2009 Eighth IEEE International Symposium on Network Computing and Applications

A Case for Meta-Triggers in Wireless Sensor Networks

Goce Trajcevski Nikolay Valtchanov Oliviu C. Ghica Peter Scheuermann
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois 60208
Email: {goce,nikolay,ocg474,peters}@eecs.northwestern.edu

***Abstract*—This work addresses the problem of managing the reactive behavior in Wireless Sensor Networks (WSN). We consider settings in which the occurrence of a particular *event*, detected in a state that satisfies a given *condition*, should fire the execution of an *action*. We observe that in WSN settings, both the event and condition may pertain to some continuous phenomena that are monitored by distinct groups of nodes and, in addition, their respective detection may impose an extra communication overhead, if a correct executional behavior is desired in terms of firing the action. Towards that end, we propose the concept of a *meta trigger*, which essentially *translates* a particular request, so that the communication overhead among the entities participating in its processing is minimized. We discuss a proof-of-concept implementation which demonstrates the benefits of the proposed methodology on an actual small-size network, and we present a detailed simulation-based experimental evaluation in large-scale networks. Our experiments indicate that the meta-triggers can yield substantial savings in the energy (and bandwidth) expenditures of the network, while preserving the intended executional correctness.**

I. Introduction and Motivation

A wireless sensor network (WSN) typically consists of a large collection of *sensor nodes* , i.e. devices characterized by limited energy resources, computational power, memory space and communication capabilities [1], that are capable to form a network and coordinate their activities in order to achieve a particular task. In order for the WSN to provide a satisfactory level of a Quality of Service (QoS) (e.g., in terms of data latency, accuracy) for a particular task, careful algorithmic designs are needed at the application, routing and media access layer (MAC) [2]. However, given the limited power-resources of the individual nodes, an important parameter is the time-extent during which a WSN is operational, namely, the network's *lifetime* [3]. Lifetime maximization and energy-consumption minimization are correlated but distinct problems [3] and it has been shown that the network lifetime optimization problems are NP hard (c.f. [4]). Various techniques have been proposed to reduce the energy-costs associated with particular network tasks, e.g., in-network data aggregation [5], filtering [6], compression [7] and optimal-path routing [8]. All of them consider the energy-efficiency aspect as part of the solution to the lifetime maximization problem.

From acquisitional point of view, the WSN can be seen as a distributed database, in which *instantaneous* and/or *continuous* queries are being processed [9]. Specifically, the TinyDB project [10], provides an SQL-like interface where the users can state their queries pertaining to the data observed by a given WSN, and the system transparently performs various query-optimization tasks. TinyDB is a distributed query processor that runs on the individual nodes in the WSN, offering many of the features of a traditional query processor (e.g., *select*, *project*, *join* and *aggregate* data). However, it also has a number of other features designed to minimize power consumption via acquisitional techniques. As a specific example of the SQL syntax, consider the query (cf. [10]):

```
SELECT AVG(volume),room
FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(volume) > threshold
SAMPLE PERIOD 30s.
```

This query partitions motes on the 6th floor according to the room where they are located (which may be hard-coded in each device, or determined via some localization technique). The answer reports all rooms where the average volume is over a specified threshold, updated every 30s. The query described above is *continuous*, in the sense that it will be periodically re-evaluated until it is de-registered from the system. One of the central concepts to the acquisitional query processing are the *event-based* queries. Events denote the occurrence of "something of interest" that is generated explicitly, either by another query or by a lower-level part of the operating system (in this case, the TinyOS) and the code that generates the event must have been compiled into the sensor node. As an example, consider (cf. [10]):

```
ON EVENT bird-detect(loc):
SELECT AVG(light), AVG(temp), event.loc
FROM sensors AS s
WHERE dist(s.loc, event.loc) < 10m
SAMPLE PERIOD 2 s FOR 30 s.
```

This query reports the average light and temperature level at sensors near a bird nest, however, it does so *only when a bird is detected*. Clearly, in case the arrival of a bird in a given nest is *not* a frequent event, the above query will generate substantial savings compared to a similar query that would, for example, continuously sample the temperature and the light intensity values.

Although the TinyDB paradigm provides event-based behavior for the purpose of efficient query processing, a methodology that enables a full-fledge exploitation and management of the *reactive behavior*, available in almost any commercial Database Management System (DBMS) via *triggers* is still lacking. Triggers, which resulting from the Active Databases (ADB) research [11], [12], are statements of the form:

ON **E**VENT
IF **C**ONDITION
THEN **A**CTION

specifying the (ECA) rules of reactive behavior. They have been part of the SQL standard since the 1990s and, typically: the `EVENT` is an elementary transactional operation (e.g., `Insert`); the `CONDITION` part is an SQL-query over the state (current and/or past) of the database; and the `ACTION` part is again an SQL statement.

