# Augmenting Film and Video Footage with Sensor Data

Norman Makoto Su*, Heemin Park†, Eric Bostrom*, Jeff Burke‡, Mani B. Srivastava†, Deborah Estrin*
*Department of Computer Science,
†Department of Electrical Engineering,
‡School of Theater, Film and Television
University of California, Los Angeles
normsu@cal.berkeley.edu, {hmpark,mbs}@ee.ucla.edu, {ebostrom,destrin}@cs.ucla.edu, jburke@hypermedia.ucla.edu

***Abstract*— With the advent of tiny networked devices, Mark Weiser's vision of a world embedded with invisible computers is coming to age. Due to their small size and relative ease of deployment, sensor networks have been utilized by zoologists, seismologists and military personnel. In this paper, we investigate the novel application of sensor networks to the film industry. In particular, we are interested in *augmenting* film and video footage with sensor data. Unobtrusive sensors are deployed on a film set or in a television studio and on performers. During a filming of a scene, sensor data such as light intensity, color temperature and location are collected and synchronized with each film or video frame. Later, editors, graphics artists and programmers can view this data in synchronization with film and video playback. For example, such data can help define a new level of seamless integration between computer graphics and real world photography. A real-time version of our system would allow sensor data to trigger camera movement and cue special effects. In this paper, we discuss the design and implementation of the first part of our embedded film set environment, the augmented recording system. Augmented recording is a foundational component for the UCLA Hypermedia Studio's research into the use of sensor networks in film and video production. In addition, we have evaluated our system in a television studio.**

## I. Introduction

It remains to be seen what the real-world wireless sensor network applications (WSNA) will be. Much of the ongoing motivation for WSNs are for monitoring natural phenomenon such as seismic propagation in buildings [1], animal habitats [2] or red-tide concentrations [3]. In such applications, WSNs are an ideal fit because they are minimally intrusive to their surrounding environment, as well as being disposable and easily deployed. As of late, however, wireless sensor networks have also been viewed as an enabler of ubiquitous environments. One can imagine sensors being deployed in groceries, schools or shopping malls. The requirements for a ubiquitous environment parallel those of the aforementioned monitoring applications. Indeed, these sensors will, as Mark Weiser prophetically stated, "weave themselves into the fabric of everyday life until they are indistinguishable from it" [4]. Sensor networks promise to provide a way to actuate on and record the dynamic events that are part of our everyday lives.

We believe WSNs will also prove immensely useful for *filmmaking* and *media production*. In traditional production, these networks will enable cost savings through efficient and detailed data gathering as well as offer increased expressive capabilities for the creative team through decision support and control of existing automated equipment. Our research seeks to explore the potential application of WSN deployments on film and television sets to support the daily work of production. Using WSNs, we can gain the following:

*Seamless Integration:* Any devices that are deployed on the set or on the performers ought to provide as little interference as possible to actors, directors and technicians.

*Mobility:* In addition, filming is often done in various harsh environments where deploying large equipment can be difficult. Thus, robust, lightweight and easily configured equipment is ideal.

*Increased Expressiveness:* Special effects and complex shots are heavily scripted and precisely choreographed to achieve the desired results. In the future, data from a WSN could be used to adjust these processes on-the-fly according to creative rules defined by the director or cinematographer.

In this paper, we describe the first step in realizing a ubiquitous film set where sensors are deployed on a set to monitor and actuate based on lighting conditions, acoustic events, director cues and performer gestures. We have built a system infrastructure to *augment* film footage with sensor data. Augmented footage is a sensor data record synchronized with the film camera's 'privileged' visual record[1]. At this early stage, the system supports display of sensor data corresponding to each frame when video proxies from a shot are reviewed.

While recording a scene, sensors will collect vital data that is *synchronized* with each film or video frame. In this post-facto synchronization scheme, one can later view the recorded film and observe its corresponding sensor data. Such synchronized data can be utilized, for example, by digital artists to add CG to scenes according to the environment. In the future (as yet unimplemented), a real-time version of this augmented recording system could actuate camera movements and special effect cues. From hereon, "augmented recording system" shall be abbreviated as ARS.

This paper is organized as follows. Section II describes related work, and Section III describes the requirements for ARS. This is followed by Sections IV-VIII, an outline of

[1]The film camera's viewpoint is 'privileged' because for film production, all that matters at the end of a shoot is what has been captured by that camera.

Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04)
0-7695-2090-1/04 $ 20.00 © 2004 IEEE

ARS's system architecture and modules. Evaluations are detailed in Sections IX and X. Finally, Section XI states our conclusion and future works.

## II. Related Work

Previous work has been done at UCLA's Hypermedia Studio in theatre production. In a production of Eugene Ionesco's *Macbett* [5], a complete system for controlling theatrical lighting and sound based on performer position and movement was developed. For example, tracking devices were embedded into the witch characters' staves. The performer could, at will, conjure up lightning strikes or whirling winds by throwing the staff in the air or swirling it.

At MIT's Media Lab, Joesph Paradiso's Responsive Environments Group's work on interactive environments that, for example, create music based on user gestures, foot positions and jumping is an example of an environment embedded with sensors [6]. The focus here is more on building sensors which can dynamically change environments, rather than on recording changes in environments. However, dynamically changing camera movement, etc. is a future goal of ARS.

The augmented footage concept itself is similar in spirit to the Smart Kindergarten [7] and Classroom 2000 [8] projects in that it collects a variety of time-stamped data from human inhabited environments. On the other hand, ARS requires a much more stringent synchronization requirement.

There already exists a large body of research based on image processing. Indeed, one might wonder whether cameras may make the use of sensors a moot point. With image processing one can infer location, lighting, etc. Sensors, however, can provide a granularity of accuracy hard to achieve purely with image processing techniques. Moreover, cameras require line of sight with the scene being filmed. Should we wish to monitor multiple areas of interest in a film set, numerous cameras are required. On the other hand, sensors have a small form-factor, making them easy (and cheap) to deploy. Finally, there are many phenomenon that can not be easily inferred from image processing. Factors such as temperature or wind speed, which may prove to be interesting to installation art, are easily measured with sensor networks. We do, however, believe vision technologies will provide a valuable role, *with* WSNs, in the future embedded film or TV set. For example, image processing can provide head pose/gaze direction measurements [9], motion recognition [10], and tracking [11].

