Panoptes: A Scalable Architecture for Video Sensor Networking Applications

Wu-chi Feng, Brian Code, Ed Kaiser, Mike Shea, Wu-chang Feng
Dept. of Comp. Sci. and Engg.
OGI School of Science and Engg @ OHSU
20000 NW Walker Rd
Beaverton, OR 97006
wuchi@cse.ogi.edu

ABSTRACT

Video-based sensor networks can provide important visual information in a number of applications including: video surveillance, environmental monitoring, health care, and emergency response. This paper describes the Panoptes video-based sensor networking architecture, including its design, implementation, and performance. We describe a video sensor platform that can deliver high-quality video over 802.11 networks with a power requirement of approximately 5 watts. In addition, we describe the streaming and prioritization mechanisms that we have designed to allow it to survive long-periods of disconnected operation. Finally, we describe a sample application and bitmapping algorithm that we have implemented to show the usefulness of our platform. Our experiments include an in-depth analysis of the bottlenecks within the system as well as power measurements for the various components of the system.

Keywords

Video sensor networks, video surveillance, streaming

1. INTRODUCTION

There are many sensor networking applications that can significantly benefit from the presence of video information. These applications include video-only sensor networks or sensor networking applications in which video-based sensors augment their traditional sensor counterparts. Examples of such applications include environmental monitoring, health-care monitoring, emergency response, robotics, and security/surveillance applications. Video sensor networks, however, provide a formidable challenge to the underlying infrastructure due to the relatively large computational requirements and the size of the resulting video data. The amount of video generated can consume the same bandwidth as potentially thousands of scalar sensors. As a result, video sensor networks must be carefully designed to be both low in power consumption as well as flexible enough to support a broad range of applications and environments.

To understand the flexibility required in video sensor configurations, we briefly outline three applications. In environmental observation systems, the tetherless nature of the application requires video sensors that are entirely self-sufficient. In particular, the sensors must be equipped with power that is generated dynamically via solar panels or wind-powered generators and managed appropriately. In addition, networking connectivity may be at a premium, including possibly intermittent or "programmed" network disconnection. For this application, keeping the sensor continuously running indefinitely while collecting, storing, and transmitting only the most important video is the primary goal. For video security applications, the video sensors should filter as much of the data at the sensor as possible in order to maximize scalability, minimize the amount of network traffic, and minimize the storage space required at the archive to hold the sensor data. The sensors themselves may have heterogeneous power and networking requirements. In outdoor security applications, the power may be generated by a solar panel and may use wireless networking to connect to the archive. For indoor security applications, the sensors most likely will have power access and will be connected via wireless or wireline networks. Finally, for emergency response scenarios, the video sensors may be required to capture and transmit full-motion video for a specified period of time (i.e. the duration of the emergency) The goal in these situations might be to meet a target operating time with minimal power adaptation, in order to provide emergency response personnel with the critical information throughout the incident.

In this paper, we describe the development of the Panoptes video sensor networking project that is underway at the Oregon Graduate Institute. In particular, we will describe the design, implementation, and performance of the Panoptes sensor node, a low-power video-based sensor. The Panoptes sensor consists of an Intel StrongARM-based embedded device that runs at approximately 5 watts of power. For this sensor, we have implemented an adaptive video delivery mechanism that can dynamically manage a buffer of data so that it supports intermittent and disconnected operation. Finally, we will describe a video sensor application that we have developed. In this application, we have designed a

change detection bit-mapping algorithm to allow video data to be queried efficiently.

In the following section, we provide a description of the embedded sensor platform, including the software system that was developed for the sensor. Following the description of the Panoptes video sensor, we describe a scalable video sensor application that has been designed to show some of the features of the video sensors. The experimentation section will provide an in-depth analysis of the performance of the video sensor and its subcomponents. In Section 5, we describe some of the work related to ours and how it differs. Finally, we conclude with some of our future work and a summary.

Contributions of this paper: We describe the design and implementation of a low-power, yet high-quality video sensor platform that can be used for video-based sensor networks. Then, we describe a dynamic video management and streaming system for the video sensors that allows the sensor to continue to operate with intermittent and disconnected operation. Finally, we propose and demonstrate a bit-mapping algorithm that allows change detection queries to be accomplished efficiently.

2. VIDEO SENSOR PLATFORM

In designing a video-sensor node, we had a number of design requirements that we were trying to accomplish:

- Low power: Whether power is scarce or plentiful, minimizing the amount of power required to capture the video is important. For environments where power is scarce, minimizing power can significantly increase the duration of time that sensors can operate. For environments where power is plentiful, minimizing power can significantly increase the number of sensors that can be economically deployed. For example, in private homes, owners may be willing to deploy a large number of 5-watt video sensors (equivalent to a night light) while on vacation. However, they may be unwilling to use workstation counterparts that could easily consume two orders of magnitude more power.

- Flexible adaptive buffering techniques: We expect that the video sensors will need to support a variety of latency and networking configurations, with a buffer on the sensor acting as the intermediate store for the data. Of course, the buffer can hold only a finite amount of data and will need to have some of it removed to hold new data. The data that is removed, however, is application dependent. For some applications, data older than some time prespecified time may be useless, while in other applications the goal will be to transmit as much data as possible over the network no matter how intermittent it is. Two such applications might include commuter traffic monitoring for the former case and coastal monitoring for the latter case. Thus, we require a flexible mechanism by which applications can specify both latency and a mapping of priorities for the data that is being captured.

