

Prediction-based Monitoring in Sensor Networks: Taking Lessons from MPEG*

Samir Goel

Department of Computer Science
Rutgers, The State University Of New Jersey
Piscataway, NJ 08854

gsamir@cs.rutgers.edu

Tomasz Imielinski

Department of Computer Science
Rutgers, The State University Of New Jersey
Piscataway, NJ 08854

imielins@cs.rutgers.edu

ABSTRACT

In this paper we discuss the problem of monitoring data sensed in large sensor networks. A sensor typically runs on a battery having a limited lifetime. In order to increase the lifetime of a sensor it is important that the mechanisms used in monitoring them be energy-efficient. In this paper, we propose a new paradigm called Prediction-based monitoring for energy-efficient monitoring. We show that the paradigm can be visualized as a watching of a “sensor movie” and that concepts from MPEG may be applied to it. We have implemented the proposed algorithms in a test bed of Rene Motes [2]. Experimental results show that the proposed solutions cut down the energy consumption by more than 5 times, considerably increasing sensor lifetimes, and thereby, the lifetime of the networks formed from these sensors.

1. INTRODUCTION

With the ever-increasing sophistication of hardware, it is becoming increasingly viable to deploy small units of autonomous nodes for performing various sensing tasks. Cooperative sensing, where a bunch of such nodes are thrown together to cooperate and perform a sensing task, has very useful applications in military and even common day-to-day situations. Some example applications include intrusion detection, gathering ground information like presence of snipers, tanks, etc. in the event of a war, or peak-hour traffic monitoring. cooperative sensing raises several interesting research issues. A majority of these are rooted in the fact that sensors are typically limited by their energy reserves, communication bandwidth, and computational power. Of these, the energy constraint is the most crippling, because if the sensor runs out of battery, there can be neither communication nor computation. Another constraining characteristic that is specific to sensor networks is that the cost (human and others) of managing the sensor nodes far exceeds the cost of the nodes themselves. This is a result of three factors: firstly, the environment in which they are deployed could make human maintenance difficult (for example, consider the scenario where they are scattered over a high-altitude region). Secondly, it is expected that a reasonable-size sensor network will comprise of thousands of nodes. The sheer number of these nodes in a sensor network makes it difficult to manage. Lastly, increasing hardware integration and economy of scale are driving the prices of sensors down. All

*This research work is supported in part by DARPA under contract number N-666001-00-1-8953 and a grant from CISCO systems

these factors make “use-and-throw” a very attractive option, where sensors that either have run out of battery or have failed are simply discarded and more sensors are thrown in to compensate for them.

Given these characteristics, the average lifetime of a sensor determines the cost of “running” a sensor network. Going by the current technology trend, although the computational power in a sensor is expected to follow Moore's law, battery technology is only expected to improve by 2-3% per year [17, 15]. Thus, the only possible way one can increase the lifetime of a sensor is by making use of mechanisms that are highly energy-efficient. Every operation at the sensor, be it transmitting data, receiving data, or performing a computation, consumes some energy. The goal of being energy-efficient, thus, translates into the problem of optimizing the number of these operations that need to be performed. At a sensor, the energy consumed in transmitting a packet is approximately twice the energy consumed in receiving a packet [9]. Also the energy consumed in receiving a packet is an order of magnitude higher than the energy consumed per instruction-execution [8]. Given this relative cost, one may achieve a reduction in total energy consumption by cutting down the number of high-energy operations at the cost of an increase in the number of low-energy operations.

In this paper, our focus is on proposing mechanisms for performing monitoring in a sensor network in an energy-efficient way. Clearly the mechanisms associated with a traditional centralized database paradigm are unsuitable for our purposes. In such a system, a central server maintains a database of readings from all the sensors. Sensors update this server when their readings change. Monitoring operation is supported by the server, which maintains the current state of all the sensors involved in the operation. There are too many messages sent in such a system, making it very energy inefficient in many cases. We make two key observations to significantly improve the energy-efficiency of monitoring operation. Firstly, sensors in close proximity are likely to have correlated readings, and in a majority of the cases, one can predict the reading at a sensor given the knowledge of readings of sensors around it and their past history. An entity (for example, base station, cluster-head) may exploit this observation and predict the set of readings that a sensor is going to see in the near future. These predictions are represented concisely as a “*prediction-model*” and sent to the sensor. The sensor now needs to transmit

its sensed reading to the monitoring-entity only when it differs from the reading given by the prediction model by more than a certain pre-specified threshold. This mode of working gives us a new paradigm of operation in sensor networks. We call this the *PREdiction-based MONitoring (PREMON)* paradigm.

Our second key observation is that a snapshot of the sensor network may be visualized as an (optical) image - the readings of individual sensors correspond to intensity values of pixels in the image. Since a monitoring operation can be thought of as receiving a sequence of these snapshots on a continuous basis, one may visualize monitoring as watching a continuous sequence of corresponding images; in effect, watching a “video of sensed values”. Given this visualization, we explore if the concepts of MPEG [19] (a standard for video compression) may be used for compressing this “video”. We show in section 3 that the MPEG encoder uses a paradigm that is an exact analogue of PREMON. This analogy with MPEG encoding gives us a convenient framework in which to visualize the problem. In addition, it also gives us a unique opportunity to adapt the well-established theory and algorithms of MPEG for use in sensor networks, and to examine the scope for cross-pollination between the two fields.

The PREMON paradigm prevents a sensor from unnecessarily transmitting all the readings that can be successfully predicted at the monitoring entity, thereby saving energy. This saving is obtained at the cost of extra computations at the monitoring-entity for generating prediction-models, and the extra cost of transmitting them. Given this tradeoff, clearly the effectiveness of the proposed paradigm is dependent on the accuracy with which prediction models can be generated and the percentage of readings that can be successfully predicted by them, without too much computational overhead. Based on the above observations, we propose algorithms in this paper and demonstrate their feasibility and effectiveness by implementing them on a test bed of real sensors. We show that it is feasible to generate prediction models for sensors and that significant savings can be achieved even with very simple methods of generating prediction models.

1.1 Background: Brief review of MPEG encoding process

In this section, we review MPEG-2 encoding briefly. Interested readers are referred to [19] for details.

MPEG-2 is a standard for audio and video compression. In this section, we limit ourselves to briefly describing only video compression. The video compression performed by MPEG falls in two classes:

1. *Spatial compression*: In this class of compression, a frame is encoded without reference to any other frame. This is based on the observation that in an image, there are many portions where the pixel value is the same. for example, “sky” in an image would be one such portion.
2. *Temporal Compression*: In this class of compression, a frame is encoded based on previous frames. We focus

on this form of compression in the remainder of this section.

The idea of temporal compression is based on the observation that successive images in a video rarely have significant difference. MPEG exploits this observation by sending only the difference between the current image and the previous image instead of sending the entire current image. The decoder re-constructs the current image by adding the difference to the previous picture.

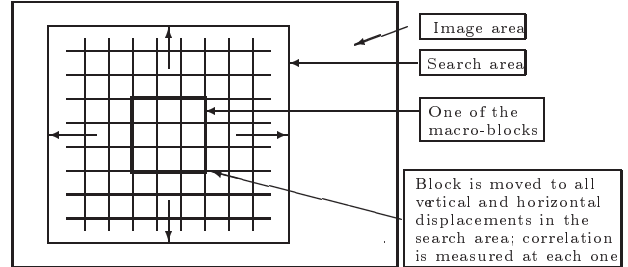


Figure 1: Block Matching Algorithm in MPEG-2

The difference between successive images may become large if there is object movement in video. MPEG deals with this problem using *motion-compensation*. The encoder contains a motion estimator that measures the direction and distance of motion between successive images, and outputs these as *motion-vectors*. The motion-vectors determine the amount of horizontal and vertical shift required. These are sent to the decoder that uses them to shift data in the previous image to more closely resemble the current image. Since motion compensation can never be ideal, the encoder still needs to send an image-difference to take care of any shortcomings in motion-compensation. To compute this image-difference, it makes use of motion decoder logic. When outputting motion-vectors, the encoder also uses them locally in a same way that a decoder will, by feeding them into the motion-decoder logic to produce a predicted picture. This is then subtracted from the actual current picture to produce a prediction error. Thus, the encoder encodes the current picture as a set of motion-vectors and a prediction error (fig. 2).

The decoder takes the previous picture, shifts it with the motion-vectors to recreate the predicted picture and then adds the prediction error to produce the actual picture. This process is shown in Fig. 2. Picture data sent as motion-vectors plus prediction error is called P-frame. The approach of sending a prediction error allows the motion estimation and compensation logic to be imperfect. Thus it allows for simple but inaccurate motion estimations in low-cost systems. If the prediction error is very large, the coder may simply transmit the current picture as it is. This constitutes an I-frame.