## III. System Requirements

To give a clearer idea of the requirements to consider when building an ARS, let us give an example scenario for video production:

1) Sensors are placed in a film set, as well as on performers.
2) A video camera starts recording a scene.
3) While recording:
   a) A *timecode generator* (Section VI-A) signal encodes frame number data onto an audio track of the video.
   b) Sensors collect frame stamped data in synch with the timecode generator.
4) The crew can view the footage with sensor readings mapped to frames (i.e. while playing back footage, the sensor data collected at each frame can be viewed).

Firstly, it is apparent that *time synchronization* with both the camera and sensors is necessary. Secondly, we require sensor data for each video frame. This requires a very high sampling rate demanded by TV standards. Thirdly, we wish to provide an easily extendible system encompassing a wide range of sensor devices. The final requirement is an interface to the system that enables users to control and manage the augmented recording system.

Preliminary mining of sensor data in augmented footage would support two access methods: 1) random access of film or video frames to provide additional scene information during film footage playback that is not apparent from footage alone and 2) sensor data-based queries of augmented footage about high-level events and lower-level environmental conditions. Currently, only item 1 is implemented via the Jini client (Section VIII).

## IV. System Architecture

In ARS, we separate software into four distinct modules:

*Sensor Node Neighborhood:* Each sensor node neighborhood is an autonomous cluster with one basestation and several neighboring sensors.

*Serial Port Server:* The serial port server directly issues commands to the various basestation sensors, interpolates missing data and will time synchronize sensors.

*Sylph Server Middleware:* Sylph [12] is a middleware solution for easily allowing higher level queries from Jini clients to manage sensors. This server translates messages between sensors and clients.

*Jini Client:* This is a Java client adhering to Jini standards. It provides an interface for the user to control ARS, store information on a database and to later inspect frame synchronized sensor data.

Figure 1 shows an overview of the key components and their connections. The lightning bolts indicate an Internet connection. The clouds represent a wireless neighborhood. Here, we describe the sequence of events leading to successful storage of frame-synched data.

1) A Jini client connects to a well-known lookup service, locates the Sylph server managing ARS and issues a specially formatted query string.
2) The Sylph server will translate the query, issue (if any) intermediary replies back to the Jini client and proceed to forward the query to the serial port server.
3) The serial port server will multiplex the message to the appropriate base station sensors.
4) The base stations will inform its surrounding sensors to begin transferring data synched with a global timecode generator.
5) As data streams back, the base stations will collect data and send it to the serial port server.

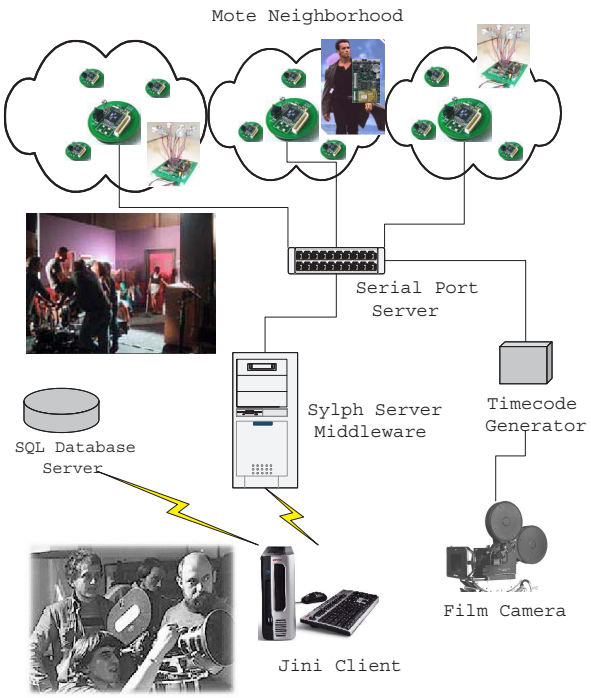


Fig. 1. System Architecture

6) The serial port server will perform error checking and interpolation on the data, format it and send it off to the Sylph server.
7) The Sylph server will announce to the Jini client that data is available.
8) The Jini client will store the ready data onto a (possibly remote) database for later perusal.

In the following sections we will go into greater detail with the software modules.

## V. Sensor Node Neighborhood

The sensors are Crossbow mica motes [13] with the standard sensor boards equipped with a thermistor, light sensor, microphone and accelerometer. In addition, we have developed a simple MAC (medium access control) protocol for high speed transfers. We chose motes because they are lightweight, easy to prototype on, have wireless radios, and there exist many sensor boards for it. In addition, it models the future vision of what many researchers believe such nodes will be—power constrained and limited in processing power. Another alternative include HP's iPAQs. However, they lack the portability and range of sensor boards that motes have.

### A. Software Platform

We are running an operating system developed by UCLA's NESL group, called PALOS (**P**seudo-Re**AL**time **O**perating **S**ystem) [14]. Currently, the bulk of today's mote programming is done with UC Berkeley's excellent TinyOS. However, for quick prototyping, we found that PALOS provides a distinct advantage. When programming with PALOS, one defines well-defined *tasks*. Each task has its own *event queue*. During the program execution, a global scheduler, by priority, runs through each event queue, dequeuing and passing the events to their respective tasks. By event, we mean a data structure which stores an identifier as well as some arbitrary data. Figure 2 shows the scheduler's cycle. Each task can also *send* messages (i.e. events) to other tasks by pushing an event into the appropriate task's queue. Programming with PALOS requires minimal use of macros and reduces the likelihood of deadlock or race conditions by enforcing a priority based schedule. However, the tradeoff is degradation in performance. Finally, code from TinyOS is easily ported to PALOS—several libraries used in ARS are based on TinyOS's.

