Panoptes: Scalable Low-Power Video Sensor Networking Technologies

WU-CHI FENG, ED KAISER, WU CHANG FENG, and MIKAEL LE BAILLIF
Portland State University

Video-based sensor networks can provide important visual information in a number of applications including: environmental monitoring, health care, emergency response, and video security. This article describes the Panoptes video-based sensor networking architecture, including its design, implementation, and performance. We describe two video sensor platforms that can deliver high-quality video over 802.11 networks with a power requirement less than 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.

Categories and Subject Descriptors: C.5.3 [**Computer System Implementation**]: Microcomputers—*Portable devices*

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Video sensor networking, video collection, adaptive video

1. INTRODUCTION

There are many sensor networking applications that can significantly benefit from the presence of video information. These applications can include both video-only sensor networks or sensor networking applications in which video-based sensors augment their traditional scalar 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 and bandwidth requirements of the resulting video data. The amount of video generated can consume orders of magnitude more resources than their scalar sensor counterparts. As a result, video sensor networks must be carefully designed to be both low power as well as flexible enough to support a broad range of applications and environments.

To understand the flexibility required in the way the video sensor are configured, we briefly outline three example applications:

This work is based upon work that was supported by the National Science Foundation (NSF) under grants ANI-0087761 and EIA-0130344.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Authors' addresses: Department of Computer Science, Portland State University, P.O. Box 751, Portland OR 97207-0751; email: {wuchi,edkaiser,wuchang}@cs.pdx.edu; mikael.le_baillif@enseirb.fr.

—*Environmental Observation Systems*: For oceanographers that want to understand the development of sandbars *beneath* the water's surface, video sensors can provide an *array* of sensors. Using image processing techniques on the video, the oceanographers can determine the evolution of such sandbars over time. 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 running indefinitely while collecting, storing, and transmitting only the most important video is the primary goal.

—*For Video Security and Surveillance Applications*: In these 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 sensor power may be generated by solar panels 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.

—*Emergency Response Systems*: A video-based sensor network may be deployed in order to help emergency response personnel assess the situation and take appropriate action. The video sensors may be required to capture and transmit high-quality 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 they need throughout the incident.

In this article, we describe the Panoptes video sensor networking project at Portland State University. In particular, we will describe the design, implementation, and performance of the Panoptes sensor node, a low-power video-based sensor. The sensor software consists of a component-based infrastructure that can have its functionality altered on-the-fly through Python-connected components. We also describe an adaptive video delivery mechanism that can manage a buffer of data so that it supports intermittent and disconnected operation. This buffering mechanism allows the user to specify how to gracefully degrade the quality of the video in the event that it is unable to transmit all the video data. Finally, we will describe a video sensor application that we have developed. In this application, we have designed an efficient algorithm to allow video data to be queried without analyzing pixel data directly.

In the following section, we provide a description of the embedded sensor platform, including the systems software and architecture of 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.

2. VIDEO SENSOR PLATFORM

In designing a video-sensor platform, we had a number of design goals that we were trying to accomplish including:

—*Low Power*. Whether power is scarce or available, minimizing the amount of power required to capture the video is important. For environments where power is scarce, minimizing power usage can significantly increase the time that the sensors can operate. For environments where power is plentiful, minimizing power usage can significantly increase the number of sensors that can be economically deployed. For example, homeowners 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 their laptop or desktop counterparts that can 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 may need to balance storing old data and new data. For some applications, data older than some prespecified time may be useless, while in other applications the goal will be to transmit as much captured data as possible (no matter how old it is) over the network. 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).

—*Adaptive Functionality*. The functionality of the sensor may need to change from time to time. Changing the functionality should not require the sensor to be stopped and reconfigured. Rather, the sensor should be able to add new functionality while running and should also minimize the amount of code transferred through the network.

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.

