type     : 0
```

• • •

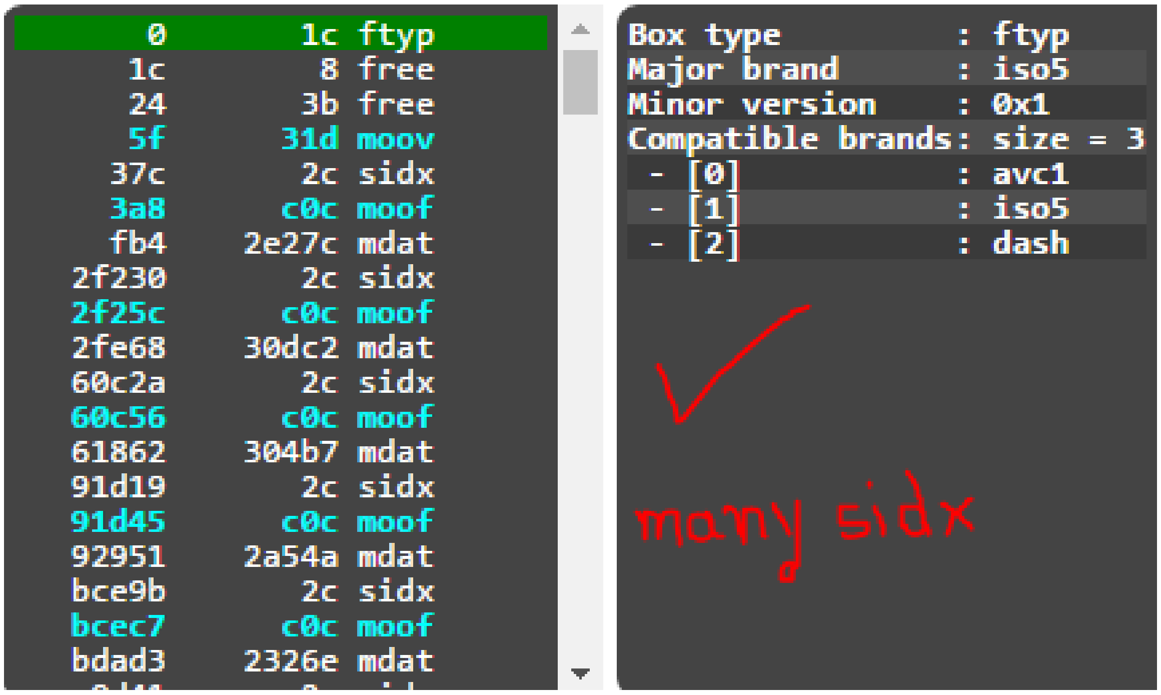
```
 4a78eb1     5f8 moof
 4a794a9  28540b mdat
 4cfe8b4     5f8 moof
 4cfeeac  254037 mdat
 4f52ee3     5f8 moof
 4f534db  26a5de mdat
 51bdab9     5f8 moof
 51be0b1  2301df mdat
 53ee290     5f8 moof
 53ee888  23ec26 mdat
 562d4ae     5f8 moof
 562daa6   c21b1 mdat
 56efc57     5f8 moof
 56f024f  259cba mdat
 5949f09     5f8 moof
 594a501   e3182 mdat
 5a2d683     5f8 moof
 5a2dc7b   9805b mdat
 5ac5cd6     1c0 moof
 5ac5e96   195b3 mdat
```

```
                          SAP_type           : 4
                          SAP_delta_time     : 0
 - [34]                   reference_type     : 0
                          referenced_size    : 931706
                          subsegment_duration: 450450
                          starts_with_SAP    : 1
                          SAP_type           : 4
                          SAP_delta_time     : 0
 - [35]                   reference_type     : 0
                          referenced_size    : 624211
                          subsegment_duration: 450450
                          starts_with_SAP    : 1
                          SAP_type           : 4
                          SAP_delta_time     : 0
 - [36]                   reference_type     : 0
                          referenced_size    : 104307
                          subsegment_duration: 112620
                          starts_with_SAP    : 1
                          SAP_type           : 4
                          SAP_delta_time     : 0
```

Figure 5.17: Illustration of the SIDX box structure detail of an actual fragmented .mp4 video container. From Google's car video used in the Shaka Player demo of MPEG-DASH over MSE. Mp4Parser [102] tool was used to visualize the container. Column 1 is the box's byte offset in the file, column 2 is the size of the box.

to dereference. Unfortunately WebM is not as widely supported as .mp4 across all browsers. The .mp4 files on the other hand need better observation of their box container components' organization. The MPEG specification has several options than can render the video unplayable depending on how it is delivered or the player itself interprets it. For example, the progressive download format is not quite the same as the fragmented format expected by MSE. So, care in capturing .mp4 video is needed to still be backward compatible with .mp4 streams. Analyzing our basic .mp4 captured file revealed the structure in Figure 5.20 below.

This plays on normal HTML5 video element/tag but not if passed in as an MSE



```
    0      1c ftyp
   1c       8 free
   24      3b free
   5f     31d moov
  37c      2c sidx
  3a8     c0c moof
  fb4   2e27c mdat
2f230      2c sidx
2f25c     c0c moof
2fe68   30dc2 mdat
60c2a      2c sidx
60c56     c0c moof
61862   304b7 mdat
91d19      2c sidx
91d45     c0c moof
92951   2a54a mdat
bce9b      2c sidx
bcec7     c0c moof
bdad3   2326e mdat
```

```
Box type          : ftyp
Major brand       : iso5
Minor version     : 0x1
Compatible brands: size = 3
 - [0]            : avc1
 - [1]            : iso5
 - [2]            : dash
```

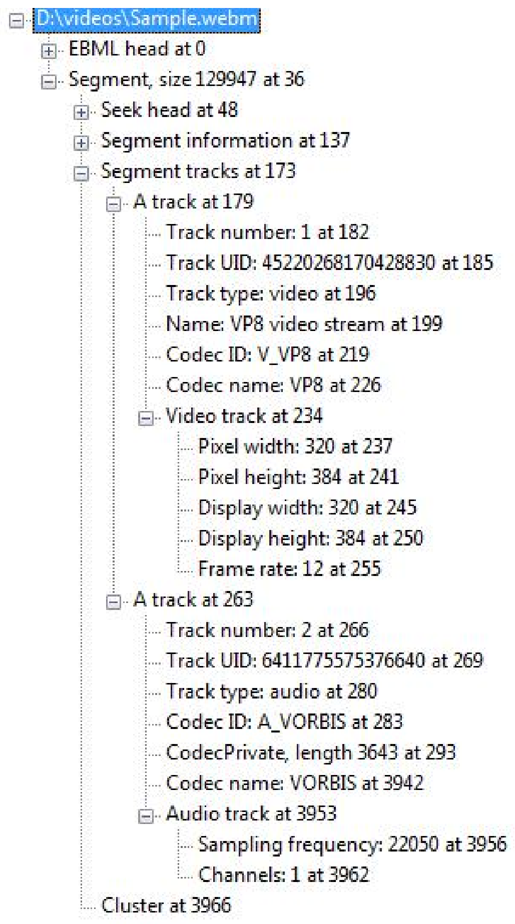
Figure 5.18: Actual partial structure of tested MSE compatible WVSNP-DASH .mp4 video container fragmented to be MPEG-DASH compatible.

source buffer. Checking the structure of the reference videos used by Google's MPEG-DASH segments used in their Shaka MPEG-DASH player [183] shows Figures 5.16 and 5.21 below.

A quick look at the structures above shows in Figures 5.20 and 5.22 that there is only one *mdat* box.

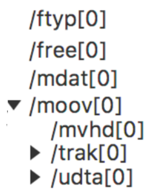
In Figure 5.16, there are multiple *mdat* boxes, which would imply fragmentation in the MSE capable containers. Since MPEG-DASH compatibility is not the goal of WVSNP-DASH, what is important is to satisfy the basic structure MSE expects to operate on as shown in the simplified standard structure in Figure 5.15 above.

All that MSE needs for an .mp4 initialization segment to be decoded is that it contains a single File Type Box *(ftyp)* followed by a single Movie Box *(moov)*. So, changing container in Figure 5.22 to Figure 5.23, works.



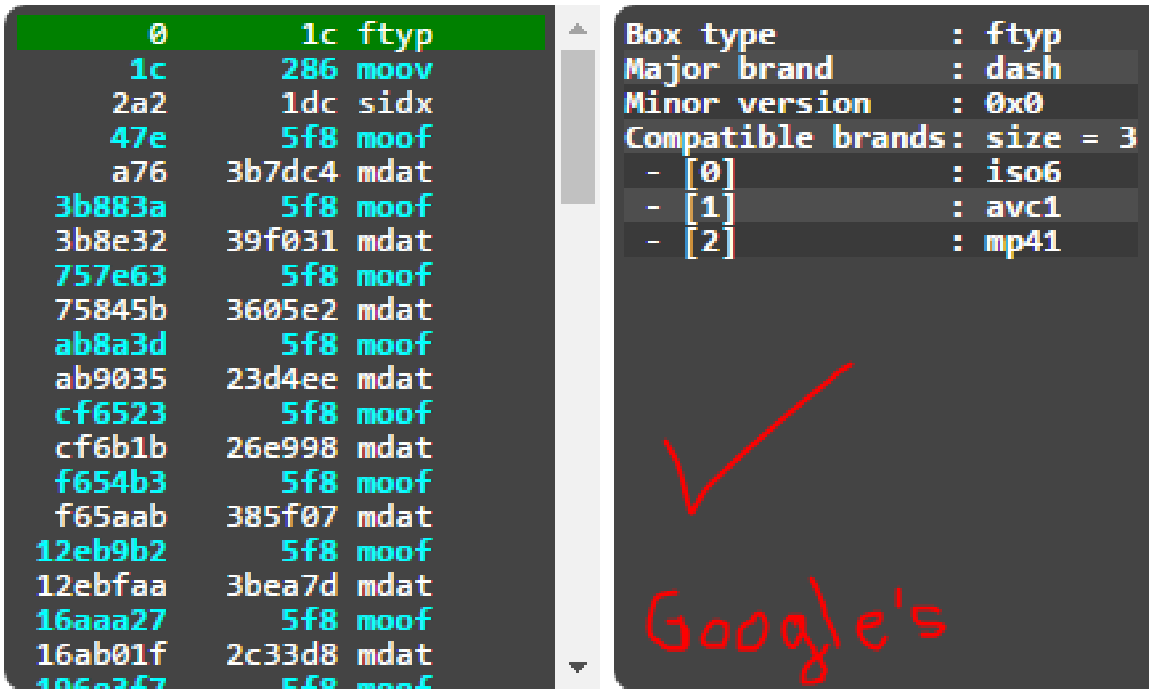
```
D:\videos\Sample.webm
  EBML head at 0
  Segment, size 129947 at 36
    Seek head at 48
    Segment information at 137
    Segment tracks at 173
      A track at 179
        Track number: 1 at 182
        Track UID: 45220268170428830 at 185
        Track type: video at 196
        Name: VP8 video stream at 199
        Codec ID: V_VP8 at 219
        Codec name: VP8 at 226
        Video track at 234
          Pixel width: 320 at 237
          Pixel height: 384 at 241
          Display width: 320 at 245
          Display height: 384 at 250
          Frame rate: 12 at 255
      A track at 263
        Track number: 2 at 266
        Track UID: 6411775575376640 at 269
        Track type: audio at 280
        Codec ID: A_VORBIS at 283
        CodecPrivate, length 3643 at 293
        Codec name: VORBIS at 3942
        Audio track at 3953
          Sampling frequency: 22050 at 3956
          Channels: 1 at 3962
    Cluster at 3966
```

Figure 5.19: WebM digital multimedia container file format. Example Image from [158].



```
/ftyp[0]
/free[0]
/mdat[0]
▼ /moov[0]
    /mvhd[0]
  ▶ /trak[0]
  ▶ /udta[0]
```

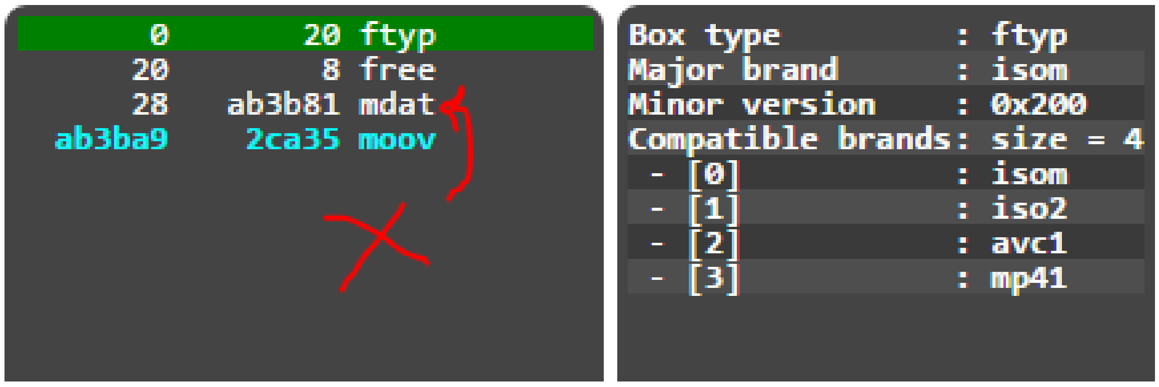
Figure 5.20: Original basic WVSNP .mp4 video container structure.



```
       0        1c ftyp
      1c       286 moov
     2a2       1dc sidx
     47e       5f8 moof
     a76    3b7dc4 mdat
  3b883a       5f8 moof
  3b8e32    39f031 mdat
  757e63       5f8 moof
  75845b    3605e2 mdat
  ab8a3d       5f8 moof
  ab9035    23d4ee mdat
  cf6523       5f8 moof
  cf6b1b    26e998 mdat
  f654b3       5f8 moof
  f65aab    385f07 mdat
 12eb9b2       5f8 moof
 12ebfaa    3bea7d mdat
 16aaa27       5f8 moof
 16ab01f    2c33d8 mdat
```

```
Box type          : ftyp
Major brand       : dash
Minor version     : 0x0
Compatible brands : size = 3
 - [0]            : iso6
 - [1]            : avc1
 - [2]            : mp41
```

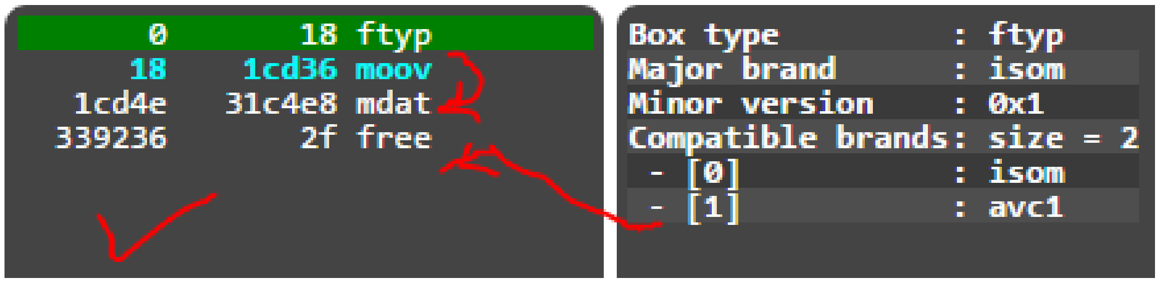
Figure 5.21: Actual partial structure of the Google car's .mp4 video used in the Shaka Player MPEG-DASH via MSE demo [183].



```
       0        20 ftyp
      20         8 free
      28    ab3b81 mdat
  ab3ba9     2ca35 moov
```

```
Box type          : ftyp
Major brand       : isom
Minor version     : 0x200
Compatible brands : size = 4
 - [0]            : isom
 - [1]            : iso2
 - [2]            : avc1
 - [3]            : mp41
```

Figure 5.22: Actual partial structure of tested WVSNP-DASH .mp4 video container NOT playable by MSE.



```
       0        18 ftyp
      18     1cd36 moov
   1cd4e    31c4e8 mdat
  339236        2f free
```

```
Box type          : ftyp
Major brand       : isom
Minor version     : 0x1
Compatible brands: size = 2
 - [0]            : isom
 - [1]            : avc1
```

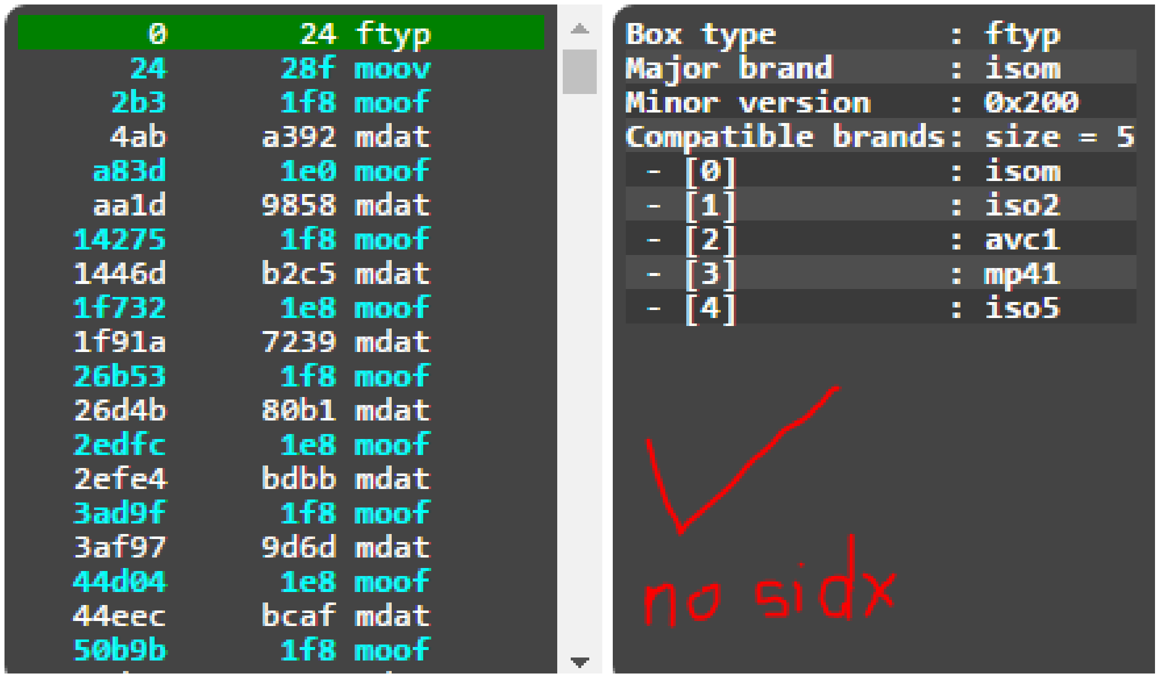
Figure 5.23: Actual partial structure of tested WVSNP-DASH .mp4 video container NOW playable by MSE.

MSE standard further states that *"Valid top-level boxes such as* ***pdin****,* ***free****, and* ***sidx*** *are allowed to appear before the* ***moov*** *box. These boxes must be accepted and ignored by the user agent and are not considered part of the initialization segment ...".* MSE [227] expects a container fragment or Media Segment to have a Segment Type Box *(styp)* followed by a single Movie Fragment Box *(moof)* that is followed by one or more Media Data Boxes *(mdat)*.

Since *(styp)* is optional, the segment must comply with whatever File Type Box (ftyp) specified in the initialization segment. After the container decoder has decoded an initialization segment or fragment segment, the boxes (*ftyp, moov, styp, moof, and mdat*), can be decoded and ignored. As long as they are before the beginning of the next fragment/media segment.

Though the MSE [227] specification above seems confusing, the previous sentence actually leaves an opening where we can move *(mdat)* box in Figure 5.20 above to below the *(moov)* box. This allows any WVSNP-DASH segment file to be decoded by MSE even if it is neither a DASH initialization file nor a media/fragmented .mp4 file, see Figures 5.23 and 5.18. This can be achieved in many ways at capture time and in post processing. For example with ffmpeg, adding flags: `frag_keyframe+empty_mov` to the `-movflags` switch achieves the goal. Both at capture and over an existing file.

This is not all the innovation needed. Since WVSNP-DASH does not use manifest



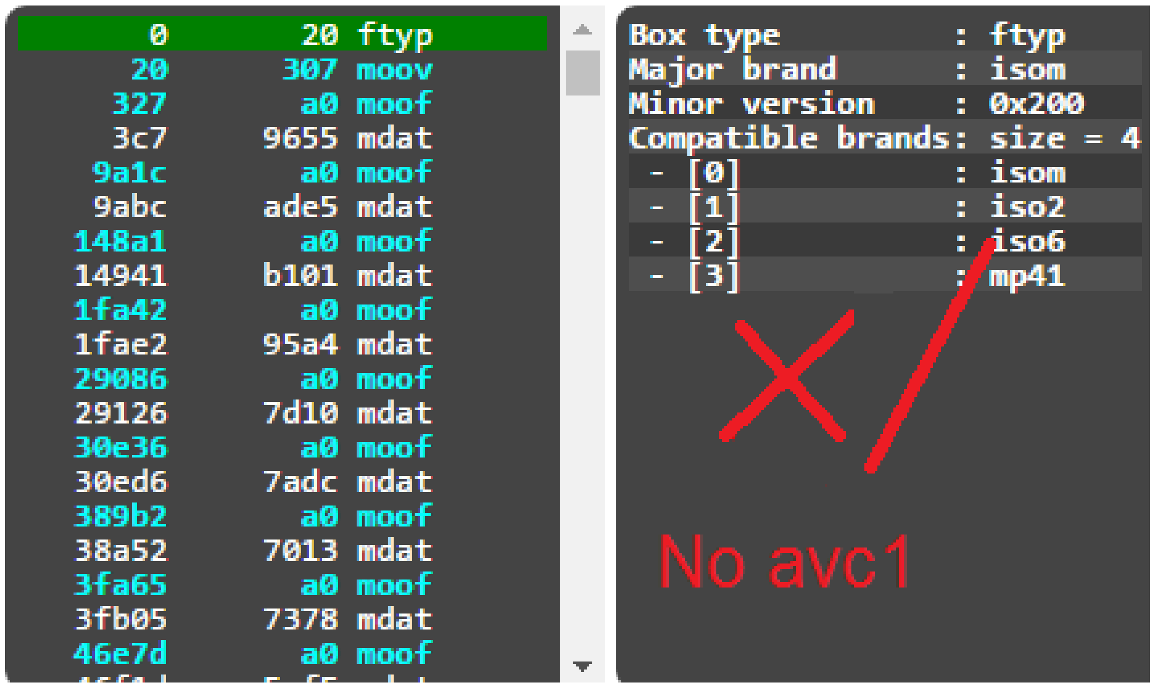
```
    0         24 ftyp
   24        28f moov
  2b3        1f8 moof
  4ab      a392 mdat
 a83d        1e0 moof
 aa1d      9858 mdat
14275        1f8 moof
1446d      b2c5 mdat
1f732        1e8 moof
1f91a      7239 mdat
26b53        1f8 moof
26d4b      80b1 mdat
2edfc        1e8 moof
2efe4      bdbb mdat
3ad9f        1f8 moof
3af97      9d6d mdat
44d04        1e8 moof
44eec      bcaf mdat
50b9b        1f8 moof
```

```
Box type          : ftyp
Major brand       : isom
Minor version     : 0x200
Compatible brands: size = 5
 - [0]            : isom
 - [1]            : iso2
 - [2]            : avc1
 - [3]            : mp41
 - [4]            : iso5
```

Figure 5.24: Actual partial structure of tested WVSNP-DASH .mp4 video container NOW playable by MSE with no SIDX and capable of range fetches within a WVSNP-DASH segment.

