# Detection and Tracking of Discrete Phenomena in Sensor-Network Databases [*]

M. H. Ali[1] Mohamed F. Mokbel[1] Walid G. Aref[1] Ibrahim Kamel[2]

[1]Department of Computer Science, Purdue University, West Lafayette, IN
{mhali,mokbel,aref}@cs.purdue.edu
[2]College of Information Systems, Zayed University, U.A.E.
Ibrahim.Kamel@zu.ac.ae

## Abstract

*This paper introduces a framework for Phenomena Detection and Tracking (PDT, for short) in sensor network databases. Examples of detectable phenomena include the propagation over time of a pollution cloud or an oil spill region. We provide a crisp definition of a phenomenon that takes into consideration both the strength and the time span of the phenomenon. We focus on discrete phenomena where sensor readings are drawn from a discrete set of values, e.g., item numbers or pollutant IDs, and we point out how our work can be extended to handle continuous phenomena. The challenge for the proposed PDT framework is to detect as much phenomena as possible, given the large number of sensors, the overall high arrival rates of sensor data, and the limited system resources. Our proposed PDT framework uses continuous SQL queries to detect and track phenomena. Execution of these continuous queries is performed in three phases; the joining phase, the candidate selection phase, and the grouping/output phase. The joining phase employs an in-memory multi-way join algorithm that produces a set of sensor pairs with similar readings. The candidate selection phase filters the output of the joining phase to select candidate join pairs, with enough strength and time span, as specified by the phenomenon definition. The grouping/output phase constructs the overall phenomenon from the candidate join pairs. We introduce two optimizations to increase the likelihood of phenomena detection while using less system resources. Experimental studies illustrate the performance gains of both the proposed PDT framework and the proposed optimizations.*

* This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

## 1. Introduction

The wide spread of sensor network applications calls for new online query processing techniques to deal with the continuous arrival of sensor data. Examples of these applications include surveillance [25] and environmental monitoring [26]. Within a sensor network, each individual sensor sends a stream of data to a sensor network database. Although the individual readings of each sensor is useful by itself, the overall processing of the data in the sensor network database as one unit provides a global view of the underlying environment.

Recent research literature focuses on leveraging database and data stream management systems to handle the massive amount of received data from sensor networks, e.g., see [5, 8, 9, 10, 12, 17, 19, 30]. The main goal is to provide efficient query processing techniques for sensor data. In this paper, we focus on extending data stream management systems to support sensor network applications. In particular, we focus on *Phenomena Detection* and *Tracking*, (*PDT*, for short). We propose a framework that can be plugged into any data stream management system to provide an online and efficient phenomena detection and tracking.

As a first step towards *phenomena detection*, we propose a *crisp* definition of a *phenomenon*. Then, we simplify the definition by considering the discrete case of the phenomenon. The proposed definition relies on two main parameters; *strength* ($\alpha$) and *time span* ($w$). A phenomenon is of strength $\alpha$ and time span $w$ when it occurs $\alpha$ times in the last $w$ time units. The main idea of our proposed *phenomena detection* and *tracking* framework (*PDT*) is to join different readings from various sensors using a *multi-way* join algorithm for data streams. The output of the multi-way join algorithm feeds a *connectivity* graph that takes into consideration both the *strength* and *time span* of the required phenomenon. Continuously maintaining the *connectivity* graph *tracks* the sensor network phenomena. Moreover, we furnish our proposed *PDT* framework with a *phenomenon-*

*aware* optimizer where the execution of the *PDT* framework is tuned based on the received feedback from the query result.

In general, the proposed *phenomena detection* and *tracking* (*PDT*) framework has three phases; the *joining* phase, the *candidate selection* phase, and the *grouping/output* phase. The *joining* phase takes the raw data from the sensor network as its input and produces as output a set of sensor reading pairs that have similar values. The output of the *joining* phase is input to the *candidate selection* phase. The *candidate selection* phase strictly enforces the phenomena definition by filtering the input to produce only sensor pairs with the specified strength ($\alpha$) and the time span ($w$). Finally, the *grouping/output* phase constructs the overall phenomenon from the candidate join pairs produced by the candidate selection phase. Moreover, the *candidate selection* phase gives a feedback on the query result to the *joining* phase. Based on the query feedback, we introduce two *phenomenon-aware* optimizations that aim to tune the performance of the *PDT* framework.

All the proposed ideas and algorithms in this paper are implemented inside the *Nile* data stream management system [14]. *Nile* is a research prototype that is currently being developed at Purdue University. In general, the contributions of this paper can be summarized as follows:

1. We introduce a *crisp* definition of a phenomenon that takes into consideration both the strength and the time span of the phenomenon.
2. We propose an efficient technique for *phenomena detection* and *tracking* (*PDT*). The proposed technique adheres to the proposed phenomenon definition.
3. We propose two *phenomenon-aware* optimizations where the query result, i.e., the detected phenomenon tunes the execution of the *PDT* framework.
4. We provide , based on a real implementation inside a prototype data stream management system, an experimental evidence of the efficiency and performance gains of the *PDT* framework.

The rest of the paper is organized as follows: Section 2 introduces the *phenomenon* definition. The SQL queries that initiate the processing of the *PDT* framework are presented in Section 3. Section 4 introduces our proposed framework for *phenomena detection* and *tracking* (*PDT*). The *phenomenon-aware* optimization techniques are presented in Section 5. Experimental results that are based on a real implementation of the proposed *PDT* framework inside a data stream management system are presented in Section 6. Section 7 highlights related work. Finally, Section 8 concludes the paper.