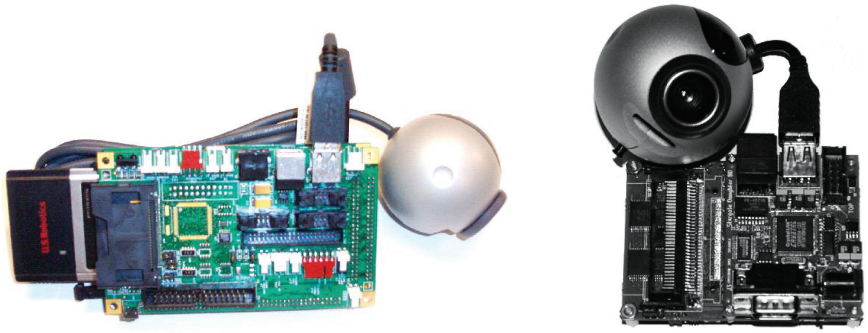
2.1 Panoptes Sensor Hardware

In designing the video sensor, we had a number of options available to us. The most prevalent 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. 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 with different input capabilities.

Our initial investigation into embedded devices uncovered a number of limitations unique to embedded processors that are not generally found in their laptop or desktop counterparts.

—*Limited I/O Bandwidth*. Many of the low-power devices today have either PCMCIA or USB for their primary I/O interconnects. Using PCMCIA-based devices typically requires significant power. For USB, low-power embedded devices do not support USB 2.0, which supports 455 Mb/sec. The main reason for this is that a fairly large processor would be required to consume the incoming data. For USB 1.0, the aggregate bandwidth is 12 Mb/sec, which cannot support the uncompressed movement of 320×240 pixel video at full frame rate.

—*Floating Point Processing*. The Intel-based embedded processors such as the StrongArm processors and the Xscale processors do not support floating point operations. For video compression algorithms, this means that they either need to have floating point operations converted to integer equivalents or the operations need to be emulated.



(a) (b)

Fig. 1. The Panoptes Video Sensors. (a) The Applied Data Bitsy platform. The sensor board is approximately 5 inches long and 3 inches wide. (b) The Crossbow Stargate platform. The Stargate platform is approximately 3.5 inches by 2.5 inches in size.

—*Memory Bandwidth*. The ability to move data in and out of memory is relatively small compared to their desktop counterparts. Image capture, processing, and compression, however, can involve processing a large amount of image in real-time.

The initial video sensor that we developed is an Applied Data Bitsy board utilizing the Intel StrongARM 206-MHz embedded processor. The device is approximately 5 inches long and approximately 3 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, lower frequency RF radio device. By switching to a USB-based camera platform, we were able to remove the power required to drive the PC-Card. After reporting our initial findings [Feng et al. 2003], we have ported the code base to work with the Crossbow Stargate platform. There are a number of advantages to this platform. First, it is made by the company that provides many of the motes to the sensor community. The Stargate was originally meant for use as a data aggregator for the motes. Second, while it has twice the processing power as the Bitsy board, it also consumes less energy. The video sensors are shown in Figure 1.

As far as we know, these are the first viable video sensors that can capture video at a reasonable frame rate (i.e., greater than 15 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 in architecting the software on the video sensor. The Panoptes video sensor that we have developed uses the Linux operating system. We chose Linux because it provides the flexibility necessary to modify parts of the system to specific applications. Furthermore, accessing the device is simpler than in other operating systems. The functionality of the video sensing itself is split into a number of components including capture, compression, filtering, buffering, adaptation, and streaming. 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 (USB 1.0) 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.

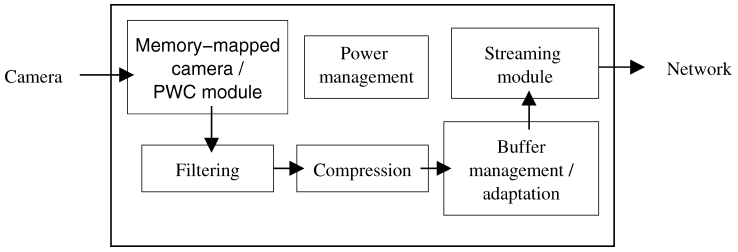


Fig. 2. Panoptes sensor software components.