files, there were more capture design tweaks to enable low power WVSNP-DASH operation. Though WVSNP-DASH .mp4 segments by default do not have the Segment Index box, *(sidx)*, the concept translates directly to the name based fetches by treating any prior playedback segment (if the current segment is not the first one fetched) as if it is an initialization segment and then inferring *(moof)* and *(mdat)* chunk ranges from that to minimize possibly wasteful fetching of redundant headers as explained in detail earlier in this sub-section 5.3.5, just before Figure 5.14.

Another .mp4/MSE feature to get around is that the MSE BufferSource needs to strictly know the *"Codecs String"* in the container ahead of time. So, to decode, the container must be forced to `"avc1.42E01E"`. This is the *H.264 Constrained Baseline Profile Level 3*. It is pretty much supported by all browsers. This makes MSE's `"MediaSource.isTypeSupported"` evaluate to `"true"`. Figure 5.25 shows



```
        0        20 ftyp
       20       307 moov
      327        a0 moof
      3c7      9655 mdat
     9a1c        a0 moof
     9abc      ade5 mdat
    148a1        a0 moof
    14941      b101 mdat
    1fa42        a0 moof
    1fae2      95a4 mdat
    29086        a0 moof
    29126      7d10 mdat
    30e36        a0 moof
    30ed6      7adc mdat
    389b2        a0 moof
    38a52      7013 mdat
    3fa65        a0 moof
    3fb05      7378 mdat
    46e7d        a0 moof
```

```
Box type          : ftyp
Major brand       : isom
Minor version     : 0x200
Compatible brands: size = 4
 - [0]            : isom
 - [1]            : iso2
 - [2]            : iso6
 - [3]            : mp41
```

Figure 5.25: Actual partial structure of tested WVSNP-DASH .mp4 video container NOT playable by MSE due to wrong "Codecs String".

MSE supported *(box)* header order but not playable due to the wrong Codecs String.

So, based on the empirical evaluation, we believe that for short WVSNP-DASH segments (2s and 5s), the best .mp4 container would be Figure 5.23. For longer segments (10s or more), the best .mp4 container organization would be Figure 5.24 with equal *moof* plus *mdat* box sizes within a segment except for the last segment of the stream. The longer segment in Figure 5.24 can easily be substituted by the MPEG-DASH compatible initialization segment in Figure 5.21 with a negligible extra size overhead of the *sidx* box which can be ignored by the parser.

5.4 Profiling Conclusions and Future Work

This paper provided an extensive power profiling of video capture, streaming, architectural path and framework flow choices on a wireless video sensor node (WVSn).

This paper thus provides empirical baseline power consumption data based on the architectural components of a node as well as the effects of streaming frameworks and applications. As part of the evaluations of better DASH frameworks suitable for WVSNs, this paper introduced Wireless Video Sensor Network Platform compatible Dynamic Adaptive Streaming over HTTP (WVSNP-DASH) framework. The WVSNP-DASH framework specifies a naming syntax for independently playable video segments. Existing DASH frameworks convey video meta data through a manifest file and begin video streaming with a special initialization video segment; subsequent video segments depend on the manifest file and initialization segment for playback. In contrast, the WVSNP-DASH video segments convey essential meta data through their name and can be played independently, i.e., each individual WVSNP-DASH segment is fully playable without reference to any other file or segment. This file independence simplifies the video capture and video file segment creation and streaming by a sensor node and hence providing power saving opportunities at both the node and the client.

The additional comparative evaluation of a WVSNP-DASH versus HLS and indirectly MPEG-DASH frameworks has indicated that the independently playable WVSNP-DASH video segments create significant potential for power savings on the sensor node serving the video. To the best of the authors knowledge the presented evaluation is the first to examine the effects of different DASH frameworks, node capture and streaming data path on sensor node power consumption.

The mp4 and HLS's .ts containers were the only ones used for WVSNP video storage and streaming analysis, but for future work other containers should be tried for WVSNP videos because power consumption and size of the segments with other containers might be lesser than mp4 container. Much more sensitive power meters might improve the results where the differences in power consumption if not very clear due to the resolution of the current meter and sampling rate. As recommended in detail under 5.3.4, there is a lot of insight to be derived from this to improve the design of WVSn. The next step is to incorporate these revelations into the ongoing improvement of the WVS node designed by the authors (Flexi-WVSNP). This node is designed from the ground up to support sensor data fusion with video as part of the data elements. The node is the first one to support WVSNP-DASH as default with the ability to integrate with non TCP/IP sensor networks and the greater Internet of Things.

The data provided by this work will enable DASH client adaptation designers to improve dynamic adaptation algorithms, including those that take into account the power budget of the server node or client. Evaluation results in 5.3.4 and 5.3.5 also brought in recommendations to create optimized capture applications to speed up efficient LIVE playback from the node. This would reduce need for large buffers that delay live segment playback.

Further refinement of this evaluation will be helpful in the future. For example, evaluating the cost of initializing every segment both on client and server. This can be compared to initializing only one and assuming that initialization data of one satisfies future dynamic switches.

Since recently MSE player support has improved considerably across all major browsers, the next step is to perform a pure side by side testbed between a pure MSE WVSNP-DASH player versus the recent hls.js [95] versus dash.js [55] versus Shaka player [183]. This will normalize a lot of variables as previously this was impossible to compare with exactly the same underlying client technologies across all browsers.

Chapter 6

CURRENT STATE OF THE PLATFORM

6.1 Overview

As mentioned in introduction, a WVSNP that can be useful and still be relevant in the modern era of cell phones, big data and the ever changing Internet of Things interfaces needs to be highly flexible, low power, lowly coupled low cost, highly cohesive and yet scalable to a large number of applications. Over the past few years, a lot of applications have been tested with goals of verifying many assumptions of how a WVSN should work. Many iterations of boards from CMUCam3, to Beagle-board, i.MX53, different configurations of i.MX6 and have been tested and applications created to verify the Zigbee, Camera, Operating System, WiFi, Bluetooth and the WVSNP-DASH framework this node is built around. A lot of technologies have changed along the way, and a lot of changes have happened which help test the flexibility of the multi-radio scheme and more.

6.1.1 From SoC to SOM Architecture

As elaborated in the Chapter 3.1.2, technology changes fast. We started with the concept of identifying highly configurable System-On-Chip (SoC) that can handle a lot of power saving techniques expected in WVSN. After many tests and use cases we settled on an SoC that can be scaled from very low power with many components turned off to very high end with the ability to be configured from single core to quad core as needed without changing the Software applications nor the operating system.

To save on the off the shelf low cost basic components, we followed the new trend in industry where a System Designer no longer spends time designing a PCB around an SoC nor memory. The design now uses a System-On-Module architecture which has a standard size that can be configured by the manufacturer for many applications using mass produced identical parts with other companies. The SoM architecture (SoM) reduces cost. SoM helps future proof parts maintainability. For example if one needs a SoM that had four CPUs they can order one without any change in the node design nor software. Same applies to an application that needs only one CPU for very low power operations. The HW platform as it is, is easily re-target-able for different use cases. SoM reduces the learning curve as many components and software have already been written and used on other applications. To add to that, the SoM can be purchased as a stand alone module re-usable from other parts as long as the design follows the modern Electronic Design Module (EDM) standard [65].

Figure 6.1 shows a depictions of a development board that was used to create the core skeleton of the platform middle ware and WVSNP operating system image.

Figure 6.2 is a depiction of a free running SoM module.

It contains a Yocto [240, 197, 174] packaged and built WVSNP OS Image (WOS) that has everything needed to power up, run and be discovered wirelessly. All dangling flexible attachments can be added to extend the node or target it for a particular application. The key goal of the platform is to make addition of the extra dangles very easy via an open WVSNP carrier board as depicted in Figures 6.3 and 6.2.

The schematics, footprint and Gerber files for the WVSNP carrier board are shown in 6.4, 6.5 and 6.6.

The second spin original board consisted of flex-headers that were originally thought to make the new dangles easier and more snug as shown in the Figures 6.7 and 6.8 below.

From a few dangle module tests and application re-targeting testing, it was decided

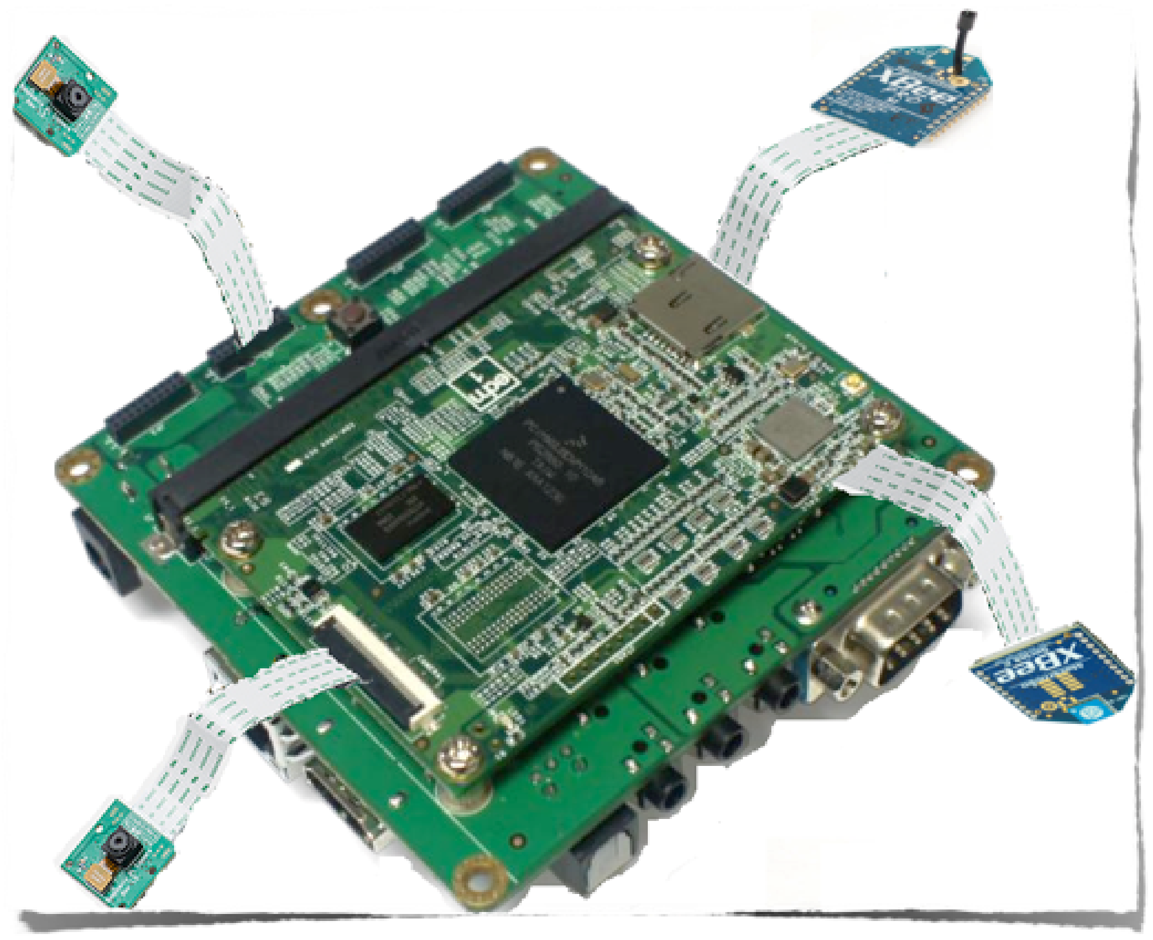


Figure 6.1: The depiction of the original final development board used for most of the testing and benchmarking SoC.

that the flex headers narrowed down re-target-ability goal of the board. The Flex headers are generally more expensive and harder to find, therefore not reducing the cost. They also are harder to solder to the board, It is harder to create mating dangle headers and cables for them for future dangles. So the WVSNP carrier board was re-made to cost even less, be easily populated when needed. This new style of headers are cheaper and easy to populate much later when needed by an application. There is no need to force future users to try to follow specific narrowed down designs around common shields as is the case with Raspberry Pi and Arduino boards. You can create

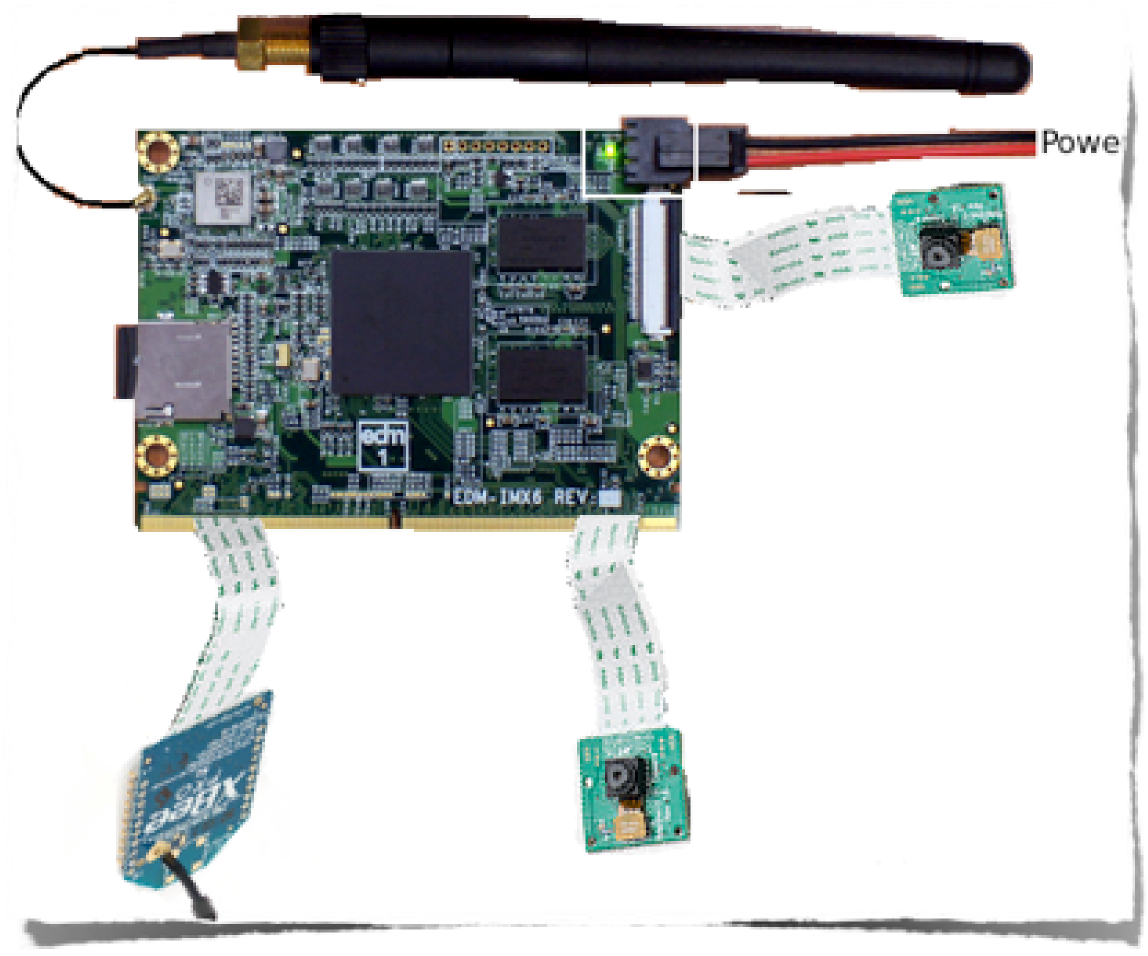


Figure 6.2: A depiction of an independently running SoM module of the Node.

your own header as long as you can connect wires to the open and free pins that come with the board. The new board is shown in Figures 6.10 and 6.9 below.

Latest board is also made much more accessible by reducing it from four (4) layers to two layers. The important goal is to always have a working WVSNP carrier board that can be ordered on demand with the specific headers needed by a customer populated and those not wanted, unpopulated. We have a streamlined concept of the manufacturing and ordering process to make this a very easy task for choosing the flexible dangles needed. From a website, one should be able to click to choose, order

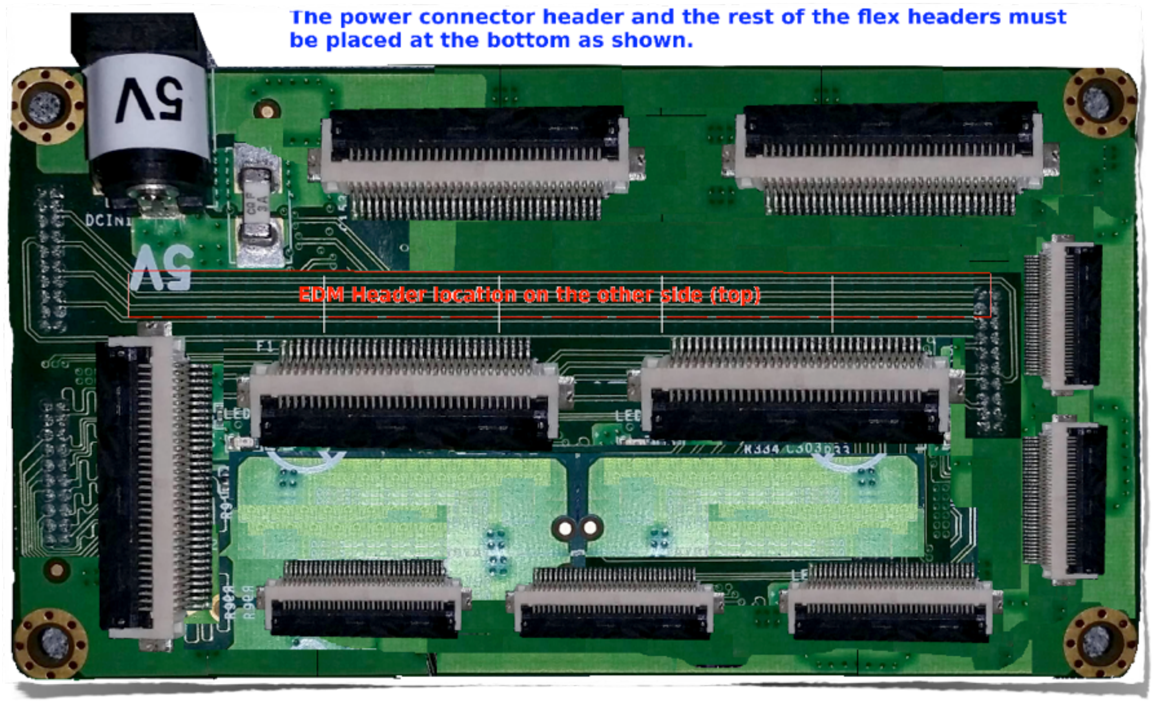

The power connector header and the rest of the flex headers must be placed at the bottom as shown.
5V
DCIN
5V
EDM Header location on the other side (top)
F1
LED
LED

Figure 6.3: The envisioned EDM re-targetor.

and confirm. The website should be able to show you the 3D of the board you want for your application. We have a tested mass manufacturer we work with. But for research, the Gerber files can be manufactured with any manufacturer of your choice.

6.1.2 Flexible IO, Data and Power Lines

As elaborated in the Section 3.1.2 and in Figure 3.2, we want all swappable HW modules to be able to be powered and shut down by applications. As shown in Figures 6.11 and 6.12 all headers above have individual power lines. The green markings are by default not populated. Virtually all pins of the processor are accessible if really needed by the application. Most headers have more than one power supply lines.

As the key shows in Figures 6.11 and 6.12 the dark red shows 5V and pink shows 3V. Blue shows ground. Black shows data or signal lines.