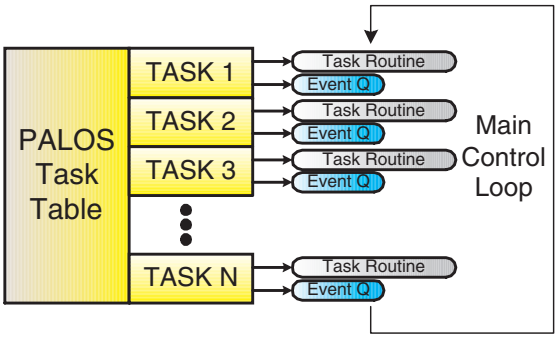


Fig. 2. PALOS Task Flow

In ARS, for example, we define the following tasks for a basestation mote:

*UART Task:* The UART Task waits for a fixed size data packet from the serial server and decides on an appropriate action. For example, when it receives a STOP_SENDING packet, it will tell the BMAC Task to send a packet wirelessly telling the neighboring motes to stop sending.

*Basic MAC (BMAC) Task:* The BMAC Task receives messages from the UART Task to send packets over the wireless link. It also sends messages to the Data Collection Task when it receives a wireless packet. So, for example, upon receiving a sensor reading from a neighboring mote, the BMAC Task will notify the Data Collection Task of this incoming packet.

*Data Collection Task:* The Data Collection Task will receive messages from the BMAC Task and forward the packet onward to the Serial Port Server. For example, it will receive an event with the data packet from the BMAC Task and then forward the data packet to the Serial Port Server.

The following code snippet in Listing 2 registers the UART and BMAC tasks:

```
1  UART0_Event uart0EventQ[UART_Q_SIZE];
2  BMAC_Event bmacEventQ[BMAC_Q_SIZE];
3
4  void globalInitTask() {
5      BMAC_id = PalosTask_register(StopWatch_init,
6          BMAC_task,
7          BMAC_XCNTR,
8          BMAC_RCNTR,
9          sizeof(StopWatch_Event),
10         BMAC_Q_SIZE,
11         (void *)bmacEventQ);
```

```
12      UART0_id = PalosTask_enroll(UART0_task,
13          UART0_XCNTR,
14          UART0_RCNTR,
15          sizeof(UART0_Event),
16          UART_Q_SIZE,
17          (void *)uart0EventQ);
18  }
```

Listing 1. Registering Tasks in PALOS

Lines 1 and 2 are the queues for their respective tasks. As data streams in from the UART or BMAC, the appropriate task routine pushes an event onto these event queues. The `globalInitTask()` function registers the two tasks (lines 5-17), informing PALOS to pass the popped event to either `BMAC_task` or `UART0_task`. The XCNTR and RCNTR variables can be redefined to control the frequency at which each task's event queue is processed.

When the main control loop, for example, reaches the UART0's event queue, it will pop the event in a FIFO fashion and pass it to a handler, `UART0_task`:

```
1  CHAR UART0_task(void *ev) {
2      UART0_Event *event = (UART0_Event *)ev;
3      UCHAR *recvData = event->data;
4      switch(recvData[0]) {
5         case OPCODE_NUM_NEIGHBORS:
6      ...
7      return PALOS_TASK_DONE;
8  }
```

Listing 2. An Event Handler in PALOS

The `ev` (line 1) parameter is the event popped off the queue. As evidenced by the code, a UART0 event has a `data` member. We then do something with that data, and finally return `PALOS_TASK_DONE` (line 7). `PALOS_TASK_DONE` tells the scheduler to pop the event off the queue permanently. Other variants include `PALOS_TASK_NEXT`, pop the event off the queue and immediately handle the next event from the *same* queue (i.e. for this example, we would immediately process the next UART event waiting in the queue, instead of perhaps processing the BMAC task next) and `PALOS_TASK_KEEP`, re-push the event onto the queue and process the next event queue normally. PALOS's complete source code, as well as coding examples, can be found at its Sourceforge page [14].

The MAC protocol is built on top of a physical layer which was ported from Wei Ye's communication stack [15]. While Berkeley's stack allows a longer transmission range, Ye has shown that his stack achieves high reliability (near 100%) and effective collision avoidance mechanisms with greater than 4 simultaneous transmitters. Our BMAC protocol is a very simple, barebones contention based protocol relying on carrier sense and ACKs—it does not use CTS and RTS. Our motivation behind this was to develop a reliable, quick protocol for sending short packets.

### B. Clustering

We chose to implement several "clusters" of mote neighborhoods. The reasoning behind not simply having one central basestation mote is simple—limited bandwidth. Let us assume we need approximately 20 motes deployed in the film set and packets of 30 bytes are transferred every frame (each frame lasts $\frac{1}{29.97}$ sec—Section VI-A has full details). Motes have an approximate maximum transfer rate of 10 kbits per sec. This would require transferring 30 bytes per 1.7 ms which is far greater than the 10 kbits per sec. In ARS, we define a cluster as a basestation and its neighboring motes. For a small film set this one-hop topology is sufficient; however, for larger sets requiring hundreds of sensors, a multi-hop tiered architecture would be more appropriate [16].

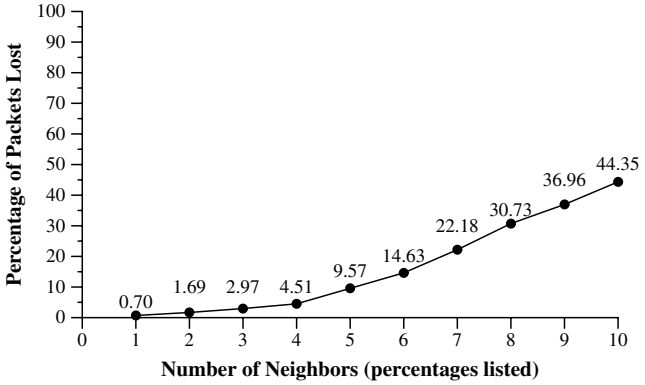


Fig. 3. Packet Loss Test Results

In general, the maximum number of neighbors allowable for each basestation depends on the environment, as well as the bandwidth. We conducted empirical tests in our lab to determine the lower bound on neighbor size. With a packet size of 30 bytes or, equivalently, 13 frames of data, we placed motes around a basestation (at approximately 4 inches away) and had each neighbor mote send packets at a rate of approximately $\frac{1}{29.97} \times 13 \approx 433\ ms$. The test stopped once 2000 packets were successfully sent from any neighboring mote. Results are shown in Figure 3. In ARS, we currently would like to limit our data loss rate to 10%. Given the results, setting the neighborhood to 5 or 6 motes seems reasonable.

