Efficient Pattern Detection in Extremely Resource-Constrained Devices

Michael Zoumboulakis
School of Computer Science and Information Systems
Birkbeck College
University of London, WC1N 3QS
Email: mz@dcs.bbk.ac.uk

George Roussos
School of Computer Science and Information Systems
Birkbeck College
University of London, WC1E 7HX
Email: gr@dcs.bbk.ac.uk

***Abstract*—We present a novel approach for the on-line detection of Complex Events in Wireless Sensor Networks. Complex Events are sets of data points that correspond to unusual patterns that can not be detected using threshold-based techniques. Our method uses an efficient implementation of SAX, a mature data mining algorithm, that transforms a stream of readings into a symbolic representation. Complex Event Detection is then performed via four alternative modes: (a.) multiple pattern detection using a suffix array, (b.) distance-based comparison, (c.) unknown pattern detection, and (d.) probabilistic detection. The method allows users to specify complex events as patterns or to search for interesting changes without supplying any information. The appropriateness of the approach has been verified by applying it to four sensor data sets. In addition, we have developed an efficient implementation for the TinyOS operating system, and further validated our assertions by collecting and analyzing data in real-time.**

I. Introduction

One of the most common uses of Wireless Sensor Networks (WSNs) is to monitor the physical environment using a variety of sensors that largely depend on the goal of the application. Typically such sensors include temperature, humidity, light, pressure, acceleration, vibration, air quality, and so on. Monitoring can be passive or reactive; in the former model readings are sent to a location outside the sensor network where they are aggregated, processed and stored for future use. The alternative is for time-sensitive response applications that need to react *if* and *when* certain conditions are met. Mining sensor data is essentially the process of finding interesting and/or unusual patterns in sensor readings. Often, in this reactive scenario mining has to be performed as close to real-time as possible, depending on how strict the requirements of the application are. This mode of operation is conceptually close to Event-Condition-Action (ECA) rules in conventional database systems where an action has to be executed in response to an event and one or more satisfied conditions.

Implementing the reactive model in desktop and server class machines is well-understood. However WSNs introduce some unique challenges with significant effects in system and application design. Extending the useful lifetime of a WSN [1] is one such challenge that perhaps carries the most weight in terms of importance. The lower-end type of WSN that we consider comprises devices that are extremely resource constrained. In particular, power is the most scarce resource that often imposes to the application designers the need to regulate the usage of other components such as CPU, radio, sensors, external flash and so on. A classic example of this is the radio: sending a single bit over the radio can consume the same energy as executing approximately 1,000 CPU instructions [2]. It is therefore clear that sending every reading over the radio to a base station for processing and out-of-network mining will significantly shorten the lifetime of the WSN. One way to address this is to perform as much of the computations as possible locally and only communicate with neighbors and the outside world when it is absolutely necessary.

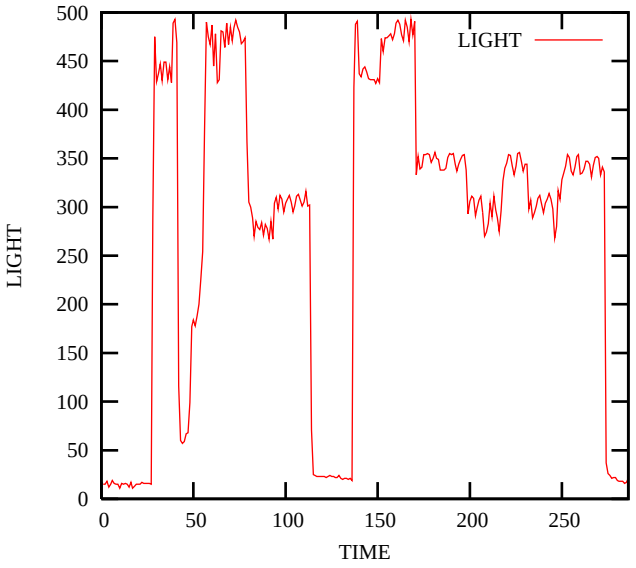
The target platform considered is the lower-end WSN comprised of nodes such as the TMote Sky [3]. This type of node is equipped with an ultra-low power Texas Instruments MSP430 F1611 16-bit RISC micro-controller. Additional components such as storage and sensors can be added through the nodes' expansion port. The hardware offers 10Kb of RAM, 48Kb of Flash and is powered by a single pair of AA batteries. Lifetime can range from over a year to a few hours depending on how heavy the load on the resources is. To achieve prolonged lifetime many applications enter low-power modes by switching off or switching to stand-by components that are not used. Spending as much time as possible in low-power modes is therefore an important goal for almost any WSN application.

A. Problem Statement

A Complex Event is an interesting or unusual pattern in the collected data that can not be captured by threshold-based techniques. Figure 1 shows such a complex event defined by 286 light sensor readings. The pattern is caused by switching the lights on and off and the variations in the brightness are induced by a dimmer. Readings below 50 Lux indicate a poorly lit room and readings above 300-350 Lux indicate a well-lit room. Although this example is oversimplified, it aims to show that such a pattern can be extremely difficult and inefficient to detect using thresholds.

For the remainder of this paper we will use the terms *pattern* and *Complex Event* interchangeably. Note that our definition of a Complex Event is somewhat different to the notion of Complex Event Processing (CEP); CEP aims to

Fig. 1. Example of a complex event: dimmer-induced variations in the brightness of light in a room. We assume that the variations in the light compose the pattern. Even this simple pattern would be difficult to detect using traditional threshold-based techniques.



process multiple individual events. We consider a Complex Event a single conceptual entity that can not be broken down to or described by individual events.