In order to generate motion-vectors in MPEG-2, the screen is divided into areas called macro-blocks, which are 16x16 pixels square. Motion vectors are generated for each macro-block. One simple technique used for generating these is called *block-matching* (Fig. 1). In this technique, motion-vector for a macro-block is obtained by moving it around

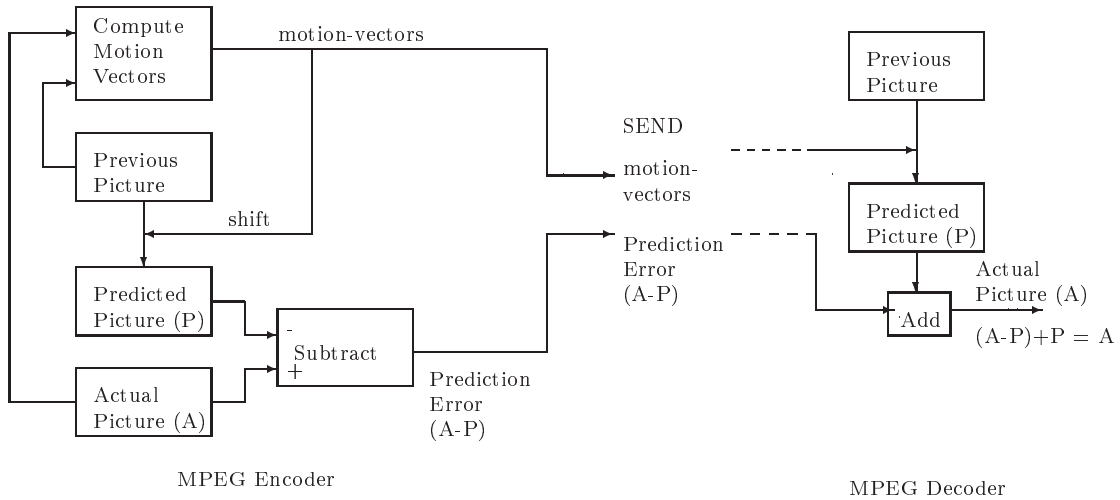


Figure 2: MPEG encoding and decoding process

over the next consecutive picture looking for matching pixel values. When a match is found, the horizontal and vertical displacement needed to obtain it is used as a basis for computing motion-vectors for this macro-block. Thus the location of the boundaries of a macro-block is fixed and a motion-vector does not move the macro-block. Instead the vector tells the decoder where to find pixel data for the macro-block in another frame. The pseudo-code for the block-matching algorithm appears in appendix A.1.

1.2 Organization of the paper

We first describe the model of sensor networks considered in this paper and state our assumptions about it. We then describe the mechanisms based on PREMON paradigm for performing energy-efficient monitoring in sensor networks, in section 3. We have implemented these mechanisms in a test bed of sensors. In section 4, we describe the experiments we have performed and the results we have obtained. In section 6, we present the related work in the field and contrast our approach. In section 7, we briefly describe the scope for future work in this area and present our conclusions in section 8.

2. MODEL

There are many classes of sensor networks [18], each with its own characteristics. Mechanisms meant for one class may not be appropriate for other classes. Before we discuss the problem of optimizing these mechanisms, it is therefore important that we define the class of sensor networks under consideration in this paper.

In this paper, we adopt as a model one of the class of sensor networks described in [18] that is large (in terms of number of sensors), and has a non-deterministic¹ topology. Also, the sensors in this network may be multiple hops away from the nearest wired node. This, according to [18], is the most challenging class of sensor networks.

We assume that the sensors organize themselves in clusters,

¹sensors are thrown randomly in the target field

and that every cluster has a cluster-head². The specific mechanism for achieving this is orthogonal to our focus of supporting energy-efficient monitoring. A possible mechanism that may be used for this purpose has been proposed in [8]. We assume the existence of some such mechanism.

As is typically the case in these networks, we assume that sensors and base stations have limited energy and communicate sensed data over wireless link. Also, we assume that within a cluster, the base station and the sensors use TDMA at the link layer to communicate [8]. The TDMA schedule is decided by the base station, which gets the first slot in the schedule. At the time the cluster is formed, the base station publishes the schedule, thereby communicating it to all the sensors in the cluster. They use it to appropriately switch between active and sleep mode.

We assume that the base stations in the sensor network communicate with the outside world via a few nodes on the wired network. We call these nodes the *access points* (wired base station in the figure) (fig. 3). We assume that every base station is statically pre-configured to communicate via one access point.

We assume that the sensors in a cluster know their location relative to one another. Determining the location of a sensor dynamically in a sensor network is a research area in its own right, and orthogonal to the problem being addressed in this paper. Several promising approaches ([13, 16, 4]) have been proposed to solve this problem. We assume the existence of some such mechanism.

3. ENERGY-EFFICIENT MONITORING

We start by describing a simple scheme for performing monitoring operations in sensor networks. We then describe how we may save energy by employing PREMON. Next, we present the details of the mechanisms based on the PREMON paradigm. Finally, we evaluate these mechanisms with simple experiments and show their effectiveness.

²we use the words cluster-head and base station interchangeably in the paper to refer to the cluster-head

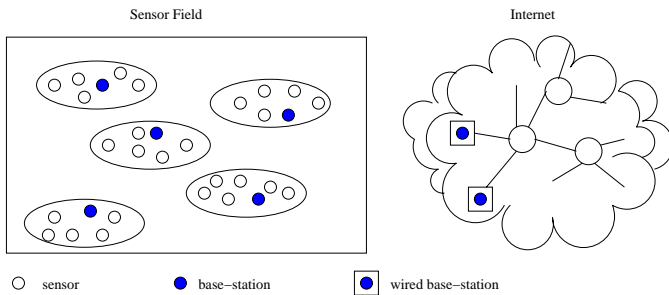


Figure 3: Model of Sensor Network

3.1 A naive approach to monitoring

The simplest approach to supporting monitoring in sensor networks would be to have all the sensors report their sensed readings to base station whenever it changes. We refer to this mode of operation of sensors as “*update-mode*”. The base station collects these updates and sends them to an access point. Thus, at any point, a base station maintains the database of current readings of all the sensors in its cell. Also, the access points collectively maintain a database of current readings of all the sensors in the sensor field. Users interested in monitoring the sensor field may register their interest with the appropriate access points [10]. We refer to this as the centralized approach.

3.1.1 Prediction Based Monitoring (PREMON)

“It is not news if one can predict it”

It has been argued in [14] that a group of sensors (possibly of different types) is more effective in performing a sensing task than one powerful sensor. This argument favors a sensor network with a large number of sensors. In such a sensor network, a group of spatially proximate sensors are very likely to have correlated readings. The correlation may be spatial, temporal, or spatio-temporal. Also, these three types of correlations are not mutually exclusive and they may be present in varying degrees in a sensor network. We aim to exploit this correlation. We believe that in many instances one can predict a sensor’s reading based on the recent history of readings of the sensors around it and based on its own recent history. The sensor need not transmit its reading when it can be predicted by the monitoring entity. This gives us a new paradigm of operation in sensor networks. We call this the PREdiction-based MONitoring (PREMON) paradigm. The basic paradigm is as follows: the base station monitors the reading of sensors and generates *prediction-models*³. It sends these to the appropriate sensors. On receiving a prediction-model sensors change their behavior — instead of sending an update whenever their reading changes, they now send an update only when their readings differ from the one predicted by the prediction model.

3.1.2 Classes of prediction models

The prediction models may be classified into four categories:

- *Spatial*: These prediction models specify the reading

³A prediction-model is a concise representation of the readings that a sensor is expected to sense in the near future

at a sensor as a function of the readings at nearby sensors. Example: “*Reading at sensor X in time slot t is the same as reading at sensor Y during the same time slot*”.

- *Temporal*: These prediction models specify the reading at a sensor as a function of its readings in the past. Example: “*Reading at sensor X in time slot t is 2 greater than its reading in the previous time slot*”.
- *Spatio-temporal*: A combination of the above two. Example: “*Reading at sensor X in time slot t is the same as the reading of sensor Y in the previous time slot*”.
- *Absolute*: These prediction models directly specify the readings that a sensor is going to sense in the near future. The readings may be specified as a list of tuples $\langle \text{time}, \text{reading} \rangle$, or they may be specified by a function. Unlike spatial or temporal models however, the function neither depends on any past reading of the sensor, nor on the reading of any nearby sensor. Example: “*Readings at sensor X in time slots t, t+1, and t+2 will be 32, 34, and 35 respectively*”.

In some cases, one may need to represent spatial, temporal, or spatio-temporal prediction model as an absolute prediction model. This happens when the readings on which future readings depend, are not available at the sensor. For example, if a sensor only stores a limited amount of history, then readings needed to compute a temporal model may not be available. Similarly, if a sensor can hear only a few neighboring sensors, then the readings of sensors needed to compute a spatial model may not be available to it.

A prediction model is always qualified with a time interval in which it is valid. At the end of the time interval, the prediction model expires and the sensor reverts to the *update-mode* described earlier.

3.1.3 Key characteristics of PREMON

This paradigm trades increased computation (for computing prediction models) and slight increase in number of receptions (for receiving the prediction-models) for savings in number of typical transmissions. Note that the cost of transmission is orders of magnitude higher than the cost of computation and is twice the cost of reception [9]. If one is able to generate prediction-models without “too much overhead” and if they are correct a “majority” of times, we can achieve significant savings in energy at the sensors. Thus, performance of any system based on PREMON paradigm is a function of how computationally expensive is the operation of generating prediction-models, and how good is the “predictability” of the sensor readings. “Predictability” of sensor readings increases with the increase in degree of correlation among readings of sensors in close proximity, and with the increase in degree of correlation between recent history and the readings that a sensor is going to see in near future. At the same time, it also decreases with the decrease in the degree of correlation. For example, readings of sensors sensing wind direction, or readings of seismic sensors sensing earthquakes, have bad temporal correlation in general. Fortunately, a number of real world phenomenon like temperature, pressure, and motion, tend to have characteristics

amenable to PREMON. We expect that generating a prediction model that makes precise predictions most of the times will be computationally expensive if not impossible. However, a more modest goal of generating a prediction model, which can predict within a small margin of error, should be achievable without too much computational overhead. Since “some” amount of error in reported data is tolerable in sensor networks (as is “small” delay), we expect the paradigm of prediction-based monitoring to work well in practice. The experimental results presented in section 4 bear this out.