## 2. Phenomena Definition and Applications

In this section, we introduce the definition of a phenomenon along with some applications that can benefit from our proposed definition.

**Definition 1** *In a sensor network $SN$, a phenomenon $P$ takes place only when a set of sensors $S \subset SN$ report similar reading values more than $\alpha$ times within a time window $w$.*

Two parameters control the *phenomenon* definition, the *strength* ($\alpha$) and the *time span* ($w$). The *strength* of a phenomenon indicates that a certain phenomenon should occur at least $\alpha$ times to qualify as a phenomenon. (This measure is similar to the notion of support in mining association rules, e.g., see [3].) Reading a value less than $\alpha$ times is considered noise, e.g., impurities that affect the sensor readings. The time span $w$ limits how far a sensor can be lagging in reporting a phenomenon. $w$ can be viewed as a time-tolerant parameter, given the common delays in a sensor network. (This measure is similar to the notion of gaps in mining generalized sequential patterns [24].)

In this paper, we focus on discrete phenomena that are produced by sensors whose reading values are discrete. In this case, the notion of similarity among sensor readings reduces to equality. Several applications benefit from the detection of discrete phenomena. Examples of these applications include:

- Tracing pollutants in the environment, e.g., oil spills in the ocean, or gas leakage out of a container. To be considered a phenomenon, the sensor should report the pollutant ID at least $\alpha$ times per $w$ time units.
- Reporting the excessive purchase of a certain item at different branches of a retail store in the same day. The purchase of an item is considered a phenomenon when the number of purchases exceeds $\alpha$ times in the last $w$ time units, e.g., in the last day.
- Detecting computer worms that strike various computer sub-networks over a certain period of time. When at least $\alpha$ computers are infected within a certain time window $w$, a phenomenon is reported.

Our work can be extended to detect continuous phenomena where sensors read values from a continuous range, e.g., temperature or density values, through a pre-processing phase. The pre-processing phase quantizes the sensor readings into a discrete set of value based on a user-defined function. Handling continuous phenomena is beyond the scope of this paper.

In general, a phenomenon may move in space. For example, an oil spill may surf the ocean according to the movement of the wind. A phenomenon may appear, disappear, move, expand, or shrink as time proceeds. In addition, a

```
SELECT SN1.VALUE, SN1.ID, SN2.ID
FROM SN SN1, SN SN2
WHERE SN1.VALUE=SN2.VALUE
AND SN1.ID <> SN2.ID
AND <other conditions >
GROUP BY SN1.VALUE, SN1.ID, SN2.ID
HAVING COUNT (*) >= α
WINDOW W
```

**Figure 1. PDT SQL queries**

phenomenon may have spatial properties. For example, an oil spill is a contiguous portion of the ocean surface. In this case, the spatial phenomenon is termed a *"cloud"*.

## 3. PDT SQL-Queries

To support sensor network operations, we extend data stream management systems with an abstract data type (*ADT*), called *SensorNetwork-ADT*. *SensorNetwork-ADT* handles the extraction of sensor readings from the sensor network. Sensor readings are of the form $(ID, value, loc, ts)$, where $ID$ is the identifier of the sensor that emitted the reading while $value$ and $loc$ indicate the reading value and the location of that sensor at timestamp $ts$, respectively.

Figure 1 gives the general form of SQL queries that continuously detect phenomena in a sensor network database. Basically, the sensor network *SN* is joined with itself. Any sensor $S_i \in SN$ is eligible to join with any other sensor $S_j \in SN$, $(S_i \neq S_j)$, based on an equality join of *SN.value*. Based on the application semantics, the `where` clause specifies other conditions, e.g., the spatial and/or temporal clustering of the phenomenon. The phenomenon *strength* ($\alpha$) is checked by grouping the query result by $(SN1.VALUE, SN1.ID, SN2.ID)$ and the *count* is calculated to report only sensors that join on the same value more than $\alpha$ times within window $w$. The phenomenon *time span* ($w$) is presented within the `window` clause.

Figure 2 gives an example of an SQL query that detects and tracks pollutants in the ocean, e.g., oil spills. $OC$ represents a set of sensors distributed in the ocean. The sensor network $OC$ is joined with itself based on the *liquid* value reported from each sensor. Only sensors that report a *liquid* value other than *"water"* are considered in the join. To reflect the *spatial* clustering of the detected pollutants, each sensor is restricted to join with other sensors that are at a maximum distance of ten meters. The *strength* ($\alpha$) and *time span* ($w$) of the detected phenomena are set to five and one minute, respectively.

```
SELECT OC1.LIQUID, OC1.ID, OC2.ID
FROM OC OC1, OC OC2
WHERE OC1.LIQUID=OC2.LIQUID
AND OC1.ID <> OC2.ID
AND LIQUID<>"WATER"
AND DISTANCE(OC1.LOC,OC2.LOC)<= 10
GROUP BY OC1.LIQUID, OC1.ID, OC2.ID
HAVING COUNT (*) >= 5
WINDOW 1 minute
```

**Figure 2. An example SQL query for pollution detection**

## 4. PDT Query Processing

The process of *phenomena detection* and *tracking* (*PDT*) is initiated by issuing the SQL-query given in Figure 1. *PDT* query processing is divided into three phases as illustrated in Figure 3. The first phase, the *joining* phase, accepts the input tuples streamed out of the sensors and applies an in-memory *multi-way* join over the entire sensor network to detect sensors with the same value within a time frame of length $w$ from each other. The second phase, the *candidate selection* phase, receives the joined sensor pairs and checks the sensors that qualify to be phenomena candidate members. Based on our definition of a phenomenon, the *candidate selection* phase checks the density of the phenomenon based on the user-specified strength ($\alpha$) and time span ($w$). Sensors that join at least $\alpha$ times over a time-window $w$ are reported to the *grouping/output phase*. The third phase, *the grouping/output phase*, groups the pairs of phenomena candidate members and investigates the application semantics to form and report the phenomena to the user.