- Power management: A low-power video platform is just one component of the video sensor. The video sensor also needs to be able to adapt the amount of video that is being captured to the amount of power that is available. Just as in the flexible adaptive buffering techniques, power management also needs to be flexible. For example, in one scenario, the application requirement might be to have the sensor turn on and capture as much video as it can before the battery dies. In another scenario, it might be necessary for the sensor to keep itself alive using only self-generated power (such as from a solar panel or a wind-powered generator).

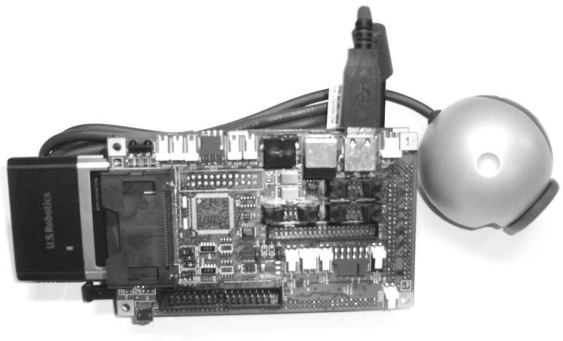


Figure 1: Panoptes Sensor

In the following section, we will describe the hardware platform that serves as the basis of our video sensor technology. Following that, we will describe the software that we have developed to help address some of the design requirements above.

2.1 Panoptes Sensor Hardware

In designing the video sensor, we had a number of options available to us. The most obvious platform in the beginning was the StrongARM based Compaq IPAQ PDA. This platform has been used for a number of research projects, including some at MIT and ISI[1]. As we will describe in the experimentation section, we found that the popular Winnov PC-Card video capture device was slow in capturing video and also required a large amount of power. The alternative to this was to find an embedded device, allowing us to move to a USB-based video camera as well as to remove the LCD video screen which was unnecessary for the video sensor. The video sensor that we developed is based on the Intel StrongARM 206 MHz embedded platform. The device is approximately 7 inches long (with the 802.11 card inserted) and approximately 4 inches wide. The sensor has a Logitech 3000 USB-based video camera, 64 Mbytes of memory, the Linux 2.4.19 operating system kernel, and an 802.11-based networking card. Note that while 802.11 is currently being used, it is possible to replace it with a lower-powered RF radio device. By switching to a USB-based camera platform, we were able to remove the power required to drive the PC-Card. The video sensor is shown in Figure 1. The complete device including the compression and transmission over 802.11 consumes approximately 5.5 Watts of power while capturing and delivering video of 320x240 resolution at 18-20 fps.

[1] http://pads.east.isi.edu/ipaq.htm

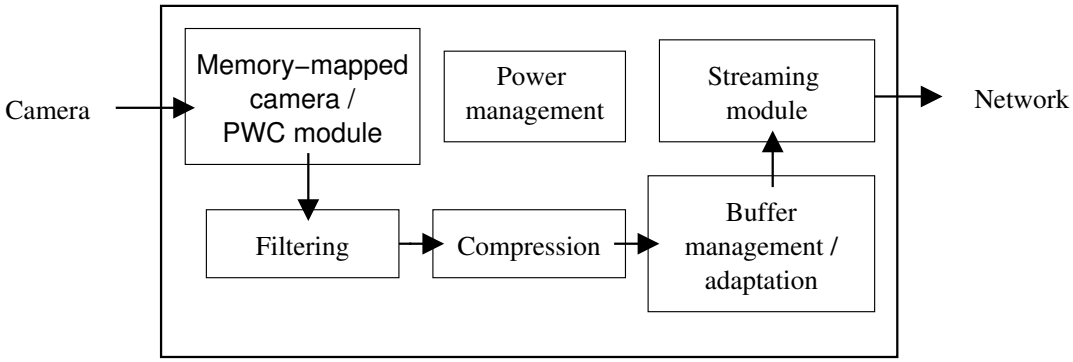


Figure 2: Panoptes Sensor Software Components

As far as we know, this is the first viable Intel StrongARM-based video sensor that can capture video at a reasonable frame rate (i.e. greater than 10 frames per second), while using a small amount of power. The other platforms that we are aware of will be described in the related work section.

2.2 Panoptes Sensor Software Architecture

There are a number of options available in architecting the software on the video sensor. The components must perform video capture, compression, filtering, buffering, adaptation, and streaming. To connect the components, a number of alternatives were considered including a single-threaded synchronous architecture, a single-threaded asynchronous architecture, and a multi-threaded architecture with each component acting as a thread. We chose to use a single-threaded architecture with asynchronous movement of data throughout the system. We felt that this allowed for the most control over the timing within the system, while allowing the parts to be specialized for individual applications. The major components of the system are shown in Figure 2. In the rest of this section, we will briefly describe the individual components.

2.2.1 Video Capture

As previously mentioned we chose a USB-based video camera. We are using the Phillips Web Camera interface with video for Linux. Decompression of the data from the USB device occurs in the kernel before passing the data to user space and allows for memory mapped access to decompressed frames. Polling indicates when a frame is ready to be read and further processed through a filtering algorithm, a compressor, or both.

2.2.2 Compression

The compression of video frames, both spatially and temporally, allows for a reduction in the cost of network transmission. We have currently set up JPEG and differential JPEG as the compression format on the Panoptes platform. Although JPEG itself does not allow for temporal compression of data, it saves on computational cost (relative to MPEG), and thus power. Compression on the Panoptes sensor is CPU bound, as a 320X240 4:1:1 YUV frame requires approximately 33 ms of CPU time (StrongARM 206 MHz). As will be shown in the experimentation section, we have taken advantage of some of Intel's performance primitives that are available for the StrongARM processor.