3.1.4 Classes of Solutions

As we mentioned earlier, the correlation in the readings of sensors in a sensor network may be spatial, temporal, or *spatio-temporal*. A wide range of techniques based on PREMON paradigm may be used to exploit each type of correlation. In this paper we focus primarily on exploiting spatio-temporal correlation. Our interest is in approaches that may be used over a wide range of sensor types. At the same time, we realize that the domain knowledge would play a significant role in giving final shape to the specific technique used, based on these approaches.

Spatio-temporal correlation is typically present in sensor networks sensing a moving phenomenon. For example consider a network of motion sensors. When an object passes through the field of these motion sensors, the readings of sensors along its path are very likely temporally correlated. In this paper, we propose techniques to exploit this type of correlation.

3.1.5 A New Framework for Visualizing the Problem

Let’s consider a specific example of a network of motion sensors (for example, magnetometer sensors) in a field. Assume that we are interested in monitoring the motion sensors lying in a small spatial region, $\langle \textit{spatial-region} \rangle$. Assume for the sake of simplicity that all the sensors within the $\langle \textit{spatial-region} \rangle$ fall within one cluster. Also assume that all the sensors operate in the *update-mode*.

When the monitoring operation begins, the sensors in the $\langle \textit{spatial-region} \rangle$ transmit their current reading to the base station. Subsequently, they transmit updates to the base station as and when their readings change. Based on the data received by the base station, one can build an activity-image of the monitored region. Each pixel in this image represents a sensor in the monitored region, the intensity of the pixel indicates its reading, and its position in the image corresponds to its location in the sensor field. When visualized this way, the sensors may be seen as collectively sending a stream of activity-maps of the monitored region, one frame at a time. Since sensors send updates to the base station only when their reading changes, sensors may be seen as collectively sending the full frame initially, and then sending only the differences from the previous frame in subsequent frames. Given this view of the monitoring operation, this whole process is analogous to how video is encoded in MPEG [19] — the initial frame that contains data from all the sensors in $\langle \textit{spatial-region} \rangle$ is analogous to an I-frame in MPEG. The subsequent frames (the difference-frames being sent by base station) are analogous to P-frames in MPEG (more precisely, P-frames with no motion-vectors). Figure 4 illustrates this analogy. In fact, we can improve the anal-

ogy by performing prediction at the base stations of sensor networks.

Thus, a base station may perform motion estimation based on the last couple of frames, and use the computed motion-vectors to predict a few frames in the future. The base station would then send the prediction to the sensors. Sensors now need to transmit only when their reading differs from the predicted value. This mode of operation is very similar to of MPEG. If a significant fraction of the predictions are correct, then substantial energy savings may be achieved at the sensor.

Although we have considered the visualization and MPEG analogy in the context of a network of motion sensors, the proposed framework is very general and it can be applied to a network of any type (or types) of sensors. This is because the algorithms used in MPEG encoding do not make any assumptions about the underlying data, i.e., in some sense they are “blind” to the specifics, and can be applied to any form of image. It is important to note that the readings of sensors need not necessarily be single values like temperature. Each reading may, in general, be a set of values (for example, imagine a network of tiny cameras, where each reading of the camera is an image) and still the same algorithms may be applied. However there are differences between sensor monitoring and MPEG many of which in fact work in the overall favor of PREMON. In the next section we elaborate on these differences.

3.1.6 Differences between MPEG and PREMON

Monitoring operations in sensor networks do not have some of the constraints that are present in video encoding. We summarize the fundamental differences below along with their effect on the algorithms:

- The most prominent difference is that live video encoding is constrained by hard real-time requirements whereas monitoring in sensor networks has soft real-time requirements (of generating the prediction and transmitting it to sensor before the predicted events occur). The hard real-time constraint causes the MPEG encoder to generate motion-vectors based on a small number of previous image frames. Also, it uses very simple algorithms for the purpose. Thus, one may view motion-vectors in MPEG as based on patterns occurring on a very small time-scale. In sensor networks, since the real-time constraint is relaxed, the algorithms may look for patterns not only on small time-scales, but also on medium and large time-scales. Furthermore, the algorithms in sensor networks may afford to spend more time in generating prediction-models.
- In sensor networks, energy of the sensors is the most precious resource. This is not so in video encoding. As a result, the PREMON paradigm attempts to save energy by generating prediction-models at base station. While in MPEG, motion-vectors are a bandwidth-efficient way of encoding the current picture based on the previous picture. Although MPEG tries to be as bandwidth efficient as possible, this requirement is only soft. Whereas, in sensor networks, better bandwidth utilization and extreme energy-efficiency is a requirement.

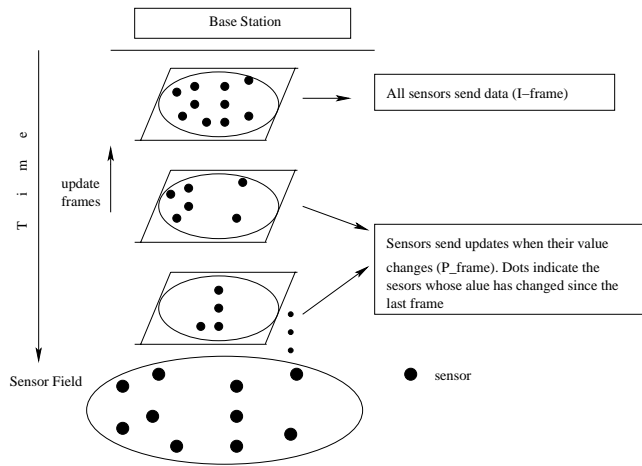


Figure 4: Illustration of MPEG analogy

Thus, there is a need for improving the accuracy of predictions and the frequency with which such predictions are made.

- The resolution of the image that MPEG works on is very high compared to that of the image that PREMON works on. Given the small radio-range of a sensor (approximately 100 feet in outdoor environment), and that all the sensors in a cluster need to be in the radio-range of the base station, we expect that the number of sensors (sensing a moving phenomenon) will be very small compared to the number of pixels in an image frame of a video. This small size significantly reduces the computation involved in performing motion-estimation.
- The frame rate in MPEG is an order of magnitude higher compared to the expected frame rate in PREMON. This allows techniques in PREMON to do motion estimation/compensation based on a set of previous frames.
- Since sensors are rarely hand-placed in the target environment, the arrangement of sensors in the field is going to be random. Thus the image visualization of the sensor network will have non-uniform placement of pixels in the image. On the other hand, the pixels in a video have a very uniform pattern. In section 3.2.6, we discuss techniques to handle this.

3.1.7 Architecture

The architecture of the system is based on the architecture of MPEG encoder. In this section, we describe in detail the processing at the base stations and at the sensors.

Processing at the Base Station

The MPEG-like encoding at the base station involves the following processing steps:

- Collect updates from the sensors. Updates received in the same TDMA cycle form a frame (of the sensor

stream). They are fed into a prediction model generation unit.

- Prediction model generation unit looks at a set of successive frames and computes prediction models. A prediction model predicts the direction of motion for a “macro-block” in the frame for next few time units. This prediction model is sent to the sensors.
- In the next TDMA cycle, the base station assumes that the prediction model predicted readings of all sensors correctly and forms the next frame. The sensors whose readings do not match with the prediction model will send an update, thereby overriding their assumed readings in the frame.
- If sending the prediction model resulted in fewer updates, then the base station encodes the frame as a set of prediction models, followed by the updates necessary to override wrong predictions (prediction errors). It sends this to an access point.

Going by the MPEG analogy, if a base station is low on energy, it may send out “low-quality images” in order to cut down the amount of data being sent. Here, by “low-quality” we mean less accurate information of the sensor readings. For example, the base station may divide its cluster in spatial blocks and may only send average reading of each block to the access point.

Processing at the Sensor

A sensor is in update mode by default - it sends an update to the base station whenever its reading changes. When it receives a prediction-model from the base station, it alters its behavior slightly. For the duration for which the prediction-model is valid, it compares its sensed reading with the reading predicted by the prediction-model. It sends the former to the base station only when it differs from the predicted reading by more than some pre-configured margin of error. When the prediction-model expires, the sensor reverts back to the default update-mode.

Computing a Prediction Model

In this section we discuss the prediction-model generation mechanism in more detail. Although there may be many different ways of generating prediction models, our focus is on using mechanisms that are generally applicable to a wide range of sensor networks.

We use the block-matching algorithm in MPEG to compute prediction models. Note that the position of sensors in a cell is not uniform, and hence, in the corresponding visualized image the pixels will not be uniformly positioned. The block-matching algorithm cannot be applied directly to such an image. In order to correct this problem, one may use the sensor readings to interpolate (or extrapolate) the readings at regular grid points in the image using one of the several standard techniques [6]. As a first cut, we employ a very simple interpolation mechanism. This mechanism assigns the reading of the closest sensor to a grid point. If more than one sensor happen to be closest to a grid point, the mechanism assigns the average of their readings to the grid point. Since the sensors may potentially be unevenly distributed in the region, it is possible that a grid point in the visualized image does not have any sensor “nearby”. We deal with this problem by labeling such a grid point as “transparent”. The significance of such a labeling will become clear shortly.

After interpolation, the visualized image will have pixels at regular intervals. Some of these pixels may be “transparent”. To deal with the “transparent” pixels we modify the standard block-matching algorithm. As in standard block-matching algorithm, the image is divided into macro-blocks. For each macro-block in a frame, the algorithm looks for a match by moving it over the next consecutive frame. If a match is found, it computes the motion-vectors. During this process, we give special treatment to “transparent” pixels. We assume that a “transparent” pixel matches any other pixel. Also, we try to compute the motion vectors for a macro-block only when the percentage of transparent pixels in it is less than a certain threshold. Appendix A.2 gives the pseudo-code for the modified block-matching algorithm. When determining the degree of match between two macro-blocks, the original MPEG algorithm (appendix A.1) would simply subtract the intensity values of the corresponding pixels and sum these up. However, in a sensor network, each reading of a sensor may not be a single value but a set of values. In such a case, a simple subtraction will not work. Instead, we compute the degree of match between the two readings. The way this is computed is dependent on the type of sensor.