Complex Events are inherently difficult and in extreme cases even impossible to describe. In some applications users wish to be informed of *interesting* changes but are able to describe what constitutes interesting using only fuzzy and ambiguous constructs. For instance, users of a soil moisture monitoring application such as [4] are interested in rain. But they are not generally interested in common rain; they would prefer to be informed when it rains in a interesting and unusual way, e.g. rain that can produce flash floods. It is difficult to capture this requirement in a program using constructs such as `if` statements. Detection can be even impossible if the event has not been witnessed yet. Hence a requirement was non-parametric event detection. In this scenario WSN nodes *learn* from a portion of data that is known to be normal and they are then able to detect events that do not fit in the learned state of affairs. We will describe exactly how this is done in later sections.

Another requirement for Complex Event Detection (CED), is the case where the user is interested in a large number of events. Even if the events can be described using programmatic control flow, having a large number of conditional statements and function calls is inefficient and wasteful. Bear in mind that the sampling frequency of the data collection may be determined by the application and this defines additional requirements. A sample of the sensors once every second allows for approximately one second — minus the time it takes to sample the sensors — to perform other operations such as CED, radio communication, logging of readings and so on. We therefore need an approach that scales well as the number of events a user is interested in increases.

Some example applications that share the above requirements to different extents include monitoring scenarios where events are relatively infrequent and complex in nature — this *boredom punctuated by panic* modus operandi is characteristic of a full class of WSN applications. Some example applications that could benefit from the reactive model proposed in this paper include wildfire detection [5], vineyard environmental conditions alerting [6], oceanography networks [7] for tsunami detection [8] and out-patient activity alerting [9], [10], [11].

The performance benefit of altering an application from passive monitoring and out-of-network processing to reactive alerting is substantial. Components such as the external flash chip for logging data and the radio for communication are inherently slow and power-hungry. Opting to use them only if interesting patterns are detected can prolong the lifetime of the network by an order of magnitude. The aim is to support this goal by providing a framework for efficient CED.

B. Our Contribution

To perform efficient CED, we have adapted Symbolic Aggregate ApproXimation (SAX) [12], [13], [14], [15], [16], [17], a mature and efficient approach that is used for traditional data mining tasks, to the unique requirements of WSNs. SAX converts a set of sensor readings to a string and CED is then performed by operating directly on the strings instead of the sensor time-series data. We introduce an efficient implementation of SAX using integer scaling that makes the application of the technique feasible in WSNs. In addition, we developed a toolbox of efficient CED methods that include:

1) *Many-pattern Detection with a Suffix Array* — essentially we are binary searching a pruned suffix array structure for the occurrence of a pattern,
2) *Exact or Approximate Matching* — where the pattern is known beforehand,
3) *Non-parametric Detection* — where no pattern is specified in advance and sensor nodes *learn* from a training portion of the time series known to be normal, and
4) *Probabilistic Detection with a Markov Chain* — again this relies on training on a portion of data known to be normal; from this character frequencies and transition probabilities are built. These are used to flag highly improbable sequences as events.

We provide an experimental evaluation of the above methods and present insights learned from the implementation process and running time analysis. All the code for the approaches discussed in this paper has been implemented for nodes running the TinyOS [18] operating system.

II. Efficient Symbolic Conversion

SAX is used for the symbolic conversion of sensor readings. Essentially SAX maps one or more readings $r_n \in \mathbb{Z}$ to a letter from a finite alphabet Σ according to a set of well-defined rules. Formally the input is a set of readings

$R = \{r_1, r_2, \ldots, r_n\}$ and the output is a set of letters $S = \{s_1, s_2, \ldots, s_m\}, s_i \in \Sigma$ and $0 < m \leq n$. In this respect SAX is another quantization method.

Once the conversion is complete we have the set S which we refer to as the *string*. CED is then performed on this string, essentially reducing the detection problem to a search and match. Each sensor node can be thought of as a process that outputs text and therefore CED is analogous to finding patterns of interest within that text.

In a different system, the search for one or more patterns can be performed offline which would be straightforward. But in WSNs radio communication is expensive and for this reason we aim to limit the usage of the radio component by enforcing nodes to do the conversion locally and on-line. This means that events are detected in-the-network and very close to their physical occurrence location. In particular, in this section we deal with the complexity that this fact introduces. We address the unique constraints of the WSN by aggressively optimizing every operation of the algorithm.

During the conversion process, we choose an appropriate compression ratio (i.e. $m < n$) for the time-series data. Different applications carry different information content which we learn empirically in our experiments over various data sets. Data sampled at frequency higher than 10Hz is best discretized using no compression. In contrast data such as humidity and temperature sampled at low-frequencies can be discretized with a compression ratio of $2:1$ or even $4:1$ without significant information loss.

Some knowledge of the nature of the time-series is therefore essential. This is a reasonable assumption given that WSNs are not general-purpose machines and applications are developed to achieve a very specific and well-defined goal. During application development, which is often intertwined with hardware engineering, data can be collected and processed in order to make a judgment about the information content and the choice of the appropriate compression ratio. One way of doing this is converting the data using different compression ratios and evaluating the impact to CED accuracy.