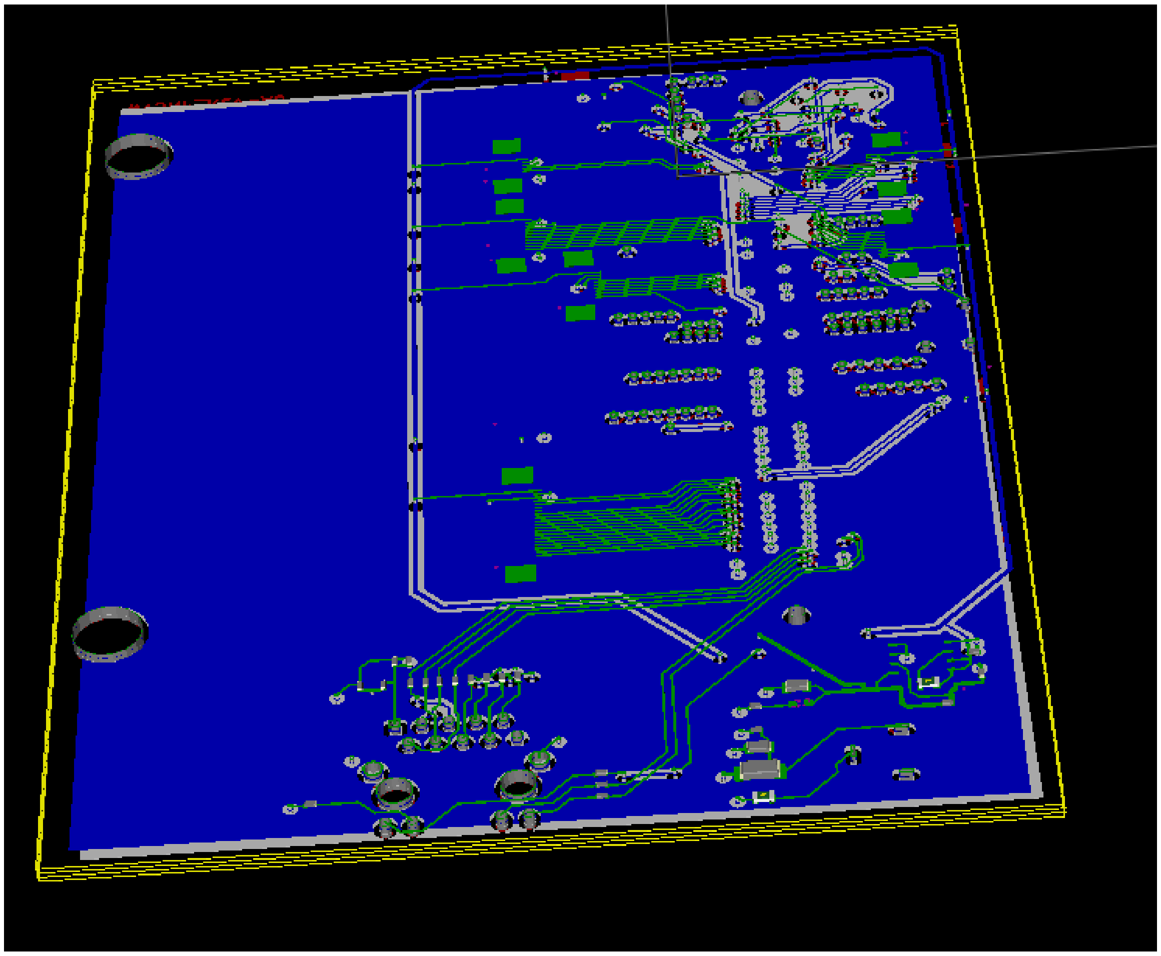


Figure 6.4: The second spin original 3D bottom layer of the WVSNP carrier board.

6.1.3 WVSNP IO Board modules

The WVSNP node will not be useful much by itself if all it did was to capture video and stream it to your phone. It also needs peripheral modules to enable the platform to remotely sense and actuate remote sensors. These IO modules allow us to create networks that create data fusion across traditional networks, proprietary, Zigbee, Bluetooth, thread and so forth.

Figures 6.13 and 6.14 below show the WVSNP IO modules created for the platform as base remote the work horses. They are are independent sensor and actuator

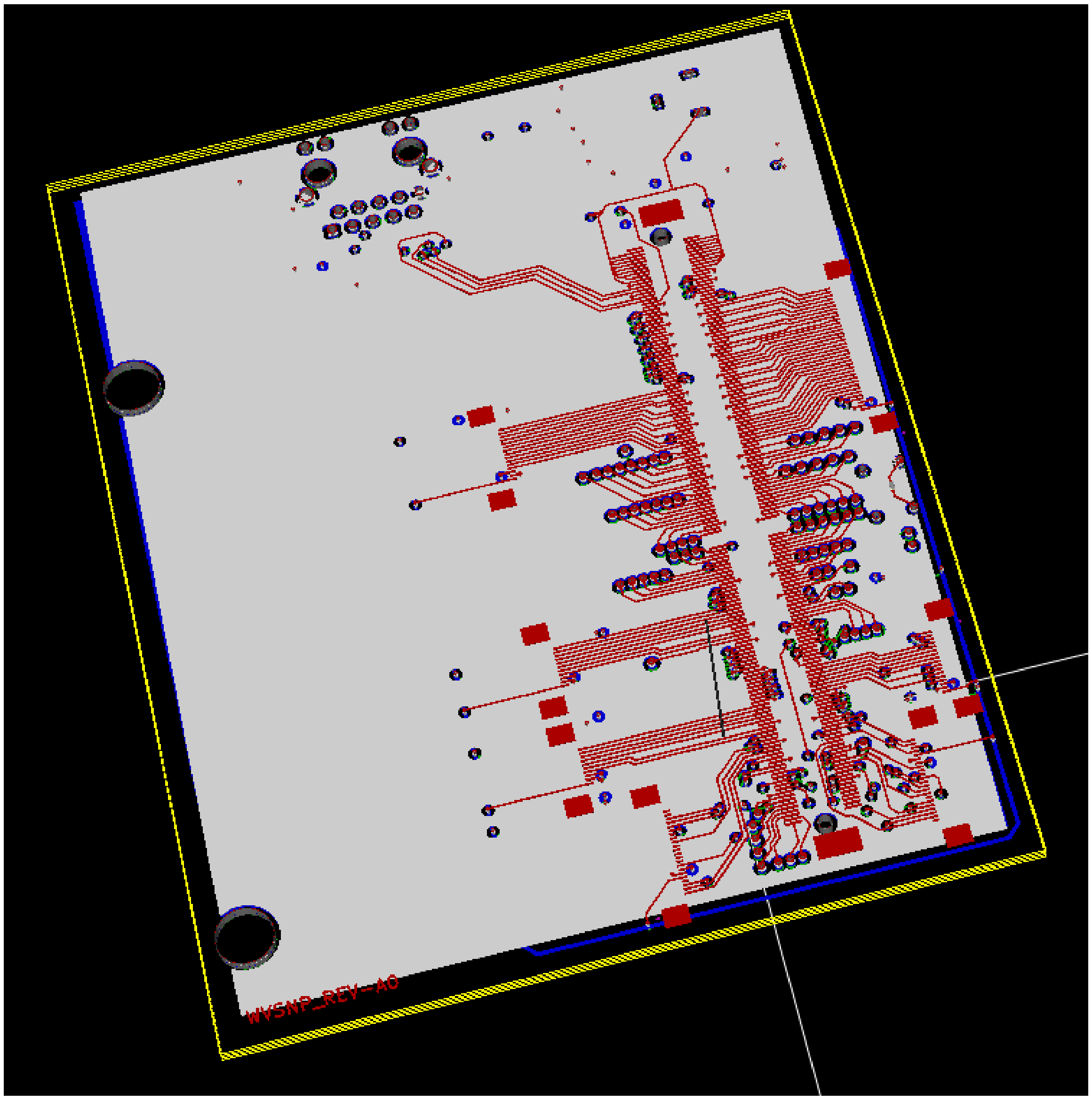


Figure 6.5: The second spin original 3D top layer of the WVSNP carrier board.

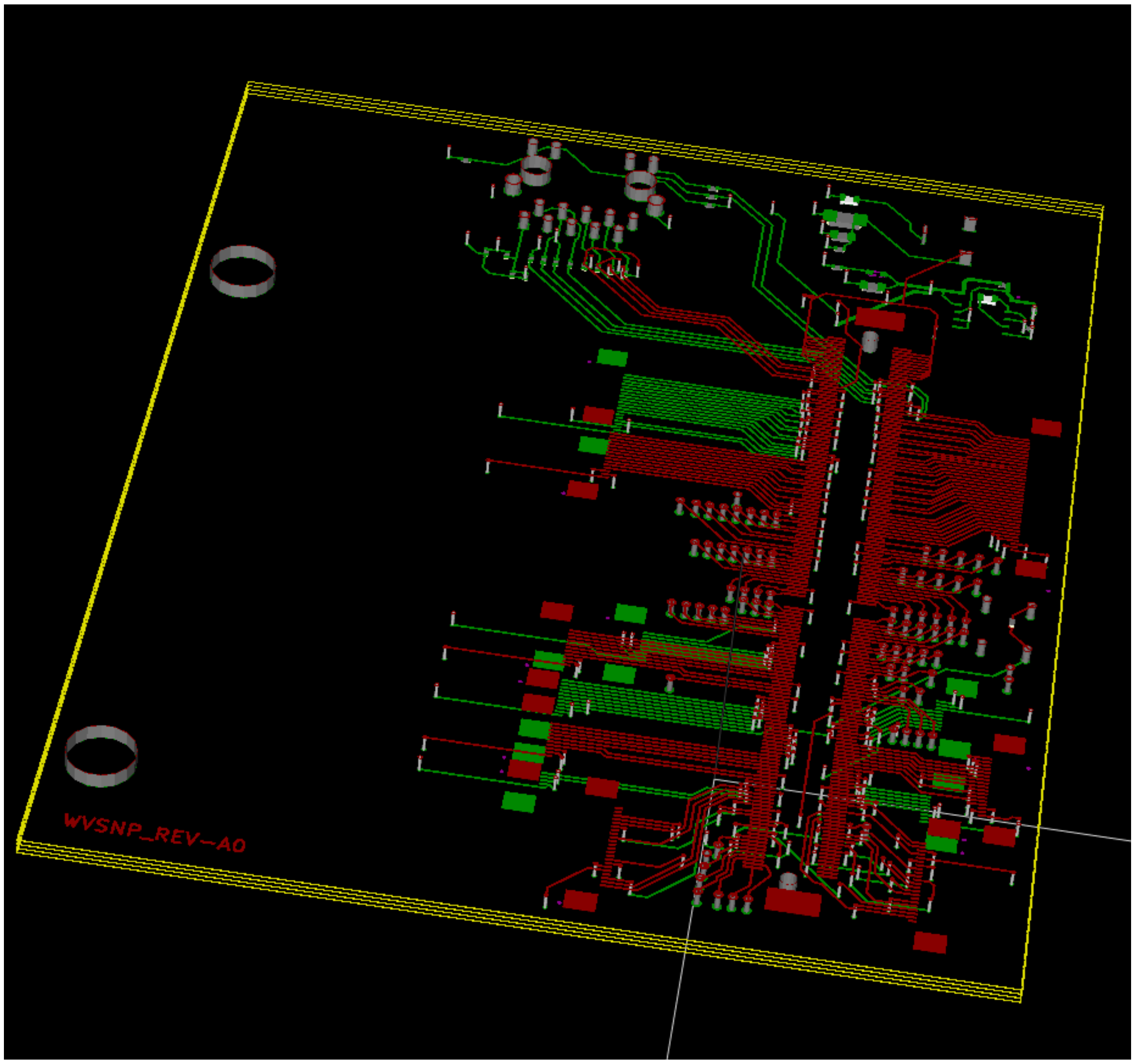


Figure 6.6: The second spin original 3D inner layers of the WVSNP carrier board.

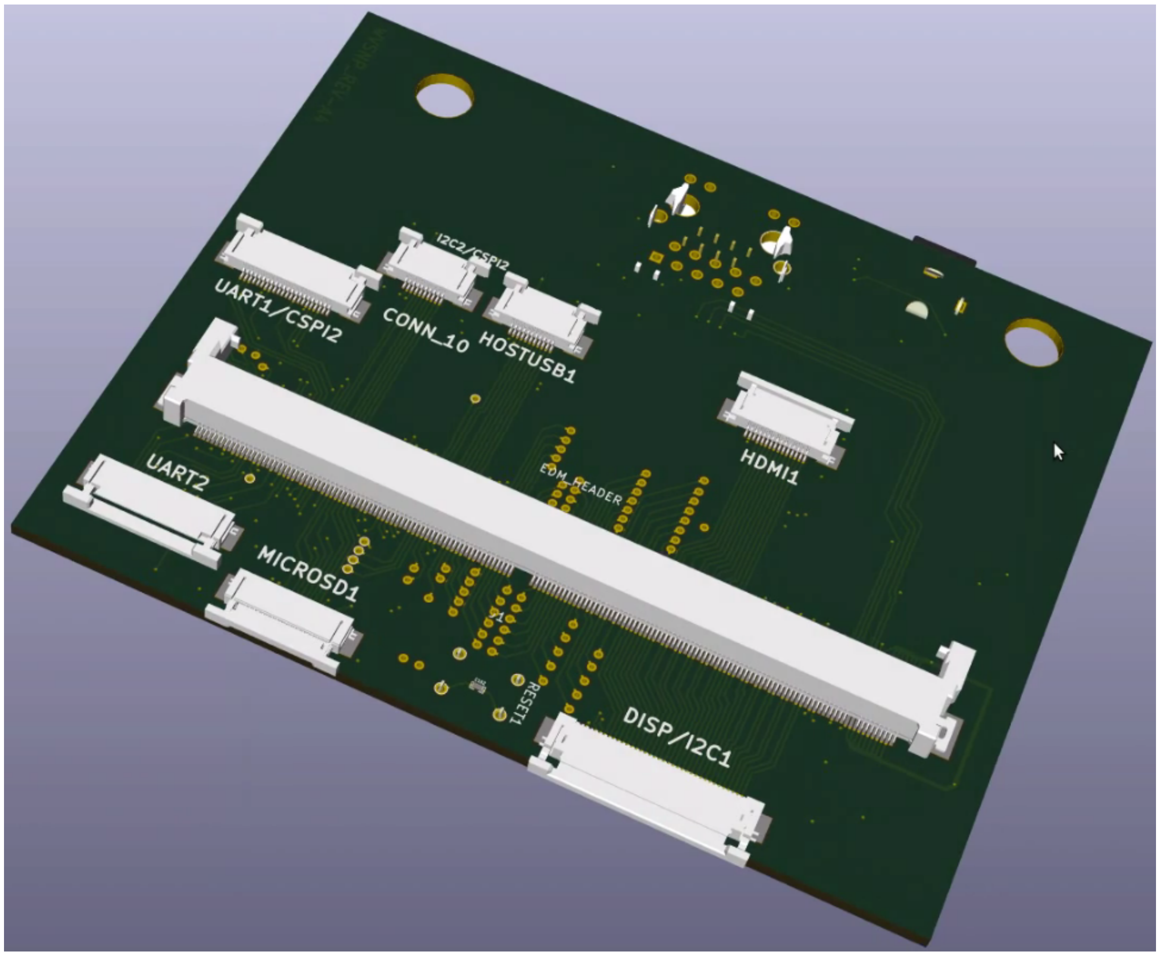


Figure 6.7: Top of the second spin original 3D WVSNP carrier board.

modules that can be controlled or monitored by the WVSNP node. The modules were tested using XBee Zigbee and XBee WiFi radios shown in the Figures 6.13 and 6.14. While a few samples were made. Any quantities can be ordered to target any sensor to deliver us the data in any quantity we can afford. These can be (UART or SPI) or (WiFi or Zigbee), and future Thread. Additionally there are also Bluetooth Low Energy (BLE) modules that were used as peripheral remote sensors or actuators just like the Zigbee modules.

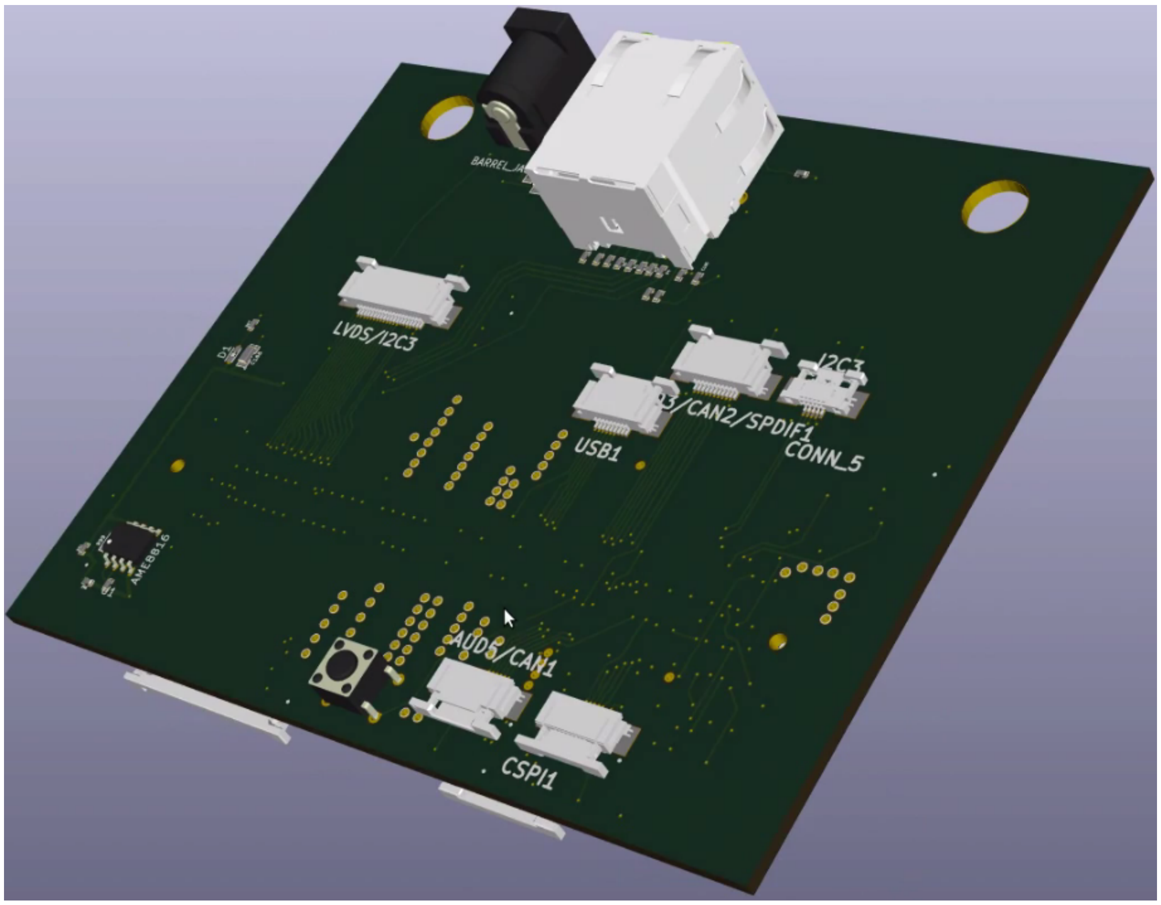


Figure 6.8: Bottom of the second spin original 3D WVSNP carrier board.

6.2 How the Node Works.

To understand how the node works, we first have to understand the boot sequence of the WVSNP node. The bootup/init routine is triggered immediately once the OS has finished booting. The routine then launches the node's own WiFi network via software enabled access point (SoftAP), and then Bluetooth and eventually Zigbee. The node will then announce that it is ready for work. When bootup is complete it makes sure the following servers are up and running: HTTP server: mongoose. Bluetooth server: rfcomm server (for chat like interaction and parsing commands). Bluetooth server: obex server (for object and file pushes). Zigbee server: There is

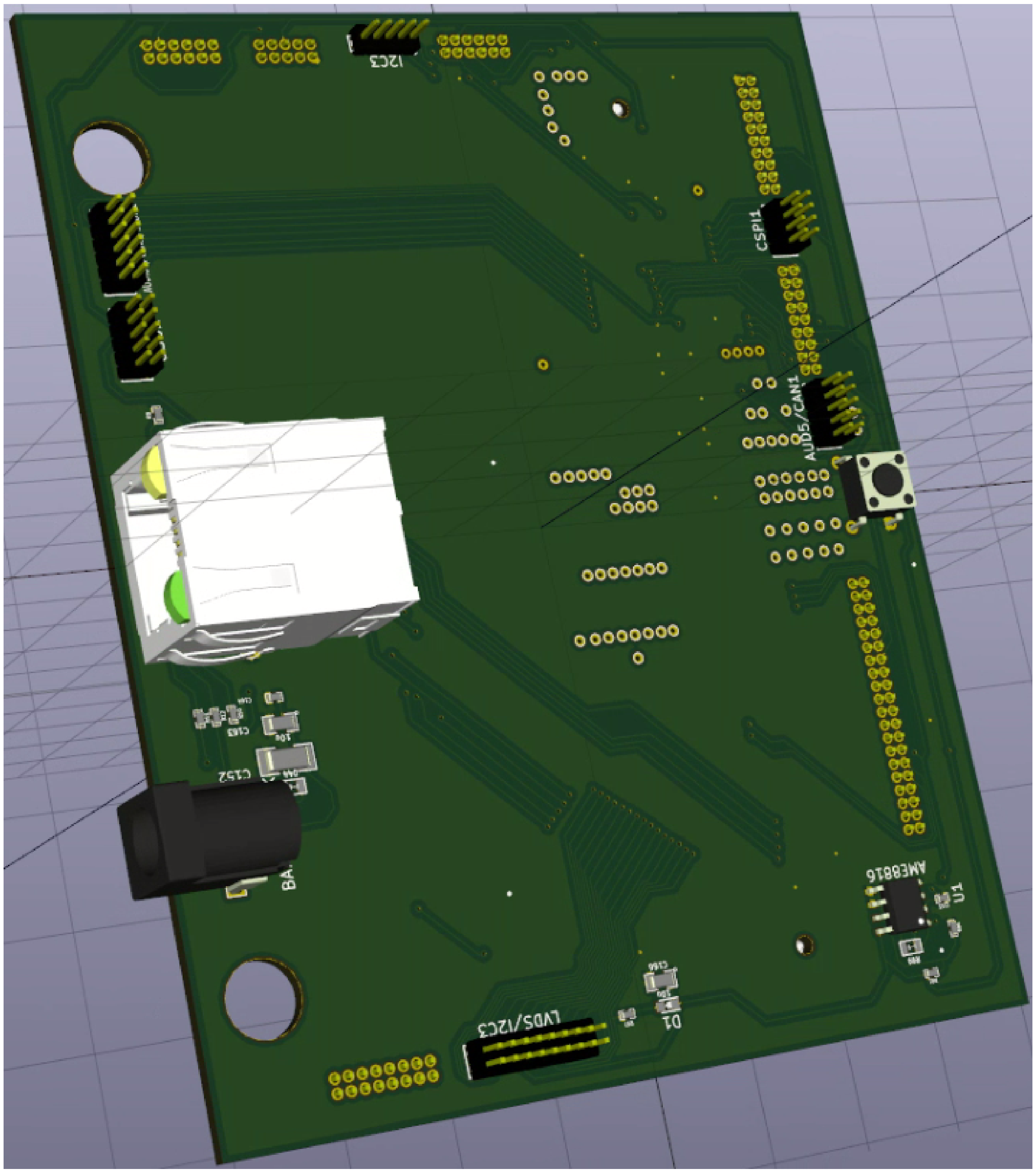


Figure 6.9: Top of the second spin original 3D WVSNP carrier board.