In WSN settings, some of the main aspects of managing ECA-like reactive behavior are:
(1) The `EVENT` may be associated with an occurence of a phenomenon that has a *duration/validity_interval* associated with it, thereby making it continuous in nature;
(2) The `CONDITION` part may also be a *continuous* query which, in addition, may pertain to a region that is *spatially* different from the region in which the `EVENT` was detected;

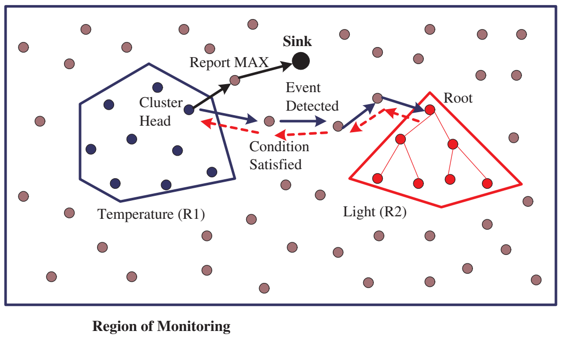


Fig. 1. Distributed Reactive Behavior

Specifically, consider the following request:
Rq1: *"Whenever the average temperature over the last 2 minutes exceeds 85F in region R1, if the average light intensity is* ≥ 80 *lumens in region R2 over the past 60 seconds, double the sampling-frequency of the temperature sensors and report the maximum readings to the Sink."*
An illustrating scenario is provided in Figure 1, where we observe that the notification of the temperature-event detection must be transmitted to the root (or cluster_head) of the nodes monitoring the light intensity which, in turn, respond with the answer of the condition-query part. An observation that motivates this work is that the current state-of-the-art in the query processing in WSN has not addressed all the aspects of processing such reactive behavior. Specifically, the reduction of the communication overhead between the (roots/cluster_heads) of the respective groups of sensors, especially when both the event and/or condition require evaluation of *continuous* phenomena over a (sliding) time-interval, has not been formally addressed.

The main contribution of this work is the introduction of the concept of *meta-triggers*. A meta-trigger is a module which compiles an ECA-like trigger into groups of triggers that are subsequently *distributed* and still guarantee behavioral/executional correctness, while providing energy-savings due to minimizations of communication overheads. As a proof of concept, we have actually programmed a small network of TelosB motes and observed the benefits of the meta-triggers on the oscilloscope (a tool readily available in the TinyOS distribution). In addition, we have conducted extensive simulation-based experiments to provide quantitative data regarding the potential benefits of the meta-triggers in large scale WSN.

The rest of this paper is structured as follows. In Section 2 we recollect some background. Section 3 introduces the meta-triggers, and presents our observation form the proof-of-concept implementation. In Section 4, we discuss the large-scale experimental observations. Section 5 positions our work with respect to the relevant literature, and Section 6 concludes the paper.

II. Preliminaries

In this section, we present a brief historic overview of the different paradigms that have, in one way or another, addressed the management of reactive behavior. Subsequently, we discuss some specifics of TinyDB that are used as a foundation for our work.

A. Overview of Formalisms

Historically, there are two extremes for specifying and managing reactive behavior:
(1) *condition* $\rightarrow$ *action* rules (IF *condition* holds, THEN execute *action*) were introduced in the Expert Systems literature (e.g., OPS5 [13]). Basically, the inference engine of the system "cycles" through the set of such rules and, whenever a left-hand side of a rule is encountered that matches the current status of the knowledge-base (KB), the action of the right-hand side of that rule would execute. Clearly, some kind of "implicit" event, along with a corresponding formalism, is needed so that the condition part of the ECA paradigm can incorporate the behavioral evolution of the database. In general, the very concept of *evolution* must be well-defined, for example: the *state* of a given instance, and what changed it. An approach offering such formal tools for database triggers, assuming a "clock-tick" as an elementary implicit-event, was presented in [14], based on temporal logic as an underlying mechanism.
(2) *event* $\rightarrow$ *action* – as another extreme, one may consider the type of rules with a missing condition part. In this case, the detection of events must be empowered with the evaluation of a particular set of facts in a given state of the database (i.e., the evaluation of the "C" part of the ECA, must be embedded within the detection of the events [15]).

More recently, the efficient management of events and queries has spurred research works in contexts different from ADBs. One example is the field of event-notification systems (ENS), in which various users can, in a sense, "subscribe" for notifications that, in turn, are generated by entities that have a role of "publishers" – all in distributed settings[16].

Various formalisms (e.g., *event algebras*) have been proposed to specify a set of *composite events*, based on the operators that are applied to the *basic/primitive events* [15]. For example, the expression $E_1; E_2$ *(sequence)* denotes a composite event which is true whenever (an instance of) E_2 is detected *after* (an instance of) E_1 has been detected. As another example, the focus of Continuous Queries (CQ) processing [17] is on efficient management of user queries over time, without forcing the users to re-issue their queries. The data values may arrive as streams which the system has to process on the fly and, furthermore, it may be multi-dimensional in nature.

B. Triggers in TinyDB

Strictly speaking, TinyDB [10] does not provide triggers in the traditional SQL sense. However, motivated by the need for efficient query processing, as part of the acquisitional query language, it provides the abilities to:
(1) Start evaluating a query (similar in spirit to *condition*) in response to a detected *event*, and
(2) Execute an *action*, pending the result of the query
Specifically, one of the types of queries identified in the TinyDB framework are the, so called, *actuation queries* which enable users to execute an action in response to a query, exemplified by:

```
SELECT nodeit, temp
FROM sensors
WHERE temp > threshold
OUTPUT ACTION power-on(nodeid)
SAMPLE PERIOD 10s.
```

This query actually states that the actuator, i.e. a cooling fan at *nodeid*, should be turned on, in response to detecting a higher temperature-level – the OUTPUT ACTION clause specifies the *external command* to be invoked. Adding an `ON EVENT` construct (cf. Section 1), could readily provide an ECA-like capability of the TinyDB. However, the fully fledged ECA paradigm has a long tradition in the ADB community [11], [12], and many issues that have been investigated there are still lacking in WSN settings: e.g., composite events; the variety of *coupling modes* between the continuous events' detection and queries evaluation; etc. For instance, the procedural semantics of the trigger-like statements in TinyDB is:
(1) When event is detected by a particular node;
(2) That node disseminates the query, specifying itself as the query root;
(3) That node collects query results, and delivers them to the basestation or a local materialization point (cf. [10]);

As we will explain in the next sections, there are aspects of the management of the ECA-like reactive behavior in WSN settings, that can yield significant energy-savings when processing users' request, without sacrificing the correctness of the executional behavior.