Guided by the detected phenomena candidate members in the *candidate selection* phase, the processing is tuned to increase the likelihood of phenomena detection while using less resources. A phenomenon-aware feedback is provided to the *joining phase* to draw the attention to regions where phenomena tend to be active. For example, the input buffers that are associated with sensors contributing to phenomena are given higher priorities than those that do not contribute to any phenomena. Similarly, in the *joining phase,* the join probing sequence is tuned to favor the joins that affect the appearance or the disappearance of a phenomenon. The rest of this section is dedicated to the three phases of the proposed *PDT* framework. *phenomenon-aware* optimizations are presented in Section 5.

### 4.1. Phase I: Joining

Two alternative approaches exist for implementing the multi-way join operator for $N$ streams: as a series of cas-

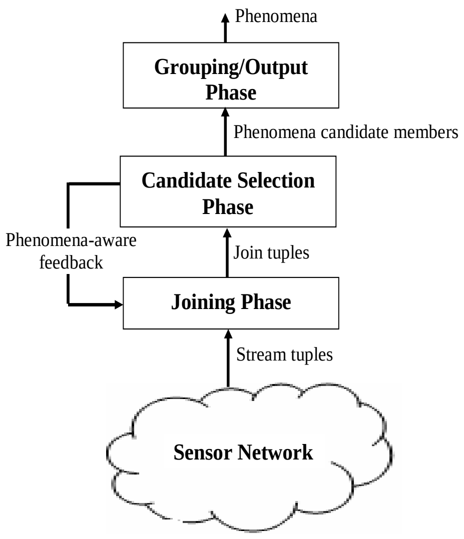


**Figure 3. PDT query processing phases**

caded $N - 1$ binary join operators where only two streams are joined at a time, or as a single operator that takes $N$ streams as its input. The MJoin operator [28] employs the second approach where it produces join results with a faster rate than the tree of binary joins. Thus, MJoin [28] is more suitable for data streaming applications. The main idea of MJoin is to maintain a hash table for each stream, i.e., sensor. Once a tuple arrives from one stream, it is inserted into the stream's corresponding hash table. Then, the incoming tuple probes the hash tables of other streams. Since a joined tuple is reported only if it appears in *ALL* streams, the MJoin algorithm stops probing hashing tables once the probed value is missing in one of the streams. To avoid unnecessary processing, the probing sequence is chosen based on the join selectivity among the streams.

To support *phenomena detection* and *tracking* (*PDT*), we employ three main modifications to the original MJoin algorithm [28]. These modifications are summarized as follows:

1. The modified multi-way join algorithm does not stop once the join value is missing in one of the streams. Instead, it continues to examine the remaining streams to produce partial results. Notice that a phenomenon need not span all the sensors in the sensor network.

2. The notions of positive and negative tuples [13] are utilized. A positive tuple is reported when a join occurs. A negative tuple is reported when one of the previously-reported join tuple components expires, i.e, becomes old enough to get outside of the most recent time-window $w$. The negative tuple is important to invalidate the candidate members of a phenomenon, if the sensors stop showing the same behavior over a time-window $w$.

3. The probe sequence and the stream sampling rate are guided by the detected phenomena to favor the probe sequence and the streams that participate in a phenomenon. This phenomenon-aware optimizations are discussed in detail in Section 5.

## 4.2. Phase II: Candidate Selection

The *joining* phase produces a tuple if the same reading is observed by two streams within the specified time-window. These two streams are considered phenomena candidate members if they persist to join with each other $\alpha$ times within the same time-window. The *candidate selection* phase employs a *connectivity graph* that is used to record the number of joins between each pair of sensors. Each sensor $S_i$ is represented by a node in the *connectivity graph*. For any two sensors $S_i$ and $S_j$, ($i \neq j$), an edge $E(v, i, j)$ is added to the connectivity graph only if $S_i$ and $S_j$ are joined together at least once in the last $w$ time units over value $v$. The weight of the Edge $E_{ij}$ is the number of times that $S_i$ and $S_j$ are joined together in the last $w$ time units, i.e., the strength of the phenomenon.

Figure 4 gives the processing of input pairs received from the *joining phase*. The input is either a positive or a negative tuple with the format $\pm(SN1.VALUE, SN1.ID, SN2.ID)$. The tuple represents the join value and two joining sensors. This tuple updates the weights of the edges in the connectivity graph. The weight of each edge is monitored. If the weight of an edge increases to reach $\alpha$ (i.e., $weight = \alpha$), a positive tuple is reported to denote the appearance of the candidate member $(SN1.VALUE, SN1.ID, SN2.ID)$. If the weight of an edge drops below $\alpha$ (i.e., $weight = \alpha - 1$), a negative tuple is reported to denote the disappearance of that candidate member.