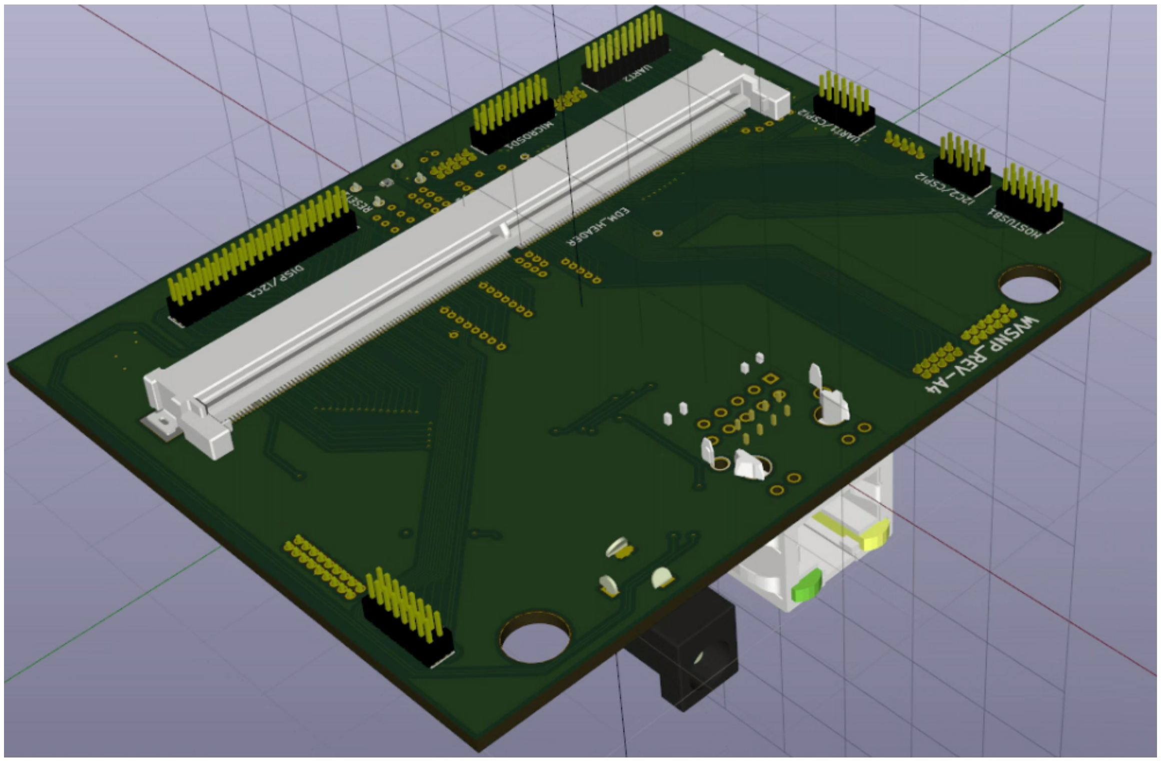


Figure 6.10: Bottom of the second spin original 3D WVSNP carrier board.

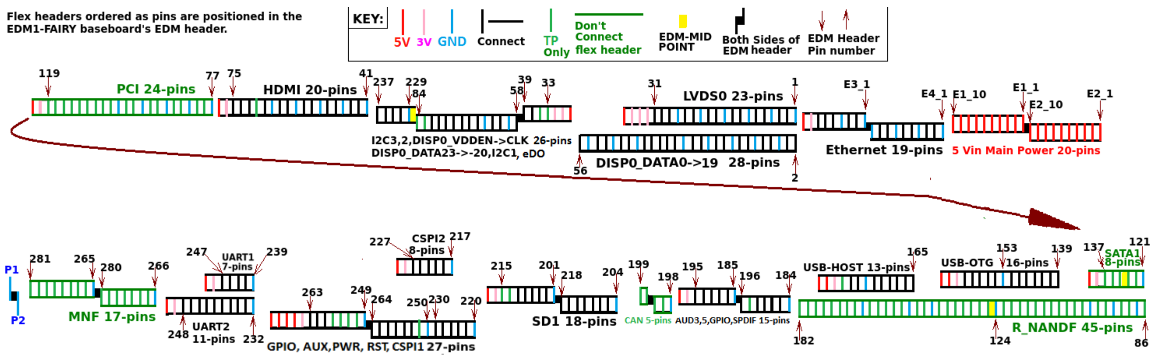


Figure 6.11: Header placement showing controllable red power lines. Green shows unpopulated.

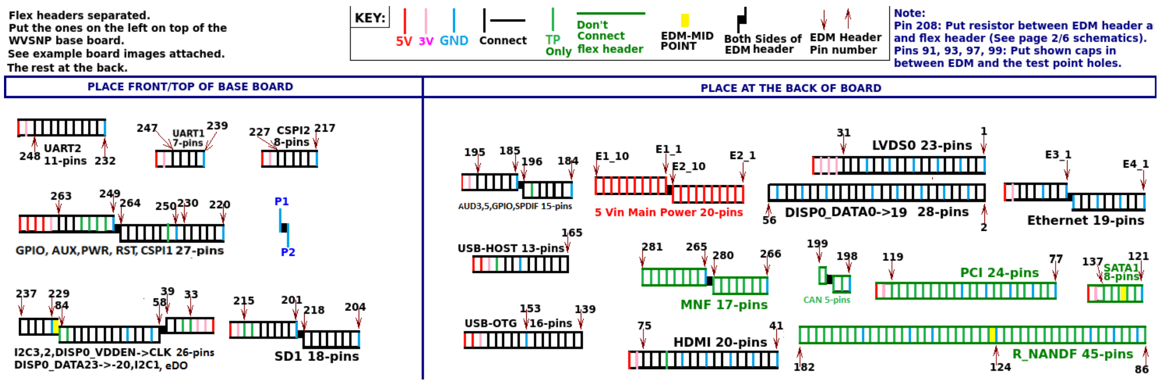

Flex headers separated.
Put the ones on the left on top of the WVSNP base board.
See example board images attached.
The rest at the back.
KEY:
5V 3V GND
Connect
TP Only
Don't Connect flex header
EDM-MID POINT
Both Sides of EDM header
EDM Header Pin number
Note:
Pin 208: Put resistor between EDM header a and flex header (See page 2/6 schematics).
Pins 91, 93, 97, 99: Put shown caps in between EDM and the test point holes.
PLACE FRONT/TOP OF BASE BOARD
PLACE AT THE BACK OF BOARD
UART2 11-pins
UART1 7-pins
CSPI2 8-pins
GPIO, AUX,PWR, RST, CSPI1 27-pins
I2C3,2,DISP0_VDDEN->CLK 26-pins DISP0_DATA23->-20,I2C1, eDO
SD1 18-pins
AUD3,5,GPIO,SPDIF 15-pins
5 Vin Main Power 20-pins
LVDS0 23-pins
DISP0_DATA0->19 28-pins
Ethernet 19-pins
USB-HOST 13-pins
MNF 17-pins
CAN 5-pins
PCI 24-pins
SATA1 8-pins
USB-OTG 16-pins
HDMI 20-pins
R_NANDF 45-pins

Figure 6.12: Header top and bottom showing controllable red power lines.

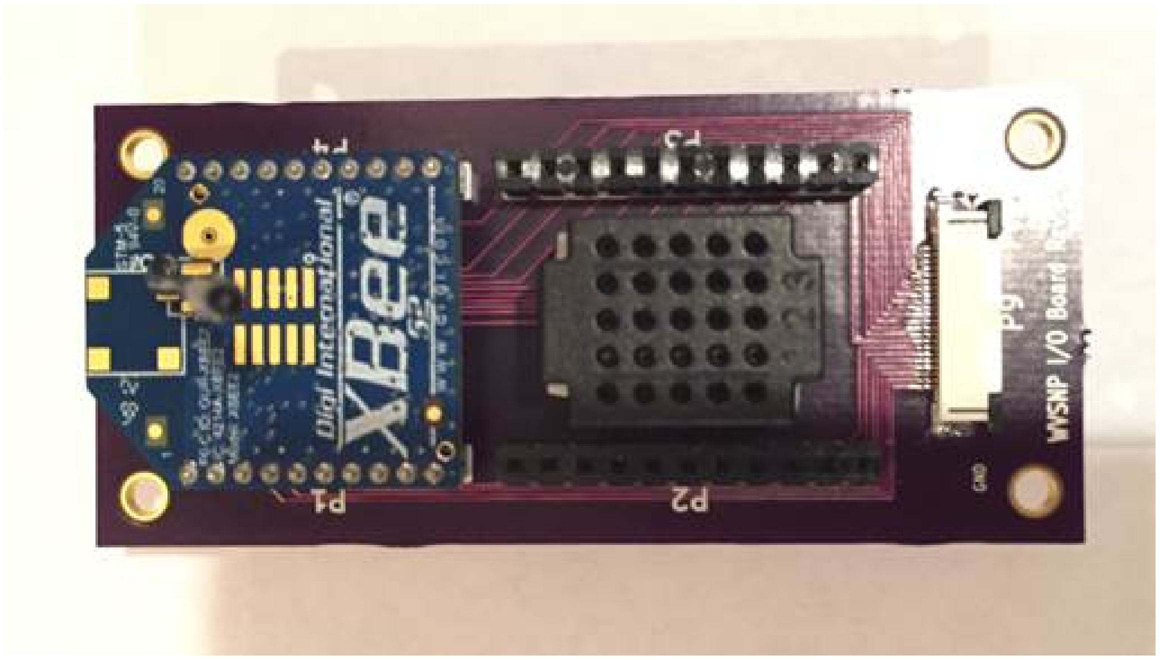


Figure 6.13: WVSNP IO Board finished module top.

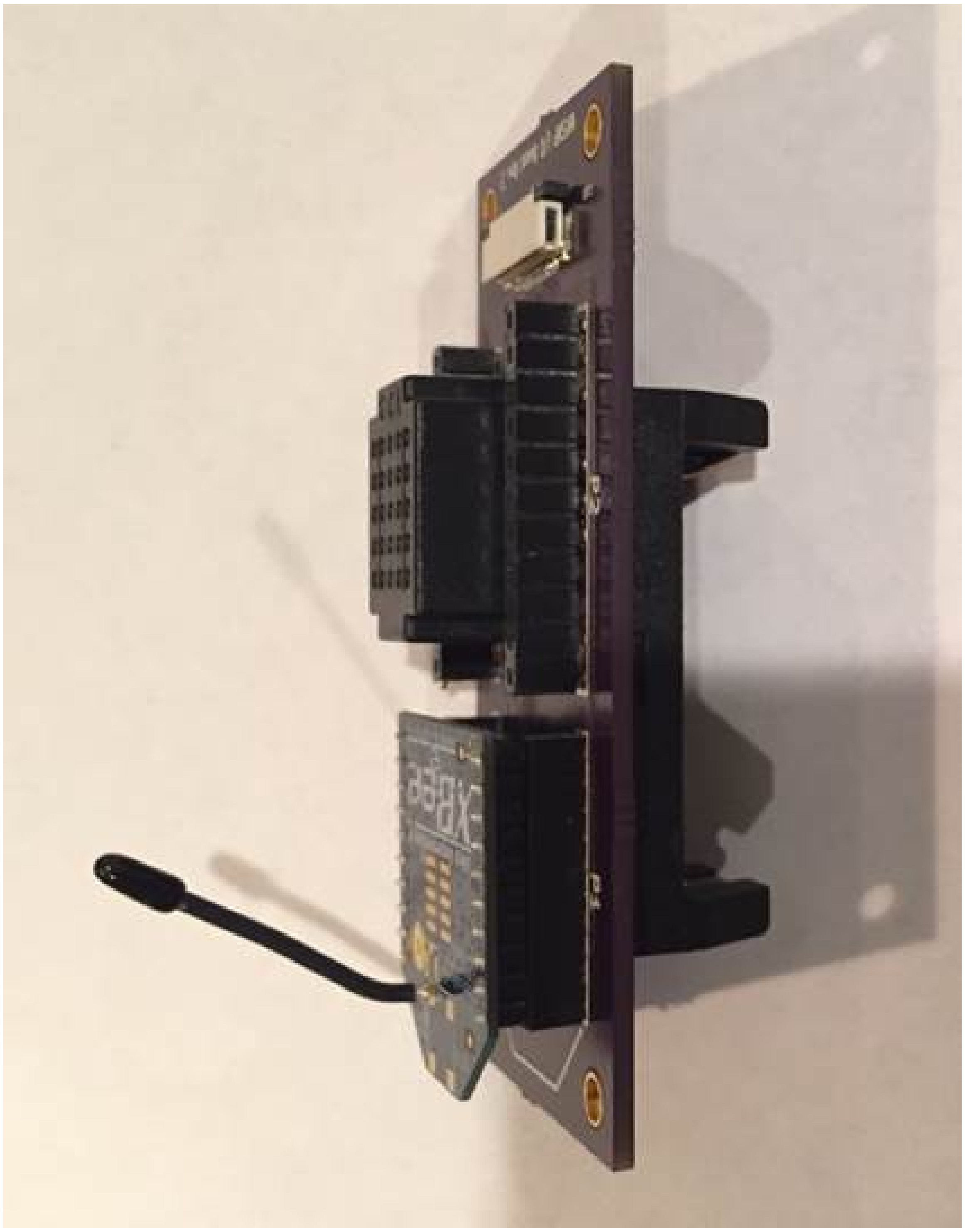


Figure 6.14: WVSNP IO Board finished module side.

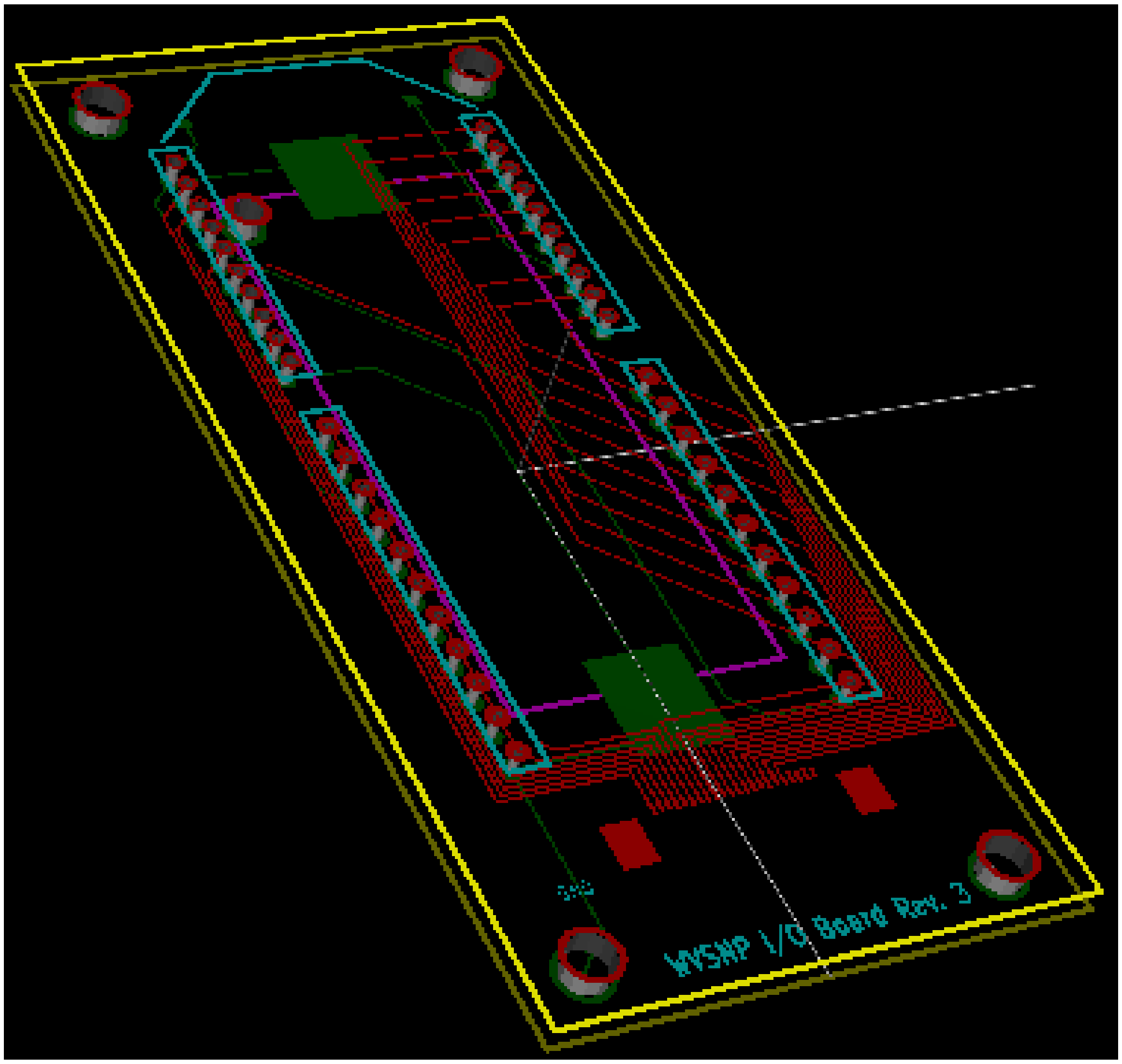


Figure 6.15: WVSNP IO Board modules Finished Layout 3D.

already a file/client server/data setup. (It needs to be incorporated later to parse JSON commands). SSH server: secure remote command line. Drop Bear sftp server: secure command line plus file transfer. To test HTTP server, you should be able to see the `wvsnp_master` WiFi SSID (SoftAP) we can connect to. You should be able to go to http://ip:addr:of:master and land on the WVSNP player page. This can be configured to be LIVE video playing video of whatever the node camera is pointing at, for now it is just the WVSNP player with Video-On-Demand (VoD) ready to play when you click the PLAY icon. And the rest of the dashboard function tabs/menu/settings. Typing http://ip:add:of:master/info OR clicking on dashboard's "Network Info" link should show us all the critical info needed to talk to or interact with the WVSNP network. e.g. MAC Address on Bluetooth, Zigbee, WiFi and other nodes connected to it. Refreshing this link should show any updated information. Especially new nodes that joined since. This is a CGI program that runs to read a JSON file that has this information about nodes. It just parses and outputs the contexts of the `wvsnp_nodes.json` file to the browser calling it. You can upload files to the node and get remote temperatures of whatever you are monitoring for example.

The `wvsnp_nodes.json` file is an important file that can be manually edited to add the nodes connecting to the WVSNP node. It also gets updated anytime a node joins the WVSNP network. If a node exits the network or dies, the WVSNP node hosting this file will remove it from the list after a few heartbeat communication tries that fail. So anytime a new pairing or new communication is established, the new node will get added to the file. Note that this can be edited by the HTTP server daemon programs/CGI programs or Bluetooth/Zigbee daemon programs. To interact with the Bluetooth server: rfcomm server (for chat like interaction and parsing JSON commands), you will first have to make sure board and device are paired and connected. The Bluetooth server uses obex server (for object and file pushes). They all should respond to specific JSON commands. Syntax defined later in the document and updated in a live project page.

The wireless servers are actually duty cycled daemons that run at bootup. They can be stopped/started/talked to via CGI via Mongoose HTTP server. Zigbee server and sensor functions. There is already file/client server/data setup. The node can also control remote WVSNP IO modules discussed earlier in the document.

To securely access the node, an SSH server secure remote command line should already work at bootup. After bootup and Wifi connection, one should be able to ssh into the board and control it from command line using screen window manager in the command line. screen is part of the WVSNP OS Image packages. DropBear OR sftp server secure command line plus file transfer should already work for manual transfer of files or mounting sftp to manually transfer files.

Once the above work, and they have been tested, applications can be tested or developed for the node. There is a `wvsnp_mware.h` library API with functions that can be called by applications running on the node. e.g.

Listing 6.1: Middleware API example.

```
wvsnp_send_data(
  char * none_name,
  char * data,
  int size,
  int timeout
);
```

These functions have CGI equivalents. e.g. `http://ip.addr.of.svr/send_data?data=blhaa&name=N1&size=233`

All CGI functions can be called by the Dashboard App and others developed by users. e.g. One can request a file via Bluetooth and then play that file on their player once received. One can request Temperature from a Zigbee sensor and display that value. `http://ip.addr.of.svr/send_data?data=temp&name=N3`

Additionally one can request temperature just using Bluetooth directly without using HTTP. Then your Bluetooth daemon will catch that request and relay to the Bluetooth node and respond back to the client.

An HTML5 application can also bypass the node altogether and talk directly to the bluetooth enabled sensor via its clients bluetooth radio.

When we scan from the phone for bluetooth devices around us, we should be able to see `wvsnp_master` and mac address in the devices found. We should be able to initiate pair with it from the phone. We should be able to issue ”info” command and get similar info as above in JSON format to be explained in protocol later in the document and live project page.

In the platform’s Zigbee tutorial, there is an example of searching for nodes in a Zigbee network. Running this program from another Zigbee client should be able to find `wvsnp_master` as one of the nodes. There is no pairing concept in Zigbee. The client after finding the MAC address and name from this scan, should be able to send JSON commands to the master and see corresponding JSON responses. One should be able to issue ”info” command and get similar info as above in JSON format to be explained in protocol later in the document and live project page. You can use the Zigbee File Transfer feature in the Dashboard as a starting point to start parsing a JSON command that is requesting a file.

There is a detailed Architectural Platform Tutorial document explaining how to setup, the node, test and use or develop applications using the WVSNP NFS and Middleware.

6.2.1 Peer to Peer (P2P) capability of the platform.

In addition to its ability to switch communication between different protocols on the fly depending, on the application needs, the WVSNP node also provides capability to communicate via Peer to Peer with other announced members of the WVSNP group. This has been tested using the latest WebRTC protocol once the client has launched the Dashboard via the browser. WebRTC is just a the latest tool used to prove this capability but the protocol the platform uses is applicable to any Peer to Peer network. The tested capability uses the Data Channel component of WebRTC to enable segment or data transfer between the node and its peer. There is a detailed document on how to setup and use the WVSNP P2P feature.

In summary, to be able to use WebRTC feature for file transfer, the node needs to have ran the Dashboard/webserver at least once to have access to the `wvsnp_peer.json` list which dynamically updates as neighbors/members announce themselves in a P2P room.

Suppose that client A is being run at a Starbucks restaurant while playing some videos from the server over celllular network. If another client B in a laptop belonging to another customer happens to have the files already in their laptop or phone, it might be better to fetch the next segment from the neighbor instead of going over the cellular network. All WVSNP clients can be configured to add themselves to the `wvsnp_peer.json` list with the segment types that they are willing to share, "published segments".