The logical steps taken by SAX are: (a). initialization, (b). normalization (z-standardization) of the time-series, (c). construction of a Piecewise Aggregate Approximation (PAA) and (d.) construction of string from the PAA representation. We refer readers not familiar with the PAA representation and the SAX algorithm to one of the SAX papers [14] that explains the approach in more detail. Figure 2 shows the process; the dotted-line box represents alternative detection options that they share one common goal: they take the string as input and attempt to determine whether this string represents an unusual or interesting set of data points.

Having provided an overview of the algorithm, we now turn our attention to our efficient implementation of SAX and its suitability for the WSN setting. Our first implementation attempt was a simple line-for-line port of SAX from MATLAB to nesC, the programming language used by TinyOS. The symbolic conversion relied on numerous floating-point operations, and since our target platform has no Floating-Point Unit (FPU) on board, the resulting performance of the code was rather disappointing. The various floating point operations performed in software slowed down a single conversion to approximately 113ms and this time was almost impossible to improve without removing the floating point operations. The importance of reducing active time stems from the fact that there is a direct relationship of CPU active time and power consumption and for the specific CPU used, idle time reduces current consumption by up to a factor of 33.

Fig. 2. Steps followed in a symbolic conversion. Dotted lines represent alternative choices for CED.

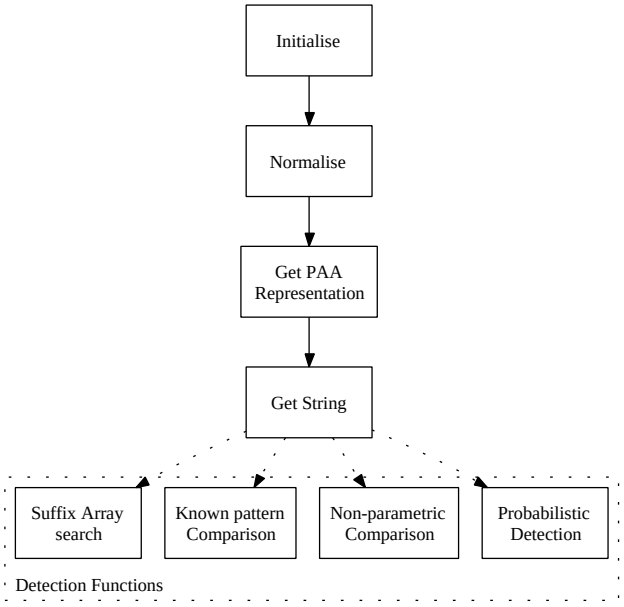


TABLE I
PERFORMANCE TIMES FOR SYMBOLIC CONVERSION.

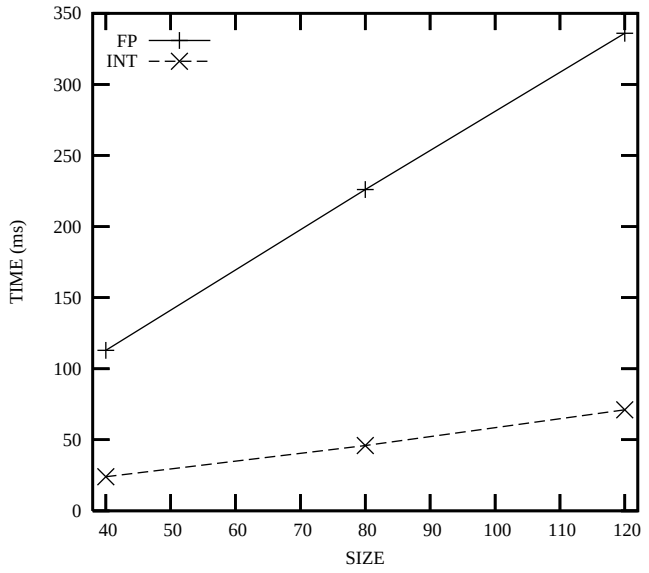
	Size	
Operation	*40*	*80*
FP time (ms)	113	226
FP Power Consumption (mA)	251.74	448.98
Int time (ms)	24	46
Int Power Consumption (mA)	96.39	134.79

In order to improve the slow performance of the algorithm we targeted the floating-point operations and decided to replace them with integers. There are a few approaches to arithmetic approximation for chips without FPU, including binary scaling, fixed-point and modular arithmetic. We use the integer scaling approach, although we relax the requirement that the scaling factor needs to be a power of 2.

One of the trade-offs that we had to settle is whether for the sake of simplicity and performance to sacrifice a little precision. Since we are eliminating the need for floating-point precision by scaling, and at the same time we avoid using arbitrary large storage types such as bit vectors, we sacrifice in terms of accuracy.

The idea behind scaling, is multiplication by an integer which is a power of 2 and then, if necessary, division by

Fig. 3. Comparison of integer approximation against floating point conversion. The SIZE attribute refers to the input size of the time series to be converted by SAX.



a power of 2. Care has to be taken so the "multiply" stage produces no integer overflow which would lead to undefined behavior.

As a simple example of scaling consider the way TinyOS implements timers: a `Timer<TMilli>` interface provides millisecond accuracy by assuming 1 second equals 1,024 milliseconds. In this manner the floating point number 1.875 seconds can be represented by an `int` with the value 1,920 milliseconds. Thus by sacrificing a little accuracy for the sake of performance big savings can be made in current consumption.