While we are not researching low-power video coding techniques, we expect that other compression technologies can be incorporated into the video sensor relatively easily.

2.2.3 Filtering

The main benefit of a general purpose video sensor is that it allows for application specific video handling to be accomplished. For example, in a video security application, having the video sensor filter uninteresting data without compressing or transmitting it upstream allows the sensor network to be more scalable than if it just transmitted the data upstream. For environmental observation, the filter may create a time-elapsed image, allowing the data to be compressed as it is needed by the application as well as minimizing the amount that needs to be transmitted [12]. Finally, in applications that require meta information about the video (e.g. image tracking), the filtering component can be set up to run the vision algorithms on the data.

The filtering subcomponent in our system allows a user to specify how and what data should be filtered. Because of the relatively high cost of DCT-based video compression, we believe that fairly complex filtering algorithms can be run if they reduce the number of frames that need to be compressed. For this paper, we have implemented a brute-force, pixel-by-pixel algorithm that detects whether or not the video has changed over time. Frames that are similar enough (not exceeding a certain threshold) can be dropped at this stage if desired.

2.2.4 Buffering and adaptation

Buffering and dynamic adaptation are important for a number of reasons. First, we need to be able to manage transmitting video in the presence of network congestion. Second, for long-lived, low-power scenarios, the network may be turned off in order to save precious battery life. In our case, using 802.11 networking accounts for approximately 1/3 of the power consumption. Finally, in the event that the buffer within the video sensor fills up, efficient mechanisms need to be in place that allow the user to specify which data should be discarded first.

For our sensor, we employ a priority-based streaming mechanism to support the video sensor. The algorithm presented here is different from traditional video streaming algorithms that have appeared in the literature (e.g. [5, 3]). The main

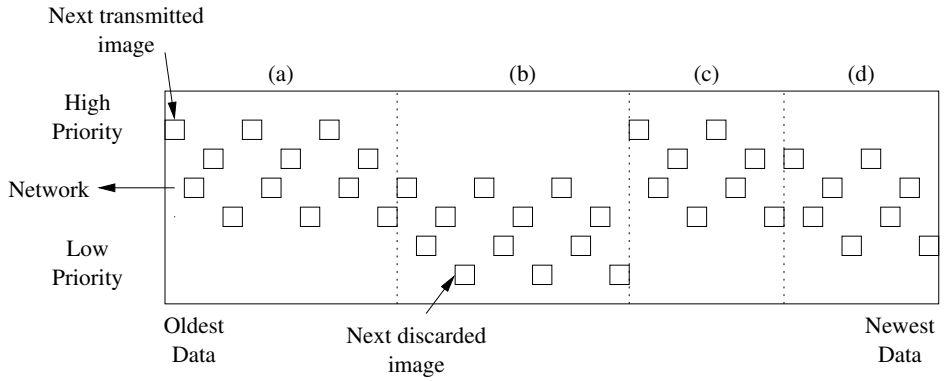


Figure 3: A Dynamic Priority Example

difference is that in traditional video streaming algorithms, the video data is known in advance but needs to be delivered in time for display. For most non-real-time video sensor applications, the video data is being generated at potentially varying frame rates to save power and the data being captured is being generated on the fly.

Priority-Based Adaptation: We have defined a flexible priority-based streaming mechanism for the buffer management system. Incoming video data is mapped to a number of priorities defined by the applications. The priorities can be used to manage both frame rate and frame quality. The mapping of the video into priorities is similar to that in [5, 8]. The buffer is managed through two main parameters: a high-water mark and low-water mark. If the buffered data goes beyond the high-water mark (i.e. the buffer is getting full), the algorithm starts discarding data from the lowest priority layer to the highest priority layer until the amount of buffered data is less than the low-water mark. Within a priority level, data is dropped in order from the oldest data to the newest. This allows the video data to be smoothed as much as possible. It is important to note that the priority mapping can be dynamic over time. For example, in the environmental monitoring application, the scientist may be interested in higher quality video data during low and high tides but may still require video at other times. The scientist can then incrementally increase the quality of the video during the important periods by increasing the priority levels. Figure 3 shows one such dynamic mapping.

Data is sent across the network in priority order (highest priority, oldest frame first). This allows the sensor to transfer its highest priority information first. We believe that this is particularly important for low-power scenarios where the sensor will disconnect from the network to save power and scenarios where the network is intermittent. As shown in the example, frames from the regions labeled (a) and (c) are delivered first. Once the highest priority data are transmitted, the streaming algorithm then transmits the frames from regions (a), (c), and (d). Note, that the buffering and streaming algorithm can accept any number of priority layers and arbitrary application-specific mappings from video data to priority levels.

3. THE LITTLE SISTER SENSOR NETWORKING APPLICATION

Video sensor networking technologies must be able to provide useful information to the applications. Otherwise, they are just capturing data in futility. In order to demonstrate the usefulness of video-based sensor networking applications, we have implemented a scalable video surveillance system using the Panoptes video sensor. The system allows video sensors to connect to it automatically and allows the sensors to be controlled through the user interface. We will provide a demonstration of our system at the ACM Multimedia conference. The video surveillance system consists of a number of components, including the video sensor, a video aggregating node, and a client interface. The components of the system are shown in Figure 4 and are described in the rest of this subsection.

3.1 The User Interface

The user interface for the Little Sister Sensor Networking application that we have deployed in our lab is shown in Figure 5. In the bottom center of the application window is a list of the video sensors that are available for the user to see. The list on the right is a list of events that the video sensor has captured. The cameras are controlled by a number of parameters which are described in the next section. The video window on the left allows events to be played back. In addition, it allows basic queries to be run on the video database. We will describe the queries that our system can run in the Query Manager section.