### C. Basestation

The basestation motes multiplex on messages from the serial port server. Via a broadcast, it tells the neighboring motes to start/stop sending data. The basestation can also assign *roles* to neighboring motes (e.g. "record light" or "record sound"). The basestation will listen for data from the neighboring motes and forward them to the serial port server.

*1) Potentiometer (POT) Calibration Phase:* Before the basestation begins requesting data from its neighbors, it performs a potentiometer (radio power) calibration phase. The POT calibration phase serves three purposes: discovering neighboring motes' ids, assigning each neighbor mote a basestation and reducing the total area of the cluster. The later allows us to minimize interference between multiple clusters.

Note that a higher potentiometer setting implies a shorter range. For sake of clarity, we shall only refer to power settings from now on. The algorithm's input is $X$, the number of neighbors we want a basestation to find. The basestation broadcasts a message (at full power) requesting all neighboring motes to set their power to a minimal value and to reply back.

```
/* Need succ/trials to go on valid list */
succ <- 3
trials <- 4
neigh_found <- 0
init_power <- 0
/* constant factor to increment power */
power_inc <- 5
neigh <- Make-Zero-Hash(X)
valid <- Make-False-Hash(X)

BaseStation-Algo(X)
    Adjust-Power(0,X,init_power)

Adjust-Power(t,X,power)
    Broadcast-Change-Power(power,valid)
    Start-Timer()
    while not Timer-Expired()
        do msg <- BMAC-Read()
            if neigh[msg.src] = 0
                then neigh_found <- neigh_found + 1
            neigh[msg.src] <- neigh[msg.src] + 1
            if neigh_found = X
                then return
            else if t = trials
                for k in neigh.keys
                    do if neigh[k] >= succ
                        then valid[k] = true
                neigh <- Make-Zero-Hash(X)
                Adjust-Pot(0,X,power+power_inc)
            else Adjust-Pot(t+1,X,power)
```

Listing 3. Basestation POT Calibration Pseudocode

```
Neighbor-Algo(my_id)
    loop
        do msg <- BMAC-Read()
            for i in msg.valid
                do if i = my_id
                    then break
            Set-Power(msg.power)
            Unicast-Power-Set-ACK(msg.src)
```

Listing 4. Neighbor POT Calibration Pseudocode

The basestation adds those neighbors whose reply successfully reached it to its valid list. Then, the basestation rebroadcasts the same message, except this time, the power value is increased by some constant factor and the broadcast contains the valid list. Those neighbors who are already on the valid list ignore the message. The other neighbors will increment their power and reply back. This process is repeated until we find $X$ neighbors. All neighboring motes will remember the basestation which calibrated it and only reply to that basestation. For full details, the reader can see the pseudocode in Listings 3 and 4. In the current implementation, we require that for a neighbor to be pushed onto the valid list, it must successfully transmit to the basestation `succ` out of `trials` times. Our scheme is similar to Kumar's COMPOW work [17].

### D. Neighbors

Upon receiving a START_SENDING message from the basestation, the neighboring motes will buffer readings for $N$ frames. All data is time stamped with the mote's *own* internal film or video frame counter which is from 2 to `MAX_INT`.


Fig. 4. Neighbor→Basestation Packet Format

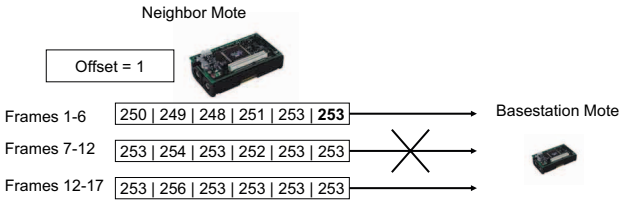


Fig. 5. Filtering Mechanism

This counter is based on the mote's internal clock. We start the mote from frame 2 due to our latency measurements in Section VI-C. Of course, clock drift and skewing will occur; a solution is also described in Section VI-C. After $N = 13$ frames are read, the readings are sent out in one packet to the basestation via unicast. A frame number $F$ is attached to indicate the frame number of the first data item; the packet will have $F$ to $F + N$ frames of data. The packet format is shown in Figure 4.

The nodes do a very simple filtering scheme. The rule is if the last frame of the previous packet is equivalent to all the data of the current packet to be sent, discard and don't send the packet. Equivalence may not be exact; for example, light readings off the Analog to Digital Converter (ADC) often fluctuate $\pm 2$ even though the light environment is unchanging. An "offset" value of $n$ makes data $\pm n$ apart equivalent (Figure 5).

Additionally, if a packet is waiting to be sent (for example, if carrier sense saw that the channel was busy or it is waiting for an ACK), we drop all further readings until the successful (or we give up after retrying some $k = 3$ number of times) transmission of the packet. This is handled by the network layer's BMAC module.

## VI. Serial Port Server

The serial port server waits for connections from the Sylph Middleware Server. It then translates and forwards these messages to each basestation mote. On initial startup, it sends a NUM_NEIGHBORS command to each of its basestation motes to find out how many neighbors surround each basestation and their respective IDs. Besides multiplexing messages from the Sylph Middleware Server, the serial port server has three important software modules, the Timecode Decoder Module, Interpolation Module, and Time Synchronization Module. The basestation sensors are connected to the serial port server via Inside Out Networks's Edgeport/8 (an 8 port to RS-232 converter).

### A. Timecode Decoder Module

The SMPTE, or Society of Motion Pictures and Television Engineers, timecode standard is an international industry standard timecode signal used in films to achieve time synchronization with frames. We use the Horita TG-50 Mini SMPTE

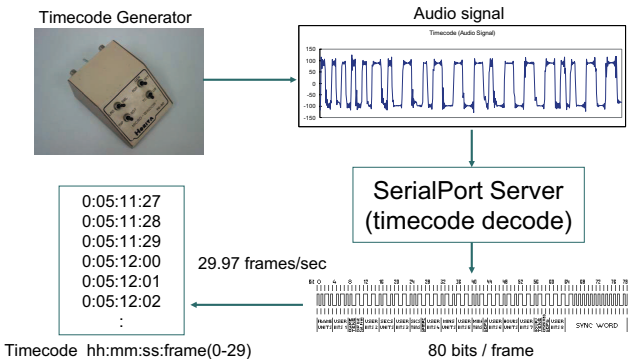


Fig. 6. SMPTE Decoding Process

```
OPCODE|Hour:Min:Sec:Frame|MoteID, Data|MoteID, Data|...
```