In order to generate a prediction model, the base station works with the four most recent frames. It first applies the above algorithm to frames 1 and 2. Let’s assume that this gives us the set, M , of motion-vectors. Each motion-vector in M indicates “motion” (the case of no motion can be seen as a special case of “motion”). For each motion vector in M , the base station checks frames 2 and 3, and 3 and 4, to see if the motion continues and if it can be described by the same motion-vector. If so, we say that the motion-vector “holds”. As an example, consider the four consecutive frames shown in figure 5. The motion vector $\langle dx, dy \rangle$ “holds” for these four frames. If a motion-vector does not hold, it is discarded. If a motion-vector holds, the base station assumes that the motion described by it will continue for the next few frames

in the future, and generates an absolute prediction model based on it. This model contains the readings of sensors in the macro-block of frame 1. Depending on the type of sensors, this information may be encoded in a more efficient form. For example, in our test bed, the magnetometer sensors [2] output a binary value - LOW/HIGH. LOW indicates that no (metallic) object is present in the “vicinity”. HIGH indicates that an object is in the vicinity of the sensor. A macro-block of readings of magnetometer sensors consists of 0s and 1s. In this case, instead of sending the entire macro-block, one can find the largest rectangular block of 1s and send the coordinate of this rectangle (8 bytes of information), thereby substantially reducing the data that needs to be sent. The prediction model would then be valid for only the sensors within the rectangle and not for the entire macro-block.

This model is sent to the geographic area containing the sensors that are going to see the motion in the next four time frames. Note that at any time instant only a portion of the target area is going to see the motion. In order to indicate this to the sensors, we mention the target of the model as of type GEO_TEMPORAL. This allows the sensors in the target region to compute their predicted value correctly.

A special case occurs when a motion-vector indicates “no motion” (i.e., the horizontal and vertical displacement are both zero) and it holds for four most recent frames. This indicates that the readings of sensors in a macro-block are not changing over time. In such a case, instead of sending the entire macro-block, the base station simply sets a flag to indicate that the values are not expected to change for the next few frames. Appendix A.3 and A.4 give the pseudo-code for generating prediction models.

Communicating a Prediction Model

Any prediction model can be communicated to sensors by specifying four components in a message:

- *Type*: Prediction models may be one of the four types:
 - Absolute
 - Spatial
 - Temporal
 - Spatio-temporal
- *Model*: The representation of the model changes with the type. It may be represented as a list of tuples $\langle \text{time}, \text{reading} \rangle$, or as a function (for example, polynomial).
- *Destination*: The destination of the prediction model decides the sensors for which it is targeted. The destination may be specified as a broadcast address (the targets are all the sensors in the cell), sensor id (the target is a specific sensor), or a spatial polygon (the targets are all the sensors that are physically present inside the polygon).
- *TTL*: This field gives the time duration for which the prediction model is valid.

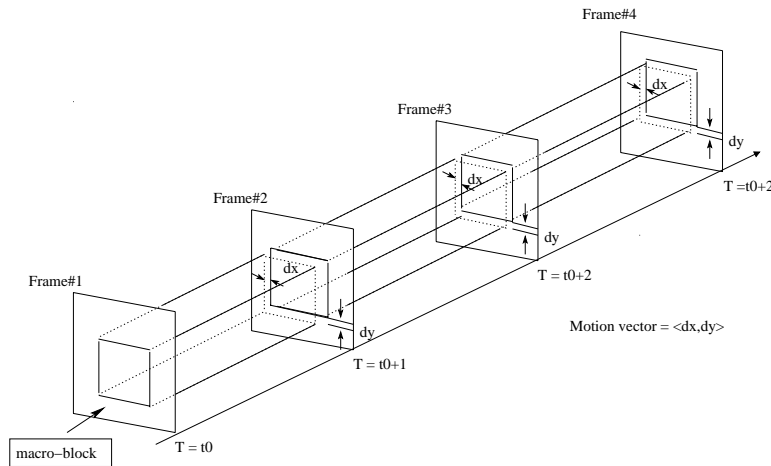


Figure 5: Movement of a macro-block in four consecutive frames

3.1.8 Effect of losses

Packet losses are quite common on wireless links. They are especially common in sensor networks because of the primitive channel encoding used. We briefly look at their effect on the sensors operating in PREMON paradigm.

When a prediction is lost on the wireless link, the sensor continues to operate in default mode. The only effect of this loss is that the sensor will end up transmitting its reading even though it was expected/predicted at the base station. A more interesting case occurs when updates are lost while a prediction is active at a sensor. Suppose that a prediction is active at a sensor and the sensor's reading does not match with the predicted value. This would cause the sensor to send an update to the base station. If this update is lost on the wireless link, the base station would wrongly assume that the sensor's reading conformed with the prediction — a false positive. We believe that this problem is no worse than the one caused by loss of updates (from sensors operating in default mode) on the wireless link, which causes the base station to not know what is going on in the sensor field. Since it is quite common to have group of sensors in close proximity in sensor networks, these problems caused by loss of updates will be controlled by the inherent correlation in their readings. As long as at least one of the updates of sensors from a group monitoring the same phenomenon reaches the base station, we will be able to effectively monitor the sensor field despite channel errors.

4. EXPERIMENTS: SETUP AND RESULTS

In this section we present the results from experiments that we have conducted to evaluate the PREMON paradigm. We first describe the experimental setup.

4.1 Experimental Setup

Our experimental setup consists of a set of light sensors. Each light sensor is fitted on a Rene Mote [2] (fig 6), a tiny computation device (about the size of a quarter dollar) with communication capability. A Mote has a 8-bit micro-processor (ATMEL AVR 90LS8535 running at 4 MHz) and a small flash memory (8KB program memory and 512 bytes data memory). The Motes run a tiny micro-threaded op-

erating system called TinyOS [9]. A set of tools [3] allows the Motes to be extensively and easily programmed using a combination of C language and a component description language. One of the main design criteria for Motes was low power consumption. It draws 19.5 mA of current in active mode⁴ and can run for around 30 hrs on 2 AA batteries. In inactive mode⁵, the Mote draws only 10 μ A of current and can run for a year [9].

A Mote consumes 1 μ J (on an average) for transmitting one bit and 0.5 μ J to receive 1 bit. Furthermore it consumes 0.8 μ J of energy for executing 208 machine cycles (roughly 100 instructions) [9]. A Mote has a radio (RFM TR1000) on-board that works at 916 MHz. The radio can support a maximum bit rate of 19.2Kbps (on/off keying).

The light sensor is not soldered on the Mote. Instead, it sits on a tiny sensor board that can be plugged onto the Rene Mote via a small on-board connector (fig 6). The light sensor measures light intensity and gives a reading between 0 and 7, where 0 represents lowest level and 7 represents the highest level.

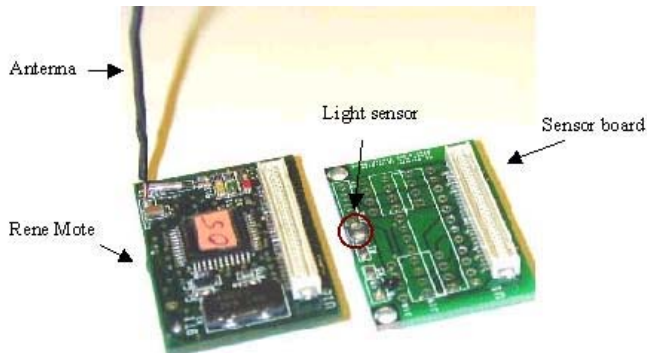


Figure 6: Rene Mote and sensor board

4.1.1 Creating a motion-sensor

⁴in active mode, processor, sensors, and radio are ON

⁵in inactive mode, processor and radio are ON, and radio is listening but not transmitting

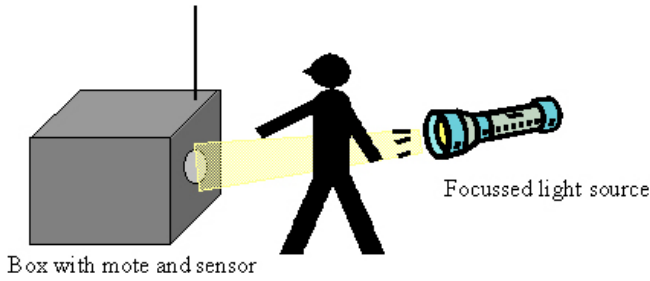


Figure 7: A motion-sensor assembly

We create a motion-sensor for our experiments by using a Mote fitted with a light sensor (we refer to this assembly as *sensor-mote*) and a focused light source. The sensor-mote is placed in a small box in such a manner that light from the pin-hole falls directly on the light sensor. The light source is placed at a distance from the box, with its light falling on the pin-hole (Fig 7).

In order to be able to detect rapid events, the sensor-mote samples the light sensor 8 times per second. However, in order to make efficient use of the channel bandwidth, the sensor-mote transmits the light sensor readings only *once* per second. Each transmission contains the eight most recent readings of the light sensor.

When no activity occurs, the light sensor reads the maximum value. When an object passes between the light source and the box containing sensor-mote, thereby obstructing the light path, the readings of the light sensor changes. Fig 8 shows the typical fluctuation in light sensor readings as an object crosses the light path. These fluctuations are reported the next time the Mote transmits. The next set of readings transmitted by the Mote will report these changes in the light readings.