3.2 Video sensor software

In this application, the video sensors are fully powered and employed 802.11 wireless networking to network the sensor to the aggregating node. To maximize the scalability of the system, we have implemented a simple change detection filtering algorithm. The basic goal of the motion filtering is to identify events of interest and have it capture video for the event. This algorithm does a pixel by pixel comparison in the luminance channel. If sufficient pixels within a macroblock are greater than some threshold away from their reference frame, then the image is marked as different and recording of the video data begins. The video is then recorded until motion stops for a user-defined time, *event_end_time*. The *event_end_time* allows us to continue recording in the event that the object being recorded stops

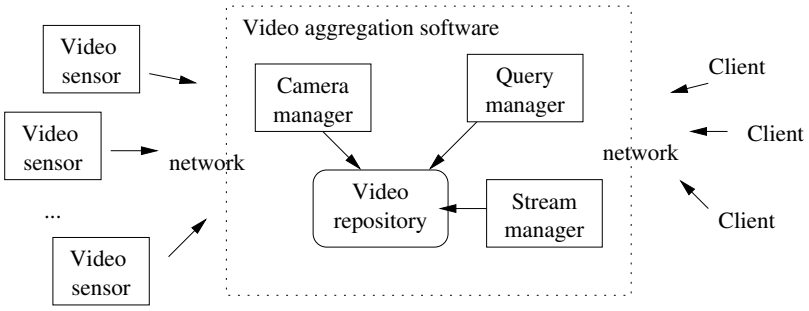


Figure 4: The Little Sister Sensor Software Components

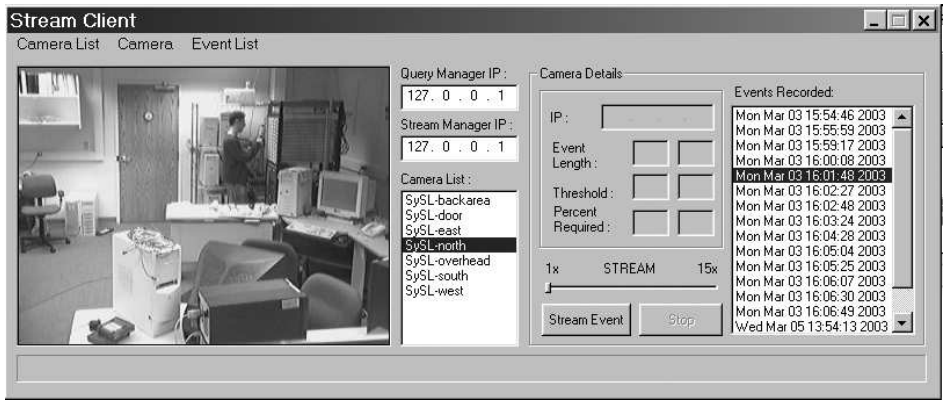


Figure 5: The Little Sister Sensor Networking Application

for a while and then continues movement. For example, a person walking into the room, sitting down to read a few web pages, and then leaving may have 5 second periods where no motion is perceived (i.e. the person is just reading without moving).

In addition to event recognition component, we propose a simple bitmapping algorithm for the efficient querying and access to the stored video data. To accomplish this, we create a map of the video data as an event that it is recording unfolds. For each image, a binary bitmap is created where each bit represents whether or not the macroblock[2] has changed. This allows us to create an *image bitmap* of where the interesting areas of the video are. Furthermore, as will be described in the next section, the video aggregation node can use this to expedite queries for the users.

Upon activation, the sensors read their configuration file to set up the basic parameters by which they should operate, including frame rate, video quality, video size, IP address of the video aggregator, etc. While we statically define the parameters by which they operate, one can easily imagine incorporating other techniques for managing the sensors automatically.

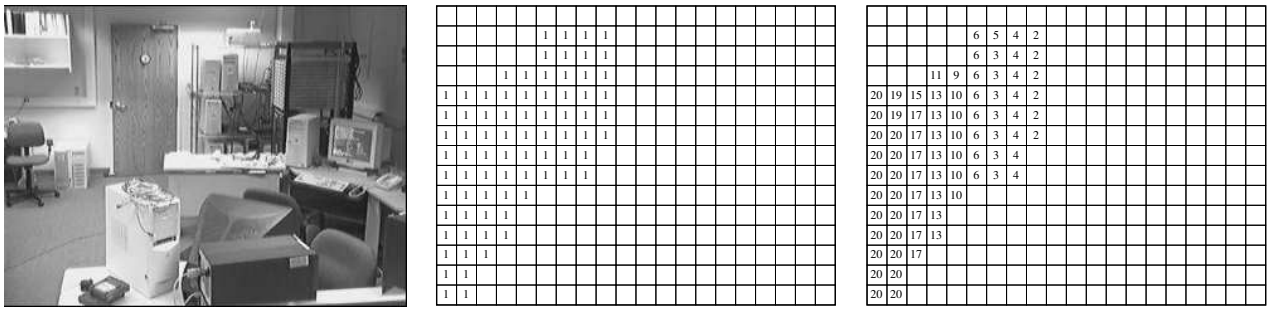
3.3 Video Aggregation Software

The video aggregation node is responsible for the storage and retrieval of the video data for the video sensors and the clients. It can be at any IP connected facility. There are a number of components within the video aggregation node. The three principle parts are the *camera manager*, the *query manager*, and the *stream manager*.