Fig. 7. Serial Port Server→Sylph Sensor Module Packet Format

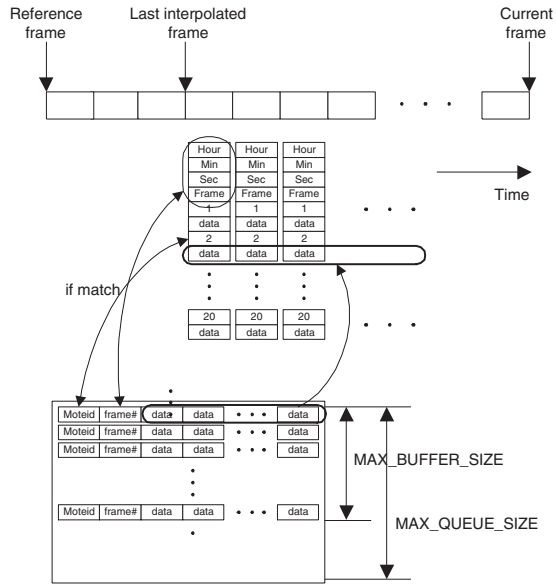


Fig. 8. Interpolation Process

Timecode Generator which generates LTC (longitudinal time code) audio signals (Figure 6) that are typically recorded on a special audio track on a tape, called the "strip". For television, such a signal has a sampling rate of 29.97 fps. Film has a rate of 24 fps. The audio signal encodes the hours, seconds, minutes and frames that have passed since the timecode generator was started. Frames cycle from 0 to 29, while hours, seconds and minutes continue to iterate. How the device generates 29.97 fps is out of the scope of this paper; [18] has a good introduction on the SMPTE timecode synchronization system.

The timecode decoder module discretizes the SMPTE timecode and generates an event for each decoded period. Such information is used by the Timecode Synchronization Module and Interpolation Module.

### B. Interpolation Module

When a Jini client requests information from ARS, it expects to get back information for each and every frame. The interpolation module is responsible for combining all the data received at the multiple basestations and sending it to the Sylph Middleware Server. Since the network connection between the serial server and sensor module is assumed to be much more reliable (i.e. Internet or LAN), we pack all sensor readings into one packet. One packet contains one frame of data for all motes. Figure 7 shows the exact packet format. As mentioned in Section V, motes will not send redundant data. The interpolation module will interpolate missing data. Consider that we may receive packets from a mote for frames (these frame numbers are the *mote's* internal frame count) 2 to 15 and then 19 to 30. In such a case, the interpolation module will fill in frames 16 to 18 with the same data as in frame 15.

The interpolation module has an internal buffer with two threshold values, `MAX_BUFFER_SIZE` and `MAX_QUEUE_SIZE`. The two sizes allow us to continue receiving packets during the interpolation process. The internal buffer is filled with data from each of the basestations. This buffer queue is of size `MAX_QUEUE_SIZE` and once the queue reaches `MAX_BUFFER_SIZE`, the interpolation module will construct packets from the last interpolated frame (i.e., the last frame it sent up to the Sylph Middleware Server) to the current frame. By comparing the mote's internal frame number with the frame number calculated from the reference frame counter (i.e. the timecode generator's counter), the module can combine the data packets into the format described above and send it. Figure 8 illustrates the interpolation module process.

### C. Time Synchronization Module

Our time synchronization scheme is based on two factors—that there exists an upper bound on time drift for the motes and that broadcast messages are received at approximately the same time at all receivers. Other time synchronization schemes exist for WSNs [19]. Based on tests conducted at our lab [20], the time drift between two mica motes does not exceed 7 µs per second. Since the duration of one frame is $\frac{1}{29.97}$ seconds, we have:

$$T_{sync} = \frac{\frac{1}{29.97}}{7 \times 10^{-6}} = 4766.7 frames$$

Hence, if we sync the timers in the sensor nodes every 4766.7 frames, then the time drift shall not exceed one frame. Taking a conservative route, we set $T_{sync}$ to be 4500 frames and thus we resynchronize approximately every 2.5 minutes.

The SYNC command is broadcasted from the basestation motes. This command instructs the neighbors to stop sending data, reset their internal frame counters to 2 and then immediately start sending data again. Since broadcast messages are inherently unreliable due to their lack of ACKs, ARS will instruct the basestations to continue sending SYNC until they receive data marked with frame number $< 13 \times T$, from *all* its neighbors. Note that after synching, there is a chance some

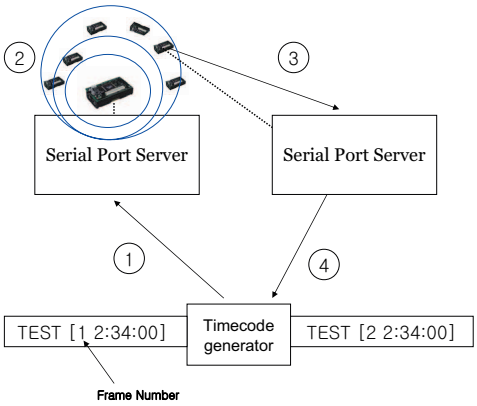


Fig. 9. Latency Test


Fig. 10. Jini Client GUI

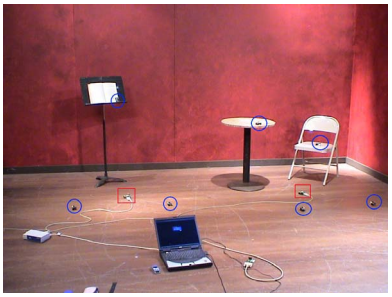

Fig. 11. Deployment in a TV Studio (Circles are neighboring motes, squares are basestation motes)

packets are lost, even though the timer did in fact reset. This is why we set a value $T$, rather than having the basestation mote listen for a certain frame number like 0 or 13 to tell us that SYNC succeeded.

The timecode generator is the reference frame counter. Upon sending a SYNC command, the Serial Port Server records the time from the timecode generator. Any packets (which are timestamped with the sending *mote's* counter) received afterwards are in reference to that recorded reference frame counter.