4.1.2 Setup

Our experimental setup consisted of four motion sensors placed along a busy corridor⁶ (fig 11). Another Mote acts as the base station. All motion sensors report to the base station once every second. Following the PREMON paradigm, the base station analyzes the readings of the motion sensors and generates predictions about future readings of the motion sensors. The motion sensors store the predictions received and transmit their readings only when they do not match with the predicted values.

4.1.3 Generating predictions

For the base-station mote, all four sensors reside in the same cluster. It divides the cluster into four macro-blocks such that there is one sensor per macro-block. It makes predictions based on the last two frames received.

The base station generates two types of predictions: *constant value prediction* and *movement prediction*. We briefly describe each type of prediction and the specific technique used to compute them.

⁶in our case we had placed them on the third floor of our department's building

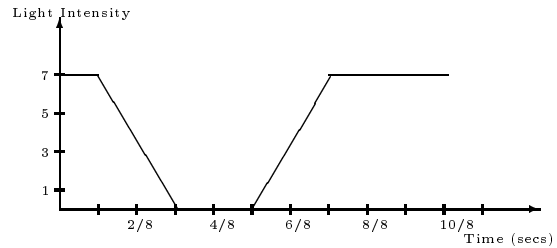


Figure 8: Typical variation in readings of a light sensor when an object passes by

Constant value predictions

These predict that the reading of a sensor-mote is not going to change over time. The base-station mote generates this type of prediction when the readings in a macro-block do not change for two consecutive frames. It always associates a TTL of infinity with this type of prediction. A sensor-mote's reading changes only when an object blocks the light path as it passes by. In our experiments, the periods of activity (from the perspective of a sensor-mote) were very short (a person passing by) compared to the periods of inactivity. This prediction prevents the sensor from transmitting during periods of inactivity.

The packet carrying constant prediction carries two fields: type of prediction and the stable value of the sensor. It is 5 bytes in length (including 3 bytes of header).

Movement prediction

These predict that a sensor is about to see a pattern in its reading. It is computed as follows: The block-matching algorithm used by the base-station considers two motion-vectors $\langle dx=1, dy=0 \rangle$ and $\langle dx=-1, dy=0 \rangle$ corresponding to two possibilities: *motion from left-to-right* and *motion from right-to-left*, respectively. For each macro-block in a frame, it looks for a match in the next consecutive frame by applying these motion-vectors. As soon as a match is found, a movement prediction is generated and sent to the next sensor along the direction of motion.

Note that sensor-motes transmit their readings to the base station at a maximum rate of once per second resulting in a frame rate of 1 frame/sec. However, it is possible that an object takes more than one second to cover the distance between two consecutive sensors. As a result, the base station will see a sequence of zero or more frames reporting no activity (*no-activity* frame) in between two frames with a macro-block reporting (*activity* frame). In order to handle this, the block-matching algorithm considers only the most recently received activity frames, discarding any intervening no-activity frames. However, the time difference between two consecutive activity frames used for generating a movement prediction is not allowed to exceed a pre-defined constant value.

Determining the degree of match

In order to determine if two macro-blocks match (appendix A.2), it is important to specify the method for determining the degree of match between readings of two sensor-motes. Since each transmission of sensor-mote consists of 8 readings, the naive approach of subtracting the two readings

cannot be applied here. We observe that as a person crosses the light path, the readings of the light sensor are going to drop from its stable value, stay at a lower value, and finally, return back to the original stable value (see fig 8). We refer to the string of values between the two stable readings as *pattern-string* (for example, the readings shown in fig 8 contains the pattern string: 3,0,0,0,3). We compute the degree of match between readings of two sensor-mote by computing the difference in the lengths of their pattern-string.

Movement prediction packet

In the movement prediction packet, the base-station mote specifies the interval in which the next sensor (along the direction of motion) is most likely to see the pattern, and the expected length of the pattern-string. The time interval is determined based on the assumption that the person moves at a constant speed. The expected length of the pattern-string is set to the average of the length of the two pattern-strings.

A movement-prediction is said to hold at a sensor if it sees a pattern-string during the specified time interval Δ , and the length of the pattern-string is within δ specified length. Thus, if the specified time interval is $[t1, t2]$, then we allow the pattern to occur in the interval $[t1-\Delta, t2+\Delta]$. Also, if the expected length of the pattern-string is $(l1+l2)/2$, then all pattern-strings with length in the interval $[(l1+l2)/2-\delta, (l1+l2)/2+\delta]$ match with the prediction.

A movement prediction not only specifies the pattern that a sensor-mote is about to see, but it also specifies the stable value that the sensor-mote is expected to read once it has seen the pattern. This exploits the observation that sensor-motes typically see a constant reading after an object has passed by.

The packet carrying movement prediction contains 5 fields: type of prediction, the lower and higher time interval, the predicted length of the pattern, and the expected stable reading of the sensor once the event has occurred. The size of the prediction packet is 8 bytes (including the 3 bytes of header).

4.2 Results and Analysis

We ran two experiments using the setup described in section 4.1.2 (fig 11). The duration of each experiment was half an hour. During the experiments, the sensor-motes and the base-station mote operated in “PREMON mode” (case# 3 below). The table 2⁷ presents the summary of the results averaged over both the runs. Given the preliminary nature of the experiments, the results hold a qualitative significance rather than a quantitative one.

Although the motes operated in PREMON mode during the experiment, we collected enough data to determine their energy consumption for each of the following three cases:

1. Case#1: *All sensor-motes operate in update mode:* In this case, the sensor-motes transmit their readings periodically (once every second). The base station does not transmit any prediction.

2. Case#2: *Base-station mote generates only constant-value predictions:* In this case, while the sensor-motes operate in “PREMON mode”, the base-station mote transmits only constant-value predictions. We consider this case in order to determine the incremental gain of generating movement predictions.
3. Case#3: *Base-station mote generates both constant-value and movement predictions:* In this case, all the motes (sensor-motes and the base-station mote) operate in “PREMON mode”, and follow the techniques presented in this paper.

The summary of results for each of the three cases appear in figure 9. The graph in the figure shows the ratio of energy consumption in case#1 to that in case#2 and case#3.

The energy consumed in the Mote for each of the basic operations is listed in table 1⁸. In all the calculations below, we assume that there are no packet losses.

Operation (Symbol)	Packet Type	Packet Size (bytes)	Per-bit cost ($\mu\text{J}/\text{bit}$)	Cost (μJ)
Transmit (Tu)	Update	11	1	88
Receive (Ru)	Update	11	0.5	44
Transmit (Tcp)	Constant value prediction	5	1	40
Receive (Rcp)	Constant value prediction	5	0.5	20
Transmit (Tmp)	Movement Prediction	8	1	64
Receive (Rmp)	Movement Prediction	8	0.5	32

Table 1: The cost of basic operations in the Mote

Sensor Addr.	Number of Predictions received		Number of transmissions saved due to:	
	Constant Predictions	Movement Predictions	Constant Predictions	Movement Predictions
4	81	60	972	595
8	77	66	924	645
12	86	54	1020	525
14	72	58	864	720

Table 2: Table summarizing the results of experiments (case#3)

4.2.1 Case#1: Default mode

In this mode, the energy consumption at each sensor-mote is easy to compute. Since the experiment runs for 30 mins, each sensor will send 1800 (30x60) updates (at the rate of once/sec) during the experiment. Therefore, the energy consumed at each sensor-mote during the entire run of the experiment is given by:

⁷Sensors in the experimental setup (fig 11) were assigned addresses: 4, 8, 12, and 14, from left-to-right

⁸Table 1 also assigns a symbol for each of the basic operation. We use these symbols in the rest of the paper.

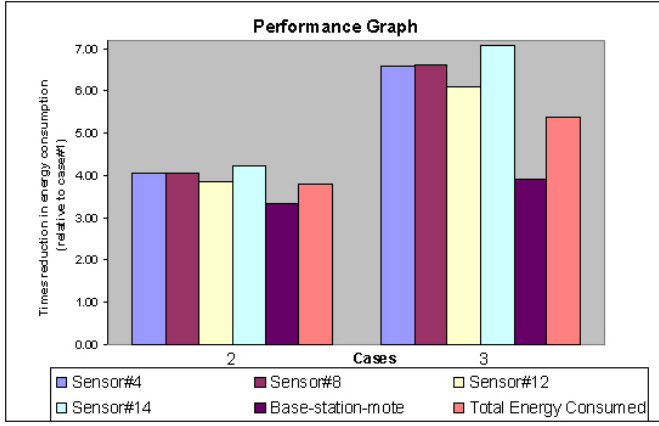


Figure 9: Performance graph showing the energy consumption in cases#2 and case#3 relative to that in case#1. The y-axis shows the times by which the the energy consumption is reduced in case#2 and case#3

$$E_s = 1800 \text{ (updates)} \times T_u = 158.4 \text{ mJ}$$

The base-station mote in this mode receives 1800 updates from each sensor during the entire run of the experiment. Therefore, the energy consumed at the base-station mote is given by:

$$E_b = 4 \times 1800 \times R_u = 316.8 \text{ mJ}$$

The total energy consumed in the system is given by:

$$TE_1 = 158.4 \times 4 + 316.8 = 950.4 \text{ mJ}$$

4.2.2 Case#2: Constant-value predictions only

We show the computation of energy consumption at sensor with address 4. The energy consumption at other sensor-motes can be similarly computed.

Total Energy consumed (TE_4) = Energy consumed in transmitting updates + Energy consumed in receiving predictions

The base-station makes 60 movement predictions. In the absence of these, there will roughly be 60 instances where an object will pass by this sensor (assuming, for simplicity, that predictions were correct). In each of these instances, the sensor will generate 3 updates – one update is generated when it detects a change in reading. The other two are generated because base-station mote does not make a constant value prediction until it has seen two consecutive sets of constant readings. Thus, the sensor will see 141 (81+60) constant value predictions and it will save 415 (595 - 60x3) transmissions of updates.