III. WSN and Meta-Triggers

Now we present the main results of this work. Firstly, we introduce some basic notation and define the WSN settings considered. Subsequently, we introduce the concept of the *meta-triggers*, define its role/purpose, and describe its operational behavior. Lastly, we discuss a proof-of-concept implementation that we completed for the purpose of getting some realistic observations regarding the benefits of the meta-triggers in actual small-scale WSN.

A. Network settings and requests specifications

We assume a collection $SN = \{sn_1, sn_2, \ldots, sn_N\}$ of N sensor nodes, deployed over a given area of interest, where each node $sn_i \in SN$ has a unique, fixed physical location in the 2D geographic space, represented as a pair (x_i, y_i) corresponding to the X and Y coordinates in a given reference system. Nodes are assumed to have the capability of determining their location (x_i, y_i) at run-time, either by means of a location hardware, such as a GPS device, or by implementing a location discovery algorithm [18], [19]. Each node is equipped with an omnidirectional radio device that can be used to establish communication links with other nodes within distance R – the communication range. Due to the limited spatial coverage of the radio device, each node $sn_i \in SN$ can communicate directly with only a subset of nodes from the network, which form its set of *neighbors* $NB(sn_i) = \{sn_j | d(sn_i, sn_j) \leq R, i \neq j, sn_j \in SN\}$, where $d(sn_i, sn_j)$ represents the Euclidian distance $\|(x_i, y_i) - (x_j, y_j)\|$ between the locations of the two nodes sn_i and sn_j. We also assume that the nodes are behaving in a cooperative manner [20], in the sense that no node will maliciously refuse to forward any packets.

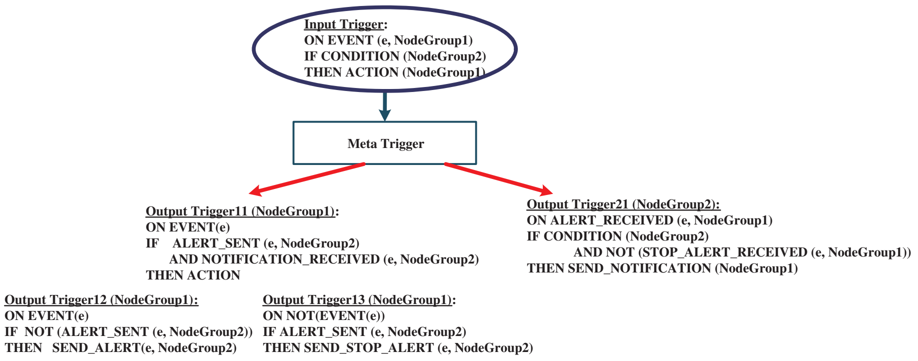
When it comes to users' requests, without loss of generality, we assume that they consist of 3 parts:
(E): *Event* – which is detected whenever a certain predicate regarding the set of measured values, over a time-interval (window), has been satisfied;
(C): *Condition* – a continuous query pertaining to a certain predicate (including aggregates) over the set of measured values, spanning over a time-interval.
(A): *Action* – which currently has an actuation-like nature over the components in the motes (e.g., adjusting the sampling frequency; activating the sensing of a particular physical value to-be-measured; etc).

Most importantly, we assume that the physical measurements whose values are used to detect the occurrence of the event "E", and/or answer of the condition "C", are being processed by *disjoint sets of nodes*. For instance, we assume that the nodes in $SN_E \subseteq SN$ are in charge of detecting the event, and the nodes in $SN_C \subseteq SN$ are in charge of monitoring the condition, and $SN_E \cap SN_C = \emptyset$. Note that this need not imply a non-overlap of the geographical regions in which SN_E and SN_G are deployed – although, in Figure 1 (Section 1), these nodes were actually physically separated in two different regions.

Hence, throughout the rest of this work, we assume that the ECA-based request will have a syntax of the form

```
Rq:  ON EVENT (NodeGroup1)
     IF QUERY (NodeGroup2)
     OUTPUT ACTION (NodeGroup1)
```

where each of the `EVENT` and `QUERY` pertains to monitoring some continuous phenomena by using disjoint sets of



```
Input Trigger:
ON EVENT (e, NodeGroup1)
IF CONDITION (NodeGroup2)
THEN ACTION (NodeGroup1)

Output Trigger11 (NodeGroup1):
ON EVENT(e)
IF   ALERT_SENT (e, NodeGroup2)
     AND NOTIFICATION_RECEIVED (e, NodeGroup2)
THEN ACTION

Output Trigger21 (NodeGroup2):
ON ALERT_RECEIVED (e, NodeGroup1)
IF CONDITION (NodeGroup2)
          AND NOT (STOP_ALERT_RECEIVED (e, NodeGroup1))
THEN SEND_NOTIFICATION (NodeGroup1)

Output Trigger12 (NodeGroup1):
ON EVENT(e)
IF  NOT (ALERT_SENT (e, NodeGroup2))
THEN   SEND_ALERT(e, NodeGroup2)

Output Trigger13 (NodeGroup1):
ON NOT(EVENT(e))
IF ALERT_SENT (e, NodeGroup2)
THEN SEND_STOP_ALERT (e, NodeGroup2)
```

Fig. 2. The Meta-Trigger Module

sensor nodes `(NodeGroup1)` and `(NodeGroup2)`. The `OUTPUT ACTION` part is executed by `NodeGroup1`. where `NodeGroup1` $\cap$ `NodeGroup2` $= \emptyset$.