Just like the unique filename of a WVSNP segment, the name of the segment file implies a WebRTC room that is created by whichever client first creates it. Peer can join this room and leave as they wish. Joining this room implies publishing that you can provide these segment types.

Client A therefore can create a room and add itself to the `wvsnp_peer.json` for that segment type room/channel. Client B during a routine call back to the WVSNP server can join the room as well, which signals that it is ready to share.

After a room is created or joined each node creates a file with the necessary network information to be exchanged with the other peer who intends to join the room. This is saved in the server room directory in the JSON format and also saved in client's local machine, e.g. `roomX/client_session_a.json`, This data file is the session variable.

Client B can download `roomX/client_session_a.json` file.

Client B must also upload its `roomX/client_session_b.json` file created when it joined.

Client A can look up Client B in `wvsnp_peer.json` and then grab Client B's session from the `roomX/` directory.

Client A uses this session data to establish direct session connection with Client B.

Once this is done, either client can fetch segments that belong to this room from each other without the need for a server anymore. This will be a direct browser to browser peer connection.

An interesting thing to note here is that the session setup can be done via either Zigbee or Bluetooth or any other protocol aside from HTTP. This is very important feature of the WVSNP node that will be showcased a lot as more of its implications become apparent.

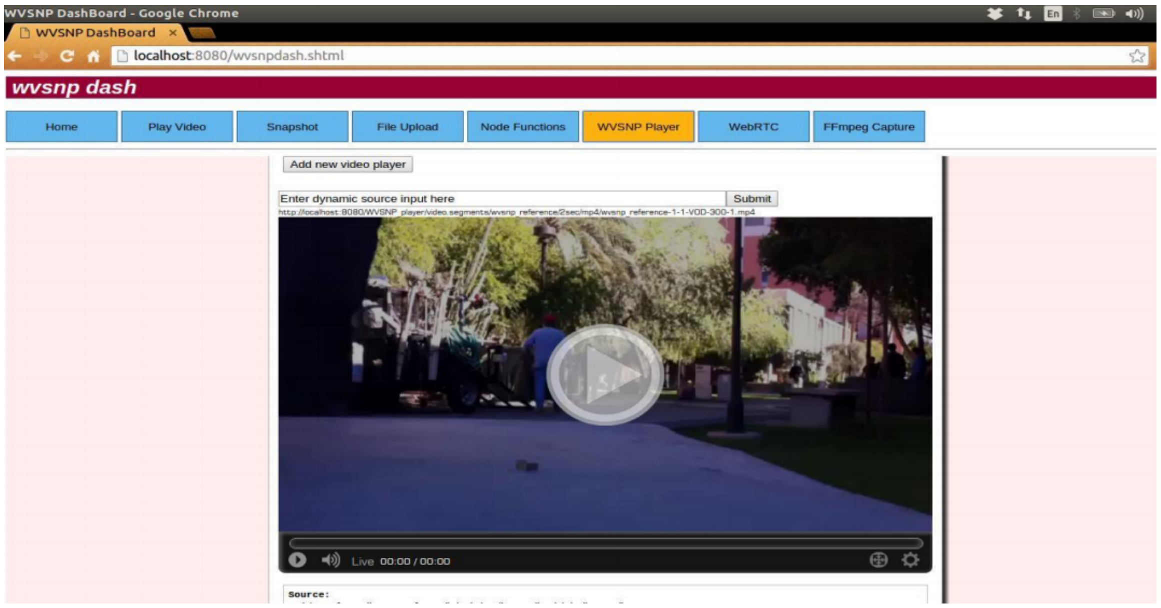
6.2.2 WVSNP OS Image.

As mentioned earlier, the WVSNP node has its own minimal embedded Linux operating system (WOS) customized for WVSNP applications. There is a detailed and exhaustive platform tutorial on how to build, configure and test a WVSNP OS and development environment. Yocto is the main toolchain used to build WOS. A WOS image contains the expected dynamic device tree to adapt to different boards without recompiling, a higher level u-boot for critical early bootup services like Ethernet, USB, serial communication and some Trivial File Transfer Protocol (TFTP) capability. The minimal core Linux Kernel can be loaded to SD card, local flash, or loaded at boot over the network if the application setup requires frequently updating kernel. The kernel has only the bare modules needed for networking, camera support and hardware accelerated codecs and security modules. The minimal default root file system package list added to the image beside the Linux kernel are `vi` as default editor; `imx_tests` for testing all HW on the board after bootup; `mongoose` web server; `gstreamer` multimedia framework for video processing (including plugins `_good, _bad, and _ugly`); `openssh` for secure remote login and control; `vsftpd` for secure remote file transfer; `x264` for software based Advanced Video Codec (AVC) compression; `screen` for managing multiple windows and processes over the Linux shell command line locally or remotely; `ffmpeg` for multimedia processing; `libxbee` for managing Zigbee networking modules; `WiFi` and the SoftAP WiFi hot-spot capability; and `Bluetooth` for bluetooth wireless compatibility Additional packages include development tools when needed that come with Network File System (NFS) setup for easy application development.

Also included of course is the WVSNP Dashboard, which is the user facing module served by the node's HTTP server. It contains the WVSNP video player module shown in Figure 6.16 below, video capture module as shown in Figure 6.18, browser interface to remote Zigbee, Bluetooth and other protocols as shown in Figure 6.17 below and other convenient functionalities to manage and control the WVSNP node from any device with a browser as shown by other tabs of the Dashboard.

Additional packages for DASH and HLS segmentation can be added if Ffmpeg and Gstreamer are not enough for a user's application.

It is important to note that the WOS uses loadable modules for most of its hardware driver modules which satisfy the DyCOMs concept of an ideal WVSNP design



http://localhost:8080/WVSNP_player/video.segments/wvsnp_reference/2sec/mp4/wvsnp_reference-1-1-VOD-300-1.mp4

Figure 6.16: WVSNP Dashboard video player module.

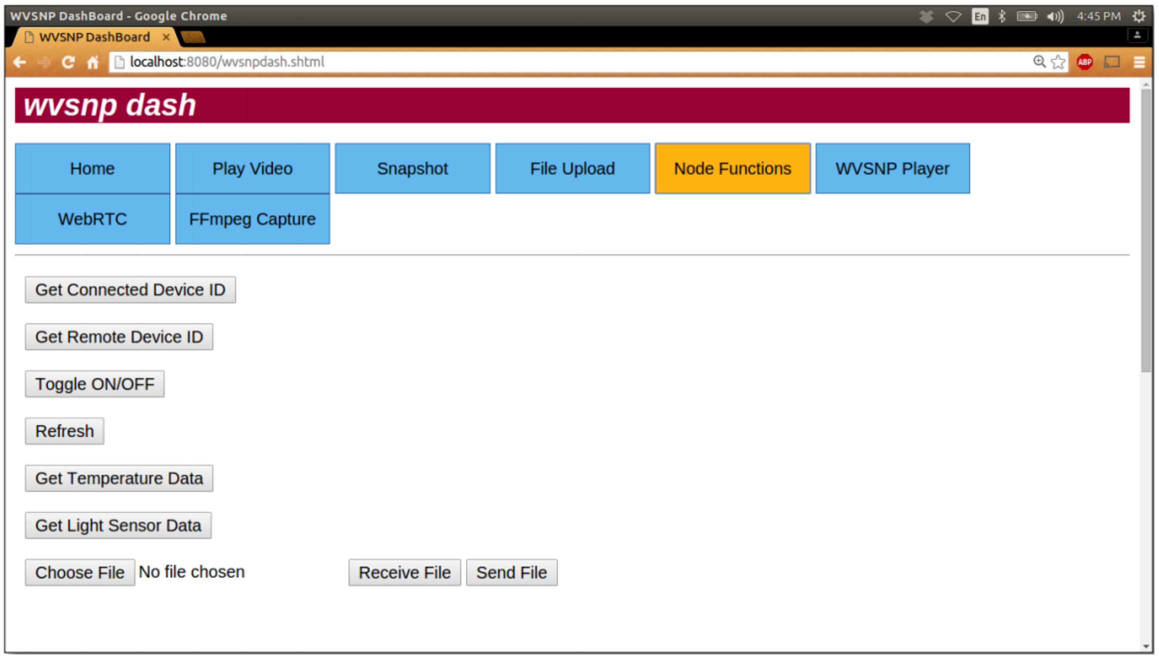
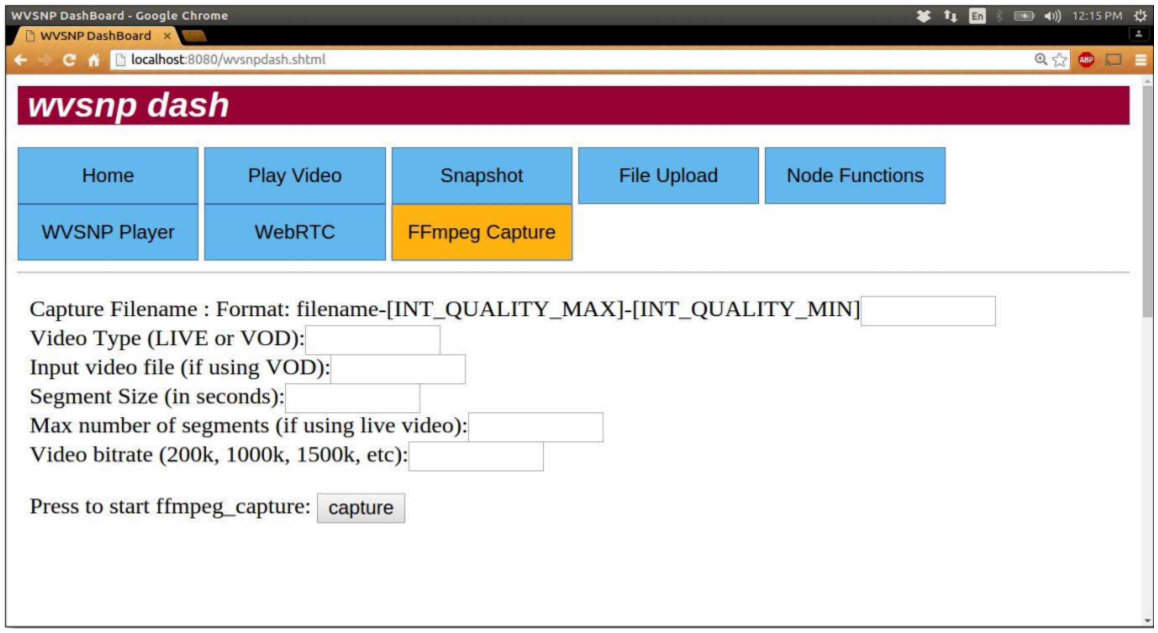


Figure 6.17: WVSNP Dashboard remote node functions module.



Capture Filename : Format: filename-[INT_QUALITY_MAX]-[INT_QUALITY_MIN]
Video Type (LIVE or VOD):
Input video file (if using VOD):
Segment Size (in seconds):
Max number of segments (if using live video):
Video bitrate (200k, 1000k, 1500k, etc):

Press to start ffmpeg_capture:

Figure 6.18: WVSNP Dashboard video capture module.

stated earlier in section 3.3.

6.2.3 Security.

As they say in real estate, location, location, location. In IoT we might as well say security, security, security. First on the hardware side of this the i.MX SoC used by the WVSNP is probably one of the most comprehensive security capable chip this research has seen for such low cost scalable SoC. almost all needed security tools fo IoT are HW accelerated and ready to be used by loading the appropriate security module and as needed by the application. Disclosed i.MX6 CAAM HW security features (those with no NDA) show an impressive HW Cryptographic Acceleration Assurance Module (CAAM). At high level, CAAM is a DMA master supporting the following capabilities. The inherited NXP Linux BSP layer contains a CAAM module to make use of the security features via the Linux CryptoAPI.The driver itself is integrated with the Crypto API kernel service in which the algorithms supported by CAAM can replace the native SW implementations. Additionally the SoC support ARM TrustZone, Secure Non-volatile Memory, Tamper Detection, High Assurance Boot (HAB) and a Real Time Integrity Checker.

In addition to ARM Trust zone, the WVSNP node's SoC has a Secure Memory feature with HW enforced access control. Cryptographic authentication features include: Hashing algorithms: MD5, SHA-1, SHA-224, SHA-256; Message authentication codes (MAC): HMAC-all hashing algorithms; AES-CMAC, AES-XCBC-MAC, Auto padding, ICV checking; and Authenticated encryption algorithms: AES-CCM (counter with CBC-MAC). the Symmetric key block ciphers offered are: AES (128-bit, 192-bit or 256-bit keys), DES (64-bit keys, including key parity), 3DES (128-bit or 192-bit keys, including key parity). They have the cipher modes: ECB, CBC, CFB, OFB for all block ciphers, and CTR for AES. CAAM also has symmetric key stream ciphers ArcFour (alleged RC4 with 40 - 128 bit keys), and Random-number generators. Entropy is generated via an independent free running ring oscillator. The oscillator is off when not generating entropy; for lower-power consumption. The generator is NIST-compliant, and its pseudo random-number generator is seeded using hardware generated entropy. There are more protocols supported such as the Public Key Infrastructure via the PKHA and also IPsec for example.

No matter how impressive this might look what is important is how these features are used by the application. There are cases where using HW acceleration might consume more power without a good net gain in throughput. Some applications might actually not use the HW module correctly resulting in inefficiency versus SW only solutions. All these cases can be sorted out by application profiling before finalizing the application design.

For IoT, there are resource constraints that precludes small nodes from managing huge database and networks of keys and server and third party certificate authorities

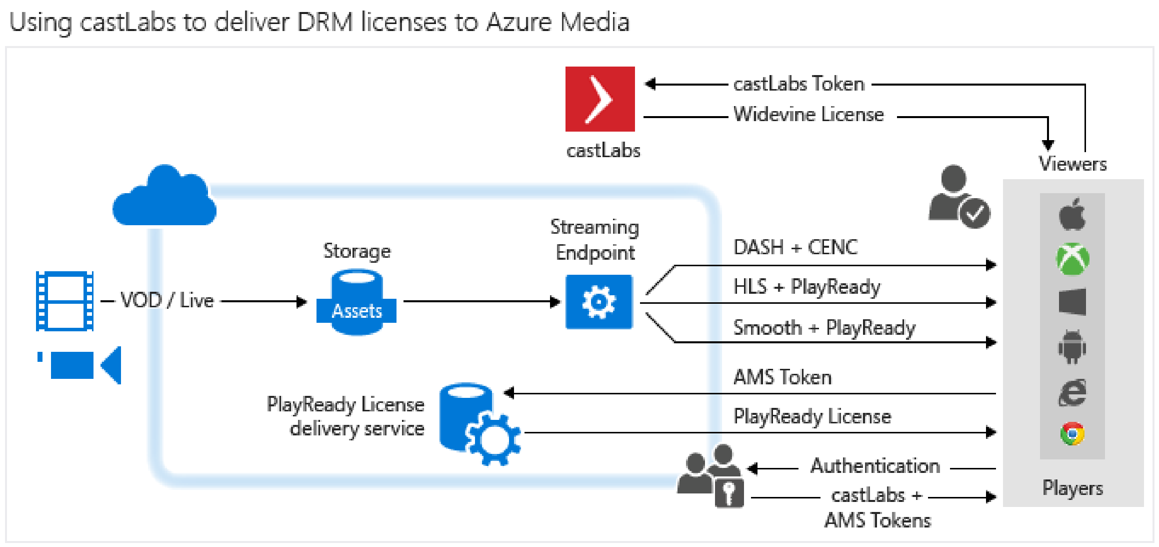


Figure 6.19: Example Widevine and Playready DRM setup for DASH (Cast Labs) [235].

and so forth. Examples in Figure 6.19 and Figure 6.18 shows just how complicated these setups for Widevine and Playready [235, 243] data protection setups can be.

The problem with setups like Figure 6.19 and Figure 6.20 is not that they are complicated. The protocols for managing the keys assumes TCP/IP and HTTP delivery of the video content. They also do not seem to consider power consumption much. For WVSNP nodes that mix up different protocols and that can send data from one HTTP server to a Bluetooth receiver and others this is not adequate. Additionally once data is decrypted, there seems to be trust given to the source of the data (server or certificate authority) instead of the data itself.

The WVSNP framework is a data first framework that is generally agnostic of the data source nor physical medium or layer the data is transmitted through. So it is important that if there is need for security, each piece of data be protected end to end by itself and unlocked only by the requesting client. This fits very well withe the core concepts of Information Centric Networks (ICN) which are a good use case for

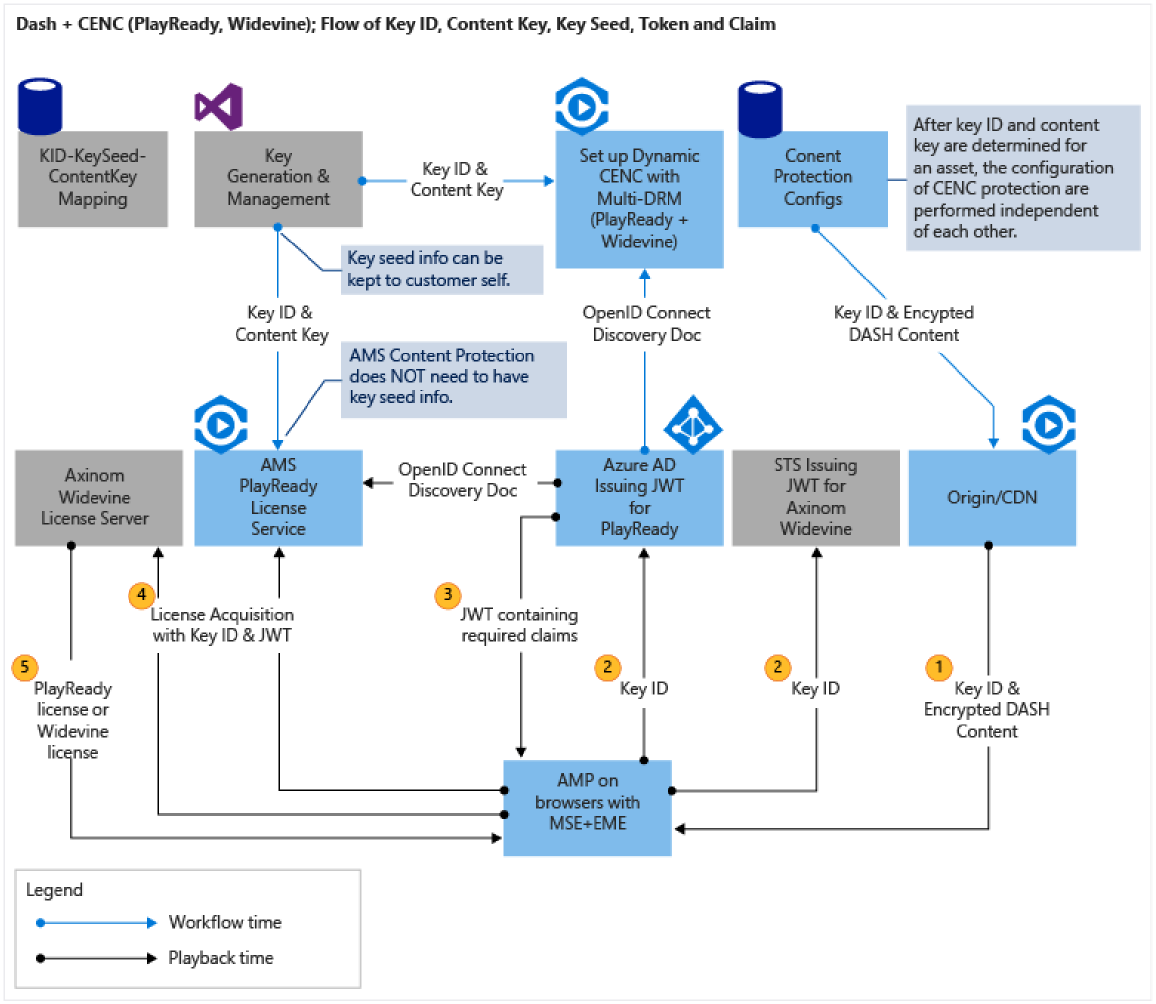

Dash + CENC (PlayReady, Widevine); Flow of Key ID, Content Key, Key Seed, Token and Claim
KID-KeySeed-ContentKey Mapping
Key Generation & Management
Key ID & Content Key
Set up Dynamic CENC with Multi-DRM (PlayReady + Widevine)
Conent Protection Configs
After key ID and content key are determined for an asset, the configuration of CENC protection are performed independent of each other.
Key seed info can be kept to customer self.
Key ID & Content Key
OpenID Connect Discovery Doc
Key ID & Encypted DASH Content
AMS Content Protection does NOT need to have key seed info.
Axinom Widevine License Server
AMS PlayReady License Service
OpenID Connect Discovery Doc
Azure AD Issuing JWT for PlayReady
STS Issuing JWT for Axinom Widevine
Origin/CDN
4 License Acquisition with Key ID & JWT
3 JWT containing required claims
5 PlayReady license or Widevine license
2 Key ID
2 Key ID
1 Key ID & Encrypted DASH Content
AMP on browsers with MSE+EME
Legend
Workflow time
Playback time

Figure 6.20: Example Widevine and Playready DRM setup for DASH (Axiom) [243]

the WVSNP-DASH framework.

In ICN, the client requests content. The client does not need to know the providing host. The path to the data is established by the request receiver to the client. Communication follows a receiver-driven approach with the data following the reverse route of the request. The ICN is responsible for mapping the requested data and its location. ICN focuses on providing efficiency is naming. Content must be named independent of the node providing the content. A provider just has to publish one of the segments (for WVSNP example) for the Content Centric Node (CCN) router to know available publishers of those segments. WVSNP node then has to check their provider table several times to see if content they are interested in available. Similar to the WVSNP P2P protocol we defines in section 6.2.1 above.

An ICN subscriber expects data objects to carry security metadata for authenticating the integrity of the objects. The object itself must be secure not where it is from. In ICNs the named data, rather than its physical location, is the key component of routing in ICN. This can easily be encoded into the name of the WVSNP segment.