The camera manager is responsible for dealing with the video sensors. Upon activation, the video sensors register themselves with the camera manager. This includes information such as the name of the video sensor. The camera manager also handles all the incoming video from the video sensors. In order to maximize the scalability of the sensor system, multiple camera managers can be used. One important part of the camera manager is that it creates an *event_overview_map* using the bit-mapped information that is passed from the video sensor. The purpose of the event_overview_map is to create an overview of the entire event to aid in the efficient querying of the video data. The event_overview_map can be constructed in a number of ways. In this paper, we describe two such techniques. Before describing the techniques, it is important to remember that our goal is to make the system more scalable. The bit-maps allow for events of interest to be retrieved. Technologies such as object tracking and object recognition can be used in conjuction with this system.

Union maps take all the image bitmaps for a single event and combine them together into the *event_overview_map* using a bitwise OR. This allows the system to quickly find events of interest (e.g. *Who took the computer?*). An example of the union map for someone walking through our lab (Figure 6(a) is shown in Figure 6(b). *Trail maps* extend the notion of union maps by incrementing a counter for every n images.

[2] 16x16 pixel block. Refer to [10] for details.



(a) (b) (c)

Figure 6: Union Map and Trail Map Examples

This allows an event characterization to be accomplished. For example, the trail bitmap can be used to distinguish people entering the lab or exiting. In Figure 6(c), we have shown a sample trail map for someone entering the lab and walking through it, while not touching any computers in the right side of the lab. We have currently incorporated union maps into our sample application.

The query manager is responsible for handling requests from the clients. Queries are entered into the video window. The user can right click to highlight 16x16 pixel regions within the video screen. The user can select any arbitrary shape of regions of interest. Upon receiving the query, the query manager finds all events within the system that have one of the regions in its *event_overview_map*. The list of matching events is then returned to the user. As an example, we have shown a sample query, in which the user highlighted part of the computer at the bottom of the image. The query manager responded with only three events. Compared with the large list of events from the same camera in Figure 5, the simple bitmapping algorithm has reduced the number of events considerably. Note that the last event on the list is a video clip capturing a student moving the computer to his cube.

The stream manager is responsible for streaming events of interest to the clients. We have implemented the camera, query, and stream managers as separate components in order to maximize the scalability of the system. While we have all three components running on a single host, it is possible to have them on separate hosts.

4. EXPERIMENTATION

In the first part of this section, we will describe the experimental results that we obtained from the various components of the video sensor including metrics such as power consumption, frame rate, and adaptability to networking resources.

4.1 USB Performance

One of the interesting limitations of using USB to receive the video data from the camera is that its internal bandwidth is limited to 12 megabits per second. This 12 megabits includes USB packet header overhead so that the actual usable bandwidth is less. For a typical web camera capturing in 4:2:0 YUV at 320x240 pixel resolution, the theoretical maximum frame rate sustainable is only 13 frames per second. Fortunately, or unfortunately, most USB cameras provide primitive forms of compression over the USB bus using mostly proprietary algorithms. The alternatives to this are firewire and USB 2.0. Most of the low-power embedded processors do not support either technology because the manufacturers feel that the processors are unable to fully utilize the bandwidth.

Image Size	Compression	fps	% System CPU
160x120	0	29.63734	4.475
	1	29.76638	22.289
	3	29.88162	15.708
320x240	0	4.87626	2.849
	1	28.71610	67.166
	3	29.68214	44.495
640x480	0	-	-
	1	14.13746	82.656
	3	14.73258	77.651

Table 1: Effect of USB Compression on Framerate and System Usage

When testing the video capture capabilities of the sensor, we set it up to grab video frames from the camera as quickly as possible, and then simply discard the data. For each resolution and USB compression setting, we recorded the frame rate as well as the amount of load that doing so puts on the sensor. We measured two metrics for a variety of parameters over 3,000 captured frames: (i) the average frame rate captured and (ii) the amount of load placed on the system. To measure frame rate, we took the total frames captured and divided it by the time required to capture all of the frames. The latter measurement shows us the load that the driver places on the system. To measure this, we ran the experiment to capture 3000 frames and then used the *rusage* system call to find out the user, system, and total time of the experiment. We then calculated system load by summing the user and system times and dividing this by the total time.

Table 1 lists the performance of the video sensor using the various compression settings and frame sizes. The Philip's based video camera can only be set to three different resolutions: 160x120, 320x240, and 640x480. As shown in the table, the sensor is easily able to capture 160x120 video. This is not unexpected as the total bandwidth required to