Ensuring overflow is avoided is essential therefore we plan and provide for this in our source code. This is straightforward to implement since we know the maximum sensor reading value beforehand which is 4095 for the 12-bit Analog-to-Digital Converter (ADC) of the TMote. For instance when we calculate the sum of the readings vector, we know that as long as the sum stays below 2^{64} no overflow can occur. The largest built-in type supported by the compiler is 8-byte long unsigned 2^{64} which allows for readings vectors with sizes well over 10^6 — a lot more than our application would need. Similar planning logic applies for other calculations and for unsafe code that was not guaranteed to stay within arithmetic bounds, we apply necessary operations such as division. A good example of this is the PAA calculation or the integer square root.

The integer-based approximation of SAX reduced the time for a single conversion from 113ms to just 24ms (Table 1), with a corresponding significant reduction in power consumption. The "Size" attribute shown on the table and on Figure 3 refers to the input size which is the length of the time-series data that we pass on to SAX for symbolic conversion.

In terms of accuracy we tested the approximation method using a sample from the uniform distribution. The results deviated slightly from results obtained by the same algorithm with floating-point operations running on a desktop PC. Empirically, the arithmetic mean of the differences was 0.0857 while the median of the differences was much smaller at 0.0026. These differences are taken by comparing strings using the SAX distance metric that lower-bounds the Euclidean distance.

Approximating SAX using integers instead of floating-point numbers, allows for more efficient operation by the WSN nodes. The CPU idle time is increased and the gain is a significant power saving that enables us to perform a range of other functions such as task the sensors, write to or read from the external flash chip, use the radio if necessary, and so on. Most importantly, further power savings can be gained by reducing radio communication. There is no need anymore to report or log every single reading. The radio component can be switched off and only periodically power up to transmit interesting patterns in the text that denote unusual activity in the monitored phenomenon.

III. Detection Functions

Up to this point we described *how* we obtain the string using SAX. Now we shift focus on *what* we use the string for and *how* we perform Complex Event Detection with it.

We provide four different modes of detection which we will describe in more detail.

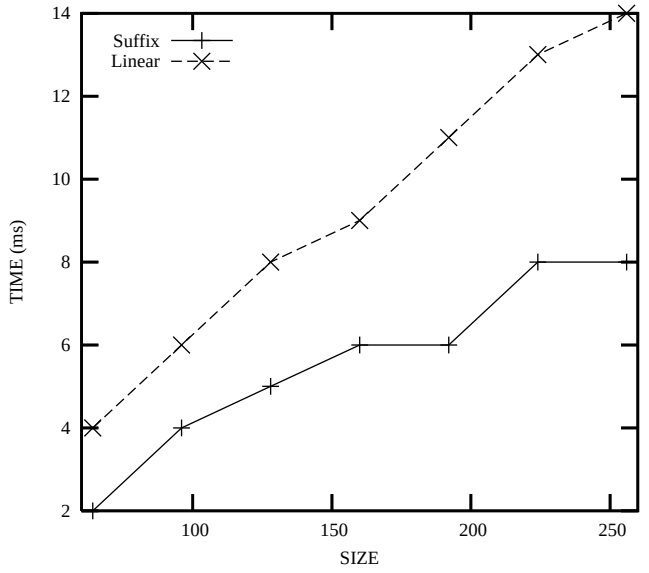
A. Fast Multiple-pattern Matching with a Suffix Array

When a large numbers of patterns, each representing a complex event, is known in advance and matching is performed by searching against this database of patterns, an efficient technique is needed that scales well as the cumulative size of the patterns grows. The output of the symbolic conversion algorithm is always fixed in size and is equal to the size of readings array divided by the compression ratio. While this is sufficient for the case where the user specifies a pattern that equals this fixed size, it scales poorly if a user wishes to supply a pattern of arbitrary size or many patterns of fixed or arbitrary size.

As an example consider a user who is interested in 12 different events described by patterns that are 8-character long each. We now have a choice of either matching the output of the symbolic conversion to each one of these 12 patterns or concatenating the 12 patterns making sure each one is terminated i.e. by appending $\backslash 0$ so there are no spurious matches at the end of one string and beginning of another.

It is clear that individual matching scales poorly as the number of patterns increases. So concatenation and linear search used to determine whether the string S generated by symbolic conversion of the sensor readings exists in the text T submitted by the user, is a better option and the problem is now equivalent to the well-documented substring matching problem. The linear search algorithm has a worst-case running time of $O(nm)$ where n is the length of the text T and m is the size of the sensor-generated string S. This is without any

Fig. 4. Comparison of suffix array binary search against linear search. The SIZE attribute refers to the cumulative size of the concatenated patterns (each pattern representing a complex event).



pre-processing and without using any of the available exact matching algorithms such as the Boyer-Moore or the Knuth-Morris-Pratt [19].

But it turns out there is a better and much faster way. A data structure that is used extensively for fast string matching is the suffix tree that can be used to locate a substring in the tree in $O(m)$ time, assuming that Σ character comparisons take constant time. We have decided to use a similar data structure called the Suffix Array [20] for the following reasons:

- It is three to five times more space efficient,
- It can be used to quickly find every occurrence of a substring in the text at a time competitive and in some cases slightly better than that of suffix trees,
- It is a good fit for TinyOS — the most popular WSN platform — that offers no Dynamic Memory Allocation and thus would make the suffix tree construction wasteful since we would have to pre-allocate space for the maximum possible number of descendants at every node.

The optimal cost of searching for a substring in a suffix array is $O(m+\log n)$.

A suffix array is defined as follows [19]: given an n-character string T, a suffix array for T called Pos is an array of the integers in the range of 0 to $n-1$, specifying the lexicographic order of the n suffixes of the string T.