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, differential JPEG, and conditional replenishment as the compression formats on the Panoptes platform. Although JPEG itself does not allow for temporal compression of data, it saves on computational cost (relative to formats such as MPEG), and thus power. Compression on the Panoptes sensors is CPU bound. As will be shown in the experimentation section, we have taken advantage of Intel's performance primitives for multimedia data that are available for the StrongARM and Xscale processors to make higher frame rates possible. While low-power video coding techniques are not the focus of this article, we expect that other compression technologies can be incorporated into the video sensor easily.

2.2.3 *Filtering.* The main benefit of a general-purpose video sensor is that it allows for application specific video handling and transformation to be accomplished at the edge of the sensor network, allowing for more sensors to be included in the system. 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 all 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 [Stockdon and Holman 2000]. 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 article, 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. Specifically, 802.11 networking consumes approximately one-third of the power. 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., Ekici et al. [1999] and Feng et al. [1999]). The main 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.

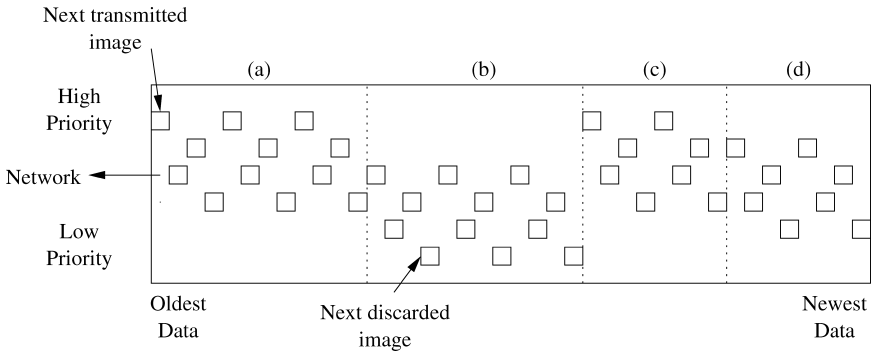


Fig. 3. A dynamic priority example.

While traditional video streaming algorithms can be employed for live streaming, we focus on adaptive video collection in this article.

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 Feng et al. [1999] or Krasic et al. [2003]. 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, the areas labeled (a) and (c) have been given higher priority than the frames in (b) and (d). Thus, the 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.

2.2.5 *Providing Adaptive Sensor Functionality*. In our initial implementation of the sensor, we simply modularized the code and connected the code via function calls. In order to change one of the parts such as the compression algorithm or the way filtering is accomplished, requires all of the code to be recompiled and sent to the sensor. In addition, the video sensor needs to be stopped and started with the new code. Thus, our goals for providing adaptive sensor functionality are to minimize the amount of code required to be compiled and transmitted to the sensor and to allow the sensor to dynamically alter its functionality without having to be manually stopped and restarted. As it turns out, the language Python allows for these goals to be met.

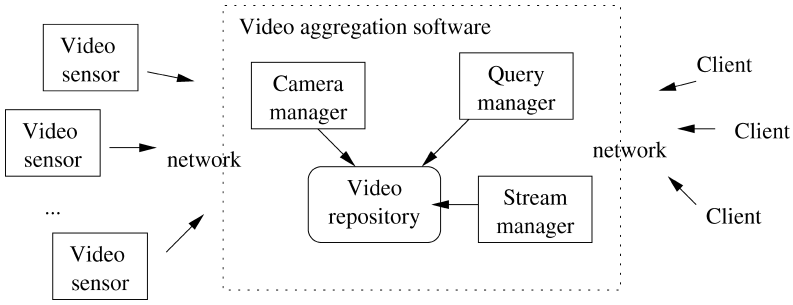


Fig. 4. The Little Sister Sensor software components.