However, how can we ensure that the motes will start sending data at the exact same time a SYNC command is executed? For this, we constructed the experimental setup detailed in Figure 9. We hooked up a basetation mote and sensor node to serial ports 1 and 2, respectively. At a certain instance of time, the current timecode address is transferred to the basestation mote through serial port 1. Then the basetation mote will broadcast this packet to the sensor nodes. Once the sensor node receives this packet, the contents of the packet are printed out through serial port 2. When receiving packets from serial port 2, we compare the contents of the packet and the current timecode address. We found that the latency is at most 3 frames after 20 trials. This time corresponds to about 100 ms. This latency includes the transmission time over the wireless channel, all the processing time in the motes, the serial port server processing and any delay in serial cables. To utilize this latency information, we set the initial frame value in the sensor node using this latency value. Even such a small skew of 2 or 3 frames (0.1 seconds) can have a detrimental effect on some applications of ARS—lip synchronization, for example, with audio becomes noticeable with only a few frames of skew (100-150 ms) [21].

## VII. Sylph Server Middleware

The Sylph [12] system handles registration, multiplexing and higher-level query translation for sensor networks. Sylph was chosen because it is particularly well-suited for supporting multiple types of sensor nodes and allows one to write minimal new Jini code for each new type of sensor. It has been successfully used in sensor network projects at UCLA such as Smart Kindergarten. On the lower-layer, Sylph is based on using SET and GET operators on attributes. Once one defines an appropriate *Sensor Module* for their sensors and its attributes, it is easy for a Jini client to change or get these attribute values. For example, in our Augmented Recording Sensor Module, we define attributes such as light, period and command. To begin sending data, the Jini client queries a service discovery domain for sensors and then issues queries to each of them of the form "READ LIGHT EVERY 30 SECONDS". Such a query gets translated to SET and GET requests—"SET PERIOD 30 SECONDS" and "SET COMMAND=STARTSENDING". Thus, Sylph provides a common interface for Jini clients to interact with sensor nodes.

## VIII. Jini Client

The client will receive data for each frame and for each mote and will store this information at a database. We use Microsoft Access to store our information and access/store information by using a RMI-JDBC bridge driver.

In designing a GUI to couple with the Jini client, our first phase will be to design a GUI that will allow us to "scrub" through film or video frames. In other words, one can fast-forward, or rewind through frames and, at the same time, sensor data corresponding to the displayed frames will be shown. The client allows one to create clusters of nodes, find their neighbors and to start or stop collection of data. Figure 10 shows the client displaying sensor data during playback of film recorded for the evaluation in Section X.

Fig. 12. Example of gradual light changes in the studio.

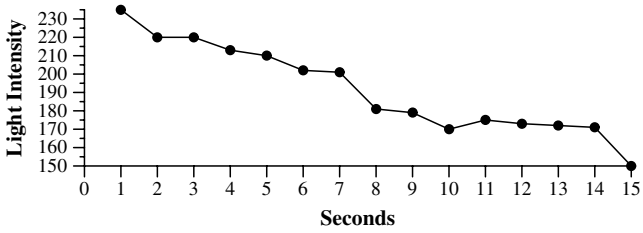


Fig. 13. Graph of light changes from Figure 12.

## IX. Evaluation I

We deployed 10 sensors (2 basestations, each with 4 neighbors) in a television studio. Figure 11 shows the overall layout of the studio. The test area was 1232x115 centimeters. The studio has lighting controls accessed from a control room. The control room allows the operator to finely adjust the light intensity of certain overhead lamps. It also allows one to quickly flash lights. In this paper, we discuss test results from gradually changing the light intensity and flashing the light quickly in the studio. The metric here is the latency involved between an event captured on video and on the ARS database. In the tests, the clustering algorithm took an average of 2.59 minutes to complete. In the three tests, we recorded for approximately 2.5, 2.5 and 10.0 minutes. The total amount of data transferred from neighbors to basestation in 2.5 minutes is 1114884.0 bytes (recall that the packet size is 31 bytes):

$$(2.5 \times 60) \div \frac{1}{29.97} \times 31 = 139360.5 \; bytes$$
$$139360.6 \times 8 = 1114884.0 \; bytes$$

For 10.0 minutes, this is 4459536 bytes. Database sizes were about 390 MB (using MS Access) for all tests.

We only tested with light readings. However, our system can easily be extended to work with other sensors such as temperature, acoustics, etc. that can be attached to our nodes.

### A. Gradual Light Intensity Changes

Figure 12 shows footage of a gradual change of lighting. The maximum light intensity value is 255. In these three frames, the values were (from left to right) 235, 200 and 150. Figure 13 shows the corresponding values from the leftmost frame to the rightmost frame in the aforementioned figure. From 7 to 8 seconds, there is a large drop—the change is not gradual. This is due to the SYNC command issued from the basestation. In our tests we found that the broadcast mechanism was far from reliable. Any broadcast to request a SYNC *must* be received by all the neighboring motes, before ARS can continue. Otherwise, if a SYNC is not acknowledged (i.e., all packets have reset their frame counter to 0), the serial server will continue to broadcast. We found that continual transmissions can sometimes last almost 10 seconds (in Figure 13's case, 2 retries of SYNC were required). Perhaps polling, or unicasting, each neighbor to SYNC would be better, since ACKs are utilized. However, this might cause the neighboring motes to start sampling data at slightly different times.

Fig. 14. Beginning and ending frames for a light flash.

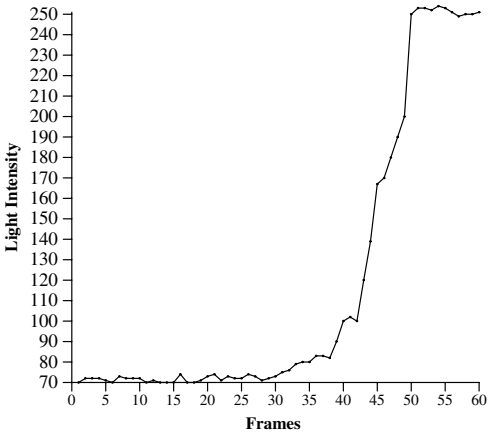


Fig. 15. Light intensity recorded, where frame 1 equals the left frame of Figure 14.