The steps we follow to build a suffix array for the patterns submitted by the user are: (i.) we first build a suffix array for each pattern, (ii.) we then merge the results dropping duplicates, and (iii.) finally we prune the array by removing small suffixes. For instance, if the minimum user-supplied pattern has a length of 5 or over then any suffix with size smaller than 5 can be removed from the structure.

To search for a substring using the suffix array we use binary search thus giving the algorithm an optimal time of $O(m\log n)$. The key in the search is that if S occurs at all in T then all the locations of those occurrences will be grouped consecutively in Pos. Using the binary search we start at the middle position of the suffix array, that is $Pos(\lfloor \frac{n}{2} \rfloor)$; if S is lexically smaller than the string in the middle position, then S will be in the upper half.

The true behavior of the algorithm depends on how many long prefixes of S occur in T. If very few long prefixes of P occur in T then it will rarely happen that a specific lexical comparison will actually take $\Theta(m)$ time and therefore the bound $O(m\log n)$ is rather pessimistic [19]. The expected time of the algorithm is much closer to $O(m+\log m)$, and in any case this time bound can be guaranteed by augmenting the data structure with longest common prefix (Lcp) information. This effectively accelerates the binary search by reducing the number of character examinations to at most one per iteration.

Another practical trick [21] used to accelerate the binary search is to "unroll" the loop expression. The positions that the binary search will examine are known in advance. So therefore the binary search loop can be "unrolled" by hard-coding those positions.

For the sake of simplicity the version of suffix array we implemented is basic. The binary search of the data structure is performed without any tricks to accelerate it. The array is built using insertion sort and since we only build the structure once at start-up, the construction cost is not that important. The search performance carries much more significance since it is performed at every timer tick i.e. every time a string is generated by the streaming sensor readings, we search for that string in the pruned suffix array structure.

The results of a comparison between binary searching the suffix array and performing a linear search are shown in Figure 4. The *Size* attribute refers to the size of T, the cumulative size of the concatenated patterns, and the *Time* attribute refers to the time it took a WSN node to perform each type of search. We can see that as that size grows the linear search becomes increasingly slower. None of the test cases represent a worst-case for the linear search. The worst-case is when S is not found at all in T. In addition there are plenty more performance statistics comparing a suffix tree to a suffix array at [20].

With this detection method the user can specify arbitrarily large numbers of patterns. Our search is both space and time efficient, scaling well as the number of patterns increases.

B. Exact and Approximate Matching

In this section we introduce the case where a user is either interested in one type of pattern or generally interested in unusual patterns. In the former case the pattern supplied is converted to a string using the symbolic conversion process outlined earlier and submitted to the nodes of the WSN either at the pre-deployment phase or dynamically during WSN operation. In the latter case if no pattern is supplied the nodes spend a fraction of time training with data known to be normal.

They can then use the distances observed during training to deduce what may constitute an unusual pattern.

1) Known Pattern: In this case each string generated by the sensor node is compared against the known pattern supplied by the user. The comparison itself is distance-based and the SAX distance function used lower-bounds the Euclidean distance [12].

If the distance is zero it means either that the two strings are equal or that the difference between the two strings is in only one character i.e. $dist(a, b) = 0$. It is worth noting that assuming zero-distance for adjacent characters is the default setting but it can be changed in an application-dependent manner. The distances look-up table can be very easily adapted so that adjacent characters have distance greater than zero. This may be attractive for applications with greater detection sensitivity requirements or if it has to be determined whether two strings are identical.

The distance-based function allows for approximate event detection since an explicit distance from the given pattern can be specified. Furthermore, other sophisticated techniques can be used on a per-application basis to augment the brute-force matching. One of the advantages of using string representations is that application developers benefit from a huge library of string algorithms that are known to produce good results in other fields.

Alternatively, other metrics such as edit distance, city block distance, and so on, can be used. We have experimented with a selection of metrics including the ubiquitous Euclidean distance, global and local alignment and so on, but we found that the SAX distance metric is sufficiently accurate and fit for the purpose of CED.

2) Unknown Pattern: Another perhaps more interesting mode of operation is when the pattern is not supplied in advance i.e. non-parametric event detection. In this mode, we train one or more nodes with a set of training data that is known to be normal. During the learning phase, a pointer to the maximum distance between temporally adjacent strings is recorded. At the end of this phase the maximum distance effectively becomes the *learned* distance and it is then used to decide whether or not an event has occurred.

A benefit of using this Machine Learning inspired approach is that we can decide when to increase the sensors' sampling frequency on-the-fly thus achieving *Dynamic Sampling Frequency Management* (DSFM). If the maximum distance set during learning phase is exceeded, then we can make a decision to double or simply increase the timer sampling frequency. This is an important feature since it allows nodes to do less work during times of relative calm and then sample faster during periods of interest. If we take as an example application the prediction of the onset of a forest fire, during periods of inactivity we can choose to take one temperature reading every few minutes. If readings indicate the possibility of a fire then we can dynamically increase the timer frequency without user intervention.

DSFM reduces the active CPU time of sensor nodes and therefore complements the general objective of power savings and improved longevity. The added benefit of DSFM is that nodes deployed in hostile and unreachable terrains can now make autonomous decisions about when to increase their sampling frequency instead of communicating with stations located far away over expensive links.