Figure 4.2 gives an example of the connectivity graph for five sensors over a window of 5 time units. The connectivity graph starts from scratch and records each join tuple by increasing the weight of the edge between the two joining sensors. Edges that exceed $\alpha$, which is set to four, are marked as bold lines to denote phenomenon candidate members. Notice that, as the window slides, the value $10$ from sensor $S1$, that came at $t1$, will expire, and consequently the edge between $S1$ and $S3$ will drop below $\alpha$ generating a negative tuple to invalidate that candidate member.

## 4.3. Phase III: Grouping/Output

The *grouping/output* phase receives phenomena candidate members on the form of a tuple that consists of the IDs of the two joining sensors and the join value. Each tuple

```
INPUT: the join tuple (SN1.VALUE, SN1.ID, SN2.ID) OUTPUT: the phenomena candidate members

Upon receiving a positive tuple,

if CheckEdge(SN1.VALUE, SN1.ID, SN2.ID)
    // if edge exists increase its weight
    Edge(SN1.VALUE, SN1.ID, SN2.ID).weight++
    // check the appearance of a candidate
    if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=$\alpha$)
        Output +(SN1.VALUE, SN1.ID, SN2.ID)
else
    // create a new edge with weight=1
    CreateEdge(SN1.VALUE, SN1.ID, SN2.ID)
    Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=1;
endif

Upon receiving a negative tuple,

// Decrease the weight of the edge by 1 and
Edge(SN1.VALUE, SN1.ID, SN2.ID).weight−−
// check the disappearance of a candidate
if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=$\alpha - 1$)
    Output −(SN1.VALUE, SN1.ID, SN2.ID)
// remove the edge if its weight becomes zero
if (Edge(SN1.VALUE, SN1.ID, SN2.ID).weight=0)
    RemoveEdge(SN1.VALUE, SN1.ID, SN2.ID)
endif
```

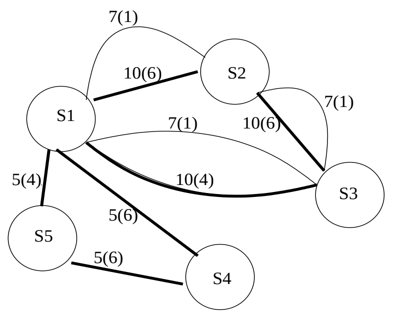
**Figure 4. Pseudo code of the candidate selection phase**

can be positive or negative to denote the appearance or disappearance of a candidate member. A positive/negative tuple indicates that the number of joins between the two sensors over the last $w$ becomes above/below $\alpha$. Upon receiving a positive tuple, based on the application semantics, the *grouping/output* phase may start a new phenomenon, add one sensor to an existing phenomenon, or merge two phenomena together. Similarly, upon receiving a negative tuple, the *grouping/output* phase may delete a phenomenon, remove a sensor from an existing phenomenon, or split one phenomenon into two separate phenomena.

In addition, the *grouping/output* phase has the flexibility to apply application-dependent semantics. For example, forcing a minimum number of sensors to form a phenomenon, determining the outer contour or the convex hull of the phenomena . Also, the application may consider the spatial clustering of sensors. The clustering or spread of the phenomenon implies whether there is a single source or multiple sources for the phenomenon. For example, multiple disconnected oil spills implies leakage out of more than one container. Application-dependent semantics can include the density of the phenomena as well. The density is measured by the ratio of the number of sensors reading the same phenomenon to the number of sensors not reading that phenomenon in a specified region. All these issues are application-dependent and are addressed by the *grouping/output* phase.

|  | **t1** | **t2** | **t3** | **t4** | **t5** |
|---|---|---|---|---|---|
| Sensor 1 | **10** | **7** | **10** | **5** | **5** |
| Sensor 2 | 15 | **10** | **7** | **10** | **10** |
| Sensor 3 | 19 | **7** | 3 | **10** | **10** |
| Sensor 4 | 22 | **5** | 9 | **5** | **5** |
| Sensor 5 | 18 | **5** | **5** | 4 | 23 |

(a) Sensors' input over 5 time instants



(b) The connectivity graph

**Figure 5. An example of the connectivity graph**

## 5. Phenomenon-Aware Query Optimization

This section proposes two *phenomenon-aware* optimizations that aim to provide a scalable execution for the proposed *phenomena detection* and *tracking* (*PDT*) framework. These optimizations tune the processing towards tuples that contribute in producing the phenomena. The main idea is to utilize the processing of the *candidate selection* phase to provide feedback to the *joining* phase. The feedback contains information about the sensors that contribute to the currently tracked phenomena. The two phenomenon-aware optimizations are: (1) Controlling the sampling rate of each sensor, and (2) Choosing the join probing sequence.

### 5.1. Controlling the Sampling Rate of Sensors

The number of sensors in a sensor network can grow large. These sensors may be generating stream data with high rates. They may show a bursty behavior as well. As a result of all of the above reasons, the query processing engine faces periods of heavy load. In these periods, it will not be possible to cope with every sensor reading. To overcome this problem, a sampler for each sensor is employed to control the input rate of its generated stream. Instead of a random sampler, a *phenomenon-aware* sampler is preferred to favor sensors that contribute heavily to the phenomena. The *phenomenon-aware* sampler gets feedback from the *candidate selection* phase about the phenomena that are currently monitored.