B. The Meta-Trigger Module

The main motivation behind the meta-triggers is that, due to the continuous nature of the events and the conditions, there may be a large communication overhead between the nodes in `NodeGroup1` and `NodeGroup2`.

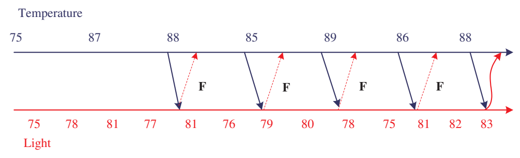


Fig. 3. Communication Pattern Example

An illustrating scenario is presented in Figure 3. Recalling the request **Rq1** (cf. Section1), the top part of the figure shows the average temperature values sampled every 60 seconds by the temperature-sensors, corresponding to `NodeGroup1`. On the other hand, the bottom part of the Figure illustrates the average of the light readings obtained by the light-sensors (`NodeGroup2`), sampled every 30 seconds. What can be observed is that the root/cluster_head of the `NodeGroup1` has sent a total of *five* messages to the root/cluster_head of the `NodeGroup2` requesting the value of the average readings in the last 1 minute, and the first *four* such messages resulted in *False* (F).

Clearly, the source of the problem is that the processing of the trigger is done in the *pull* mode, in the sense that with every new *primitive* temperature-event, that satisfies the *composite continuous* temperature event, `NodeGroup1` needs to know whether the condition of the trigger has been satisfied. However, with a slight modification of the original trigger, the entire communication between the `NodeGroup1` and the `NodeGroup2` can be set to a *push* mode, thereby reducing the traffic of unnecessary messages.

The *meta-trigger module* is essentially a translator which: (1) Takes a trigger-like request (e.g., **Rq1**) as an input; (2) Generates a collection of triggers as an output.

The triggers thus generated introduce some additional events, that may be derived from the one(s) in the input trigger. An illustration of the meta-trigger module is presented in Figure 2. As shown, as a result of the input trigger, four new triggers are generated in the output, three of which (*OutputTrigger11, OutputTrigger12* and *OutputTrigger13*) pertain to the `NodeGroup1`, and one (*OutputTrigger21*) pertains to the `NodeGroup2`. The new events are as follows:

- `ALERT_SENT(NodeGroup2)`, associated with *OutputTrigger11*, signaling that when the `EVENT` was detected for the first time, a notification was also sent to the `NodeGroup2` that the result of the *Condition* is needed;
- `NOTIFICATION_RECEIVED`, associated with *OutputTrigger11* is an event generated by the `NodeGroup2` (due to *OutputTrigger12*, signaling that the result of the *Condition* is T (true).
- `ALERT_RECEIVED` is a local event of *OutputTrigger21*, signifying that the `NodeGroup1` has requested a notification whenever the *Condition* is satisfied. Note that the *OutputTrigger21* will send the notification only if the outcome of the *Condition* is still of interest for the `NodeGroup1`, which is indicated via the event *NOT (STOP_ALERT_RECEIVED)*, as a result of *SEND_STOP_ALERT* (cf. *OutputTrigger13*). Also note that the main role of the *OutputTrigger12* and *OutputTrigger13* is to ensure that the `NodeGroup2` is properly notified that the value of *Condition* that they monitor is or is no-longer needed. Lastly, observe that the events are *parameterized* for the purpose of distinguishing among different instances of the same type of event (denoted by `e` in Figure 2).

C. Proof of Concept

We have implemented the ECA-like triggers in nesC, on top of TinyOS[1]. Our network setup, illustrated in Figure 4, consisted of seven TelosB motes, one of which was the sink, attached to a PC. For this setup, we have applied the principles of the meta-trigger, and generated an equivalent collection of nesC programs representing the set of *OutputTriggers*, which we used to program the TelosB motes. To monitor the effects, we used the *Oscilloscope* application, readily

[1]Publicly available at *http://www.tinyos.net/*

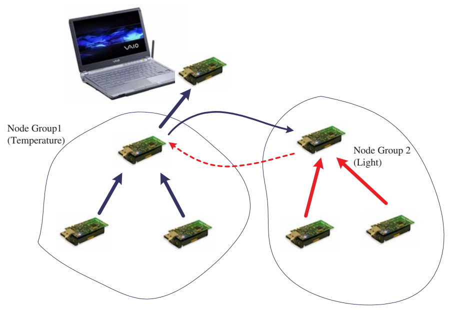


Fig. 4. ECA-triggers in nesC – TelosB motes (Configuration)

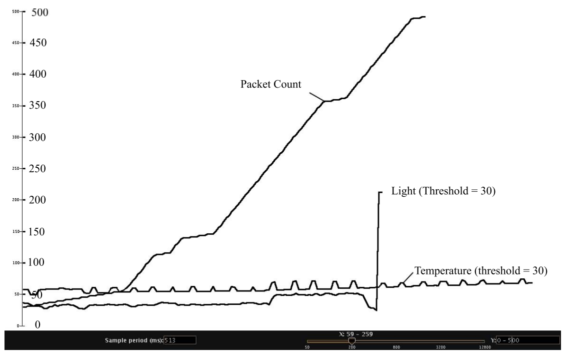


Fig. 5. ECA-triggers execution (pull mode)

available with the TinyOS distribution. Figure 5 illustrates the communication load when the ECA-like behavior is executed in a "brute-force" (pull) manner, whereas Figure 6 shows large decrease in the communication when the meta-trigger is employed to ensure a correct execution in a push-based manner[2].