Python is an interpreted programming language similar in vein to TCL or Perl that allows natively compiled functions in high-level languages such as C to be assembled together through a scripting language. As a result, it allows one to take advantage of the speed of optimized compiled code (e.g., JPEG compression routines, networking, etc.), while having the flexibility of a scripting language in construct. For our video sensor software, each component is compiled into its own object code with an appropriate Python interconnect. The Python script can then be constructed to stitch the components together. In order to change the functionality of the video sensor such as its compression algorithm, one need only compile the object for the new compression algorithm, load the object onto the sensor, and update the script that the sensor is using. We have set up the video sensor code to automatically re-read the script every 10 seconds so a change in the script will change the functionality of the sensor on the fly. Our performance measurements of the Python-based interconnects shows an overhead of approximately 0.5 frames per second for the system, or approximately 5%. We believe that the additional flexibility gained by such a system is worth the overhead.

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. 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 section.

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 Section 3.3.

3.2 Video Sensor Software

In this application, the video sensors are fully powered and employ 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

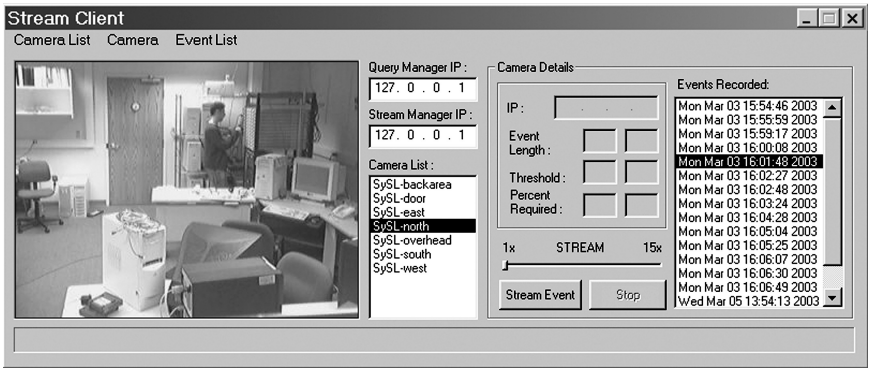


Fig. 5. The Little Sister Sensor Networking client interface.

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 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).

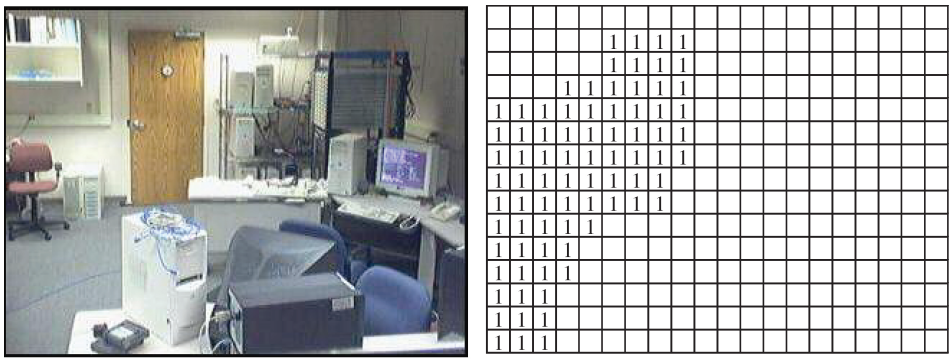
In addition to event recognition component, we propose a simple bitmapping algorithm for efficient querying and access to the stored video data. We create a map of the video data as an event is recording. For each image in the event, an *image bitmap* is created where each bit represents whether or not the luminance block has changed from the first image in of the event. This image bitmap indicates 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 one relatively simple technique.



(a) (b)

Fig. 6. Event bitmapping example.