### B. Sudden Light Intensity Changes

We found it difficult to gauge the latency purely by watching the video and recorded sensor data stream side-by-side. So, with this test, we stopped the video player (Window Media Player) at the frame the instant the flash occurs. This test had a measure of subjectivity in that it depends on the tester's judgment of when a light change is visible. A flash is not truly instantaneous, but we would expect the data to show a sharp increase in light intensity within a short period.

Figure 14 shows the first frame we chose. The graph at Figure 15 starts off at that frame. As shown, the light intensity does not start rising until about frame 28. This indicates a latency of nearly one second. This is a rather large latency (when working within the frames metric) and should be fixed. In another test we left running for 5 minutes, the latency was at most 2 seconds and the average latency was 1.3 seconds. So while the SYNC was working to prevent the time drift, there was still a significant latency (solved in Section X).

## X. Evaluation II

After the evaluation in Section IX, we determined that the observed latency was caused by inaccuracies with how the

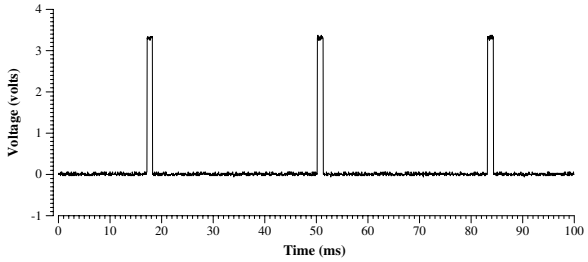


Fig. 16. Mote Timing Test: Pin Readings

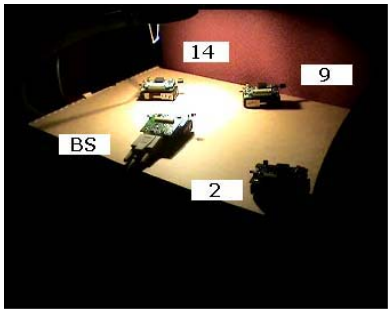


Fig. 17. Evaluation II Setup: One Basestation, Three Neighbors (IDs are shown). The lamp is above mote 14.

mote was counting 33 ms. The TinyOS Timer module, which multiplexes on one of the mica mote's timers (called "timer0"), was found via oscilloscope measurements to count 35 ms ticks, rather than 33 ms ticks. A 2 ms differential is rather large—every 17 seconds, a one second delay is aggregated. Instead, we dedicated timer1 for counting samples in ARS.

Figure 16 shows a sample of readings taken at a 10 ms resolution on an oscilloscope with timer1. A pin on the mote was set every time an ADC reading was requested and cleared once the ADC readings were ready to be processed. As the graph verifies, the pin is raised every 33 ms and the ADC data is ready approximately 1 ms after the request. The later measurement is important in that it tells us that the ADC data can be processed before the next frame's data becomes available.

With the aforementioned modifications, we reconducted our tests on a smaller scale. A halogen lamp with an attached light dimmer was placed over four motes—one basestation and three neighbors. The screenshot displayed in figure 17 shows the setup. In this evaluation, we used a different video player, RadLight, because of its ease in jumping directly to user-specified frames.

### A. Delay Measurements

From 46 to 57 seconds of the recording, we quickly raised and dimmed the light (mimicking the tests in Section IX-B). The light intensity measurements over that period are shown in Figure 18.

We determined that the first time the light is raised is at approximately frame 11 frame and then, it is quickly dimmed at approximately frame 45 . Snapshots of these frames are in Figure 19. We did several short bursts of raising and dimming

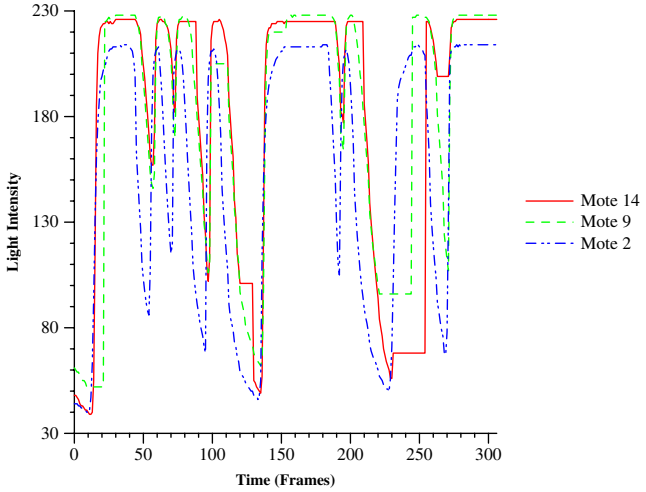


Fig. 18. Light Readings from 00:00:46-00:00:57

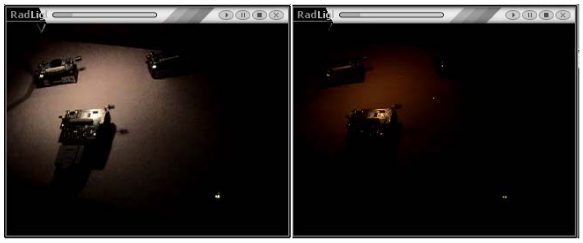

Fig. 19. Snapshots of Frame 11 and 45 from Figure 18

the intensity until frame 120 and then we let the light go dim for an extended period. This was followed by brightness for an extended period and then some random bursts of brightness and darkness. As the figure shows, the data nicely corresponds to the light changes. When viewing the data with the frames, there is no noticeable delay in sensor readings with frames Any delay is most likely a delay of about 10 frames and most certainly less than 20 frames.

In addition, we measured delays immediately before and after the SYNC command was issued. In both cases, the delay was less than 20 frames.

## XI. Conclusion and Future Work

We have described our implementation of the Augmented Recording System and a brief evaluation in a television studio. Our system is fully functional and demonstrates a maximum latency time of 667 ms (20 frames). ARS integrates the Sylph Middleware Server to allow easy interoperability with standard Jini clients. We have used tools such as PALOS to easily design mote programs. With augmented footage users can view film or video frames mapped to sensor data readings. This is the first step to realizing a film set embedded with wireless sensor networks at UCLA's Hypermedia Studio.