Notice that if the weight of an edge between two sensors is highly below or highly above $\alpha$, it is less likely to provide new information. If it is highly below $\alpha$, it is less likely to increase instantly to $\alpha$ and form a phenomenon candidate member. Similarly, if it is highly above $\alpha$, it is less likely to decrease instantly below $\alpha$ and eliminate a candidate member. Sensors with edges that are close to $\alpha$ are considered strong candidates to form or eliminate a phenomenon. Hence, as given in Equation 1, the *edge strength (ES)* between two sensors $i, j$ is inversely proportional to the absolute difference between the edge weight and $\alpha$ ($|Edge(v, i, j).weight - \alpha|$), where $Edge(v, i, j).weight$ is the weight of the edge between sensors $i$ and $j$ based on the value $v$ (or zero if no edges at all). However, there may be more than one edge between the two sensors if they join over multiple values. To be conservative, the edge gets the maximum strength over all of these values and the *sensor strength (SS)* is considered to be the maximum strength over all of the edges connecting that sensor to its neighbors (Equation 2). To favor sensors that are involved in phenomena, the *sampling factor ($SF_i$)* of each sensor $i$ is proportional to its strength as given in Equation 3.

Let $R^*$ be the global desired sampling rate over all sensors, and let $R_i$ be the sampling rate of each sensor. $R^*$ is formed by adding the sensor rates $R_i$ after they are being adjusted by $SF_i$ (Equation 4). Equation 4 is rewritten again in the form of a summation in Equation 5. In Equation 6, we substitute for $SF_j$ by Equation 3. From Equation 7, we can obtain $SF_i$ given the rate of each sensor, the strength of each sensor and the desired rate $R^*$. These parameters are continuously updated as the streams are running. The stream rates ($R_is$) and the desired rate ($R^*$) are updated periodically to avoid unnecessary fluctuations and bursty behaviors of sensors. The sensor's strength ($SS_i$) is updated with the arrival of each tuple to eagerly detect the new phenomena.

$$ES(i, j) = MAX_v\{\frac{1}{1 + |Edge(v, i, j).weight - \alpha|}\} \tag{1}$$

$$SS_i = MAX_j(ES(i, j)) \tag{2}$$

$$\frac{SF_i}{SF_j} = \frac{SS_i}{SS_j} \tag{3}$$

$$SF_1 \cdot R_1 + SF_2 \cdot R_2 + \cdots + SF_n \cdot R_n = R^* \tag{4}$$

$$SF_i \cdot R_i + \sum_{j=1, j\neq i}^{n} SF_j \cdot R_j = R^* \tag{5}$$

$$SF_i \cdot R_i + \sum_{j=1, j\neq i}^{n} \frac{SF_i \cdot SS_j}{SS_i} \cdot R_j = R^* \tag{6}$$

$$SF_i \cdot (R_i + \frac{\sum_{j=1, j\neq i}^{n} SS_j \cdot R_j}{SS_i}) = R^* \tag{7}$$

Reducing the stream sampling rate of a sensor may lead to delaying the discovery or to entirely missing new phenomena because they will take a longer time to increase the edge weights between participating sensors till they reach to the desired $\alpha$. A sensor needs to be persistent in producing the phenomenon and to increase its strength gradually till the phenomenon is discovered. The difference between the time at which the phenomenon is formed and the time at which it is reported is known as the response time. We trade the response time of discovering new phenomena for the sake of monitoring already existing phenomena efficiently. Otherwise, monitoring all sensors with the same quality would degrade the whole system's performance and may result in losing phenomena.

## 5.2. Choosing the Join Probing Sequence

Once a tuple arrives from one sensor, it is used to probe the hash tables of other sensors looking for matches. The sequence in which the tuple probes other hash tables affects the performance of the join operator. In the original MJoin [28] algorithm, the selectivity factors among the joins are taken into consideration. The least selective join is evaluated first. Consequently, the number of partial output tuples is reduced at early steps. In our context, choosing the join order based on the selectivity is not of great benefit where we are interested in partial results as well. Moreover, in large sensor networks, probing hundreds or thousands of sensors may be prohibitive, specially when the sensor joins with a very small subset of the sensors. Instead, we choose a probing sequence in which probing is based on the likelihood of producing a tuple that contributes to a phenomenon.

If a tuple arrives at sensor $i$, it probes the hash table of sensor $j$ with probability $P$, where

$$P = \frac{1}{1 + |Edge(v, i, j).weight - \alpha|} \tag{8}$$

Equation 8 adjusts the probing probability based on how close the edge between sensors $i$ and $j$ on value $v$ to $\alpha$. The

```
INPUT: a tuple from sensor $S_i$ with value $v$
OUTPUT: the join probing sequence

for j=1 to NoOfSensors
begin
    $P = min(BaseProb, \frac{1}{1+|Edge(v,i,j).weight-\alpha|})$
    Generate a random variable $U$ between $0, 1$
    if ($U \leq P$)
        ProbeSensor(j)
end
```

**Figure 6. The join probing sequence**

join probing sequence is evaluated as given in Figure 6. On the arrival of a tuple with value $v$ from sensor $i$, a complete traversal over all sensors is performed. Because the probing operation is costly, we decide to either probe or skip sensor $j$ based on the probability $P$. Notice that $P$ should not go below a minimum $BaseProb$ to avoid the zero join probability among sensors with no edges in between. This policy reduces the number of joins dramatically and focuses on joins that contribute in phenomena. Eliminating the cost of unnecessary joins allows our technique to be scalable with respect to the number of sensors. Similar to the case of Section 5.1, a delay may be observed in detecting new phenomena. A sensor needs to strengthen the edge between itself and other sensors gradually to get a higher probability in the join operation.