Thus,

$$TE_{S4} = [1800 - (972 + 415)] \times T_u + 141 \times R_{cp} = 39.16 \text{ mJ}$$

Following similar approach, the total energy consumed at

sensors with addresses 8, 12, and 14 is given by:

$$TE_{S8} = 411 \times T_u + 143 \times R_{cp} = 39.02 \text{ mJ}$$

$$TE_{S12} = 435 \times T_u + 140 \times R_{cp} = 41.08 \text{ mJ}$$

$$TE_{S14} = 396 \times T_u + 130 \times R_{cp} = 37.40 \text{ mJ}$$

The total energy consumed at base-station mote in receiving updates and sending predictions is given by:

$$TE_b = (413+411+435+396) \times R_u + (141+143+140+130) \times T_{cp} = 94.98 \text{ mJ}$$

We assume that the cost of generating each constant-value prediction is negligible.

The total energy consumed in the system is given by:

$$TE_2 = 39.16 + 39.02 + 41.08 + 37.40 + 94.98 = 251.64 \text{ mJ}$$

4.2.3 Case#3: Constant-value and movement predictions

Results for this case appear in table 2. We assume that the energy consumed in computing each movement prediction at the base-station mote is given by C_m .

$$TE_{S4} = [1800-(972+595)] \times T_u + 81 \times R_{cp} + 60 \times R_{mp} = 24.04 \text{ mJ}$$

$$TE_{S8} = [1800-(924+645)] \times T_u + 77 \times R_{cp} + 66 \times R_{mp} = 23.98 \text{ mJ}$$

$$TE_{S12} = [1800-(1020+525)] \times T_u + 86 \times R_{cp} + 54 \times R_{mp} = 25.88 \text{ mJ}$$

$$TE_{S14} = [1800-(864+720)] \times T_u + 72 \times R_p + 58 \times R_{mp} = 22.30 \text{ mJ}$$

The total energy consumed at base-station mote is the sum of the energy consumed in receiving updates, generating predictions, and sending predictions. For the sake of simplicity, we assume for all updates, except ones that result in constant-value predictions, the amount of processing performed by the block-matching code is the same. This is clearly the worst-case assumption. The total energy consumed at the base station is given by:

$$TE_b = (233 + 231 + 255 + 216) \times R_u + (81 + 77 + 86 + 72) \times T_{cp} + (60 + 66 + 54 + 58) \times T_{mp} + (233 + 231 + 255 + 216 - 81 - 77 - 86 - 72) \times C_m = 935 \times R_u + 316 \times T_{cp} + 238 \times T_{mp} + 619 \times C_m = 69012 + 619 \times C_m$$

We can determine the value of C_m to a good approximation as follows: The tools provided with the Motes allow one to generate a map of the Mote's memory. Using this memory map, one can determine the size of any function that is part of the program. We determined that the block-matching function (for computing movement prediction) occupied approximately 1,200 bytes of program memory. In the worst case, every byte will be a separate machine instruction. Thus, the function has 1,200 machine instructions. Since we consider only two motion-vectors ($\langle dx=1, dy=0 \rangle$ and $\langle dx=-1, dy=0 \rangle$), at worse, all the instructions in the block-matching code will be executed twice for ev-

ery update received. Thus, the worst-case cost of Cm is the energy consumed in executing 2×1200 instructions. This is given by:

$$\begin{aligned} Cm &= 2400 \times 0.8/100 \mu J/\text{update} - \text{processing} \\ &= 19.2 \mu J/\text{update} - \text{processing} \end{aligned} \quad (1)$$

With this worst-case computational cost, we can compute the energy consumed at the base-station mote. This is given by:

$$TE_b = 69012 + 619 \times 19.2 = 80.9 \text{ mJ}$$

This represents an upper bound on the energy consumed at the base-station mote. The total energy consumed in the system is given by:

$$TE_3 = 24.04 + 23.98 + 25.88 + 22.30 + 80.9 = 177.1 \text{ mJ}$$

Thus, the total energy consumption in this case is approximately **5.3 times** lesser than one in case#1, and **30%** lesser than one in case#2. We expect case#3 to perform significantly better than the other cases when more sensors are involved.

4.3 When is movement prediction more energy efficient ?

In this section, we establish conditions under which a scheme generating motion-prediction and constant-value prediction (case#3) will perform better than the one generating only constant-value prediction (case#2).

Let us assume that the average success rate for movement predictions is s (expressed as percent of the total number of movement predictions), and let the cost of computing a movement prediction be Cm .

Assume that a motion-sensor is in steady state (i.e., the light sensor reading is constant over time). When an object passes by, the motion-sensor will see a pattern and report this to the base-station. Assuming that the object took less than one second to pass by, the reading of this motion-sensor in the next frame would be its constant stable value. It will report this constant value for two consecutive frames before the base station generates a constant-value prediction. Thus, whenever an object passes by a motion-sensor, it results in three updates being sent to the base station. If the base-station is able to correctly predict the movement of an object, it can save these three transmissions. But in order to predict correctly, it needs to process the readings of two motion-sensors that this object triggered before passing the current motion-sensor. Thus, the total energy consumed in this case is given by:

$$\begin{aligned} TE_3 &= s \times \text{energy consumed when the prediction succeeds} \\ &+ (1-s) \times \text{energy consumed when the prediction fails} \\ &= s \times (\text{energy consumed in sending, receiving, and} \\ &\quad \text{processing updates sent by the two motion} \\ &\quad \text{sensors} \\ &+ \text{energy consumed in sending constant value} \\ &\quad \text{predictions to the two sensors} \\ &+ \text{energy consumed sending and receiving a} \\ &\quad \text{movement prediction}) \end{aligned}$$

$$\begin{aligned} &+ (1-s) \times (\text{energy consumed in sending, receiving,} \\ &\quad \text{and processing updates sent by the two} \\ &\quad \text{motion sensors} \\ &+ \text{energy consumed in sending constant value} \\ &\quad \text{predictions to the two sensors} \\ &+ \text{energy consumed in sending and receiving a} \\ &\quad \text{movement prediction} \\ &+ \text{energy consumed in sending and receiving} \\ &\quad \text{update due to failed prediction} \\ &+ \text{energy consumed in sending constant value} \\ &\quad \text{predictions to the sensor}) \end{aligned}$$

$$\begin{aligned} TE_3 &= s \times (2 \times (132 \times 3 + Cm + (40 + 20)) + (64 + 32)) \\ &+ (1-s) \times (2 \times (132 \times 3 + Cm + (40 + 20)) \\ &\quad + (64 + 32) + 132 \times 3 + (40 + 20)) \end{aligned} \quad (2)$$

In case#2, the total energy consumed would be:

$$\begin{aligned} TE_2 &= \text{energy consumed in sending and receiving updates} \\ &\quad \text{from the three motion-sensors} \\ &+ \text{energy consumed in sending constant value} \\ &\quad \text{predictions to the three sensors} \\ &= 3 \times (132 \times 3 + (40+20)) \end{aligned} \quad (3)$$

In order for case#3 to perform better than case#2, the following condition must be satisfied:

$$TE_3 \leq TE_2$$

From equations 2 and 3, we get,

$$\begin{aligned} &s \times (2 \times (132 \times 3 + Cm + (40+20)) + (64+32)) \\ &+ (1-s) \times (2 \times (132 \times 3 + Cm + (40+20)) + (64+32) \\ &\quad + 132 \times 3 + (40+20)) \\ &\leq 3 \times (132 \times 3 + (40+20)) \end{aligned}$$

Simplifying, we get,

$$s \geq (2 \times Cm + 96)/456 \quad (4)$$

Substituting the value of Cm from equation 1, we get,

$$s \geq 0.294$$

Thus, even if the base station predicts successfully 30% of the time, the scheme with movement prediction (case#3) will perform better than the scheme with just constant-value prediction (case#2).

Solving equation 4 for Cm , we get,

$$Cm \leq (456 \times s - 96)/2 \quad (5)$$

From table 2, we observe that the average success rate is approximately 75%. Substituting this value of s in equation 5, we get,

$$Cm \leq 123 \mu J$$

Thus, as long as we consume less than $123 \mu J$ of energy in processing an update, the scheme with movement prediction (case#3) will perform better than the scheme with just constant-value prediction (case#2). Given that 100 instructions consume $0.8 \mu J$ of energy in the Mote, $94.5 \mu J$ is enough to perform 15,375 instructions. This, we believe,

is more than enough to generate movement predictions to a good accuracy.

5. QUALITY OF MONITORING

The constraints on the quality of the answer to a monitoring query required by a user govern the amount of energy savings that may be achieved. The quality of answer is controlled by two parameters:

1. *Error tolerance*: Controls the amount by which the received answer may deviate from its actual reading
2. *Delay tolerance*: Controls the interval from the time a sensor's reading changes to the time it is received at the user.

The more the error tolerance, the less precise the prediction-models need to be, and hence, more is the potential for saving energy. In the extreme case when the error tolerance is infinite, the sensors do not have to transmit anything. Also, the more the delay tolerance, the more time is available to generate prediction-models. This increases the potential for generating close-to-perfect prediction models at the base station, which, in turn, increases the potential for saving energy.

6. RELATED WORK

Researchers in sensor networks have looked at various ways of saving energy of sensors. In particular, [12, 8, 7] have proposed exploiting redundancy in sensed readings of sensors that are in close proximity. However, the approaches proposed in [12] rely on sensors to figure out among themselves the redundant part in their sensed information and suppress it from transmission.