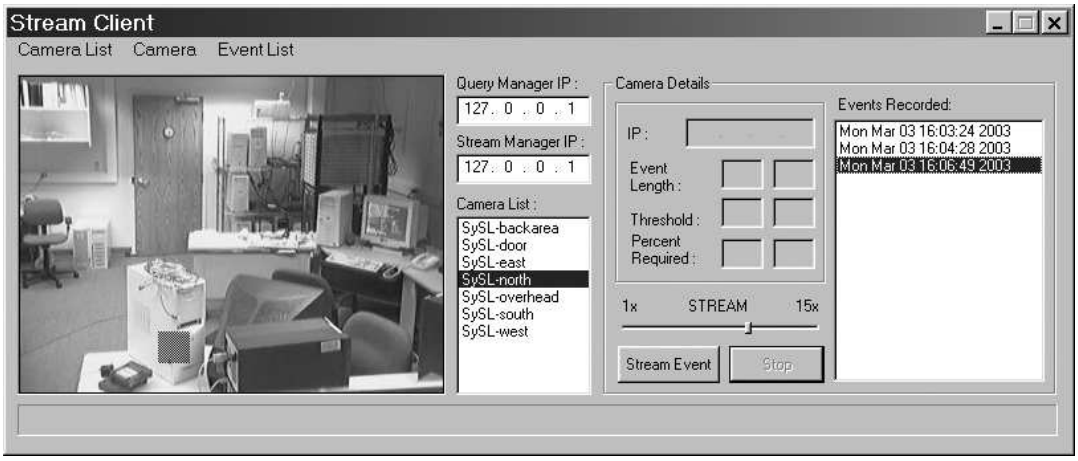


Figure 7: Example of Bit-Map Query

transmit 160x120 video at 30 frames per second is only 6.9 megabits, well beneath the USB bus bandwidth limit. For the various compression levels (1 being a higher quality stream with less compression and 3 being the lowest quality stream with high compression), we found that the system load introduced can be quite significant for the lightweight sensor. At the lowest compression setting, 22% of the CPU capacity is needed to decompress the video data from the USB camera. We believe that much of this time is spent touching memory and moving it around, rather than running a complex algorithm such as an IDCT. Using higher compression for the video data from the USB camera reduces the amount of system load introduced. We suspect that this is due to the smaller memory footprint of the compressed frame.

At 320x240, we encounter the Achilles' Heel of the USB-based approach. Using uncompressed data from the camera, we are only able to achieve a frame rate of 5 frames per second (similar to the PC-card based approaches). With higher overhead (i.e. more time for decompression), we can achieve full frame rate video capture. In addition, we see that the amount of system load introduced is less than that required for the 160x120 stream. We suspect that this is again due to I/O being relatively slow on the video sensor. At 640x480, the video camera driver will not let the uncompressed mode be selected at all. Theoretically, one could achieve about 3 frames per second across the USB bus, but we suspect that if this mode were available, only 1 frame per second would be achievable. Using compression, we are able to achieve 14 frames per second, but we pay a significant penalty in having the video decompressed in the driver.

As an aside, we are currently working on obtaining an NDA with Philips so that the decompression within the driver can be optimized as well as possibly allowing us to stay in the compressed domain.

4.2 Compression Performance

We now focus on the ability of the video sensor to compress data for transmission across the network. Recall, we are interested in using general purpose software, so that algorithms such as filtering or region-of-interest coding can be accomplished on an application-specific basis. Software compression also allow us to have control over the algorithms that are used for compression (e.g. nv, JPEG, H.261, or MPEG).

Image size	IPP (ms)	ChenDCT (ms)
320x240	26.98	73.34
640x480	104.14	278.24

Table 2: Standalone Optimized vs. Unoptimized Compression Routines

To measure the performance of compression on the 206 MHz Intel StrongARM processor, we measure the performance of an off-the-shelf JPEG compression algorithm (ChenDCT) and a JPEG compression algorithm that we implemented to take advantage of some of Intel's StrongARM Performance Primitives. In particular, there are some hand-coded assembly routines that use the architecture to speed up multimedia algorithms. Among these are algorithms to perform the DCT algorithm, quantization and Huffman encoding. For test data, we use a sample image in 4:2:0 YUV planar form taken from our lab and use it to test just the compression component. For each test, we compressed the image 300 times in a loop and averaged the measured compression times.

As shown in Table 2, we are able to achieve real-time compression of 320x240 pixel video using the Intel Performance Primitives. More importantly, it takes approximately one third the time using the primitives compared with using a less optimized software only algorithm. As shown in the second row of the table, compressing a larger image scales linearly in the number of pixels and that we are able to achieve approximately 10 frames per second using a high-quality image. It should be noted that the compression times using the IPP are dependent on the actual video content. For our experimentation, we chose a reasonable image of our lab (similar to those shown in Figure 5).

Image size	IPP (ms)	ChenDCT (ms)
320x240	30.307	102.334
640x480	120.006	304.630

Table 3: Optimized vs. Unoptimized Compression Routines

	320x640	640x480
PWC Decode	16.96	55.99
Jpeg Encode	29.24	113.20
Bitmap Compare	6.00	23.40
Image Copy	2.19	9.38
Msg Create	0.54	1.24
Other	1.347	4.75

Table 4: Average Time Per Component in ms

4.3 Component Interaction

Having described the performance of individual video sensor components, we now focus on how the various components come together.

Because the capture and compression routines make up a large portion of the overall computing requirement for the video sensor, we are interested in understanding the interaction between them. Table 3 shows the performance of the sensor in capturing and compressing video data. Interestingly, the capture and compression with the Intel Performance Primitives results in approximately 4 milliseconds of overhead per frame captured. This scales linearly as we move to 640x480, requiring an additional 16 milliseconds per frame. For the ChenDCT algorithm, using either 320x240 or 640x480 video, the overhead of capturing data introduces a 24 millisecond overhead per frame. This seems to indicate that because the ChenDCT algorithm is unable to keep up the ability to capture video data that the I/O is being amortized during compression.

To fully understand what is going on, we have instrumented a version of the code to measure the major components of the sytem. To do this, we inserted *gettimeofday()* calls within the source code and recorded the amount of time spent within each major code segment over 500 frames. The time spent in each of these components is shown in Table 4. For the 320x240 pixel images, nearly all the time is spent in the USB decompression module and compressing the video data. Our expectation is that with an appropriately optimized USB decompression module we will be able to achieve near real-time performance.

For applications where video quality and not video rate is important, we see that at 640x480 pixel video, we are able to achieve on the order of 5 frames a second. We note that this frame rate is better than previously published results for 320x240 video data. In addition, we see that the time to compress the 640x480 video stream scales quite well, which is not entirely unexpected.