C. Probabilistic Detection

With symbolic conversion, we have a discrete process that continuously produces characters from a finite alphabet. We can view this process as a Markov chain where the set of states equals the size of the alphabet. Formally the set of states S is $S = \{a, b, c, \dots\}$ and $size(S) = \Sigma$ where Σ is the alphabet.

If the process outputs x_i at time t and then it moves to x_j at time $t + 1$, the probability for this move is represented by p_{ij} and we formally say that we had a state transition from state i to j. The process can also remain to the state it is in with a probability p_{ii}, that is when the current character in the string is the same as the previous.

To encode all the possible transitions, we create and populate a square matrix called the transition matrix. We ignore starting states and we assume that no absorbing states exist. For simplicity, we assume a Markov chain of order one however this is not a strict requirement; higher order (memory) Markov chains can be used depending on the type of data and application at hand.

We then use the Markov model built during the learning phase to predict path probabilities. A path probability can be thought of as a realization of a Markov chain as a path in time through its state space [22]. So a path probability for path $(x_1, x_2 \dots x_t)$ is simply $P(X_1, X_2 \dots X_t) = (x_1, x_2 \dots x_t) = P(X_1 = x_1)p_{x_1x_2}p_{x_2x_3} \dots p_{x_{t-1}x_t}$. Probabilities for strings that are very close to 0 can be flagged as events, as the individual transitions in the string are highly unlikely, given the data segment we have used to build the transition matrix.

The same approach can be used for predicting states in dense deployments: for instance if we assume an over-simplified scenario of three nodes, a, b and c we can then have a and b outputting characters and probabilistically determine the characters for c while c is in sleep mode. We can then alternate with the remaining nodes in the network, thus saving energy.

Finally, our Markov model implementation is approximated using integer-based code. For example we also apply the integer scaling technique in this case so the row probabilities of the matrix sum up to 1024 instead of one.

IV. Experimental Results

To verify the appropriateness of SAX for CED we have subjected it to various data sets and event types. This process also helped in gaining an insight for the information content of the different data sets which in turn is valuable for determining the one-shot initialization parameters — such as SAX input size and compression ratio — needed for successful detection.

The first data set we used [23] is from a WSN of 54 nodes deployed at the Intel Lab at Berkeley. This set contains approximately 2.3 million readings of temperature, humidity, light

Fig. 5. Approximate pattern matching: suppose that the user supplies the pattern shown by the dotted line and requests to be informed of patterns similar to the one submitted. The algorithm finds a similar pattern — a pattern that minimizes the string distance. This is the simplest type of search since it requires one string comparison at each step and no further processing.

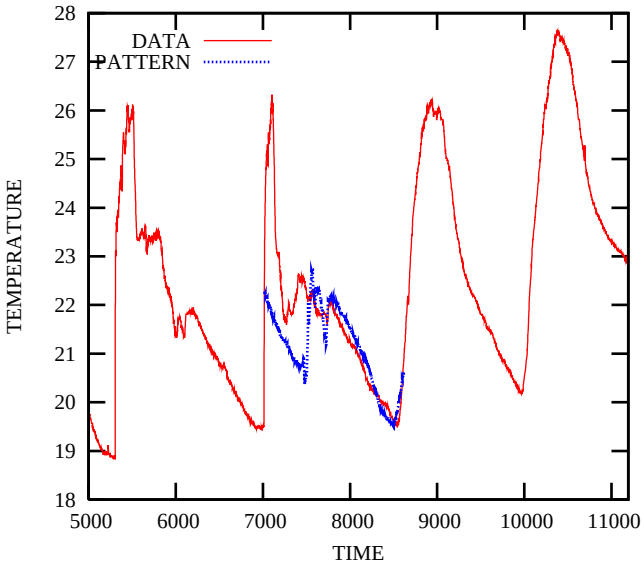
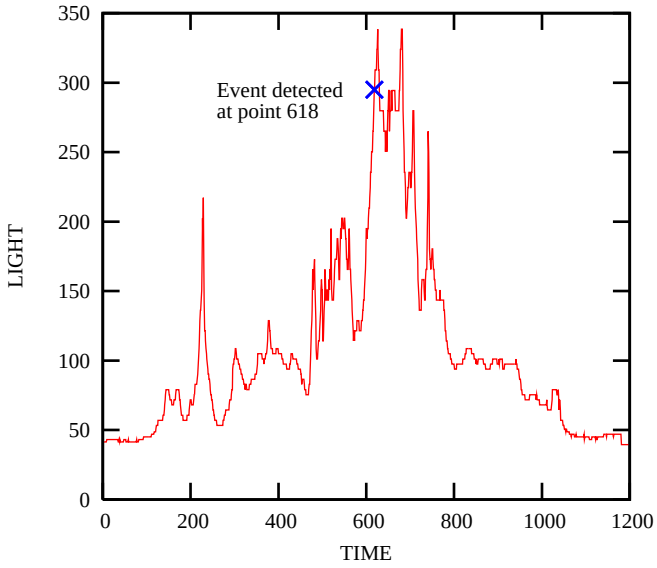


Fig. 6. Unknown pattern detection: suppose that a user is interested in a type of event but is unable to describe it. The machine learning approach uses a learning phase to acquire distance information. Then high distances are flagged as events as shown as by the spike near point 618 that is sufficiently unusual compared to the rest of the data to be flagged as an event.



and voltage sampled at a frequency of approximately 2Hz. There is a certain element of noise in the data mainly from failing sensors. It is importance to stress that in performing CED we are not interested in outliers such as faulty readings. Thus it was a requirement of the technique to be able to successfully disregard outliers.