## 6. Experiments

In this section, we conduct an experimental study of the proposed *phenomena detection* and *tracking PDT* technique. Three sets of experiments are conducted. The first set of experiments (Section 6.1) is concerned with the effect of the *PDT* parameters; the strength $\alpha$ and the time span $w$ on the number of detected phenomena. The second (Section 6.2) and the third sets (Section 6.3) of experiments study the performance of the *PDT* optimization techniques with the change of the number of sensors and data arrival rates, respectively. For the last two sets of experiments, we compare the performance of the following four versions of the proposed *PDT* framework:

1. *Simple PDT*, where processing is not guided by the detected phenomena.
2. *PDT+1*, where the sensor's sampling rate is controlled based on the detected phenomena as discussed in Section 5.1.
3. *PDT+2*, where the join probing sequence is controlled based on the detected phenomena as discussed in Section 5.2.

| $\alpha$ | w=5 | w=10 | w=15 | w=20 |
|---|---|---|---|---|
| 3 | 1010 | 4350 | 7150 | 11150 |
| 4 | 121 | 498 | 815 | 1256 |
| 5 | 20 | 220 | 270 | 320 |
| 6 | 4 | 19 | 56 | 140 |
| 7 | 0 | 5 | 11 | 18 |
| 8 | 0 | 0 | 2 | 5 |

**Table 1. The effect of application-dependent parameters**

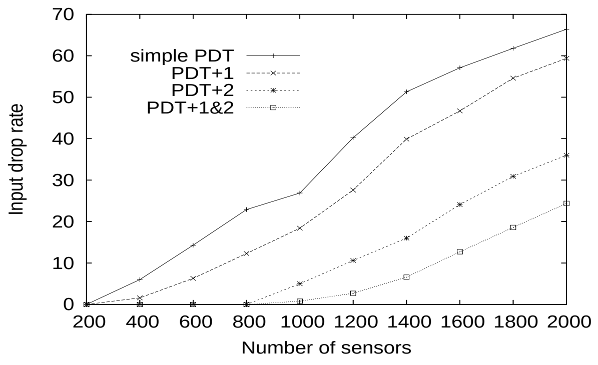
4. *PDT+1&2*, where both of the optimizations in (2) and (3) are applied together.

Various *PDT* techniques are compared with respect to three performance measures: (1) The *input drop rate* where some of the input sensor data tuples have to be dropped due to the scarcity of system resources. (2) The *response time*, which is measured by the difference between the time in which a phenomenon is reported by the system and the actual time in which it took place. (3) The *output loss rate* where some phenomena are lost as a result of losing some of the input tuples. A smart technique tries to minimize all these measures. rate.

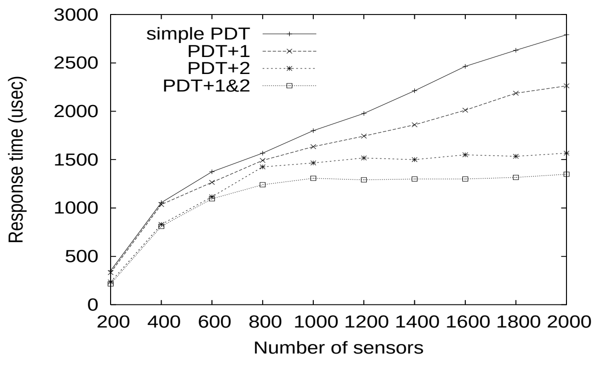
All the experiments are triggered by the execution of the continuous query given in Figure 2. A synthesized data set is used to simulate the sensor network readings. Unless mentioned otherwise, we maintain 1000 sensors uniformly distributed over a $100 \times 100$ meters rectangular space. Each sensor generates a stream of 10,000 tuples where the tuple values follow the zipfian distribution. The interarrival time of sensor data follows an exponential distribution with an average of one second. All the experiments in this section are based on a real implementation of the *PDT* framework inside the *Nile* data stream management system [14]. The Nile engine executes on a machine with Intel Pentium IV, CPU 2.4GHZ with 512MB RAM running Windows XP.

### 6.1. PDT parameters

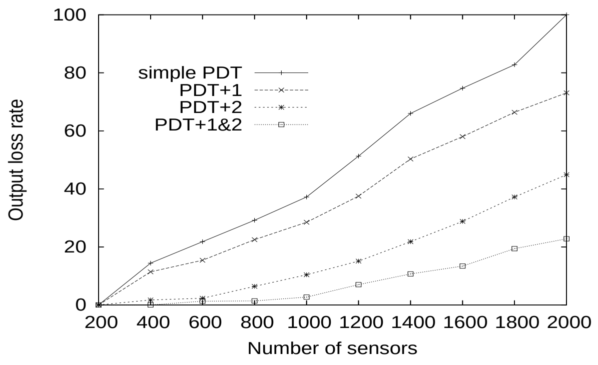
Table 1 gives the number of detected phenomena for various values of the strength ($\alpha$) and time span ($w$). As given in the table, $\alpha$ and $w$ have opposite effects. The increase in $\alpha$ results in less detected phenomena as the condition for detecting a phenomenon becomes more restrictive. On the other side, the increase in the time span $w$ relaxes the condition for the detected phenomena. Thus, more phenomena can be detected. The largest number of detected phenomena is obtained for $w = 20$ and $\alpha = 3$. At this point, *PDT* tracks up to 11,150 different phenomena over the lifetime of the experiment.