IV. Large Scale Experimental Evaluation

We performed our experiments for large networks using SIDnet-SWANS [21], a simulator and integrated development environment for wireless sensor network, built on top of the JiST-SWANS [22]. The simulation testbed contains 500 nodes, randomly distributed using a uniform distribution function in an area of 15000x15000 sqft. Nodes are homogeneous, sharing the same configuration: 40000 bps transmission/reception rate, 5 seconds time-to-sleep interval and power consumption characteristics that are based on the Mica2 Motes specs, which, for completeness, are summarized in Table I. A small battery powers each node, with an initial capacity of 35 mAh which, given the power consumption characteristics from Table I, is expected to provide energy to a node for several tens of hours, depending on the load. A smaller battery (in terms of initial capacity) has been chosen in order to reduce the simulation time while still preserving the correctness of the

[2]Since the original *Oscilloscope* application provides a black background, for clarity we are actually showing the "negative" in Figure 5 and 6. The actual screen-shots, as well as the source code of the implementation is available at http://www.eecs.northwestern.edu/˜goce/MetaTriggers.

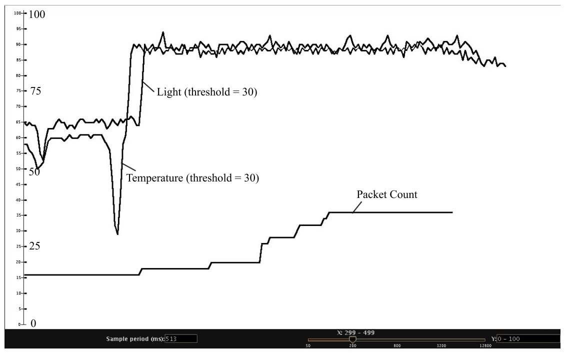


Fig. 6. ECA-triggers execution (push mode)

experimental results. The experimental platform was Intel Dual Core 3.2GHz Extreme Ed. with 2GB RAM.

TABLE I
Energy characteristics of *Mica2 Mote (MPR500CA)*

State	Based on	Energy requirement
Sensing Active	$I_s = 10mA$	$0.03mJ/ms$
Sensing Passive	$I_s = 0mA$	$0mJ/ms$
CPU Active	$I_p = 8mA$	$0.024mJ/ms$
CPU Idling	$I_i = 0.015mA$	$4.5 * 10^{-5}mJ/ms$
RADIO Transmitting	$I_t = 27mA$	$0.081mJ/ms$
RADIO Receiving	$I_r = 10mA$	$0.03mJ/ms$
RADIO Listening	$I_l = 3mA$	$0.009mJ/ms$
RADIO Off-Mode	$I_{slp} = 0.5mA$	$0.0015mJ/ms$

The triggers span across two distinct, equally sized regions of the network ($R1$ and $R2$), similar to the illustration provided in Figure 1. A TAG-based [23])aggregation tree was built in each of the regions. The data values from $R1$ and $R2$ are aggregated periodically for the duration of the query and used for evaluation of the *Event* (specified as $Q1(t,T) > Q_{th1}$), where T is the duration of the time-window (fixed to 60 seconds) and the corresponding *Condition* (IF $Q2(t,T) > Q_{th2}$). The thresholds Q_{th_1} and Q_{th_2} are expressed as percentages relative of the maximum of the measurement-values for the corresponding phenomena.

Experiments are based on a configuration space which takes into consideration: – different sampling rates – different thresholds for Q_{th1} and Q_{th2}; – and different patterns of the phenomena fluctuations. A summary of the experimental space is given in Table II.

TABLE II
Summary of Experimental Configuration Space

Query Processing Mode	Sampling Interval seconds	Phenomena speed	Q_{th_1} $>$	Q_{th_2} $>$
PULL	5	1x	25% (L)	25% (L)
PUSH	20	10x	50% (M)	50% (M)
		100x	75% (H)	75% (H)

All the possible combination of Q_{th_1} and Q_{th_2} have been considered. The average measurement-values of the phenomena over the entire network, throughout the duration of the

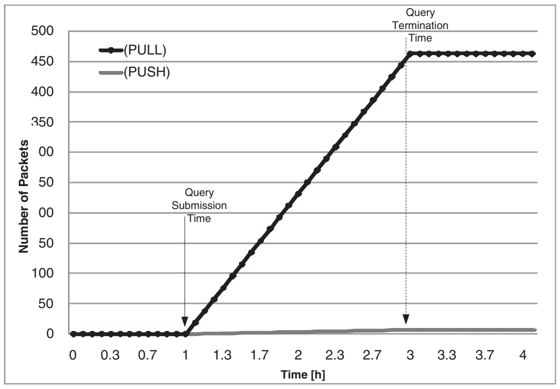


Fig. 7. Overall performance in terms of the number of messages, PULL vs. PUSH modes

experiments is 50%, with isolated samples covering the full scale. The phenomena "speed" is expressed in terms of a time-constant, with the following meaning: – "1x" represents the base time-constant of 10 minutes; – "10x" represents the mode in which the fluctuations of the values of the physical phenomena are changing 10-times faster (e.g., every minute); – "100x" respectively represents a speed-up of the phenomena by reducing the base time-constant 100-times. The total number of experiments, corresponding to each possible combination of settings and conforming to Table II within one run, is 108. These experiments have been repeated 50 times with a different, but still uniformly distributed, node placement. Thus, a total of 5400 experiments were performed.