Synthetic complex events were planted at different part of the set and the ability of the algorithm to successfully detect those events was evaluated. The synthetic events were constructed by either picking values from a uniform distribution or by altering the values by a minimum of 15 % deviation. The output of this exercise were some useful default values for parameters. For instance we have found that we are unable to detect events that are described with less than 10 data points, using an input size of 28 points or larger. By varying the input size from 16 points up to 836 we were able to establish that a useful value for this parameter is 40 points. The setting of 40 points is sufficient to successfully detect complex events between 10 points and 40 points. In addition, if the input size is approximately twice the size of the event then complex events are detected with nearly 100 % accuracy. Similarly a good choice for alphabet size is 10 after testing ranges between 5 and 20. As expected, mean and median distances grow proportionately to the alphabet size increase.

We have tested exact matching, approximate matching and unknown pattern matching. Figure 5 aids in visualizing an example of approximate matching. This temperature data is from node 4 of the Intel data set and it corresponds to the week of 1st of March to the 7th of March 2004. The pattern shown in dotted blue line is a segment from the 18th of March. The position of the match is shown on the graph and is the best match by using SAX or Euclidean distance. Figure 6 shows an example of finding the most interesting data segment, in light readings from node 1, when no pattern is supplied. After training on the first third of the set, the point 618 was identified as the start of the pattern with the higher distance, which is again correct as points 600-800 have a high percent deviation compared to the rest of the data. Similar to the example of Figure 1, this pattern indicates somebody switching the lights on for a brief moment.

We have also explored the performance of this technique on high frequency ECG data obtained from the UCR Data Mining Archive [24]. The two cases we tested were a normal ECG turning to Super Ventricular and normal turning to Malignant Ventricular. Detection was non-parametric and in both cases the exact point of change was identified giving 100 % accuracy. No compression was used for this data set since it carries a high information content.

From the same archive we have used the set named “playing” which contains accelerometer data from a Sony ERS-210 Aibo Robot. This data was sampled at 125Hz and it is 3-dimensional. We used the first third of the data as training set and a 4:1 compression ratio. We were able to detect the most interesting change in the set (point 536).

Due to space restrictions we will not review the experiments in further detail here; we refer the interested reader to our previous work [25].

Lastly, a demo was designed and presented where one or more sensor nodes train using light readings in a room. Complex events are induced using variations of the brightness caused by a flash light over the light sensor similar to the earlier example given in Figure 1. The technique used was

able to demonstrate successful real-time CED running on WSN nodes. Further experiments to the same effect with different sensor data such as accelerometer, acoustic and so on were performed using a multi-sensor [26] plug in. Finally the robustness of the code has been verified by running non-stop for a month and processing approximately one million temperature readings.

V. Related Work

Event notification is important in WSNs and complex events are commonplace in a large number of applications. However the research to address CED does not completely reflect their importance.

One of the first papers that presented an event-based Approach was Directed Diffusion [27]. In that approach a node would request data by sending interests which are conceptually similar to subscriptions in a publish/subscribe system. Data found to match those interests is then sent towards that node. This addresses individual points of interest in the data but did not offer the ability to describe more complex patterns. A different framework that is based on event classification, is the on-line state tracking [28] approach. This technique consists of two phases: the first phase is the learning process where new sensor readings are classified to states and the second phase is the on-line status monitoring phase during which nodes are collaborating to update the overall status of the network. This is interesting work in a sense that it moves away from individual nodes' readings and views the whole network as a state machine. Conceptually it is closer to our work since it allows for more complex conditions in order to move from one state to another.

Another event-based technique based on thresholding is Approximate Caching [29] whereby nodes only report readings if they satisfy a condition. A more recent paper [30] suggests a mixture of hardware and software as a solution for detecting rare and random events. The event types they consider are tracking and detecting events using the eXtreme Scale Platform (XSM) mote equipped with infrared, magnetic and acoustic sensors. Central to their architecture is the concept of passive vigilance, which is inspired from sleep states of humans where the slightest noise can wake us up when we are asleep. This is implemented with duty cycling and recoverable re-tasking.

A similar approach [31] proposes a sleep-scheduling algorithm that minimizes the surveillance delay (event detection delay) while it maximizes energy conservation. Sleep scheduling is coordinated locally in a fair manner, so all nodes get their fair share of sleep. A minimal subset that ensures coverage of the sensing field is always awake in order to be able to capture rare events. Sleep scheduling is related to our DSFM approach.

A first paper that addresses the need for CED is the one by Girod et al [32]. In their work the authors suggest a system that would treat a sequence of samples (a signal segment) as a basic data type and would offer a language (WaveScript) to express signal processing programs as declarative queries over streams of data. The language would be able to execute both on PCs and distributed sensors. The data stream management system called WaveScope combines event-stream and data management operations. Unfortunately, the paper describing the system is a position paper so there is only a high-level description of the architecture and no current implementation or mention of plans.

REED is an improvement on TinyDB [33]. It extends TinyDB with its ability to support joins between sensor data and static tables built outside the network. The tables outside the network describe events in terms of complex predicates. These external tables are joined with the sensor readings table, and tuples that satisfy the predicate indicate readings of interest e.g. where an event has occurred.