(a) The input drop rate



(b) The response time



(c) The output loss rate

**Figure 7. The effect of the number of sensors**

## 6.2. The effect of the number of sensors

Figure 7 studies the scalability of the four variations of the *PDT* framework with respect to increasing the number of sensors from 200 to 2000. The *PDT* parameters $\alpha$ and $w$ are set to 5 and 10 seconds, respectively. *PDT-1* decreases the input drop rate over the *simple PDT* because it reduces the sampling rate of sensors that do not contribute to any phenomenon and, hence, keeps the input buffers less occupied (Figure 7a). The response time of the *PDT+1* is less than that of the *simple PDT* because the size of the hash structures gets smaller after reducing the sampling rate of irrelevant sensors (Figure 7b). As a result of controlling the sampling rates, the output loss rate of the *PDT+1* is reduced (Figure 7c). Notice that as the number of sensors increases, more load is posed against the system and the difference in the output loss rate becomes significant (up to 25% for 2000 sensors). *PDT+2* reduces the response time (Figure 7b) because it favors the join over sensors that contribute to phenomena and leaves other joins to be performed with a lower probability. This behavior increases the processing time availability and reduces the input drop rate (Figure 7a). The controlled join probing sequence reduces the output loss rate through the efficient management of the time budget in useful joins (Figure 7c). *PDT+1&2* combines the features of both of the *PDT+1* and *PDT+2* techniques. *PDT+1&2* decreases the input drop rate (Figure 7a), the response time (Figure 7b), and the output loss rate (Figure 7c). There is a reduction of up to 78% in the output loss rate over the *simple PDT* for 2000 sensors.

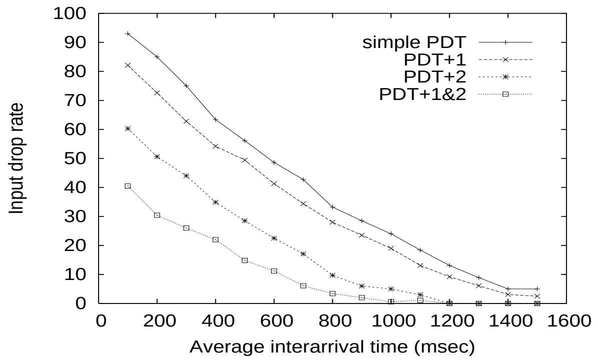
## 6.3. The effect of the stream rates

Figure 8 compares the performance of the four variations of the *PDT* framework with respect to varying the sensor data interarrival rates from 0.1 to 1.5 seconds. The *PDT* parameters $\alpha$ and $w$ are set to 5 and 10 seconds, respectively. Small interarrival times imply scarcity in resources. Large interarrival times imply an increased availability of resources, where all curves approach each other and the drop rate approaches zero. For the same reasons, as discussed in Section 6.2, both *PDT+1* and *PDT+2* decrease the input drop rate over the *simple PDT* (Figure 8a), the response time (Figure 8b), and the output loss rate (Figure 8c). The *PDT+1&2* combines the benefits of both optimizations and reduces the output loss rate by up to 45% over the *simple PDT* (for a 0.1 second average interarrival time).

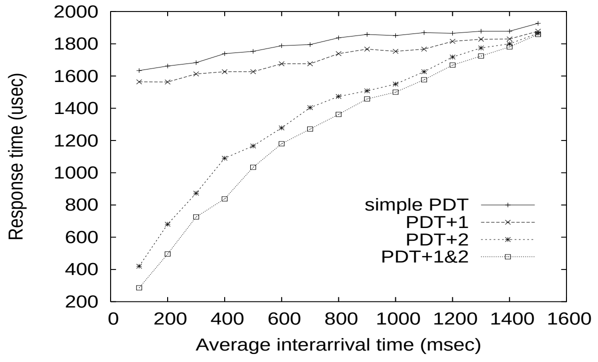
Notice that the response times of the *simple PDT* and *PDT+1* are constant over time because the join probing sequence spans all of the 1000 sensors for all stream rates. However, the *PDT+2* and *PDT+1&2* increase the length of the probing sequence with the increase of the time availability. The response time increases because the technique traverses a larger probing sequence per tuple for the sake of decreasing the output loss rate.

## 7. Related Work

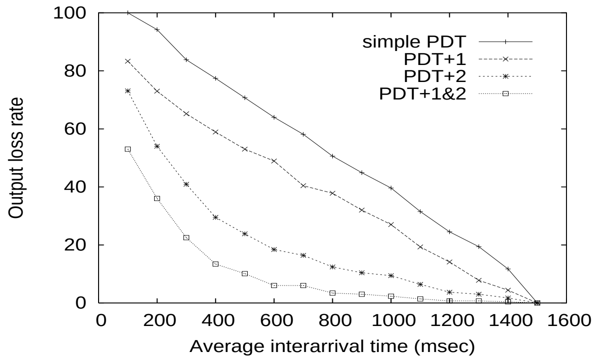
A lot of research interest has been directed recently to data stream processing. Data stream systems, e.g., Stanford STREAM [22], AURORA [1], NiagaraCQ [7], Telegraph [6], are developed to cope with the new challenges imposed by the nature of data streams [4]. The COUGAR system [5, 30] introduces a new abstract data type for sen-



(a) The input drop rate



(b) The response time



(c) The output loss rate

**Figure 8. The effect of the stream rates**

sors to facilitate the extraction of data and to process queries over sensor networks. The Borealis engine [2] proposes a scalable QoS-based optimization model to operate across sensor networks. The Fjords architecture [17] proposes an infrastructure for query processing over sensor data.