Figure 7 confirms the observations obtained in the small-scale WSN consisting of TelosB motes (cf. Section3), regarding the benefits of the meta-trigger concept in terms of the number of packets that were communicated between the roots of the $R1$ (Event) and $R2$ (Condition) during processing of a trigger. Clearly, in the "PULL" settings, which do not incorporate the meta-trigger, the number of packets that were communicated through the network grows linearly with the time, very much along the lines of our TinyOS implementation (cf. Figure 5). On the other hand, the "PUSH" mode (meta-triggers used) yields almost-constant communication overheads throughout the duration of the time-interval of interest.

Figure 8 and Figure 9 show the impact of the threshold values Q_{th1} and Q_{th2} of the phenomena, in terms of detection of the events' occurrence, and the condition evaluation, with and without the application of the meta-triggers. Looking at the rightmost part of each Figure (Q_1 enabled 75% of the time, with a *Low* threshold-value), we observe that when Q_{th2} is low, the benefits of the meta-triggers in terms of packets-transmission range within a factor of 50-100.

The next group of the experiments, illustrated in Figure 10 and Figure 11 illustrates the benefits of the meta-triggers in terms of "sensitivity" of the communication overhead, with respect to the frequency of fluctuations of the values of the phenomena that generate the detection of the event and the satisfiability of the query predicate. As can be seen,

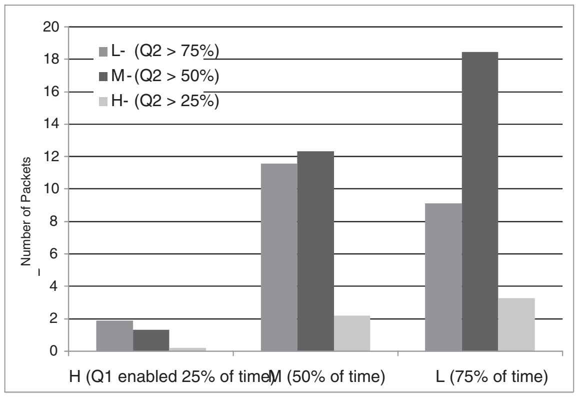


Fig. 8. Meta-trigger processing performance (PUSH mode), based on the various thresholds-values for Q1 and Q2 components of the query

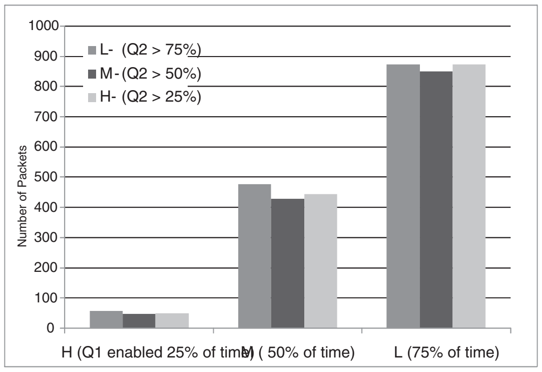


Fig. 9. Trigger processing performance in PULL mode, based on the various thresholds of Q1 and Q2 components of the query

the PUSH-mode resulting from the application of the meta-triggers is indeed growing proportionally to the number of the fluctuations, whereas the PULL-mode not only completely insensitive, but still has an order of magnitude higher overhead, while transmitting almost constant high amount of packets.

Figure 12 illustrates yet another benefit of the meta-triggers – in terms of the missed notifications. Namely, due to "race" conditions, it may be very possible that:

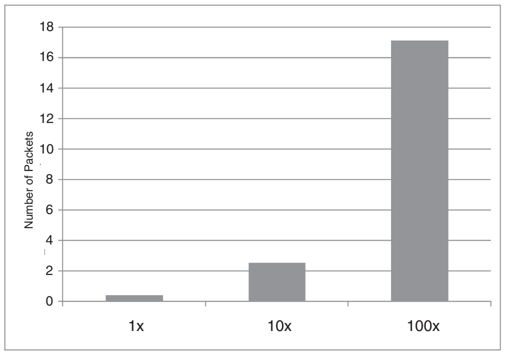


Fig. 10. Triggers processing performance vs. fluctuation-frequency of the phenomena (meta-triggers; PUSH mode)

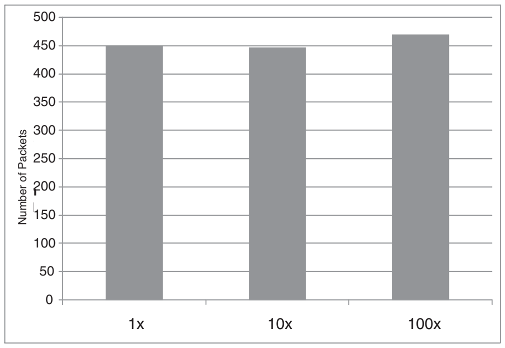


Fig. 11. Triggers processing performance vs. fluctuation-frequency of the phenomena (PULL mode)