As long as the WVSNP node can encrypt its segments with keys in an efficient way depending on the application those segments can be advertised and shared between the ICN network or even across multiple protocols intact as long as the final consumer of the segment has the public or private key needed. This enables the WVSNP application to not worry about networking issues but focus on generating and storing data efficiently as long as they encrypt it immediately. This might be a simple as just using basic AES on each segment. Because a WVSNP node has multiple protocols or networking mediums, one can see key management be provisioned in a completely separate network than the data channel. This document leaves key management to assert management policies of the data.

The WVSNP node provides the popular OpenSSH by default and the HTTP server can be configured to use `https` only. Additionally there is physical security mentioned above in addition to the 256-bit Manufacturing Protection input from the Secure Fuse Processor (SFP) as well as signed u-boot and kernel. Some hashes are fused on the SoC with permanent irreversible operation. This cannot be un-done like secure boot in many personal computers

Again, all this talk gets proven or dis-proven by power profiling, throughput, and memory, benchmarking, etc. We hope to revisit this soon.

6.2.4 IoT Relevance of the WVSNP Framework. Video Use cases.

IoT is a whole new ball game. Rethinking of video server client and network itself is very important and overdue. The WVSNP node and framework addresses these. Requirements for IoT require interaction with other networks other than just HTTP. The ability of the WVSNP framework to enable ICN use case has already been mentioned above and at length in the Security section 6.2.3. ICN enablement by WVSNP-DASH is an important contribution Flexi-WVSNP provides because it is based on communication being driven by recipients requesting named data objects. Providers publish the objects to make NDOs available to receivers. Publishers might have the same server but one might be less trusted than the other. This means that a WVSNP-DASH client can dynamically switch for the next segment depending on dynamic security changes in the previous segments. If object to object authentication fails or signs of tempering between segments and so forth. This assumes the WVSNP segments are secured as proposed in the Security section 6.2.3 using simple segment to segment encryption. Another advantage was that key exchanges can be handled in a different data channel other than the data channel if needed.

The second key use case of the WVSNP-DASH framework is detailed in the P2P section 6.2.1. This can be done by HLS and MPEG-DASH but it requires switching the entire stream and reinitializing the manifest files and initialization files.

Again for IoT, HLS and MPEG-DASH assume an HTTP server from source WVSNP-DASH does not. As shown in Figure 6.21, assume an application remotely rendering a complex mix of video, heatmaps, IR data, radiation flow in a nuclear reactor or maybe a 2D Heat gradient, Pressure Currents, and Deposition video in a semiconductor chamber from any device to (one player) screen. With WVSNP-DASH

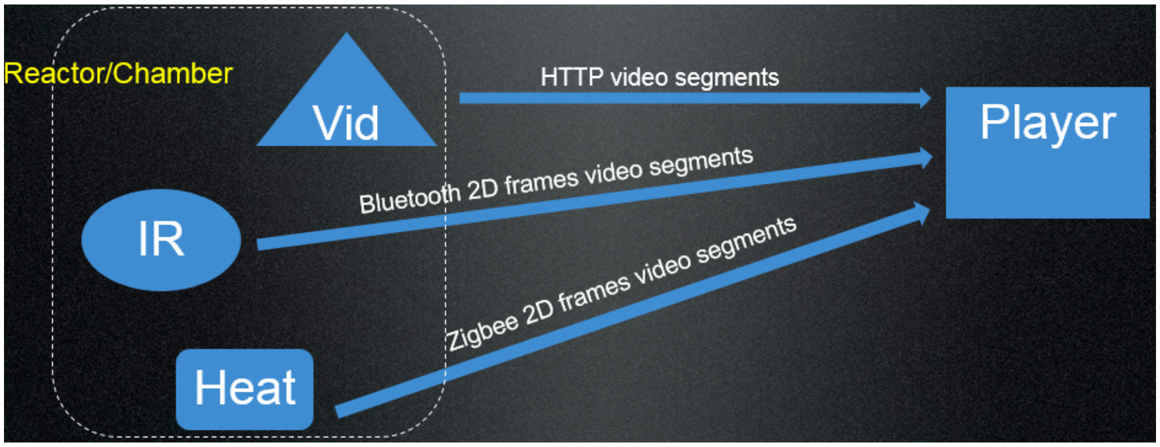

Reactor/Chamber
Vid
IR
Heat
HTTP video segments
Bluetooth 2D frames video segments
Zigbee 2D frames video segments
Player

Figure 6.21: Example mixing 2D and video segments for complex application rendering.

is just a matter of fetching the segments from each of the remote sources in round robin and deciding what to put next on the screen without need to create complex fusion re-computation of manifests. Any next segment available in the local buffer is ready for the screen rendering can be indexed in order or interleaved as and when needed. It is truly a random streaming framework once one of the segments of each stream are parsed. Again HLS and MPEG-DASH have no concept of other possible non-TCP/IP sources. Think about the mushrooming 5G and other parallel networks that are being created to support non-cellular/non-LTE traffic. e.g. LP-WAN such as LoRa and Sigfox. All these can be taken advantage of to switch networks for the next segment based on the quality or security needs.

Again, the failure to foresee IoT validates WVSNP-DASH in the scenario shown in Figure 6.22 below. Suppose that you are watching video or a sports game from a cellular network. If you notice that the video reception is slowing down and your mobile device senses other devices close by that are watching the same game, it can fetch the next segment from its WiFi-direct peers instead of from a far away server. Or in another examples assume you are watching the superbowl and you are streaming

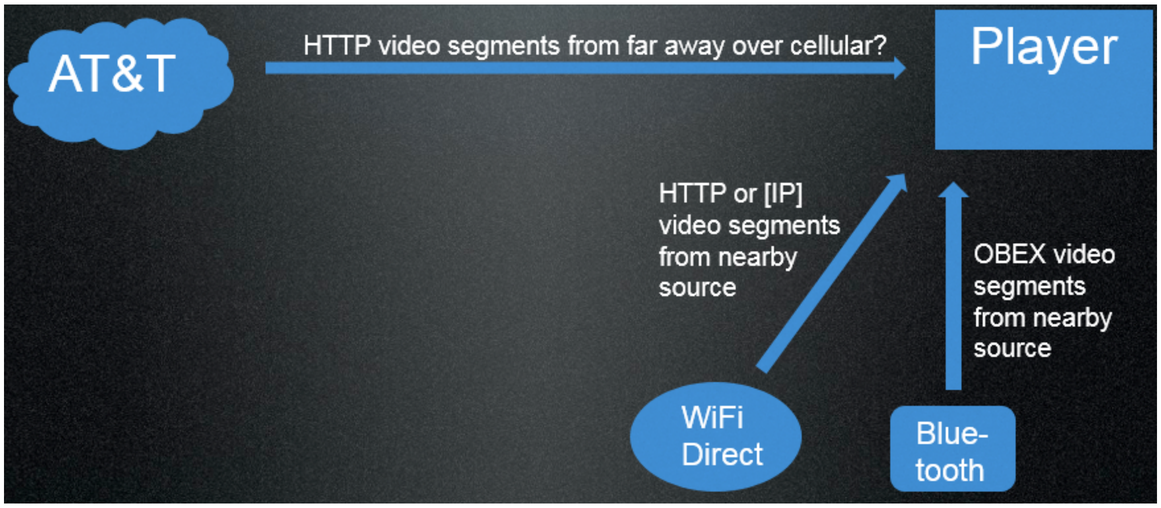

AT&T
HTTP video segments from far away over cellular?
Player
HTTP or [IP] video segments from nearby source
OBEX video segments from nearby source
WiFi Direct
Blue-tooth

Figure 6.22: Example using proximity sensing of networks to switch video source.

via your Internet Service Provider (ISP) who has a data cap. Other device in the house instead of fetching their segments from ISP they can fetch them the DVR box via Bluetooth or WiFi to avoid your data cap going high. A delay of some milliseconds can be implemented to make it appear like all devices are watching LIVE TV at the same time. There is no need for your phone or tablet or watch to go through your WiFi router via ISP to fetch its own redundant stream. A manifest file in HLS or MPEG-DASH would need to be different for each device for this to happen. Also a lot of changes in the frameworks would be needed. In WVSNP-DASH case, this are just dynamic decisions made by the intelligent client using proximity data.

For resource constrained sources like smart watches and other constrained 2D data sensors and clients, all they should worry about is capturing data and sending it when requested. Not managing manifest and initialization files that consume scarce power.

Another interesting use case for IoT by WVSNP-DASH is in area of Content Delivery Networks (CDN)s. Assume that a CDN manager keeps a table of edge network server latencies in a table every-time they fetch a segment. One could see

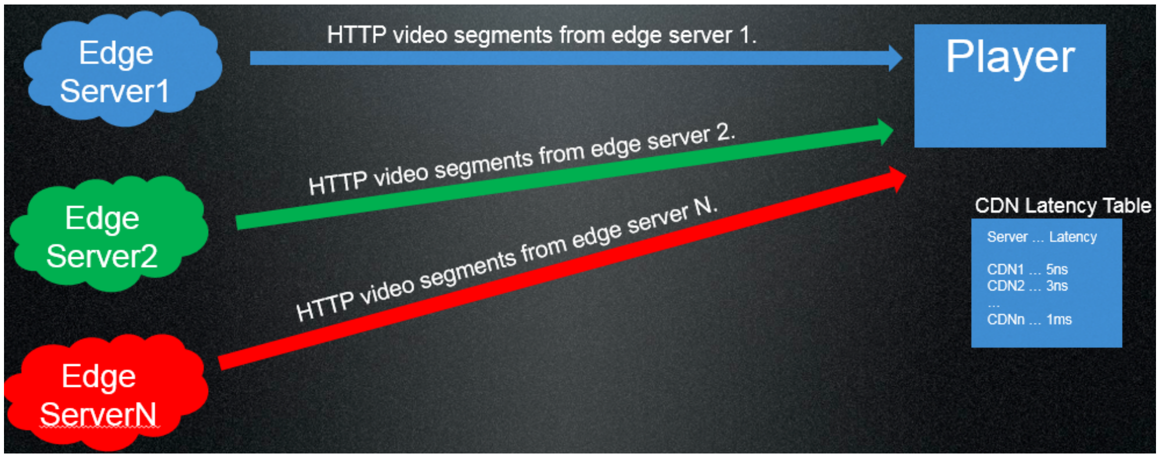

Edge Server1
Edge Server2
Edge ServerN
HTTP video segments from edge server 1.
HTTP video segments from edge server 2.
HTTP video segments from edge server N.
Player
CDN Latency Table
Server ... Latency
CDN1 ... 5ns
CDN2 ... 3ns
...
CDNn ... 1ms

Figure 6.23: Example using proximity sensing of networks to switch video source.

them fetching the next segment only from the edge with a previous lowest latency. For now they always have to keep duplicate manifest files and re-initialize files for every edge switch. This becomes more complicated for LIVE events a they have to plan switches ahead of time and keep a low performing edge up longer than needed while recomputing and reinitializing for any possible server switch. This wastes a lot of power. For now CDNs don't seem to mind power consumption much, nor the relatively little switch delay disruption they need to statically plan for. But when they start getting more and more of their data from IoT devices in the future, this will be a problem. WVNSP-DASH solves this problem.

For HLS and MPEG-DASH, multiple sources require repeated manifest files. One at a time. Imagine future applications were there is a LIVE event being streamed by multiple phones or WVSNP devices. With WVSNP-DASH, they can all provide the same set of segments. The client can then randomly get the next segment to view different views as needed without much camera switching. This cannot be done with the current DASH technologies.

To repeat, the current DASH technologies are not truly random. They always have to require initialization files for player. Not truly random access network wide. WVSNP-DASH only needs one random segment passed to it and it will be able to stream the video available to it, fast forward and rewind without any reinitialization. For example a client can search for segments on a network and if it finds maybe segment number seven, it should be able to fetch the next segment and prior segments if they exist.

The fact that Flexi-WVSNP can deal with mesh network radios gives it an upper hand in that it is able to stream data within mesh networks which other DASH technologies cannot do. They manifest file management and regular updates.

WVSNP-DASH enables LIVE updates and live commercials without any coordination needed with the source of the live video. To insert advertising segments, HLS and MPEG-DASH need a pre-prepared video list and their manifest files require hard coded network sources.

As you can see there are many unlimited possibilities that this framework and the Flexi-WVSNP it supports opens up the new world of IoT.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This work provides an extensive introduction to a new WVSNP platform that is robust and yet very highly adaptable to the new Internet of Things paradigm. There has been new concepts introduced and tested on the best way forward for building low cost and highly adaptable sensor node. Extensive power profiling of video capture, streaming, architectural path and flow choices on a wireless video sensor node (WVSn) has been done an analyzed to aid in relevant architectural choices. This work thus provides empirical baseline power consumption and throughput data based on the architectural components of a node as well as the effects of streaming frameworks and applications. As part of the evaluations of better DASH frameworks suitable for WVSNs, this work introduced Wireless Video Sensor Network Platform compatible Dynamic Adaptive Streaming over HTTP (WVSNP-DASH) framework. The WVSNP-DASH framework specifies a naming syntax for independently playable video segments. Existing DASH frameworks convey video meta data through a manifest file and begin video streaming with a special initialization video segment; subsequent video segments depend on the manifest file and initialization segment for playback. In contrast, the WVSNP-DASH video segments convey essential meta data through their name and can be played independently, i.e. each individual WVSNP-DASH segment is fully playable without reference to any other file or segment. This file independence simplifies the video capture and video file segment creation and streaming by a sensor node.

The comparative evaluation of a WVSNP-DASH against HLS and MPEG-DASH players indicated that the independently playable WVSNP-DASH video segments create significant potential for power savings on the sensor node serving the video. To the best of the authors knowledge the presented evaluation is the first to examine the effects of different DASH frameworks on sensor node power consumption, CPU usage and memory consumption. Evaluation shows that WVSNP-DASH' saves power compared to the popular DASH frameworks, especially HLS. For LIVE video playback an average of 15% or more power is saved.

Due restrictions across browser platforms only the mp4 container was analyzed extensively. For future work other containers should be tried for WVSNP videos because power consumption and size of the segments with other containers might be lesser than mp4 container. Also much more sensitive power meters might improve the results where the differences in power consumption if not very clear due to the resolution of the current meter and sampling rate. As recommended in detail under 5.3.4 there is a lot of insight to be derived from this to improve the design of WVSn. The next step is to incorporate these revelations into the ongoing improvement of the WVS node platform (Flexi-WVSNP) as we plan to mass produce it and use it in a wider scope to gather more real world data. This node is designed from the ground up to support sensor data fusion with video as part of the data elements. The node would be the first node to support WVSNP-DASH as default with the ability to integrate with non TCP/IP sensor networks and the greater Internet of Things.

To better support WVSNP-DASH for sensor networks, refinements of the prototype retriever and player noted throughout the work were added. For instance, Media Source Extensions (MSE) have recently been increasingly adapted by web browsers, thus incorporating the ability to use MSE to play WVSNP-DASH data was investigated and implemented with some discoveries along the way to make this and option and finally the default mode for the WDP to play video. This would become a useful feature for ensuring broad cross-platform support of the WVSN nodes. The WDP prototype manually selects the desired quality level of a video stream. Dynamic adaptation have been extensively covered in the literature, e.g., [56, 77, 104, 119, 128]. A module can be added to WDP for instantiating specific automated dynamic adaptation algorithms including those influenced by the power budget of the server node.

This platform introduced new ways for wireless sensor nodes to perform peer to peer WebRTC data exchanges and new ways to secure data based on object based security usually preferred by Information Centric Networks. There is universal cross platform phone application being developed as the future client interface to the Flexi-WVSNP network beyond the current browser based interface.

The DASH-WVSNP framework introduced is a concept not only limited to HTML5 and web technologies. The framework will be implemented into popular open source tools like ffmpeg, VLC and others to facilitate its adoption.

Evaluation results in Chapter 5.3.4 also brought in recommendations to create optimized capture applications to speed up the LIVE playback from the node. This would reduce the need for large buffers that delay live segment playback. Additionally this work produced patent pending real world framework that might see future adoption if its use cases are proven via sustained demonstration projects and funded large scale deployments. We plan to study the sensor network interaction of the WVSNP-DASH framework with a wide range of access networks [210, 211, 214, 213, 83, 20], including wireless, fiber-wireless (FiWi) [42], and DSL networks and especially how it can benefit Content Delivery Networks (CDNs) and [140, 137, 139]Information Centric Networks.

Beyond the research, there are plans to transform this platform into a product and create an open source and open hardware community around it. Many use case demonstration projects will be created around the platform to solicit more funding and commercial/research ventures and to use it as a starting point for many Capstone and research projects at ASU as well as other Internet of Things, big data and other video related research and startups. Ideas such as visualization of sensor nodes and sensor networks [26, 25] can benefit from this as well.

REFERENCES

[1] "BDTI Focus Report: FPGAs for DSP, Second Edition", Available at http://www.bdti.com/products/reports_fpga2006.html (2008).

[2] "Mit senseable city laboratory", URL http://senseable.mit.edu/ (2011).

[3] "Pachube - data infrastructure for the internet of things", URL http://www.pachube.com/ (2011).

[4] "Android power management on i.MX6DQ/DL", https://community.freescale.com/docs/DOC-93884/version/1 (2012).

[5] "ISO/IEC 23009-1:2014 - information technology—dynamic adaptive streaming over HTTP (DASH) – part 1: Media presentation description and segment formats", (2014).

[6] "FFmpeg", URL http://ffmpeg.org (2017).

[7] "Matroska the extensible, open source, open standard multimedia container.", URL https://www.matroska.org/technical/specs/index.html (2017).

[8] Abas, K., C. Porto and K. Obraczka, "Wireless smart camera networks for the surveillance of public spaces", IEEE Computer **47**, 5, 37–44 (2014).

[9] Accardi, K. and A. Yates, "Powertop user's guide", URL https://01.org/powertop (2007).

[10] Adams, J., "Building low power into wireless sensor networks using ZigBee technology", in "Proc. of Sensors Expo Chicago", pp. 1–33 (2005).

[11] Adzic, V., H. Kalva and B. Furht, "Optimizing video encoding for adaptive streaming over HTTP", IEEE Trans. Consumer Electronics **58**, 2, 397–403 (2012).

[12] Akhtar, F., M. H. Rehmani and M. Reisslein, "White space: Definitional perspectives and their role in exploiting spectrum opportunities", Telecommunications Policy **40**, 4, 319–331 (2016).

[13] Akyildiz, I. F., T. Melodia and K. R. Chowdhury, "A survey on wireless multimedia sensor networks", Computer Networks **51**, 4, 921–960 (2007).

[14] Akyildiz, I. F., W. Su, Y. Sankarasubramaniam and E. Cayirci, "Wireless sensor networks: A survey", Computer Networks **38**, 4, 393–422 (2002).

[15] Al-Ali, A. R., I. Zualkernan, A. Lasfer, A. Chreide and H. Abu Ouda, "GRPS-based distributed home-monitoring using internet-based geographical information system", IEEE Trans. Consumer Electronics **57**, 4, 1688–1694 (2011).

[16] Anastasi, G., M. Conti, M. D. Francesco and A. Passarella, “Energy conservation in wireless sensor networks: A survey”, Ad Hoc Networks **7**, 3, 537–568 (2009).

[17] Anderson, J. and B. Anderson, “The myth of persistence of vision revisited”, Journal of Film and Video **45**, 1, 3–12 (1993).

[18] Apple Inc., “Example Playlist Files for use with HTTP Live Streaming—Technical Note TN2288”, Tech. rep., https://developer.apple.com/library/content/technotes/tn2288/ (2012).

[19] Atitallah, A. B., P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi and P. Marchegay, “Hardware platform design for real-time video applications”, in “Proc. of the 16th International Conference on Microelectronics”, pp. 722–725 (2004).

[20] Aurzada, F., M. Lévesque, M. Maier and M. Reisslein, “FiWi access networks based on next-generation PON and gigabit-class WLAN technologies: A capacity and delay analysis”, IEEE/ACM Transactions on Networking **22**, 4, 1176–1189 (2014).

[21] axemclion, “IndexedDBShim, accessed February 19th 2014”, `https://github.com/axemclion/IndexedDBShim` (2012).

[22] Bai, Y.-W., Z.-L. Xie and Z.-H. Li, “Design and implementation of a home embedded surveillance system with ultra-low alert power”, IEEE Trans. Consumer Electronics **57**, 1, 153–159 (2011).

[23] Bayer, B. E., “Color imaging array, US Patent 3971065”, (1975).

[24] Bidelman, E., *Using the HTML5 Filesystem API* (O’Reilly Media, Inc., 2011).

[25] Blenk, A., A. Basta, M. Reisslein and W. Kellerer, “Survey on network virtualization hypervisors for software defined networking”, IEEE Communications Surveys & Tutorials **18**, 1, 655–685 (2016).

[26] Blenk, A., A. Basta, J. Zerwas, M. Reisslein and W. Kellerer, “Control plane latency with SDN network hypervisors: The cost of virtualization”, IEEE Transactions on Network and Service Management **13**, 3, 366–380 (2016).

[27] Bouaziz, S., M. Fan, A. Lambert, T. Maurin and R. Reynaud, “PICAR: experimental platform for road tracking applications”, in “Proc. of the IEEE Intelligent Vehicles Symposium”, pp. 495–499 (2003).

[28] Bourdon, A., A. Noureddine, R. Rouvoy and L. Seinturier, “PowerAPI: A software library to monitor the energy consumed at the process-level”, `http://abourdon.github.com/powerapi-akka` (2012).

[29] Bramberger, M., A. Doblander, A. Maier, B. Rinner and H. Schwabach, “Distributed embedded smart cameras for surveillance applications”, IEEE Computer **39**, 2, 68–75 (2006).

[30] Bramberger, M., R. Pflugfelder, B. Rinner, H. Schwabach and B. Strobl, "Intelligent traffic video sensor: Architecture and applications", in "Proceedings of the Workshop on Telecommunications and Mobile Computing", (2003).

[31] Brown, P., "Zigbee and low-cost wireless applications", `www.freescale.com/zigbee` (2008).

[32] Brun, P., G. Hauske and T. Stockhammer, "Subjective assessment of H.264-AVC video for low-bitrate multimedia messaging services", in "Proc. of International Conference on Image Processing (ICIP)", vol. 2, pp. 1145–1148 (2004).

[33] Buchmann, I., *Batteries in a Portable World* (Cadex Electronics Inc, 2001), 2 edn.