We see three problems in this approach. Firstly, it will take N transmissions for a group of N sensors to figure out all the redundant information, which is costly and may outweigh the advantage of removing redundancy. Secondly, the approach works well when the sensed readings are simple (like temperature, pressure, etc.). In cases when the sensed readings are complex (like image from a camera, assuming a network of cameras [1]), it is not clear how one may determine the redundant part given the limited computational power at the sensors. Thirdly, at the very best, this approach only minimizes the redundancy in information being sent. It may work well in sensor networks like network of temperature sensors where the sensed reading is not expected to vary much in a small spatial region. However, the assumption of redundancy is not true in general. Consider a network of motion sensors. Assume that the reading at a sensor (for example, magnetometer sensor) is a function of its distance from a moving object. In such a case, the readings of sensors in close proximity will not be that same but will be correlated. We distinguish between redundancy and correlation. We say that readings at two sensors have *redundancy* if there is a part that is common to both. For example, if two temperature sensors in close proximity sense 65F as the temperature, we say that there is redundancy in their information. Although the word "correlation" has many connotations, for the purpose of this paper, we say that the readings at two sensors are *correlated* if given the reading of one sensor one may derive the reading at another

sensor with reasonable accuracy. In other words, whenever the reading at one sensor is a function of the readings at the other sensor we say that two sensors have correlated readings. One may think of redundancy as a special case of correlation where one may derive reading at a sensor by employing an identity function (i.e., $f(x) = x$). Returning back to our example of sensor network of motion sensors, if the reading at a motion sensor is a deterministic function of its distance from the closest moving object, then readings at two sensors A and B are correlated (not redundant). The reading at A may be derived given the reading at B, the location of two sensors, and the location of the object. Lastly, this approach only talk about eliminating redundancy in readings sensed in the same time frame. In our proposed scheme, we specify mechanisms for identifying correlation in readings across time frames and try to eliminate the need for transmitting these if they can be derived from older readings. We expect this to translate into significant savings in energy.

In [8], authors deal with the problem of how to self-organize sensors that are thrown together, so that a meaningful sensing task can be accomplished. The paper prescribes compressing information at the nodes acting as cluster-head, before sending to a central station. However, the paper does not specify any mechanism for performing this compression operation. Also, this approach does not prevent sensors from sending redundant information.

In [11] the emphasis is on being able to "route" data sensed at sensors to where the interest is. It does not deal with cutting down the information being relayed.

7. FUTURE WORK

We plan to explore the following future directions:

1. *Learning in PREMON*:

We plan to explore how one can incorporate learning patterns of movement based on history, and use them to guide the prediction-generation logic.

2. *Duty Cycles*:

A possible way of reducing the energy usage in a sensor network is by reducing the duty cycle of sensors that are not seeing frequent activities. Thus, we would like to partition a sensor network into regions based on the level of activity they see. For sensors in these different types of areas, the duty cycle requirement is clearly different. We plan to look at protocols for dynamically partitioning a sensor field into these different areas and adjust the duty cycles of sensors appropriately to achieve near-optimal energy usage.

3. *Extracting patterns at larger time-scales*:

We observe that MPEG encoding algorithms only look at patterns that occur in smaller time-scales. We believe that one may improve the accuracy of the predictions by extracting patterns that occur frequently at medium and larger time-scales and using them to improve predictions on smaller time-scales. As an example, consider a sensor network consisting of acoustic sensors thrown over a battlefield. As a tank approaches an acoustic sensor, the sensor will record higher

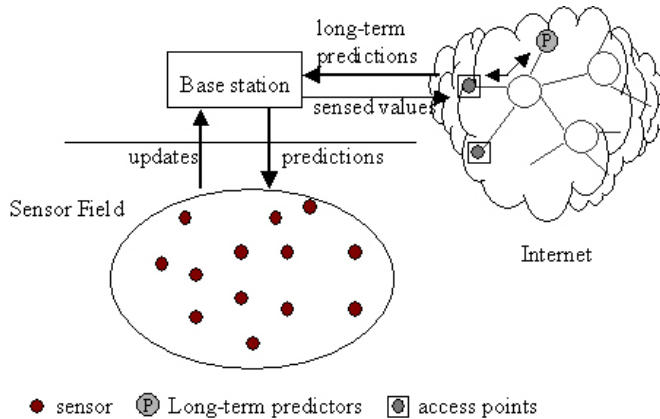


Figure 10: Architecture of the system making use of long-term patterns

amplitude in the sensed reading. The amplitude of the sensed reading will reach a peak value and will start decreasing as the tank passes by. If one were to look only at pattern on a smaller time-scale, by seeing a steady increase in amplitude of the reading, one would predict that the amplitude of the sensed reading would continue to rise. Similarly, by noticing a steady decline in amplitude, one would predict that the amplitude of the sensed reading will continue to decrease. However, by processing the data over a longer time period, one would be able to extract the signature pattern (steady rise followed by steady decline) of a tank. If this information is given to the base station, it will be able to identify the pattern just by looking at the steady increase in amplitude. It would then predict not only the steady rise in amplitude but also the steady decline following that. Assuming that the signature pattern is unique enough, this would substantially reduce the number of prediction misses.

Extracting patterns on larger time-scales however can be a computationally intensive task. We assume that a powerful wired node in the wired network will do this processing. The architecture of envisaged system is illustrated in figure 10. The sensed reading are mined by a “long-term predictor” for patterns that recur at larger time scales. These long-term patterns are given to the base station. The base station uses them as input in its prediction process.

We are investigating two approaches for extracting long-term patterns. One makes use of data-mining techniques and the other is based on compression theory (interested readers are referred to [5] for details). We intend to evaluate the effectiveness of these mechanisms.

4. Predicting aggregates:

In this paper, we have looked at predicting values of sensors. However, in many instances one is interested in the aggregate value over a small spatial region (e.g., what is the average temperature in DataMan Lab). The challenge is to apply the PREMON paradigm to predict the values of aggregates.

5. Simulations:

We are in the process of performing simulations to analyze the performance of the proposed algorithms over large sensor networks.

8. CONCLUSIONS

In this paper, we have presented energy-efficient mechanisms for performing monitoring operations in sensor networks. For energy-efficient monitoring, we have proposed a new paradigm called Prediction-based monitoring (PREMON). It exploits the spatial, temporal, and spatio-temporal correlation likely to be present in readings of spatially proximate sensors. We have shown that this paradigm is analogous in many ways to the one followed by MPEG encoder. This close analogy allows us to adapt the theory and algorithms of MPEG for use in sensor networks. We have proposed an algorithm for generating prediction models based on the block-matching algorithm in MPEG. We have implemented the proposed algorithm on a test bed of sensor-motes. The results of our experiments have shown that utilizing the PREMON paradigm helps in significantly cutting down the number of transmissions from the sensors, thereby achieving substantial energy savings.

9. ACKNOWLEDGEMENTS

We would like to acknowledge the help of Vihari Komaragiri, Sudeept Bhatnagar, Anirudha Bohra, and Rahul Pupala, for their insightful comments to the earlier drafts of this paper. We would also like to thanks the anonymous reviewer whose comments helped us in raising the quality of this paper.

10. REFERENCES

- [1] Forest of sensors project. <http://www.ai.mit.edu/projects/vsam/>.
- [2] TinyOS: An operating system for networked sensors. <http://tinyos.millennium.berkeley.edu/>.
- [3] Tools for programming rene motes. <http://tinyos.millennium.berkeley.edu/release/tos-latest.tar.gz>.
- [4] L. Doherty, L. E. Ghaoui, and K. S. J. Pister. Convex position estimation in wireless sensor networks. In *Proceedings of IEEE INFOCOM, Alaska*, April 2001.
- [5] S. Goel and T. Imieliński. Prediction-based monitoring in sensor networks: Taking lessons from mpeg. Technical Report DCS-TR-438, Rutgers University, June 2001.
- [6] R. Harding and D. Quinney. *Simple Introduction to Numerical Analysis : Interpolation and Approximation*. Adam Hilger Ltd, 1989.
- [7] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, Banff, Canada, October 2001.
- [8] W. Heinzelman, A. Chandrasekaran, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor network. In

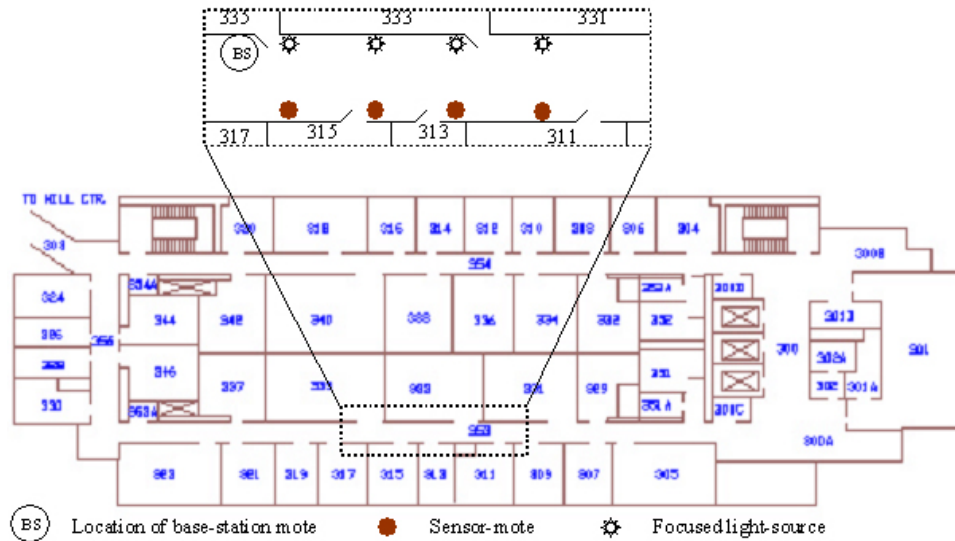


Figure 11: Floor plan indicating placement of motion-sensors

Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00), January 2000.