4.4 Power Measurements

To determine how much power is being drawn from the video sensor, we instrumented the sensor with an HP-3458A digital multimeter connected to a PC. This setup allows us to log the amount of current (and thus power) being consumed by the video sensor. To measure the amount of power required for the various components, we have run the various components in isolation or layered on top of another subsystem that we have already measured. The results of these measurements can be applied to power management algorithms (e.g. [9])

System State	Power (watts)
Idle	1.473
CPU Loop	2.287
w/Camera	3.049
w/Camera in sleep	1.617
w/Network	2.557
w/Camera & Network	4.280
Capture Running	5.268
Sleep	0.058

Table 5: Average Power Per Component in Watts

The results of the experiments are shown in Figure 8. From the beginning of the trace until about 6 seconds into the trace, the video sensor is turning on and configuring itself. During this time, the power being drawn by the sensor is highly variable as it configures and tests the various hardware components on the board. Seconds 6 to 10 show the power being drawn by the system when it is completely idle (approximately 1.5 watts). Seconds 10-13 show the video camera turned on without capturing. As shown by the differential from the previous step, the camera requires approximately 1.5 watts of power to operate. Seconds 13-16 show the camera sleeping. Thus, over a watt of power can be saved if the sensor is incorporated with other low-power video sensor technologies that notify it when to turn on. In seconds 19-22, we show the power required to have just the network card on in the system but not transmitting any data (approximately 2.6 watts). In seconds 22-27, we added the camera back into the system. Here we see that the power for the various components is pretty much additive, making it easier to manage power. That is, the jump in power required to add the camera with and without the network card in is approximately the same. In seconds 27-38, we show the entire system running. As one would expect with a wireless network, the amount of power being drawn is fairly variable over time. Between seconds 38-40, we removed the camera and the network card, returning the system to idle. We then ran the CPU in a tight computational loop to show the power requirements while being fully burdened. Here, we see that the system by itself draws no more than 2.5 watts of power. Finally, we put the sensor in sleep mode (seconds 50-55). In the sleep state, the sensor requires very little power (approximately 0.05 watts of power).

We have summarized the results in Table 5. The most important thing to draw from the experiments is that the power consumed by the sensor is relatively constant over time. The only variability comes from the network transmission. As a result, we expect that the algorithms for power management that are being worked on by others might fit into this frame work without much modification.

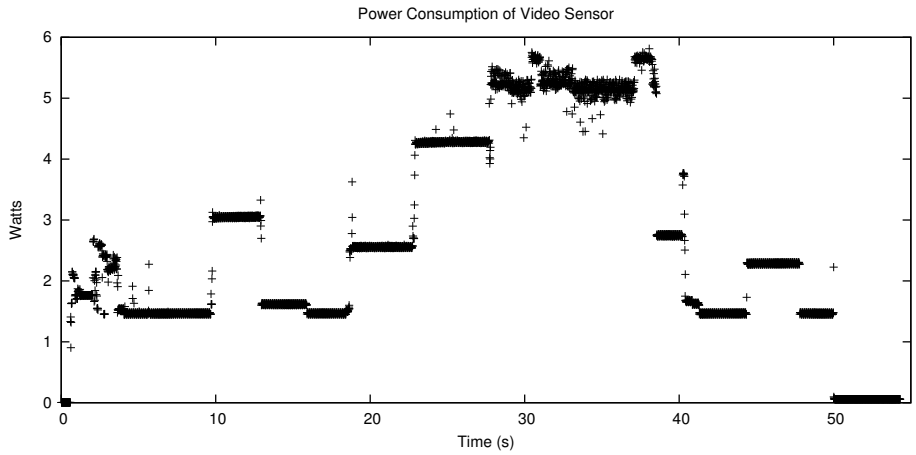


Figure 8: Power Consumption Profile

4.5 Buffering and adaptation

To test the ability of the sensor to deal with disconnected operation, we have run experiments to show how the video rate is adapted over time. In these experiments, we have used a sensor buffer of 4 megabytes with a high and low water mark of 3.8 and 4 megabytes, respectively.

For these experiments, we first turned on the sensor and had it capture, compress, and stream data. The experiment then turned the network card on and off for the times shown in Figure 9(a). The "on" times are shown as a value of 1 in the graph, while the "off" state is shown as a value of 0. As shown by Figures 9(b) and (c), the video sensor is able to cope with large amounts of disconnected time, while managing the video buffer properly. During down times, we see that the buffer reaches its high water mark and then runs the algorithm to remove data, resulting in the sawtooth graph shown in Figure 9(c). Once reconnected, we see that the buffer begins to drain with the networking bandwidth becoming more plentiful relative to the rate at which the video is being captured. Had the network been constrained, instead of off, the algorithm would converge to the appropriate level of video.

5. RELATED WORK

There are a number of related technologies to the proposed system detailed in this paper.

5.1 Sensor Networking Research

There are a tremendous number of sensor networking technologies being developed for sensor networking applications [4]. From the hardware perspective, there are two important sensors: the Berkeley Mote [7] and the PC-104-based sensor developed at UCLA [1]. The Berkeley Mote is perhaps the smallest sensor within the sensor networking world at the moment. These sensors are extremely low powered and have a very small networking range. As a results these sensors are really useful for collecting very small amounts of information. The PC-104-based sensor from UCLA is the next logical progression in sensor technologies that provides slightly more compute power. We believe the Panoptes platform is the next logical platform within the hierarchy of sensor network platforms. We expect that hybrid technologies, where Motes and the PC-104-based sensors can be used to trigger higher-powered sensors such as ours. This would allow the sensor network's power consumption to be minimized.