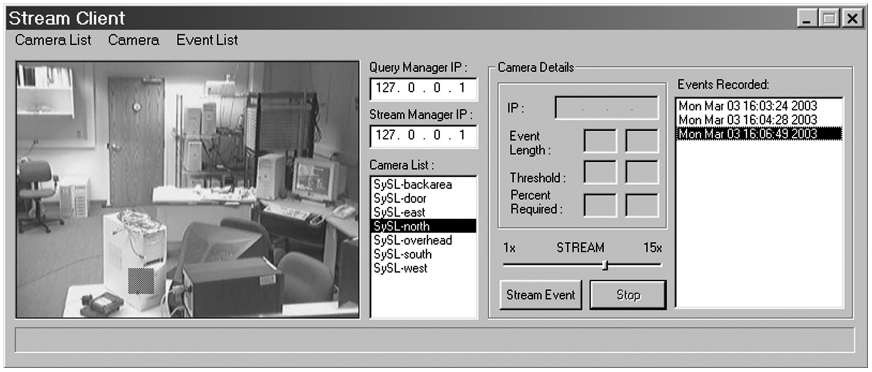


Fig. 7. Example of bit-map query. The right window shows the result of a query for events that relate to the area of the computer in the foreground.

Other techniques that track motion over time and create vectors of motion could also be integrated into the 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 that was sitting here?*). An example of the union map for someone walking through our lab (Figure 6(a)) is shown in Figure 6(b).

The query manager is responsible for handling requests from the clients. Queries are entered into the video window. The user can left click to highlight 8×8 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 (see Figure 7). 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. Although we have all three components running on a single host, it is possible to have them on geographically separated 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 standard 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 4:2:0 YUV data at 320 $\times$ 240 pixel resolution, the theoretical maximum frame rate sustainable is only 13 frames per second over USB. Fortunately, or unfortunately, most USB cameras provide primitive forms of compression over the USB bus using mostly proprietary algorithms. There are a number of implications for compression, however. First, the quality may be degraded for the application. Second, it may require additional computation cycles in the host to which it is connected. For the Logitech cameras that we are using, the compression ratio from the USB camera is very small so that it is not suitable for wireless network transmission, requiring the data to be decompressed and recompressed into a more network friendly format.

The alternatives to standard USB 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 or would spend significant amount of its power and processing dealing with such devices.

To test 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 I 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: 160×120, 320×240, and 640 $\times$ 480. As shown in the table, the sensors are easily able to capture 160 $\times$ 120 video. This is not unexpected as the total bandwidth required to transmit 160 $\times$ 120 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 for the Bitsy and 12% of the CPU for the Stargate. 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.

Table I. Effect of USB Compression on Frame Rate and System Usage

Image Size	Compression	Bitsy		Stargate	
		Frame Rate	%System CPU	Frame Rate	%System CPU
160 × 120	0	29.64	4.48	30.17	7.48
	1	29.77	22.29	30.14	12.56
	3	29.88	15.71	30.38	9.96
320×240	0	4.88	2.85	4.97	3.51
	1	28.72	67.17	30.76	63.85
	3	29.98	44.50	30.00	42.01
640×480	0	—	—	—	—
	1	14.14	83.66	13.29	99.43
	3	14.73	77.65	14.93	100.00

This table shows the ability of the sensors to capture video from the Logitech web camera and the amount of CPU required for each.

At 320 × 240, 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 160 × 120 stream. We suspect that this is again due to I/O being relatively slow on the video sensor. At 640 × 480, 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 allows us to have control over the algorithms that are used for compression (e.g., nv, JPEG, H.261, or MPEG).

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 Intel's Performance Primitives for the StrongArm and Xscale processors. 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 II, we are able to achieve real-time compression of 320 × 240 pixel video using the Intel Performance Primitives. More importantly, it takes approximately 1/3 or 1/6 the time using the primitives compared with using a non-Intel-specific software optimized algorithm on the Bitsy and Stargate, respectively. As shown in the second row of the table, compressing a larger image scales linearly in the number of pixels. That is, the 640 × 480 pixel image takes approximately four times the amount of time to compress as the 320 × 240 pixel image. Furthermore, we are able to achieve

Table II. Standalone Optimized vs. Unoptimized Software Compression Routines

Image Size	Bitsy		Stargate	
	IPP (ms)	ChenDCT (ms)	IPP (ms)	ChenDCT (ms)
320 × 240	26.65	73.69	20.18	124.55
640 × 480	105.84	291.28	85.82	518.85

This table shows the performance of the sensors compressing a single image repeatedly with no video capture.