A somewhat different method that supports geographic grouping of sensor nodes was presented in Abstract Regions [34], [35]. Abstract Regions is essentially a family of spatial operators for TinyOS that allows nodes to form groups with the objective of data sharing and reduction within the groups by applying aggregate operators such as `min, max, sum`, and so on. A different direction was presented in TINA [36] where the `TOLERANCE` keyword was introduced. Users could specify a level for tolerance e.g. if tolerance was set at 10% then only readings that differed by more than this tolerance threshold would get reported. The work by [37] extended the types of aggregates supported by introducing approximate quantiles such as the median, the consensus, the histogram and range queries.

VI. Discussion & Conclusions

We have taken a mature algorithm that is used — among other tasks — for conventional data mining in time-series data and we have re-engineered it so it can run efficiently within the strict constraints of a WSN. Our method successfully addresses the following requirements:

- It is *light-weight but powerful*. The code for the conversion is less than 1Kb (compiled RAM image) and it only takes 24ms, allowing for both high-frequency sampling and liberal use of other node components such as radio, and so on.
- It *scales well* as the number of events increase. With the pruned Suffix Array data structure we allow users to register many events without introducing a significant computational overhead.
- It *reduces* CPU idle time and can aid in limiting the use of radio communication. This helps to achieve power savings and consequently longer network lifetime.

Concluding, we have addressed a very common problem in WSN, namely the requirement for CED. The tangible outcome is an efficient implementation of SAX that allows it to run in tiny resource-constrained sensor nodes. Furthermore, by introducing a toolbox of detection mechanisms we offer more choice to the user of a reactive system. These facts complement the goal of prolonging the useful lifetime of a WSN.

Finally, all the source code for the techniques described in this paper has been implemented for TinyOS 2.x and made available to the WSN community.

ACKNOWLEDGMENTS

The authors would like to thank Dr Eamonn Keogh who patiently answered our SAX-related questions and provided relevant code and test data.

REFERENCES

[1] M. Tubaishat and S. Madria, "Sensor networks: An overview," *IEEE Potentials*, April/May 2003.
[2] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proceedings of the of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, 2002.
[3] "MoteIV: Tmote sky datasheet, www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf."
[4] I. Marshall, M. Price, H. Li, N. Boyd, and S. Bould, "Multi-sensor cross correlation for alarm generation in a deployed sensor network," in *Proceedings of 2nd European Conference on Smart Sensing and Context (EuroSSC)*, 2007.
[5] D. Doolina and N. Sitara, "Wireless sensors for wildfire monitoring," in *Proceedings of SPIE Symposium on Smart Structures & Materials NDE*, 2005.
[6] M. Ward, "Wi-fi sensor net aids wine makers http://news.bbc.co.uk/1/hi/technology/3860863.stm."
[7] J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, I. Henning, N. Boyd, C. Vincent, and I. Marshall, "Real world issues in deploying a wireless sensor network for oceanography," in *Proceedings of ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.
[8] K. Casey, A. Lim, and G. Dozier, "A sensor network architecture for tsunami detection and response," *International Journal of Distributed Sensor Networks*, vol. 4, no. 1, 2008.
[9] E. Katsiri, M. Ho, L. Wang, and B. Lo, "Real-time embedded heart-rate variability analysis," in *Proceedings of 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, 2007.
[10] "MIT house_n project http://architecture.mit.edu/house_n."
[11] "biosensornet: Autonomic biosensor networks for pervasive healthcare, www.doc.ic.ac.uk/~mss/Biosensornet.htm."
[12] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
[13] E. Keogh, J. Lin, and A. Fu, "HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence," in *Proceedings of IEEE International Conference on Data Mining*, 2005.
[14] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
[15] E. Keogh, S. Lonardi, and B. Chiu, "Finding surprising patterns in a time series database in linear time and space," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
[16] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter-free data mining," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.
[17] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.
[18] "Tinyos home page www.tinyos.net."
[19] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
[20] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," in *SIAM Journal on Computing, Volume 22, Issue 5*, 1993.
[21] J. Bentley, *Programming Pearls*. ACM Press (Addison-Wesley), 1999.
[22] J. R. Norris, *Markov Chains*. Cambridge University Press, 1998.
[23] S. Madden, "Intel berkeley research lab data, http://db.csail.mit.edu/labdata/labdata.html," 2004.
[24] E. Keogh, "The time series data mining archive, www.cs.ucr.edu/~eamonn/TSDMA."
[25] M. Zoumboulakis and G. Roussos, "Escalation: Complex event detection in wireless sensor networks." in *EuroSSC*, vol. 4793, 2007.
[26] "Easysen sbt80 multi-modality wireless sensor board www.easysen.com/SBT80.htm."
[27] C. I. R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, 2000.
[28] M. Halkidi, V. Kalogeraki, D. Gunopulos, D. Papadopoulos, D. Zeinalipour-Yazti, and M. Vlachos, "Efficient online state tracking using sensor networks," in *MDM '06: Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*, 2006.
[29] C. Olston, B. T. Loo, and J. Widom, "Adaptive precision setting for cached approximate values," in *Proceedings of SIGMOD Conference*, 2001.
[30] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a wireless sensor network platform for detecting rare, random, and ephemeral events," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.
[31] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.
[32] L. Girod, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan, and S. Madden, "The Case for a Signal-Oriented Data Stream Management System," in *CIDR 2007 - 3rd Biennial Conference on Innovative Data Systems Research*, 2007.
[33] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, 2005.
[34] M. Welsh, "Exposing resource tradeoffs in region-based communication abstractions for sensor networks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, 2004.
[35] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, 2004.
[36] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *The VLDB Journal*, vol. 13, no. 4, 2004.
[37] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004.