[34] Buschmann, C., D. Pfisterer, S. Fischer, P. F. Sándor and A. Kröller, "Spyglass: a wireless sensor network visualizer", ACM SIGBED Review **2**, 1, 1–6 (2005).

[35] Campbell, J., P. B. Gibbons, S. Nath, P. Pillai, S. Seshan and R. Sukthankar, "IrisNet: an internet-scale architecture for multimedia sensors", in "Proceedings of ACM Int. Conference on Multimedia", pp. 81–88 (2005).

[36] Cao, Z.-Y., Z.-Z. Ji and M.-Z. Hu, "An image sensor node for wireless sensor networks", in "Proc. of Int. Conference on Information Technology: Coding and Computing", pp. 740–745 (2005).

[37] Capo-Chichi, E. P. and J. M. Friedt, "Design of embedded sensor platform for multimedia application", in "Proc. of Int. Conference on Distributed Framework and Applications (DFmA)", pp. 146–150 (2008).

[38] Cerpa, A., J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao, "Habitat monitoring: application driver for wireless communications technology", ACM SIGCOMM Computer Communication Review **31**, 2, 20–41 (2001).

[39] Chandrakasan, A., F. Fox, W. Bowhill and W. Bowhill, *Design of high-performance microprocessor circuits* (IEEE Press, 2001), URL `http://books.google.com/books?id=EZoeAQAAIAAJ`.

[40] Chen, P., P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Y. Yang, C. Yeo, L.-C. Chang, J. D. Tygar and S. S. Sastry, "CITRIC: A low-bandwidth wireless camera network platform", in "Proc. of ACM/IEEE Int. Conference on Distributed Smart Cameras (ICDSC)", pp. 1–10 (2008).

[41] Chen, P., S. Oh, M. Manzo, B. Sinopoli, C. Sharp, K. Whitehouse, G. Tolle, J. Jeong, P. Dutta, J. Hui, S. Schaffert, S. Kim, J. Taneja, B. Zhu, T. Roosta, M. Howard, D. Culler and S. Sastry, "Experiments in instrumenting wireless sensor networks for real-time surveillance", in "Proc. of IEEE Int. Conference on Robotics and Automation", (2006).

[42] Chen, P.-Y. and M. Reisslein, “A simple analytical throughput-delay model for clustered FiWi networks”, Photonic Network Communications **29**, 1, 78–95 (2015).

[43] Cheng, W., *Embedded SYSTEM DESIGN AND POWER-RATE-DISTORTION OPTIMIZATION FOR VIDEO ENCODING UNDER ENERGY CONSTRAINTS*, Master’s thesis, Univ. of Missouri (2007).

[44] Chi, Y. M., R. Etienne-Cummings, G. Cauwenberghs, P. Carpenter and K. Colling, “Video sensor node for low-power ad-hoc wireless networks”, in “Proc. of Annual Conf. on Information Sciences and Systems (CISS)”, pp. 244–247 (2007).

[45] Cho, Y., S. O. Lim and H. S. Yang, “Collaborative occupancy reasoning in visual sensor network for scalable smart video surveillance”, IEEE Trans. Consumer Electronics **56**, 3, 1997–2003 (2010).

[46] Collins, M. J., “Indexed DB”, in “Pro HTML5 with Visual Studio 2012”, pp. 255–279 (Apress, 2012).

[47] Concolato, C. and J. Le Feuvre, “Live HTTP streaming of video and subtitles within a browser”, in “Proc. ACM Multimedia Systems Conference”, pp. 146–150 (2013).

[48] Corujo, D., M. Lebre, D. Gomes and R. L. Aguiar, “Furthering media independence mechanisms for Future Internet enablement”, in “Proc. IEEE ICC”, pp. 6845–6849 (2012).

[49] Culurciello, E. and A. G. Andree, “ALOHA CMOS imager”, in “Proc. of IEEE Int. Symposium on Circuits and Systems (ISCAS)”, pp. IV–956–9 (2004).

[50] Culurciello, E., J. H. Park and A. Savvides, “Address-event video streaming over wireless sensor networks”, in “Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on”, pp. 849 –852 (2007).

[51] Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana, “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”, IEEE Internet Computing **6**, 2, 86–93 (2002).

[52] D. Real de Oliveira, F., P. Chalimbaud, F. Berry, J. Serot and F. Marmoiton, “Embedded early vision systems: Implementation proposal and hardware architecture and hardware architecture”, in “Proc. of COGnitive systems with Interactive Sensors”, (2006).

[53] DALSA Corp., “Image sensor architectures for digital cinematography”, Tech. rep., http://www.teledynedalsa.com, white paper 03-70-00218-01 (2003).

[54] DASH-IF, “Dash industry forum’s reference client 1.0.0”, URL `http://dashif.org/reference/players/javascript/1.0.0/index.html` (2013).

[55] DASH Industry Forum, ., “Dash industry forum’s dash.js”, URL `https://github.com/Dash-Industry-Forum/dash.js` (2017).

[56] De Cicco, L., V. Caldaralo, V. Palmisano and S. Mascolo, “ELASTIC: a client-side controller for dynamic adaptive streaming over HTTP (DASH)”, in “Proc. IEEE Packet Video Workshop”, pp. 1–8 (2013).

[57] Decker, C., A. Krohn, M. Beigl and T. Zimmer, “The particle computer system”, in “Proc. of Fourth Int. Symposium on Information Processing in Sensor Networks”, pp. 443–448 (2005).

[58] der Auwera, G. V., P. David and M. Reisslein, “Traffic and quality characterization of single-layer video streams encoded with H.264/MPEG-4 advanced video coding standard and scalable video coding extension”, IEEE Transactions on Broadcasting **54**, 3, 698–718 (2008).

[59] Detti, A., M. Pomposini, N. Blefari-Melazzi, S. Salsano and A. Bragagnini, “Offloading cellular networks with Information-Centric Networking: The case of video streaming.”, in “Proc. WOWMOM”, pp. 1–3 (2012).

[60] Detti, A., B. Ricci and N. Blefari-Melazzi, “Peer-to-peer live adaptive video streaming for information centric cellular networks”, in “Proc. IEEE PIMRC”, pp. 3583–3588 (2013).

[61] Dionne, J. and M. Durrant, “uclinux—embedded linux microcontroller project”, (2010).

[62] Dong, W., C. Chen, X. Liu and J. Bu, “Providing os support for wireless sensor networks: Challenges and approaches”, Communications Surveys Tutorials, IEEE **12**, 4, 519 –530 (2010).

[63] Downes, I., L. B. Rad and H. Aghajan, “Development of a mote for wireless image sensor networks”, in “Proc. of COGnitive systems with Interactive Sensors (COGIS)”, (2006).

[64] Dutta, P., J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse and D. Culler, “Trio: enabling sustainable and scalable outdoor wireless sensor network deployments”, in “Proc. of Int. Conference on Information Processing in Sensor Networks (IPSN)”, pp. 407–415 (2006).

[65] EDM, S., “Guidelines for designing edm modules and carrier boards, edm specifications and design guide ver. 1.00”, URL `http://www.edm-standard.org/images/downloads/edm-specifications-100.pdf` (2014).

[66] Edwards, L., “Choosing a microcontroller and other design decisions”, in “Embedded Hardware: Know It All”, (2008).

[67] Emigh, J., “New Flash player rises in the Web-video market”, IEEE Computer **39**, 2, 14–16 (2006).

[68] Engineering Services, Avnet, "WandCam (AES-WCAM-ADPT-G)—getting started guide", URL `http://www.em.avnet.com/wandcam` (2016).

[69] Evans-Pughe, C., "Bzzzz zzz [zigbee wireless standard]", IEE Review **49**, 3, 28 – 31 (2003).

[70] Fan, T., L. Xu, X. Zhang and H. Wang, "Research and design of a node of wireless multimedia sensor network", in "Proc. of Int. Conference on Wireless Communications, Networking and Mobile Computing (WiCom)", pp. 1–5 (2009).

[71] Feng, W.-C., E. Kaiser, W. C. Feng and M. LeBaillif, "Panoptes: scalable low-power video sensor networking technologies", ACM Transactions on Multimedia Computing, Communications, and Applications **1**, 2, 151–167 (2005).

[72] Fife, K., A. E. Gamal and H.-S. P. Wong, "A multi-aperture image sensor with 0.7 μm pixels in 0.11 μm CMOS technology", IEEE Journal of Solid-State Circuits **43**, 12, 2990–3005 (2008).

[73] Free Electrons, "Power management", `http://free-electrons.com/doc/power-management.pdf` (2011).

[74] Friedman, J., D. Lee, I. Tsigkogiannis, P. Aghera, A. Dixit, S. Wong, D. Levine, D. Chao, A. Kansal, W. Kaiser and M. Srivastava, "RAGOBOT: A new platform for wireless mobile sensor networks", in "Proc. of IEEE Int. Conference on Distributed Computing in Sensor Systems", (2005).

[75] Gamal, A. E., D. X. D. Yang and B. A. Fowler, "Pixel-level processing: why, what, and how?", in "SPIE Sensors, Cameras, and Applications for Digital Photography", edited by N. Sampat and T. Yeh, vol. 3650, pp. 2–13 (1999).

[76] Gang, G., L. Zeyong and J. Jun, "Internet of things security analysis", in "Proc. Internet Technology and Applications (iTAP)", pp. 1–4 (2011).

[77] Garcia, S., J. Cabrera and N. Garcia, "Quality-optimization algorithm based on stochastic dynamic programming for MPEG DASH video streaming", in "Proc. IEEE Int. Conference on Consumer Electronics (ICCE)", pp. 574–575 (2014).

[78] Gayan, "Powerstat: Power Consumption Calculator for Ubuntu Linux", `http://www.hecticgeek.com/2012/02/powerstat-power-calculator-ubuntu-linux` (2012).

[79] Go, Y., O. C. Kwon and H. Song, "An energy-efficient HTTP adaptive video streaming with networking cost constraint over heterogeneous wireless networks", IEEE Transactions on Multimedia **17**, 9, 1646–1657 (2015).

[80] GStreamer, P., "Gstreamer, open source multimedia framework.", URL `https://gstreamer.freedesktop.org` (2016).

[81] Gundavaram, S., *CGI programming on the World Wide Web* (O'Reilly, 1996).

[82] Gupta, R., A. Pulipaka, P. Seeling, L. J. Karam and M. Reisslein, "H.264 coarse grain scalable (CGS) and medium grain scalable (MGS) encoded video: A trace based traffic and quality evaluation", IEEE Transactions on Broadcasting **58**, 3, 428–439 (2012).

[83] Gursu, H. M., M. Vilgelm, W. Kellerer and M. Reisslein, "Hybrid collision avoidance-tree resolution for M2M random access", IEEE Transactions on Aerospace and Electronic Systems, in print (2017).

[84] Haas, C., S. Munz, J. Wilke and A. Hergenroder, "Evaluating energy-efficiency of hardware-based security mechanisms", in "Proc. IEEE Int. Conf. on Pervasive Computing and Commun. Workshops (PERCOM Workshops)", pp. 560–565 (2013).

[85] Haas, C., J. Wilke and V. Stöhr, "Realistic simulation of energy consumption in wireless sensor networks", in "Wireless Sensor Networks", pp. 82–97 (Springer, 2012).

[86] Hachicha, K., S. Bouton, D. Faura, O. Romain, A. Pinna and P. Garda, "Integration of MPEG4 codec on the M.E.R.I.T.E platform for wireless sensor networks", in "Proc. of IEEE Int. Conference on Industrial Technology", pp. 1215–1220 (2004).

[87] Harris, R., "Review: Home spy camera hidden in clock", Physorg.com (2007).

[88] He, Z. and D. Wu, "Resource allocation and performance analysis of wireless video sensors", IEEE Transactions on Circuits and Systems for Video Technology **16**, 5, 590–599 (2006).

[89] Hengstler, S. and H. Aghajan, "WiSNAP: A wireless image sensor network application platform", in "Proc. of Int. IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities", (2006).

[90] Hengstler, S., D. Prashanth, S. Fong and H. Aghajan, "MeshEye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance", in "Proceedings of Int. Conference on Information Processing in Sensor Networks", pp. 360–369 (2007).

[91] Hergenroder, A. and J. Furthmuller, "On energy measurement methods in wireless networks", in "Proc. IEEE Int. Conf. on Communications (ICC)", pp. 6268–6272 (2012).

[92] Hergenroeder, A., J. Wilke and D. Meier, "Distributed energy measurements in WSN testbeds with a sensor node management device (SNMD)", in "Proc. Int. Conf. on Architecture of Computing Systems (ARCS)", pp. 1–7 (2010).

[93] Hill, J., M. Horton, R. Kling and L. Krishnamurthy, "The platforms enabling wireless sensor networks", Communications of the ACM **47**, 6, 41–46 (2004).

[94] Hill, J. L., *System Architecture for Wireless Sensor Networks*, Ph.D. thesis, University of California at Berkeley (2003).

[95] hls.js, ., "hls.js is a javascript library which implements an http live streaming client. it relies on html5 video and mediasource extensions for playback.", URL `https://github.com/video-dev/hls.js` (2017).

[96] Huang, H.-C., J.-W. Ding and Y.-M. Huang, "An implementation of battery-aware wireless sensor network using ZigBee for multimedia service", in "Proc. of Int. Conference on Consumer Electronics", pp. 369–370 (2006).

[97] Huang, J.-T., J.-H. Shiao and J.-M. Wu, "A miniaturized hilbert inverted-f antenna for wireless sensor network applications", Antennas and Propagation, IEEE Transactions on **58**, 9, 3100 –3103 (2010).

[98] Huang, Y. and G. Li, "Descriptive models for Internet of Things", in "Proc. Intelligent Control and Information Processing (ICICIP)", pp. 483–486 (2010).

[99] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282 (Proposed Standard), URL `http://www.ietf.org/rfc/rfc6282.txt` (2011).

[100] Hwang, K.-I., J. In, N. Park and D.-S. Eom, "A design and implementation of wireless sensor gateway for efficient querying and managing through World Wide Web", IEEE Trans. Consumer Electronics **49**, 4, 1090–1097 (2003).

[101] Hwang, M.-C., L. T. Ha, N.-H. Kim, C.-S. Park and S.-J. Ko, "Person identification system for future digital TV with intelligence", IEEE Transactions on Consumer Electronics **53**, 1, 218–226 (2007).

[102] Hllmarker, O., "Mpeg4 file parser.", URL `http://mp4parser.com/` (2017).

[103] IMX VPU API, "i.MX VPU application programming interface, linux reference manual, rev. 0, 03/2016", URL `https://freescale.jiveon.com/servlet/JiveServlet/download/810849-1-382106/i.MX_Linux_User's_Guide.pdf` (2016).

[104] Irondi, I., Q. Wang and C. Grecos, "Empirical evaluation of H.265/HEVC-based dynamic adaptive video streaming over HTTP (HEVC-DASH)", in "Proc. SPIE Photonics Europe", pp. 91390L–1–91390L–9 (2014).

[105] ISO BMFF, B. S. F., "Media source extensions byte stream format specification based on the iso base media file format - iso bmff byte stream format.", URL `https://w3c.github.io/media-source/isobmff-byte-stream-format.html` (2016).

[106] ISO/IEC 14496-12:2015(E), "ISO Information Technology – Coding of audio-visual objects – Part 12: ISO base media file format", Standard, International Organization for Standardization, ISO copyright office, Case postale 56, CH-1211 Geneva 20, Switzerland, URL `https://www.iso.org/standard/68960.html` (2015).

[107] Jiang, J., V. Sekar and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with Festive", IEEE/ACM Trans. Networking **22**, 1, 326–340 (2014).

[108] Jiang, X., P. Dutta, D. Culler and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale", in "Proc. Int. Symp. on Inform. Proc. in Sensor Netw. (IPSN)", pp. 186–195 (2007).

[109] Kandhalu, A., A. Rowe and R. Rajkumar, "DSPcam: A camera sensor system for surveillance networks", in "Proc. of ACM/IEEE Int. Conference on Distributed Smart Cameras (ICDSC)", pp. 1–7 (2009).

[110] Kandhalu, A., A. Rowe, R. Rajkumar, C. Huang and C.-C. Yeh, "Real-time video surveillance over IEEE 802.11 mesh networks", in "Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)", pp. 205–214 (2009).

[111] Karlsson, J., T. Wark, P. Valencia, M. Ung and P. Corke, "Demonstration of image compression in a low-bandwidth wireless camera network", in "Proceedings of Int. Conference on Information Processing in Sensor Networks (IPSN)", pp. 557–558 (2007).

[112] Khan, A. A., M. H. Rehmani and M. Reisslein, "Cognitive radio for smart grids: Survey of architectures, spectrum sensing mechanisms, and networking protocols", IEEE Communications Surveys & Tutorials **18**, 1, 860–898 (2016).

[113] Kidwai, N. R., E. Khan and M. Reisslein, "ZM-SPECK: A fast and memoryless image coder for multimedia sensor networks", IEEE Sensors Journal **16**, 8, 2575–2587 (2016).

[114] Kidwai, N. R., E. Khan and M. Reisslein, "ZM-SPECK: A fast and memoryless image coder for multimedia sensor networks", IEEE Sensors Journal **16**, 8, 2575–2587 (2016).

[115] Kleihorst, R., A. Abbo, B. Schueler and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing", in "Proc. of IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)", (2007).

[116] Kleihorst, R., B. Schueler, A. Danilin and M. Heijligers, "Smart camera mote with high performance vision system", in "Proc. of ACM Conference on Embedded Networked Sensor Systems: Workshop on Distributed Smart Cameras", (2006).

[117] Knuth, D., "Backus Normal Form vs. Backus Naur Form", Comm. ACM **7**, 12, 735–736 (1964).

[118] Kogut, G., M. Blackburn and H. R. Everett, "Using video sensor networks to command and control unmanned ground vehicles", in "Proc. of AUVSI Unmanned Systems in International Security", (2003).

[119] Kovacevic, J., G. Miljkovic, K. Lazic and M. Stankic, “Evaluation of adaptive streaming algorithms over HTTP”, in “Proc. IEEE Int. Conf. on Consumer Electronics-Berlin (ICCE-Berlin)”, pp. 1–4 (2013).

[120] Krause, M., M. V. Hartskamp and E. Aarts, “A quality metric for use with frame-rate based bandwidth adaptation algorithms”, in “Proc. of SPIE Human Vision and Electronic Imaging”, (2008).

[121] Krishnappa, D. K., D. Bhat and M. Zink, “DASHing YouTube: An analysis of using DASH in YouTube video service”, in “Proc. IEEE Conf. on Local Computer Networks (LCN)”, pp. 407–415 (2013).

[122] Kulkarni, P., D. Ganesan, P. Shenoy and Q. Lu, “SensEye: a multi-tier camera sensor network”, in “Proceedings of ACM International Conference on Multimedia”, pp. 229–238 (2005).

[123] Kun, A. J. and Z. Vamossy, “Traffic monitoring with computer vision”, in “Proc. International Symposium on Applied Machine Intelligence and Informatics (SAMI)”, pp. 131–134 (2009).

[124] Kuo, C. H., C. C. Chen, W. C. Wang, Y. C. Hung, E. C. Lin, K. M. Lee and Y. M. Lin, “Remote control based hybrid-structure robot design for home security applications”, in “Proc. of IEEE/RSJ Int. Conference on Intelligent Robots and Systems”, pp. 4484–4489 (2006).

[125] Kushalnagar, N., G. Montenegro and C. Schumacher, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals”, RFC 4919 (Informational), URL `http://www.ietf.org/rfc/rfc4919.txt` (2007).

[126] Kwan, H., C. Roy and D. Das, “Demonstration of a multimedia player supporting the MPEG-DASH protocol”, in “Proc. IEEE Visual Communications and Image Processing (VCIP)”, pp. 1–1 (2012).

[127] Ladis, E., I. Papaefstathiou, R. Marchesani, K. Tuinenbreijer, P. Langendorfer, T. Zahariadis, H. C. Leligou, L. Redondo, T. Riesgo, P. Kannegiesser, M. Berekovic and C. J. M. van Rijn, “Secure, mobile visual sensor networks architecture”, in “Proc. of IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON Workshops)”, pp. 1–3 (2009).

[128] Lederer, S., C. Mueller, B. Rainer, C. Timmerer and H. Hellwagner, “Adaptive streaming over content centric networks in mobile networks using multiple links”, in “Proc. IEEE ICC”, pp. 677–681 (2013).

[129] Lenk, J., *Simplified Design of Voltage/Frequency Converters*, EDN Series for Design Engineers (Elsevier Science, 1997), URL `http://books.google.com/books?id=a8avGaqz8AoC`.

[130] Li, Y., M. Reisslein and C. Chakrabarti, “Energy-efficient video transmission over a wireless link”, IEEE Transactions on Vehicular Technology **58**, 3, 1229–1244 (2009).

[131] Lifton, J., M. Broxton and J. A. Paradiso, "Experiences and directions in push-pin computing", in "Proceedings of Int. Symposium on Information Processing in Sensor Networks (IPSN)", pp. 57–62 (2005).

[132] Liu, H., T. Roeder, K. Walsh, R. Barr and E. G. Sirer, "Design and implementation of a single system image operating system for ad hoc networks", in "Proc. of Int. Conference on Mobile Systems, Applications and Services", pp. 149–162 (2005).

[133] Lu, M.-H. and T. Chen, "CMUseum: A location-aware wireless video streaming system", in "Proc. of IEEE Int. Conference on Multimedia and Expo", pp. 2129–2132 (2006).

[134] Lymberopoulos, D. and A. Savvides, "XYZ: a motion-enabled, power aware sensor node platform for distributed sensor network applications", in "Proceedings of Int. Symposium on Information Processing in Sensor Networks (IPSN)", pp. 449–454 (2005).

[135] Margi, C. B., V. Petkov, K. Obraczka and R. Manduchi, "Characterizing energy consumption in a visual sensor network testbed", in "Proc. Int. Conf. on Testbeds & Research Infrastr. for the DEvelopm. of NeTworks & COMmunities (TRIDENTCOM)", pp. 1–8 (2006).

[136] Martina, M., A. Molino, F. Quaglio and F. Vacca, "A power-scalable motion estimation co-processor for energy constrained applications", in "Proceedings of Int. Symposium on Signal Processing and Its Applications", pp. 603–604 (2003).