- [9] J. Hill, R. Szweczyk, A. Woo, S. Hollar, and K. P. D. Culler. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.
<http://tinyos.millennium.berkeley.edu/papers/tos.pdf>.
- [10] T. Imielinski and S. Goel. Dataspace – querying and monitoring deeply networked collections in physical space. *IEEE Personal Communication Magazine, Special issue on "Networking the physical world*, pages 4–9, October 2000.
- [11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '00)*, Boston, August 2000.
- [12] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, Seattle, WA, 1999.
- [13] D. Niculescu and B. Nath. Ad-hoc positioning system. Technical Report DCS-TR-435, Rutgers University, April 2001. To appear in the Proc. of IEEE Globecom, November 2001.
- [14] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of ACM*, 43(5), May 2000.
- [15] J. Rabaey. Silicon platforms for the next generation wireless systems - what role does reconfigurable

hardware play? In *Proceedings FPL 2000, Austria*, August 2000.

- [16] A. Savvides, C.-C. Han, and M. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '01)*, Los Angeles, August 2001.
- [17] M. Srivastava. Energy efficient wireless systems. In *Submitted for publication in DIMACS Summer School on Foundations of Wireless Networks and Applications*, August 2000.
- [18] L. Subramaniam and R. Katz. An architecture for building self-configurable systems. In *IEEE/ACM workshop on Mobile Ad Hoc Networking and Computing (MobiHOC '00)*, Boston, August 2000.
- [19] J. Watkinson. *MPEG-2*. Local Press, 1999.

APPENDIX

A. PSEUDO-CODE

A.1 Block matching algorithm in MPEG-2

```

/* Each frame can be thought of as composed of
 * macro-blocks. Size of each macro-block is
 * blockWidth x blockWidth pixels.
 * The function is passed two consecutive
 * frames: frame1 and frame2. For each
 * macro-block in frame1, this function
 * searches for the best-matching block in
 * frame2 (two macro-blocks are supposed to
 * match if the sum of the difference in
 * their corresponding pixel values is less
 * than a certain pre-specified threshold).
 * Once a matching block is found, the function
 * records the displacement (in X and Y)
 * as a motion-vector that describes the

```

```

* movement of the macro-block from frame1 to
* frame2. The set of all motion vectors
* considered are assumed to be stored in the
* array: PossibleMotionVectors
*/
function blockMatchingAlgo(frame1, frame2,
    blockWidth) returns Motion-Vectors

```

```

foreach mblock in frame1 do
{
    topLeftX1 = mblock.getTopLeftX();
    topLeftY1 = mblock.getTopLeftY();

    error = 0; minError = MAXINT;
    foreach mVector in PossibleMotionVectors do
    {
        <topLeftX2, topLeftY2> =
            applyMotionVector(topLeftX1, topLeftY1,
                mVector);
        if (isWithinFrame(topLeftX2, topLeftY2,
            frame1) {
            error =
                subtractMacroBlocks(frame1,
                    topLeftX1, topLeftY1, frame2
                    topLeftX2, topLeftY2, blockWidth);

            if (error < minError) {
                bestMatchingVector = index;
                minError = error;
            }
        }
    }
    if (minError < ErrorThreshold)
        addNewMotionVector(motionVectors,
            topLeftX1, topLeftY1, mVector);
}
return motionVectors;

```

```

/* This function finds the degree of match
* between the macro-block, starting at
* <topLeftX1, topLeftY1>, of frame1, and
* macro-block, starting at <topLeftX2,
* topLeftY2>, of frame2. Each macro-block is
* assumed to be of size-
* blockWidth x blockWidth pixels. Function
* abs() gives the absolute value.
*/

```

```

function subtractMacroBlocks(frame1,
    topLeftX1, topLeftY1, frame2, topLeftX2,
    topLeftY2, blockWidth)
    returns degree-of-match

```

```

row1=topLeftX1; col1=topLeftY1;
row2=topLeftX2; col2=topLeftY2;
error = 0;

while (row1 < row1+blockWidth) {
    while (col1 < col1+blockWidth) {
        error = error + abs(frame1[row1][col1]
            - frame2[row2][col2]);
        col1++; col2++;
    }
}

```

```

    row1++; row2++;
}
return error;

```

A.2 Block-matching algorithm used in PRE-MON

```

/* This function is a variation of the original
* block-matching algorithm of MPEG.
* The difference is that a macro-block in the
* image visualization of sensor data can
* have transparent pixels. If their number
* exceeds a threshold
* (TransparentPixelThreshold) then the macro-
* block is not processed
*/

```

```

function newBlockMatchingAlgo(frame1, frame2,
    blockWidth) returns Motion-Vectors

```

```

foreach mblock in frame1 do
{
    if (getNumTransparentPixels(mblock) >
        TransparentPixelThreshold) continue;

    topLeftX1 = mblock.getTopLeftX();
    topLeftY1 = mblock.getTopLeftY();

    error = 0; minError = MAXINT;
    foreach mVector in PossibleMotionVectors do
    {
        <topLeftX2, topLeftY2> =
            applyMotionVector(topLeftX1, topLeftY1,
                mVector);
        if (isWithinFrame(topLeftX2, topLeftY2,
            frame1) {
            error =
                newSubtractMacroBlocks(frame1,
                    topLeftX1, topLeftY1, frame2,
                    topLeftX2, topLeftY2,
                    blockWidth);
            if (error < minError) {
                bestMatchingVector = index;
                minError = error;
            }
        }
    }
    if (minError < ErrorThreshold)
        addNewMotionVector(motionVectors,
            topLeftX1, topLeftY1, mVector);
}
return motionVectors;

```

```

/* This function finds the degree of match between
* the macro-block, starting at <topLeftX1,
* topLeftY1>, of frame1, and macro-block,
* starting at <topLeftX2, topLeftY2>, of frame2.
* Each macro-block is assumed to be of size -
* blockWidth x blockWidth pixels. Function
* degreeOfMatch() gives a quantitative
* measure of the degree to which two pixel values
* match each other.

```

```

*/
function newSubtractMacroBlocks(frame1, topLeftX1,
                               topLeftX2, frame2, topLeftY1,
                               topLeftY2, blockWidth)
    returns degree-of-match

row1=topLeftX1; col1=topLeftY1;
row2=topLeftX2; col2=topLeftY2;
error = 0;

while (row1 < row1+blockWidth) {
    while (col1 < col1+blockWidth) {
        if (frame1[row1][col1].type() ==
            TRANSPARENT ||
            frame2[row2][col2].type() ==
            TRANSPARENT)
            continue;
        else
            error = error +
                degreeOfMatch(frame1[row1][col1],
                              frame2[row2][col2]);
        col1++; col2++;
    }
    row1++; row2++;
}
return error;

```

A.3 Prediction-model generation algorithm

```

/* This function takes a set of four consecutive
 * frames and computes motion-vectors that holds
 * for all these frames. It assumes that these
 * motion-vectors will hold for next 4 frames
 * (in future), assuming constant speed of
 * movement. Based on this assumption, it
 * translates the motion-vectors into
 * prediction-models.
 */
function generatePredictionModels(frames [],
                                blockWidth)
    returns Prediction-Models

validMotionVectors =
newBlockMatchingAlgo(frames[0], frames[1],
                    blockWidth);

foreach vmVector in validMotionVectors do {
    topLeftX1 = vmVector.topLeftX;
    topLeftY1 = vmVector.topLeftY;
    mVector = vmVector.mVector;

    <tempTopLeftX, tempTopLeftY> =
        applyMotionVector(topLeftX1, topLeftY1,
                          mVector);
    <topLeftX2, topLeftY2> =
        applyMotionVector(tempTopLeftX, tempTopLeftY,
                          mVector);

    error = newSubtractMacroBlocks(frame[0],
                                    topLeftX1, topLeftY1, frame[2],
                                    topLeftX2, topLeftY2);
    if (error < ErrorThreshold) {
        <topLeftX3, topLeftY3> =
            applyMotionVector(topLeftX2, topLeftY2,

```

```

                                mVector);

        error = newSubtractMacroBlocks(frame[0],
                                        topLeftX1, topLeftY1, frame[3],
                                        topLeftX3, topLeftY3);
        if (error < ErrorThreshold) {
            newModel =
                translateVectorToModel(frame[0],
                                       vmVector, blockWidth);
            addNewModel(newModel, predictionModels);
        }
    }
}
return predictionModels;

```

A.4 Pseudo-code for translating motion-vectors into prediction models

```

function translateVectorToModel(frame, vmVector,
                                blockWidth)
    returns Prediction-model

mVector = vmVector.motionVectorIndex;
topLeftX = vmVector.topLeftX;
topLeftY = vmVector.topLeftY;
newModel = NULL;

if (mVector.displacementX == 0 &&
    mVector.displacementY == 0) {
    /* Send a temporal model */
    newModel.type = TEMPORAL;
    newModel.model = {FUNCTION, f(x, t) = f(x,
                                             t-1)};

    newModel.destination = {GEO, topLeftX,
                            topLeftY, topLeftX+blockWidth,
                            topLeftY+blockWidth};
    newModel.TTL = 4; /* valid for next 4 time
                       units */

    return newModel;
}
/* Send an absolute model */
macro_block = extractMacroBlock(frame, topLeftX,
                                topLeftY, blockWidth);
newModel.type = ABSOLUTE;
newModel.model = {ABSOLUTE,
                 size=blockWidth*blockWidth, macro_block};
newModel.TTL = 4; /* valid for next 4 time units */

destPolygon = computeDestPolygon(topLeftX,
                                  topLeftY, mVector, newModel.TTL);
newModel.destination = {GEO_TEMPORAL, destPolygon};
return newModel;

```