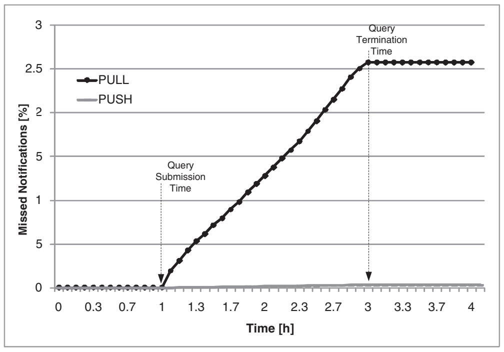


Fig. 12. Missed notifications (percentage of false-negatives a specific implementation is prone to)

(1) At $t = t1$, the sensors that are in charge of evaluating the condition part have received the *NOTIFICATION_REQUEST*;
(2) Upon evaluation (completed, e.g., at $t = t_1 + \varepsilon$), it was found that the condition is *False*, which is transmitted back to the sensors monitoring the event.
(3) The condition changes from *False* to *True* at some time $t = t_2 \geq (t_1 + \varepsilon)$
(4) The notification received at $t = t_2 + \delta$ by the sensors evaluating the condition is *STOP_ALERT_RECEIVED*.

Under this scenario, a firing of the action that should have occurred at t_2 has been missed, which is a false-negative type of error. As shown in Figure 12, the number of false-negatives grows proportionally with the time in the PULL-mode, whereas it is almost constant in the PUSH-mode when meta-triggers are used. A detailed illustration of the miss-ratio for the different frequencies of the phenomena-fluctuation is presented in Figure 13.

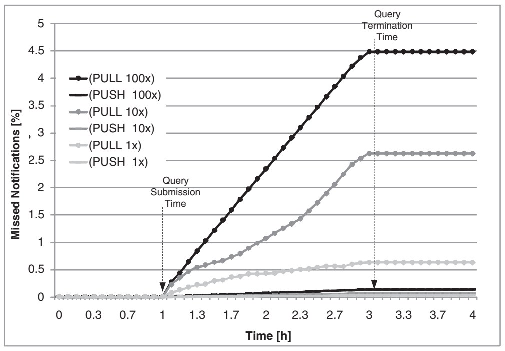


Fig. 13. Missed-notifications percentage for different frequencies of phenomena fluctuation

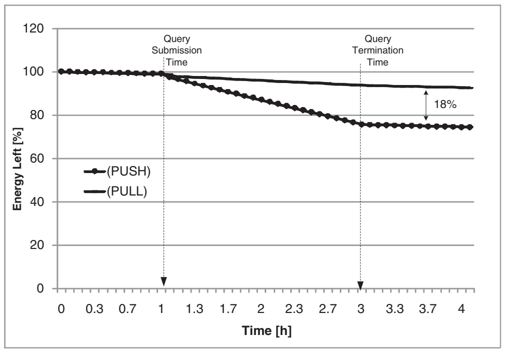


Fig. 14. Energy benefits of meta-triggers in the relay nodes

The last experimental evaluation that we present actually gives an idea of the energy-savings induced by the meta-triggers. Namely, when processing a trigger, it is not only the nodes that detect the event and evaluate the query that are spending energy due to the communication. In addition, whenever the corresponding roots of the trees in each group of nodes need to exchange packets, the relay-nodes on the route between them are also subject to energy-consumption. Figure 14 illustrates the amount of the residual energy left in the relay-nodes between the respective roots (averaged over all simulation runs). As can be seen, after 3 hours of continuous operation, the meta-triggers yield almost 20% higher energy-reserves.

V. Related Work

The topic of active databases has been extensively studied for a long time [24], [25], [26], [11], [12] and various aspects have been investigated: – *termination and confluence* [27]; – *coupling modes* between transactions which generated events vs. condition evaluation and actions execution [26]; – *event processing and consumption* [28], [29]; – *expressiveness* issues [30] and semantics of the active rules behavior, for which several formalisms have been used, e.g., action theories [31] and temporal logic [14]. In particular, the *Extended Event-Condition-Action (EECA)* model in [26] took a step towards more "event-aware" active rules. However, most of these work did not consider the peculiarities of the WSN settings, where the communication among the nodes had to be considered as an important factor of the overall processing.

The application of the active rules in WSN was proposed in [32], where the authors suggested the potential benefits of the ECA-like triggers, however, the work that is closest in spirit to ours, is the TinyDB project [10]. As we already mentioned, the TinyDB framework does provide a reactive behavior that is very similar to the ECA paradigm, except the main focus of the work was on efficient *query processing*. Hence, the specific issues of the triggers (syntax, semantics, efficient management of communication) were not investigated in detail. Specifically, the problem of communication overheads between the nodes that are responsible for the detection of an event and the ones that are in charge of evaluating the condition, was not addressed from the perspective that we presented in this paper.

VI. Concluding Remarks and Future Work

We addressed the problem of efficient management of the reactive behavior in WSN for the case when it is specified via rules conforming to the ECA paradigm. Specifically, we focused on the settings in which both the *event* and the *condition* are continuous, in the sense that their validity/values change over time. We argued that one aspect that cannot be neglected, when it comes to the overall energy consumption in the network, is the communication between the two sets of nodes that are processing the detection of the events and the evaluation of the queries. Towards that end, we proposed the concept of the *meta-trigger*, a module that translates the ECA-based trigger into a collection of triggers that cooperatively execute the intended behavior, while reducing the communication overhead. We presented a proof-of-concept implementation demonstrating the feasibility of the proposed methodology, and we also showed through extensive simulation-experiments that the meta-triggers can indeed bring substantial savings in the number of transmitted packets which, implicitly, reduces the energy expenses of the nodes, thereby prolonging the network lifetime.