Table III. Standalone Optimized vs. Unoptimized Software Capture and Software Compression Routines

Image Size	Bitsy		Stargate	
	IPP (ms)	ChenDCT (ms)	IPP (ms)	ChenDCT (ms)
320 × 240	29.20	80.63	41.05	171.46
640 × 480	115.42	319.71	164.30	725.72

This table shows the additional overhead incurred by the sensor in both capturing and compressing video.

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.

In comparing the two platforms, it appears that the Stargate platform is able to outperform the Bitsy platform using the Intel Performance Primitives but cannot outperform it using the software compression algorithm. We believe that this is due to the fact that (i) the Xscale device has a faster processor and can take advantage of it when the working set is relatively small and (ii) the memory accesses in the Stargate seem to be a little slower than on the Bitsy. Finally, we note that using grayscale images reduces all the figures by approximately 1/3. This is not entirely surprising as the U and V components are one-third of the data in a color image.

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 III 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 for the Bitsy sensor. This scales linearly as we move to 640 × 480, requiring an additional 16 milliseconds per frame. For the ChenDCT algorithm, using either 320 × 240 or 640 × 480 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 system. 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 IV. For the 320 × 240 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,

Table IV. Average Time per Software Component

Software Component	320 × 240 (ms)	640 × 480 (ms)
PWC Decode	16.96	55.99
JPEG Encode	21.08	85.85
Bitmap Compare	4.05	16.45
Image Copy	1.09	6.29
Create Message	0.43	1.27
Other	10.35	30.49

This figure shows the time spent in each of the various components within the Bitsy sensor.

we see that at 640 × 480 pixel video, we are able to achieve on the order of 5 frames a second. Finally, we note that this frame rate is better than IPAQ-based device results for 320 × 240 video data.

For the Stargate sensor, we see that capturing and compressing video adds more overhead to the system when moving doing multiple tasks. From the capture results in Table II to the capture and compression results in Table III, we see that the Bitsy board incurs only additional milliseconds per frame, while the Stargate device nearly doubles its time, adding 20 milliseconds of overhead over the compression only results. We believe that both of these can be explained by what seems to be a slower memory sub-system on the Stargate sensor. The Bitsy-embedded device has an extra I/O processor for all of the I/O, resulting in lower overhead in capturing data.

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., Kravets and Krishnan [2000]). Due to the recent move of the Systems Software faculty from the Oregon Graduate Institute to Portland State University, we were unable to set up our multimeter for testing of the Stargate sensor.

The results of the experiments for the Bitsy board 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 6–10 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,

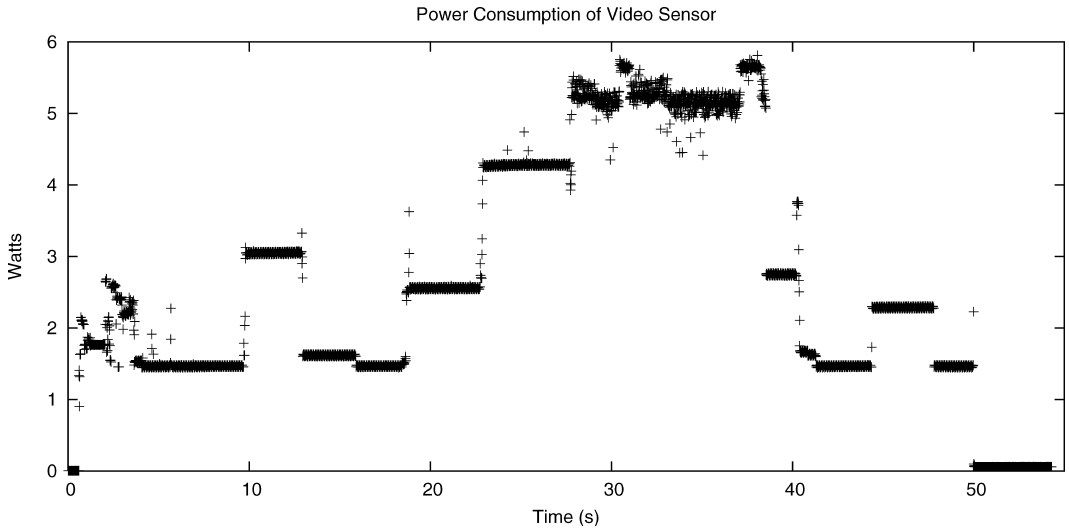


Fig. 8. Power consumption profile.