In addition to hardware sensors, there are a large number of sensor networking technologies that sit on top of the sensors themselves. These include technologies for ad hoc routing, location discovery, resource discovery, and naming. Clearly, advances in these areas can be incorporated into our video sensor technology.

5.2 Mobile Power Management

Mobile power management is another key problem for long-lived video sensors. There have been many techniques focused on overall system power management. Examples include the work being done by Kravets at UIUC, Noble at Michigan, and Satyanarayanan at CMU [9, 6, 2]. We have not yet implemented power management routines within the video platform. We expect that the work presented in the literature can be used to control the frame rate of the video being captured as well as when the networking should be turned on and off to save power.

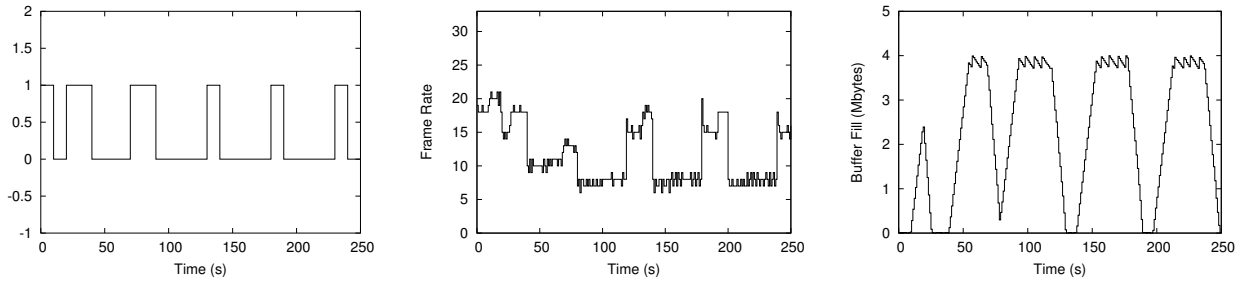
5.3 Video Streaming Technologies

There have been a large number of efforts focused on video streaming across both reservation-based and best-effort networks, including our own. As previously mentioned, the work proposed and developed here is different in that the that for streaming mainly due to the continuity requirements of streaming technologies.

For video streaming across wireless networks, there have been a number of efforts focused on maximizing the quality of the video data in the event of network loss. These schemes are either retransmission-based approaches (e.g. [11]) or forward error correction based (e.g. [13]).

6. CONCLUSION AND FUTURE WORK

In this paper, we have described our initial design and implementation of the *Panoptes* video sensor networking platform. There are a number of significant contributions that



(a) Network on/off model (b) Frame rate (c) Video sensor buffer

Figure 9: Dynamic Adaptation Results

this paper describes. First, we have developed a low-power, high-quality video capturing platform that can serve as the basis of video-based sensor networks as well as other application areas such as virtual reality or robotics. Second, we have designed a prioritizing buffer management algorithm that can effectively deal with intermittent network connectivity or disconnected operation to save power. Third, we have designed a bit-mapping algorithm for the efficient querying and retrieval of video data.

Our experiments show that we are able to capture fairly high quality video running on low amounts of power, approximately the same amount of power required to run a standard night light. In addition, we have showed how the buffering and adaptation algorithms manage to deal with being disconnected from the network.

While we have made significant strides in creating a viable video sensor network platform, we are far from done. We are currently in the process of assembling a sensor with a wind-powered generator for deployment along the coast of Oregon. Our objective is to use a directed 802.11 network to have a remote video sensor capture video data for the oceanographers at Oregon State. We have an operational goal of having the sensor stay alive for a year without power or wireline services. We are also working on creating an open source platform that can be used by researchers to include the fruits of their research. The goal is to have the sensors in use for research areas such as robotics, computer vision. Finally, we are working on similarity searching algorithms for the trail maps being generated.

7. ACKNOWLEDGMENTS

This work was supported in part by grants from the National Science Foundation (ANI-9875493 and EIA-0130344). The results and opinions expressed in this paper do not necessarily reflect the opinions of the funding agency.

8. REFERENCES

[1] N. Bulusu, J. Heidemann, D. Estrin, and T. Tran. Self-configuring localization systems: Design and experimental evaluation. *ACM Transactions on Embedded Computing Systems*, page To appear., May 2003.

[2] M. Corner, B. Noble, and K. Wasserman. Fugue: Time scales of adaptation in mobile video. In *ACM/SPIE Multimedia Computing and Networking*, January 2001.

[3] E. Ekici, R. Rajaie, M. Handley, and D. Estrin. Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proceedings of INFOCOM 99*, 1999.

[4] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Conneting the physical world with pervasive networks. *IEEE Pervasive Computing*, pages 59–69, January 1992.

[5] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev. A priority-based technique for the best-effort delivery of stored video. In *Proceedings of the SPIE Multimedia Computing and Networking*, January 1999.

[6] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, 1999.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[8] B. Krasic and J. Walpole. Priority progress streaming. In *ACM Multimedia 2001 Doctoral Symposium*, October 2001.

[9] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. *Wireless Networks*, 6(4):263–277, 2000.

[10] D. LeGall. A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[11] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proceedings of SIGCOMM 1998*, 1998.

[12] H. Stockdon and R. Holman. Estimation of wave phase speed and nearshore bathymetry from video imagery. *Journal of Geophysical Research*, 105(9), September 2000.

[13] W. Tan and A. Zakhor. Real-time internet video using error resilent scalable compression and tcp-friendly transport protocol. *IEEE Transactions on Multimedia*, 1(2), June 1999.