The physical acquisition or sampling of sensor data is explored in [19, 11, 16, 23] to extract the sensor data with low cost but still accurate methods. The work in [8, 18, 20] proposes the aggregation of sensor data tuples before reaching the data management system. Aggregates reduce the data size and, consequently, consume less power in the transmission process.

Some prior work has been conducted to track moving objects in sensor networks [12]. Similar to our work, the join operation is used to detect similar readings over the sensor network. It proposes a new non-blocking multi-way join operator over a sliding window, the W-join operator, to track the readings of the same object-ID that appears in different locations over the sensor network. Our work tracks moving phenomena, where each phenomenon is a group of sensors producing the same value, rather than tracking individual moving point objects.

The join operation over data streams has been explored in the literature. Symmetric Hash Join [29] is proposed to take care of the infiniteness of the data source. XJoin [27] provides disk management to store overflowing tuples on disk for later processing. An asymmetric window join over two data streams with different arrival rates is discussed in [15]. The Hash-Merge Join (*HMJ*) [21] is a non-blocking join algorithm that produces early join results. In our work, a modified version of the M-Join [28] is used to detect streams with similar behavior over a window of time.

## 8. Conclusions

In this paper, we proposed a framework for *phenomena detection* and *tracking* (*PDT, for short*) in sensor network databases. To identify a phenomenon, we provided a *crisp* definition for the phenomenon that takes into consideration both the strength ($\alpha$) and the time span ($w$). A *phenomenon* of strength ($\alpha$) and time span $w$ occurs at least $\alpha$ times in the last $w$ time units.

The proposed *PDT* framework has three phases: The *joining* phase, the *candidate selection* phase, and the *grouping/output* phase. The *joining* phase employs a multi-way join algorithm that joins the raw data from the sensor network and produces a set of sensor pairs with similar values. The *candidate selection* phase takes the output of the joining phase as input and applies a filter to enforce the strength and time span of the phenomena. Finally, the *grouping/output* phase is application-specific where it enforces the application semantics. Furthermore, we provided two *phenomenon-aware* optimizations that aim to : (1) Increase the sampling rate of the sensors that are part of any phenomenon, and (2) Choose the join order of the multi-way join algorithm to increase the likelihood of detecting the phenomena.

Experimental study based on a real implementation inside a research prototype for data stream management systems shows that the proposed *PDT* technique is scalable in terms of the number of streams, the stream rates, and the number of detected phenomena. The *optimized PDT* reduces the output loss rate over the *simple PDT* by up to 78% for a network of size 2000 sensors.

## References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2:120–139, August 2003.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *Proceedings of the Second Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2005.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB*, pages 487–499, Sept. 1994.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of PODS*, pages 1–16, June 2002.

[5] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Intl. Conf. on Mobile Data Management*, pages 3–14, Jan. 2001.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, and et al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet databases. In *Proceedings of ACM SIGMOD*, pages 379–390, May 2000.

[8] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of ICDE*, pages 449–460, April 2004.

[9] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB*, pages 588–599, August 2004.

[10] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of ACM SIGMOD*, pages 503–514, June 2003.

[11] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *Computer Communication Review*, 34(1):125–130, 2004.

[12] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. Stream window join: Tracking moving objects in sensor-network databases. In *Proceedings of the Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 75–84, July 2003.

[13] M. A. Hammad, T. M. Ghanem, W. G. Aref, A. K. Elmagarmid, and M. F. Mokbel. Efficient execution of sliding-window queries over data streams. Technical Report CSD-03-035, Department of Computer Science, Purdue University, June 2004.

[14] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *Proceedings of ICDE*, page 851, April 2004.

[15] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *Proceedings of ICDE*, pages 341–352, Feb. 2003.

[16] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. Quasar: quality aware sensing architecture. *SIGMOD Record*, 33(1):26–31, 2004.

[17] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of ICDE*, pages 555–566, Feb. 2002.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *USENIX OSDI*, 2002.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of ACM SIGMOD*, pages 491–502, June 2003.

[20] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks.

[21] M. Mokbel, M. Lu, and W. Aref. Hash-merge join: A non-blocking join algorithm for producing fast and early join results. In *Proceedings of ICDE*, pages 251–262, April 2004.

[22] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2003.

[23] J.-Y. Pan, S. Seshan, and C. Faloutsos. Fastcars: Fast, correlation-aware sampling for network data mining. In *GLOBECOM 2002 - IEEE Global Telecommunications Conf.*, pages 2167 – 2171, November 2002.

[24] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *In Proceedings of EDBT*, pages 3–17, March 1996.

[25] S. Srinivasan, H. Latchman, J. Shea, T. Wong, and J. McNair. Airborne traffic surveillance systems: video surveillance of highway traffic. In *The 2nd ACM international workshop on Video surveillance & sensor networks*, pages 131–135, 2004.

[26] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of ACM*, 47(6):34–40, 2004.

[27] T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2):27–33, 2000.

[28] S. Viglas, J. F. Naughton, and J. Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *Proceedings of VLDB*, pages 285–296, Sept. 2003.

[29] A. N. Wilschut and E. M. G. Apers. Pipelining in query execution. In *Proceedings of the International Conference on Databases, Parallel Architectures and their Applications*, 1991.

[30] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of the First Conf. on Innovative Data Systems Research (CIDR)*, pages 233–244, Jan. 2003.