Table V. Average Power Requirement in Watts

System State	Bitsy Power (Watts)
Sleep	0.058
Idle	1.473
CPU Loop	2.287
Camera with CPU	3.049
Camera in sleep with CPU	1.617
Networking on with CPU	2.557
Camera, Networking, CPU	4.280
Capture Running	5.268

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).

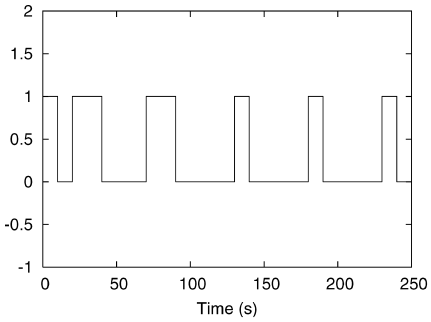
We have summarized the results in Table V. The most important thing to draw from the experiments is that in a given state, the power consumed by the sensor is relatively constant over time. The only exception comes when performing network transmission. As a result, we expect that the algorithms for power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

We suspect that the Stargate sensor will have approximately 1–2 watts less power dissipation than the Bitsy board. This will be entirely attributed to the lower power requirement of the board and the CPU. The networking and the video camera are expected to require the same amount of power.

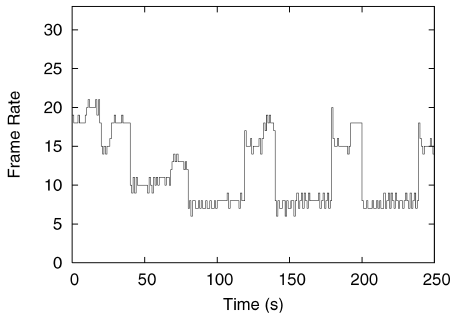
4.5 Buffering and Adaptation

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 high-and low-water marks of 3.8 and 4 megabytes, respectively.

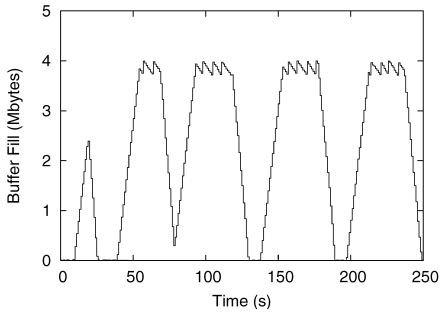
For these experiments, we first turned on the sensor and had it capture, compress, and stream data. The experiment then turned the network card on and off for the times shown in Figure 9(a). The “on” times are indicated by a value of 1 in the graph, while the “off” state is shown as a value of 0. As



(a) Network On/Off



(b) Frame Rate Delivered



(c) Video Sensor Buffer

Fig. 9. Dynamic adaptation example.

shown by Figures 9(b) and 9(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.

Larger video sensor buffers behave similar to the example in Figure 9. The only difference is that a larger buffer allows the system to be disconnected for longer periods of time.

5. RELATED WORK

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

5.1 Video Streaming and Capture Cameras

There are a number of technologies that are available that capture video data and either store the data to the local hard disk or stream the data across the network. For example, web cameras such as the Logitech 3000 camera comes with software to allow motion-activated capture of video data. The camera, however, is not programmable and cannot be networked for storage or retrieval. Other cameras such as the D-Link DCS-1000W are IP streaming video cameras. These cameras capture data and stream it to the network. They were designed specifically for video streaming and capture. Thus, they are not programmable and would not work for situations such as environmental monitoring where power is extremely important.