Our future work aims to not only improve ARS's performance, but to further couple it with the day-to-day work of cinematographers and directors on the set:

*Calibration*: During our experiments it quickly became evident that despite equivalent phenomenon, two different

sensor nodes would output readings which were different from each other. The degree of difference varied widely and some sort of normalizing of readings is needed. Work done by Bychkovskiy, et al. [22], begins to address this issue.

*More Sensors*: Light intensity is certainly not the only reading we are interested in. Sensors that detect color temperature, location, audio, etc. will be especially applicable to film or television. Cameras themselves can report aperture, shutter angle, etc.

*Semantic Indexing of Video Streams*: We would like to allow the user, in some sort of high level language or macro, to express a certain *interest* in browsing parts of video streams where some property holds. For this, we are considering WSN runtime environments such as SensorWare [23] which provide TCL bindings.

*Enhanced Activity Detection Via Sensor Fusion*: In detecting certain activities on the film set, one type of sensor may not be enough. Our current system simply collects data from various sensors. Future versions will utilize fuse these sensor readings to give a more meaningful interpretation of events occurring in the environment—for example, vision sensors (cameras) for localization could complement other sensors in a film set.

*Dynamic Control*: Currently, the lag between an event and storing it on the database is too large for real-time actuation. However, should ARS be able to fulfill the real-time requirements, equipment positions or settings could be automatically calibrated based on events on the film set, or even on scripted events entered in some sort of macro language. Examples include telling a camera to shift angle dynamically based on the actual position of the actors, in order to maintin a certain framing as the camera and actors move.

*Continuity Management*: Finally, script supervision, or continuity, ensures that different shots and scenes be logically sound and consistent within a spatiotemporal context. For example, for practical reasons, certain scenes that are at the end of a film may be filmed first. Perhaps ARS would be let one to easily merge different augmented footage frame (or different footage for the same time frame in several shots of different angles) to allow checks for continuity.

## ACKNOWLEDGEMENT

The authors would like to thank Chih-Chieh Han for his valuable help and support in developing ARS. We would also like to acknowledge the UCLA School of Theater, Film and Television for graciously allowing us to run evaluations at their television studio. Work done on ARS is based on past research partially supported by the National Science Foundation (NSF) under Grant No. ANI-0085773 and the Center for Embedded Network Sensing (CENS), a NSF Science and Technology Center. Additionally, the second author would like to express his appreciation to Samsung for their support.

## REFERENCES

[1] S. Baher, P. M. Davis, and G. Fuis, "Separation of Site Effects and Structural Focusing in the Santa Monica Damage Zone from the Northridge Earthquake," in *Bulletin of the Seismological Society of America*, vol. 92, no. 8, Dec. 2002.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," in *The Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[3] A. Requicha, D. Caron, C. Zhou, W. Zhang, and X. Liu. Detection and Identification of Marine Microorganisms. [Online]. Available: http://cens.ucla.edu/ProjectDescriptions/Microorganisms_Identification/index.html

[4] M. Weiser, "The Computer for the 21st Century," *Scientific American*, pp. 94–104, Sept. 1991.

[5] J. Burke, "Dynamic Performance Spaces for Theater Production," *Theatre Design & Technology (U.S. Institute of Theatre Technology)*, vol. 38, no. 1, 2002.

[6] J. Paradiso, C. Adler, K. Hsiao, and M. Renolds, "The Magic Carpet: Physical Sensing for Immersive Environments," in *Proceedings of the CHI '97 Conference on Human Factors in Computing Systems, Extended Abstracts*, 1997, pp. 277–278.

[7] M. Srivastava, R. Muntz, and M. Potkonjak, "Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments," in *Proceedings of the ACM SIGMOBILE 7th Annual International Conference on Mobile Computing and Networking*, July 2001.

[8] G. D. Abowd, "Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment," in *IBM Systems Journal*, vol. 38, no. 4, 1999.

[9] Y. Matsumoto and A. Zelinsky, "Algorithm and Real-time Implementation of Head Pose and Gaze Direction Measurement System Using Stereo Vision," in *Proceedings of the IEEE 4th International Conference on Face and Gesture Recognition (FG2000)*, 2000, pp. 499–505.

[10] C. Rao, A. Yilmaz, and M. Shah, "View-Invariant Representation and Recognition of Actions," in *International Journal of Computer Vision*, vol. 50, no. 2, 2002, pp. 203–226.

[11] S. L. Dockstader and A. M. Tekalp, "On the Tracking of Articulated and Occluded Video Object Motion," in *Real-Time Imaging (Special Issue)*, vol. 7, no. 5, pp. 415–432.

[12] A. Chen, R. R. Muntz, S. Yuen, I. Locher, S. I. Park, and M. B. Srivastava, "A Support Infrastructure for the Smart Kindergarten," *Proceedings the 6th International Symposium on Wearable Computers*, 2002.

[13] TinyOS: Hardware Designs. [Online]. Available: http://webs.cs.berkeley.edu/tos/hardware/hardware.html

[14] PALOS Sourceforge Page. [Online]. Available: http://palos.sourceforge.net

[15] W. Ye, J. Heidemann, and D. Estrin, "A Flexible and Reliable Radio Communication Stack on Motes," USC Information Sciences Institute," Technical Report ISI-TR-565, Sept. 2002.

[16] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the Physical World with Pervasive Networks," in *IEEE Pervasive Computing*, 2002, pp. 59-69.

[17] S. Narayanaswamy, V. Kawadia, R. S. Sreenivas, and P. R. Kumar, "Power Control in Ad-Hoc Networks: Theory, Architecture, Algorithm and Implementation of the COMPOW Protocol," in *Proceedings of the European Wireless Conference - Next Generation Wireless Networks: Technologies, Protocols, Services and Applications*, Feb. 2002, pp. 156–162.

[18] SMPTE EBU timecode by Phil Rees. [Online]. Available: http://www.philrees.co.uk/articles/timecode.htm

[19] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002.

[20] S. Ganeriwal, private communication, 2002.

[21] C. Sreenan, "To Use or Avoid Global Clocks?" in *The IEEE Workshop on Multimedia Synchronization (Sync '95)*, May 1995.

[22] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak, "Colibration: A Collaborative Approach to In-Place Sensor Calibration," in *2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003, pp. 301–316.

[23] A. Boulis, C.C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *The First International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, May 2003.