There are several extensions of the current work that we are planning to address in the future. Firstly, we would like to build the trigger-like capabilities (in terms of syntax) on top of the available SQL syntax of TinyDB [10], and integrate the meta-trigger module with the overall query processing "engine" of the TinyDB. In addition, we would also like to extend the ECA-like triggers in WSN settings with the higher-level $(ECA)^2$ (Evolving and Context-Aware Event Condition Action) triggers [33]. In the lieu of the recent works on bringing the logic-based paradigm in the WSN context, we hope to extend the existing efforts in Declarative Networking [34] with the geometric constructs (e.g., *Region R*), that can be incorporated in the queries and triggers processing.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, 2002.

[2] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, 2005.

[3] Q. Dong, "Maximizing system lifetime in wireless sensor networks," in *IPSN*, 2005.

[4] J. Park and S. Sahni, "Maximum lifetime routing in wireless sensor networks." *IEEE/TOC*, vol. 55, no. 3, 2006.

[5] K. Kalpakis, K. Dasgupta, and P. Namjoshi, "Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks." *Computer Networks*, vol. 42, no. 6, pp. 697–716, 2003.

[6] I. Kadayif and M. T. Kandemir, "Tuning in-sensor data filtering to reduce energy consumption in wireless sensor networks." in *DATE*, 2004, pp. 852–857.

[7] S. Lin, D. Gunopulos, V. Kalogeraki, and S. Lonardi, "A data compression technique for sensor networks with dynamic bandwidth allocation." in *TIME*, 2005, pp. 186–188.

[8] C.-W. Shiou, F. Y.-S. Lin, H.-C. Cheng, and Y.-F. Wen, "Optimal energy-efficient routing for wireless sensor networks." in *AINA*, 2005, pp. 325–330.

[9] A. Woo, S. Madden, and R. Govindan, "Networking support for query processing in sensor networks," *Communications of the ACM*, vol. 47, no. 6, 2004.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM TODS*, vol. 30, no. 1, 2005.

[11] N. Paton, Ed., *Active Rules in Database Systems*. Springer-Verlag, 1999.

[12] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[13] L.Brownston, K.Farrel, E.Kant, and N.Martin, *Programming Expert SystemsinOPS5: An Introduction to Rule-Base Programming*, 2005.

[14] P. Sistla and O. Wolfson, "Temporal conditions and integrity constraint checking in active database systems," in *ACM SIGMOD*, 1995.

[15] R. Adaikkalavan and S. Chakravarthy, "Formalization and detection of events using interval-based semantics," in *COMAD*, 2005.

[16] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *ACM-TOCS*, vol. 19, no. 3, 2001.

[17] J. Chen, D. DeWitt, F. Tian, and Y. Wang, "Niagaracq: A scalable continuous query system for internet databases," in *ACM SIGMOD Conference*, 2000.

[18] R. Nagpal, H. E. Shrobe, and J. Bachrach, "Organizing a global coordinate system from local information on an ad hoc sensor network." in *IPSN*, 2003, pp. 333–348.

[19] L. Fang, W. Du, and P. Ning, "A beacon-less location discovery scheme for wireless sensor networks." in *INFOCOM*, 2005, pp. 161–171.

[20] V. Srinivasan, P. Nuggehalli, C.-F. Chiasserini, and R. R. Rao, "Cooperation in wireless ad hoc networks." in *INFOCOM*, 2003.

[21] O. Ghica, G. Trajcevski, P. Scheuermann, Z. Bischoff, and N. Valtchanov, "Sidnet-swans: A simulator and integrated development platform for sensor networks applications," in *SenSys*, 2008.

[22] "Jist – java in simulation time / swans," http://jist.ece.cornell.edu, 2005.

[23] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad hoc sensor network," in *Proc. Fifth Symp. on Operating Systems Design and Implementation, USENIX OSDI*, 2002.

[24] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca, "Automatic generation of production rules for integrity maintenance," *ACM Transactions on Database Systems*, vol. 19, no. 3, pp. 367–422, 1994.

[25] U. Dayal, E. Hansen, and J. Widom, "Active database systems," in *Modern Database Systems: The Object Model, Interoperability and Beyond*, W. Kim, Ed. Addison–Wesley, 1994.

[26] P. Fraternali and L. Tanca, "A structured approach for the definition of the semantics of active databases," *Transactions on Database Systems*, vol. 20, no. 4, 1995.

[27] S. Urban, M. Tschudi, S. Dietrich, and A. Karadimce, "Active rule termination analysis: An implementation and evaluation of refined triggering graph method," *JIIS*, vol. 12, no. 1, 1999.

[28] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. Kim, "Composite events for active databases: Semantics, contexts and detection," in *20th VLDB Conference*, 1994.

[29] I. Motakis and C. Zaniolo, "Formal semantics for composite temporal events in active database rules," *JOSI*, vol. 7, no. 3, 1997.

[30] P. Picouet and V. Vianu, "Semantics and expressiveness issues in active databases," in *Priniciples of Database Systems*, 1995, full version 1996.

[31] C. Baral, J. Lobo, and G. Trajcevski, "Formal characterization of active databases: Part ii," in *DOOD*, 1997.

[32] A. Zomboulakis, G. Roussos, and A. Poulovassilis, "Eca rules for sensor networks," in *ACM SenSys*, 2004.

[33] G. Trajcevski, P. Scheuermann, O. Ghica, A. Hinze, and A. Voisard, "Evolving triggers for dynamic environments," in *Extending Database Technology (EDBT)*, 2006.

[34] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative networking: language, execution and optimization," in *SIGMOD Conference*, 2006.