5.2 Sensor Networking Research

There are a tremendous number of sensor networking technologies being developed for sensor networking applications [Estrin et al. 1992]. From the hardware perspective, there are two important sensors: the Berkeley Mote [Hill et al. 2000] and the PC-104-based sensor developed at UCLA [Bulusu et al. 2003]. 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 result, these sensors are really useful for collecting small amounts of simple information. The PC-104-based sensor from UCLA is the next logical progression in sensor technologies that provides slightly more computing 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.

5.3 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 [Kravets and Krishnan 2000], Noble at the University of Michigan [Corner et al. 2001], and Satyanarayanan at CMU [Flinn and Satyanarayanan 1999]. 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.

5.4 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 traditional streaming technologies focus on the continuity requirements for playback while streaming from video sensors does not have this restriction.

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., Rhee [1998]) or forward error correction based (e.g., Tan and Zakhor [1999]). These techniques can be directly applied to the Panoptes sensor.

6. CONCLUSION

In this article, we have described our initial design and implementation of the Panoptes video sensor networking platform. There are a number of significant contributions that this article 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. In addition, for low-power video sensor, we have discovered that the actual performance of the system involves both the CPU speed and other critical components including the I/O architecture and the memory subsystem. While we entirely expected the Stargate-embedded device to outperform the Bitsy board, we found that its memory system made it slower. The Stargate does, however, consume less power than the Bitsy boards.

Although 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, and computer vision.

ACKNOWLEDGMENT

We would like to thank Dr. Edward Epp of Intel for getting the USB drivers working on the Stargate device.

REFERENCES

BULUSU, N., HEIDEMANN, J., ESTRIN, D., AND TRAN, T. 2003. Self-configuring localization systems: Design and experimental evaluation. *ACM Trans. Embed. Comput. Syst.* (May).

CORNER, M., NOBLE, B., AND WASSERMAN, K. 2001. Fugue: Time scales of adaptation in mobile video. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference* (Jan.).

EKICI, E., RAJAIE, R., HANDLEY, M., AND ESTRIN, D. 1999. RAP: An end-to-end rate-based congestion control mechanism for real time streaming in the internet. In *Proceedings of INFOCOM 1999*.

ESTRIN, D., CULLER, D., PISTER, K., AND SUKHATME, G. 1992. Connecting the physical world with pervasive networks. *IEEE Perv. Comput.* (Jan.), 59–69.

FENG, W., LIU, M., KRISHNASWAMI, B., AND PRABHUDEV, A. 1999. A priority-based technique for the best effort delivery of stored video. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference*. ACM, New York, Jan.

FENG, W.-C, CODE, B., KAISER, E., SHEA, M., FENG, W.-C., AND BAVOIL, L. 2003. Panoptes: Scalable low-power video sensor networking technologies. In *Proceedings of the ACM Multimedia 2003*. ACM, New York, Nov.

FLINN, J. AND SATYANARAYANAN, M. 1999. Energy-aware adaptation for mobile applications. In *Proceedings of the Symposium on Operating Systems Principles*. pp. 48–63.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D. E., AND PISTER, K. 2000. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*. pp. 83–104.

KRASIC, B., WALPOLE, J., AND FENG, W. 2003. Quality-adaptive media streaming by priority drop. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (*NOSSDAV 2003*), June.

KRAVETS, R. AND KRISHNAN, P. 2000. Application driven power management for mobile communication. *Wireless Netw.*, *6*, 4, 263–277.

RHEE, I. 1998. Error control techniques for interactive low-bit-rate video transmission over the internet. In *Proceedings of SIGCOMM 1998*, ACM, New York.

STOCKDON, H. AND HOLMAN, R. 2000. Estimation of wave phase speed and nearshore bathymetry from video imagery. *J. Geophys. Res. 105*, 9 (Sept.).

TAN, W. AND ZAKHOR, A. 1999. Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE Trans. Multimed. 1*, 2 (June).

Received January 2005; accepted January 2005