[137] McGarry, M. P., M. Reisslein, F. Aurzada and M. Scheutzow, "Shortest propagation delay (SPD) first scheduling for EPONs with heterogeneous propagation delays", IEEE Journal on Selected Areas in Communications **28**, 6, 849–862 (2010).

[138] McKay, C. and F. Masuda, "Empirical study of 802.11b wireless networks in the presence of bluetooth interference", Institute for Telecommunication Sciences **2003**, 1, 1–29 (2003).

[139] Mercian, A., E. I. Gurrola, F. Aurzada, M. P. McGarry and M. Reisslein, "Upstream polling protocols for flow control in PON/xDSL hybrid access networks", IEEE Transactions on Communications **64**, 7, 2971–2984 (2016).

[140] Mercian, A., M. P. McGarry and M. Reisslein, "Offline and online multi-thread polling in long-reach PONs: A critical evaluation", IEEE/OSA Journal of Lightwave Technology **31**, 12, 2018–2028 (2013).

[141] Milenkovic, A., M. Milenkovic, E. Jovanov, D. Hite and D. Raskovic, "An environment for runtime power monitoring of wireless sensor network platforms", in "Proc. Southeastern Symp. on System Theory, (SSST)", pp. 406–410 (2005).

[142] Misra, S., M. Reisslein and G. Xue, "A survey of multimedia streaming in wireless sensor networks", IEEE Communications Surveys and Tutorials **10**, 4, 18–39 (2008).

[143] Montenegro, G., N. Kushalnagar, J. Hui and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944 (Proposed Standard), URL `http://www.ietf.org/rfc/rfc4944.txt`, updated by RFC 6282 (2007).

[144] Mooshimeter, M., "DMM-BLE-2x01A Mooshimeter Technical Specifications", URL `https://moosh.im/mooshimeter/specs/` (2016).

[145] Moritz, G., F. Golatowski and D. Timmermann, "A lightweight SOAP over CoAP transport binding for resource constraint networks", in "Proc. Mobile Adhoc and Sensor Systems (MASS)", pp. 861–866 (2011).

[146] Müller, C. and C. Timmerer, "A VLC media player plugin enabling dynamic adaptive streaming over HTTP", in "Proc. ACM Multimedia", pp. 723–726 (2011).

[147] Nachman, L., J. Huang, J. Shahabdeen, R. Adler and R. Kling, "IMOTE2: Serious computation at the edge", in "Proc. of IEEE Int. Wireless Communications and Mobile Computing Conference (IWCMC)", pp. 1118–1123 (2008).

[148] Nichols, S. and K. A. Hua, "Design and implementation of intelligent mesh nodes for wireless video stream sharing", in "Proc. Int. Conf. on Internet Multimedia Computing and Service (ICIMCS)", pp. 13–16 (2011).

[149] Nicolae, B., G. Antoniu, L. Bouge, D. Moise and A. Carpen-Amarie, "BlobSeer: Next-generation data management for large scale infrastructures", J. Parallel Distr. Comp. **71**, 2, 169–184 (2011).

[150] Noureddine, A., A. Bourdon, R. Rouvoy and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT", in "Proc. Int. Workshop on Green and Sustainable Software", pp. 21–27 (Zurich, Switzerland, 2012).

[151] Noureddine, A., A. Bourdon, R. Rouvoy and L. Seinturier, "Runtime monitoring of software energy hotspots", in "Proc IEEE/ACM Int. Conf. on Autom. Softw. Eng. (ASE)", pp. 160–169 (2012).

[152] Panda, D. P., C. A. Anderson and P. G. Michalopoulos, "A new integrated video sensor technology for traffic management applications", in "Proc. of Eighth Annual Meeting of ITS America", (1998).

[153] Pantazis, N. and D. Vergados, "A survey on power control issues in wireless sensor networks", IEEE Communications Surveys & Tutorials **9**, 4, 86–107 (2007).

[154] Parhi, K., *Vlsi Digital Signal Processing Systems: Design And Implementation* (Wiley India Pvt. Ltd., 2007), URL `http://books.google.com/books?id=APFRHFkMqG8C`.

[155] Park, C. and P. H. Chou, “eCAM: ultra compact, high data-rate wireless sensor node with a miniature camera”, in “Proceedings of the 4th International Conference on Embedded Networked Sensor Systems”, pp. 359–360 (2006).

[156] Park, C. and P. H. Chou, “Eco: Ultra-wearable and expandable wireless sensor platform”, in “Proc. of International Workshop on Wearable and Implantable Body Sensor Networks (BSN)”, pp. 162–165 (2006).

[157] Paulson, L. D., “Building rich web applications with Ajax”, IEEE Computer **38**, 10, 14–17 (2005).

[158] Permadi, F., “Introduction to webm file structure.”, URL `https://permadi.com/2010/06/webm-file-structure/` (2010).

[159] Polastre, J., J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker and I. Stoica, “A unifying link abstraction for wireless sensor networks”, in “Proc. of 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)”, (2005).

[160] Pon, R., M. A. Batalin, J. Gordon, A. Kansal, D. Liu, M. Rahimi, L. Shirachi, Y. Yu, M. Hansen, W. J. Kaiser, M. Srivastava, G. Sukhatme and D. Estrin, “Networked infomechanical systems: a mobile embedded networked sensor platform”, in “Proc. of Fourth International Symposium on Information Processing in Sensor Networks”, pp. 376–381 (2005).

[161] Project, V., “Vlc media player.”, URL `http://www.videolan.org/` (2016).

[162] Pudlewski, S. and T. Melodia, “A distortion-minimizing rate controller for wireless multimedia sensor networks”, Computer Communications **33**, 12, 1380–1390 (2010).

[163] Pulipaka, A., P. Seeling, M. Reisslein and L. J. Karam, “Traffic and statistical multiplexing characterization of 3-D video representation formats”, IEEE Transactions on Broadcasting **59**, 2, 382–389 (2013).

[164] Quacchio, E., G. Bruno and M. Grangetto, “An HTML5 player for a gstreamer based MPEG DASH client”, in “Proc. IEEE Visual Communications and Image Processing (VCIP)”, pp. 1–1 (2012).

[165] Raghunathan, A., N. Jha and S. Dey, *High-level power analysis and optimization* (Kluwer Academic Publishers, 1998), URL `http://books.google.com/books?id=R_BCk7499B8C`.

[166] Rahimi, M., R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin and M. Srivastava, “Cyclops: in situ image sensing and interpretation in wireless sensor networks”, in “Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems”, pp. 192–204 (2005).

[167] Rein, S. and M. Reisslein, “Low-memory wavelet transforms for wireless sensor networks: A tutorial”, IEEE Communications Surveys & Tutorials **13**, 2, 291–307 (2011).

[168] Rein, S. and M. Reisslein, "Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks", Ad Hoc Networks **9**, 4, 482–496 (2011).

[169] Rein, S. and M. Reisslein, "Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks", Ad Hoc Networks **9**, 4, 482–496 (2011).

[170] Rein, S. and M. Reisslein, "Scalable line-based wavelet image coding in wireless sensor networks", Journal of Visual Communication and Image Representation **40**, 418–431 (2016).

[171] Reis, C., A. Barth and C. Pizano, "Browser security: lessons from Google Chrome", ACM Queue **7**, 5, 1–8 (2009).

[172] Reisslein, M., J. Lassetter, S. Ratnam, O. Lotfallah, F. Fitzek and S. Panchanathan, "Traffic and quality characterization of scalable encoded video: A large-scale trace-based study, part 1: Overview and definitions", Arizona State Univ., Telecommunications Research Center, Tech. Rep (2002).

[173] Rowe, A., A. G. Goode, D. Goel and I. Nourbakhsh, "CMUcam3: An open programmable embedded vision sensor", Tech. rep., Robotics Institute, Carnegie Mellon University (2007).

[174] Salvador, O. and D. Angolini, *Embedded Linux Development with Yocto Project* (Packt Publishing Ltd, 2014).

[175] Sambhoos, P., A. Hasan, R. Han, T. Lookabaugh and J. Mulligan., "Weeblevideo: Wide angle field-of-view video sensor networks", in "Proc. of 4th ACM Conference on Embedded Networked Sensor Systems: Workshop on Distributed Smart Cameras", (2006).

[176] Schiller, J., A. Liers and H. Ritter, "ScatterWeb: A wireless sensornet platform for research and teaching", Computer Communications **28**, 13, 1545–1551 (2005).

[177] Schroeder, D., A. Ilangovan, M. Reisslein and E. Steinbach, "Efficient multirate video encoding for HEVC-based adaptive HTTP streaming", IEEE Transactions on Circuits and Systems for Video Technology, in print (2017).

[178] Schwoebel, L., "Browser-and codec-agnostic html5 video-streaming in the wireless video sensor network platform", (2014).

[179] Seema, A., "Blueprint for a low-cost piggyback wireless video sensor node/platform architecture, ms thesis, arizona state university, electrical engineering", (2009).

[180] Seema, A. and M. Reisslein, "Towards efficient wireless video sensor networks: A survey of existing node architectures and proposal for a Flexi-WVSNP design", IEEE Communications Surveys & Tutorials **13**, 3, 462–486 (2011).

[181] Seema, A. and M. Reisslein, “Systems and methods for name-based segmented media acquisition and distribution framework on a network”, US Patent App. 14/961,706 (2015).

[182] Seema, A., L. Schwoebel, T. Shah, J. Morgan and M. Reisslein, “WVSNP-DASH: Name-based segmented video streaming”, IEEE Trans. Broadcasting **61**, 3, 346–355 (2015).

[183] Shaka Player, ., “Shaka player is a javascript library for adaptive video streaming.”, URL `https://github.com/google/shaka-player` (2017).

[184] Sharma, A., K. Yoon, D. Vanhorn, M. Dube, V. McKenna and M. S. Bauer, “ScoutNode: A multimodal sensor node for wide area sensor networks”, in “Proc. of Int. Conference on Computer Communications and Networks (ICCCN)”, pp. 1–6 (2009).

[185] Shelby, Z. and C. Bormann, *6LoWPAN: the wireless embedded Internet* (Wiley, 2010).

[186] Shin, S. Y., H. S. Park and W. H. Kwon, “Mutual interference analysis of IEEE 802.15.4 and IEEE 802.11b”, Computer Networks **51**, 12, 3338–3353 (2007).

[187] Singh, H., K. Agarwal, D. Sylvester and K. Nowka, “Enhanced leakage reduction techniques using intermediate strength power gating”, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **15**, 11, 1215 –1224 (2007).

[188] Sitbon, P., N. Bulusu and W.-C. Feng, “Cascades: An extensible heterogeneous sensor networking framework”, in “Proceedings of the 4th International Conference on Embedded Networked Sensor Systems”, pp. 339–340 (2006).

[189] Sitbon, P., W.-C. Feng, N. Bulusu and T. Dang, “SenseTK: a multimodal, multimedia sensor networking toolkit”, in “Multimedia Computing and Networking”, edited by R. Zimmermann and C. Griwodz (2007).

[190] Slotfeldt, T., “Simplify graphical user interface and video integration for i.MX 6 series processors”, URL `https://community.nxp.com/docs/DOC-104267` (2016).

[191] Smith, J., A. Colwell and M. Wolenetz, “WebM Byte Stream Format”, W3C note, W3C, https://www.w3.org/TR/2016/NOTE-mse-byte-stream-format-webm-20161004/ (2016).

[192] Smith, J., M. Watson, M. Wolenetz, A. Bateman and A. Colwell, “MPEG-2 TS byte stream format”, W3C note, W3C, https://www.w3.org/TR/2016/NOTE-mse-byte-stream-format-mp2t-20161004/ (2016).

[193] Song, G., Y. Zhou, W. Zhang and A. Song, “A multi-interface gateway architecture for home automation networks”, IEEE Trans. Consumer Electronics **54**, 3, 1110–1113 (2008).

[194] Srinivasan, K., P. Dutta, A. Tavakoli and P. Levis, "An empirical study of low-power wireless", ACM Trans. Sen. Netw. **6**, 2, 1–49 (2010).

[195] Stockhammer, T., "Dynamic adaptive streaming over HTTP–: standards and design principles", in "Proc. ACM Conf. on Multimedia Systems", pp. 133–144 (2011).

[196] Stockhammer, T., "Mpeg's dynamic adaptive streaming over http (dash) - enabling formats for video streaming over the open internet", in "Proc. of Webinar at EBU", (Qualcomm Incorporated, 2011).

[197] Streif, R. J., *Embedded Linux Systems with the Yocto Project* (Prentice Hall, 2016).

[198] Sudevalayam, S. and P. Kulkarni, "Energy harvesting sensor nodes: Survey and implications", Communications Surveys Tutorials, IEEE **13**, 3, 443 –461 (2011).

[199] Sward, R. E. and J. Boleng, "Service-oriented architecture (SOA) concepts and implementations", ACM Ada Lett. **32**, 11–12 (2012).

[200] Tabar, A. M., A. Keshavarz and H. Aghajan, "Smart home care network using sensor fusion and distributed vision-based reasoning", in "Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks", pp. 145–154 (2006).

[201] Tan, L. and N. Wang, "Future internet: The Internet of Things", in "Proc. Advanced Computer Theory and Engineering (ICACTE)", pp. V5–376–V5–380 (2010).

[202] Tausif, M., N. R. Kidwai, E. Khan and M. Reisslein, "FrWF-Based LMBTC: Memory-efficient image coding for visual sensors", IEEE Sensors Journal **15**, 11, 6218–6228 (2015).

[203] Tausif, M., N. R. Kidwai, E. Khan and M. Reisslein, "FrWF-based LMBTC: Memory-efficient image coding for visual sensors", IEEE Sensors Journal **15**, 11, 6218–6228 (2015).

[204] Teixeira, T., A. G. Andreou and E. Culurciello, "Event-based imaging with active illumination in sensor networks", in "Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)", pp. 644–647 (2005).

[205] Teixeira, T., E. Culurciello, J. H. Park, D. Lymberopoulos, A. Barton-Sweeney and A. Savvides, "Address-event imagers for sensor networks: evaluation and modeling", in "Proc. of International Conference on Information Processing in Sensor Networks", pp. 458–466 (2006).

[206] Thang, T. C., Q.-D. Ho, J.-W. Kang and A. Pham, "Adaptive streaming of audiovisual content using MPEG DASH", IEEE Trans. Consumer Electronics **58**, 1, 78–85 (2012).

[207] The DASH Industry Forum, ., "The dash industry forum (dah-if).", URL `http://dashif.org/` (2017).

[208] Thommes, D., A. Gerlicher, Q. Wang and C. Grecos, "RemoteUI: A high-performance remote user interface system for mobile consumer electronic devices", IEEE Trans. Consumer Electronics **58**, 3, 1094–1102 (2012).

[209] Thonet, G., P. Allard-Jacquin and P. Colle, "Zigbee–wifi coexistence - white paper and test report", Tech. rep., Schneider Electric (2008).

[210] Thyagaturu, A. S., Y. Dashti and M. Reisslein, "SDN-based smart gateways (Sm-GWs) for multi-operator small cell network management", IEEE Transactions on Network and Service Management **13**, 4, 740–753 (2016).

[211] Thyagaturu, A. S., A. Mercian, M. P. McGarry, M. Reisslein and W. Kellerer, "Software defined optical networks (SDONs): A comprehensive survey", IEEE Communications Surveys & Tutorials **18**, 4, 2738–2786 (2016).

[212] Titzer, B., D. Lee and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing", in "Proc. Int. Symp. on Information Proc. in Sensor Networks (IPSN)", pp. 477–482 (2005).

[213] Tyagi, R. R., F. Aurzada, K.-D. Lee, S. G. Kim and M. Reisslein, "Impact of retransmission limit on preamble contention in LTE-advanced network", IEEE Systems Journal **9**, 3, 752–765 (2015).

[214] Tyagi, R. R., F. Aurzada, K.-D. Lee and M. Reisslein, "Connection establishment in LTE-A networks: Justification of Poisson process modeling", IEEE Systems Journal, in print (2017).

[215] Tzoc, E. and J. Millard, "For video streaming/delivery: Is HTML5 the real fix?", Code4Lib Journal , 22 (2013).

[216] Upadhyaya, B., Y. Zou, H. Xiao, J. Ng and A. Lau, "Migration of SOAP-based services to RESTful services", in "Proc. Web Systems Evolution (WSE)", pp. 105–114 (2011).

[217] Van der Auwera, G., P. T. David, M. Reisslein and L. J. Karam, "Traffic and quality characterization of the H.264/AVC scalable video coding extension", Advances in Multimedia **2008**, 164027, 1–27 (2008).

[218] Vasseur, J. and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet* (Elsevier Science, 2010).

[219] Vaughan-Nichols, S. J., "Will HTML 5 restandardize the web?", IEEE Computer **43**, 4, 13–15 (2010).

[220] Vieira, M. A. M., C. N. Coelho Jr., D. C. da Silva Jr. and J. M. da Mata, "Survey on wireless sensor network devices", in "Proc. of 9th IEEE International Conference on Emerging Technologies and Factory Automation", pp. 537–544 (Lisbon, Portugal, 2003).

[221] Virone, G., A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin and J. A. Stankovic, “An assisted living oriented information system based on a residential wireless sensor network”, in “Proceedings of the 1st Distributed Diagnosis and Home Healthcare (D2H2) Conference”, pp. 95–100 (2006).

[222] Wang, C. and M. Zink, “On the feasibility of DASH streaming in the cloud”, in “Proc. of ACM Network and Operating System Support on Digital Audio and Video Workshop”, p. 49 (2014).

[223] Wang, F. and J. Liu, “Networked wireless sensor data collection: Issues, challenges, and approaches”, IEEE Communications Surveys Tutorials **13**, 4, 673–687 (2011).

[224] Wark, T., P. Corke, J. Karlsson, P. Sikka and P. Valencia, “Real-time image streaming over a low-bandwidth wireless camera network”, in “Proc. of International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP)”, pp. 113–118 (2007).

[225] Warneke, B. A., M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser and K. S. J. Pister, “An autonomous 16mm^3 solar-powered node for distributed wireless sensor networks”, in “Proc. of IEEE International Conference on Sensors”, pp. 1510–1515 (2002).

[226] Watson, M., A. Colwell, A. Bateman, J. Smith and M. Wolenetz, “ISO BMFF Byte Stream Format”, W3C note, W3C, https://www.w3.org/TR/2016/NOTE-mse-byte-stream-format-isobmff-20161004/ (2016).

[227] Watson, M., A. Colwell, A. Bateman, J. Smith and M. Wolenetz, “Media source extensions^{TM}”, W3C recommendation, W3C, https://www.w3.org/TR/2016/REC-media-source-20161117/ (2016).

[228] WebM Container, ., “Webm container guidelines.”, URL `https://www.webmproject.org/docs/container/` (2016).

[229] Welsh, M., “Wireless communications and sensor networks”, (2005).

[230] Wien, M., H. Schwarz and T. Oelbaum, “Performance analysis of SVC”, IEEE Trans. Circuits and Systems for Video Technology **17**, 9, 1194–1203 (2007).

[231] Wolf, W., B. Ozer and T. Lv, “Smart cameras as embedded systems”, IEEE Computer **35**, 9, 48–53 (2002).

[232] World Wide Web Consortium, ., “The world wide web consortium (w3c).”, URL `https://www.w3.org` (2017).

[233] Wu, M., T.-J. Lu, F.-Y. Ling, J. Sun and H.-Y. Du, “Research on the architecture of Internet of Things”, in “Proc. Advanced Computer Theory and Engineering (ICACTE)”, vol. 5, pp. V5–484–V5–487 (2010).

[234] Yan, L. and G. Yang, "A Web video service based on Flash video technique", in "Proc. IEEE Int. Conf. on Internet Technology and Applications (iTAP)", pp. 1–4 (2011).

[235] Yan, M., "Using castlabs to deliver widevine licenses to azure media services", URL `https://azure.microsoft.com/en-us/documentation/articles/media-services-castlabs-integration` (2016).

[236] Yeap, G., *Practical low power digital VLSI design* (Kluwer Academic Publishers, 1998), URL `http://books.google.com/books?id=9ZEe15lpuk4C`.

[237] Yick, J., B. Mukherjee and D. Ghosal, "Wireless sensor network survey", Computer Networks **52**, 12, 2292–2330 (2008).

[238] Yihan, L., S. S. Panwar, M. Shiwen, S. Burugupalli and H. L. Jong, "A mobile ad hoc bio-sensor network", in "Proc. of IEEE International Conference on Communications (ICC)", pp. 1241–1245 (2005).

[239] Yin, H., C. Lin, B. Sebastien and X. Chu, "A novel secure wireless video surveillance system based on Intel IXP425 network processor", in "Proceedings of the 1st ACM Workshop on Wireless Multimedia Networking and Performance Modeling (WMuNeP)", pp. 62–69 (2005).

[240] Yocto, P., "The yocto project - an open source collaboration project to create custom linux-based systems for embedded products regardless of the hardware architecture.", URL `https://www.yoctoproject.org/` (2016).

[241] Zhang, J., G. Fang, C. Peng, M. Guo, S. Wei and V. Swaminathan, "Profiling energy consumption of DASH video streaming over 4G LTE networks", in "Proc. Int. Workshop on Mobile Video (MoVid)", pp. 3:1–3:6 (2016).

[242] Zhang, J., G. Song, G. Qiao, T. Meng and H. Sun, "An indoor security system with a jumping robot as the surveillance terminal", IEEE Trans. Consumer Electronics **57**, 4, 1774–1781 (2011).

[243] Zhang, W., "Using axinom to deliver widevine licenses to azure media services.", URL `https://azure.microsoft.com/en-us/documentation/articles/media-services-axinom-integration/` (2016).

[244] Zhao, Y., X. Gong, W. Wang and X. Que, "A rate adaptive algorithm for HTTP streaming", in "Proc. Cloud Computing and Intelligent Systems (CCIS)", pp. 529–532 (2012).

[245] Zhu, G., F. Zhang, W. Zhu and Y. Zheng, "HTML5 based media player for real-time video surveillance", in "Proc. Image and Signal Processing (CISP)", pp. 245–248 (2012).

[246] ZigBee Alliance, "Zigbee and wireless radio frequency coexistence - zigbee white paper", Tech. rep., ZigBee Alliance (2007).