Complex Event Processing in Wireless Sensor Networks

Omran Saleh
Faculty of Computer Science and Automation
Ilmenau University of Technology
Ilmenau, Germany
omran.saleh@tu-ilmenau.de

ABSTRACT

Most of the WSN applications need the number of sensor nodes deployed to be in order of hundreds, thousands or more to monitor certain phenomena and capture measurements over a long period of time. The large volume of sensor networks would generate continuous streams of raw events[1] in case of centralized architecture, in which the sensor data captured by all the sensor nodes is sent to a central entity.

In this paper, we describe the design and implementation of a system that carries out complex event detection queries inside wireless sensor nodes. These queries filter and remove undesirable events. They can detect complex events and meaningful information by combining raw events with logical and temporal relationship, and output this information to external monitoring application for further analysis. This system reduces the amount of data that needs to be sent to the central entity by avoiding transmitting the raw data outside the network. Therefore, it can dramatically reduce the communication burden between nodes and improve the lifetime of sensor networks.

We have implemented our approach for the TinyOS Operating System, for the TelosB and Mica2 platforms. We conducted a performance evaluation of our method comparing it with a naive method. Results clearly confirm the effectiveness of our approach.

Keywords

Complex Event Processing, Wireless Sensor Networks, In-network processing, centralized processing, Non-deterministic Finite state Automata

1. INTRODUCTION

Wireless sensor networks are defined as a distributed and cooperative network of devices, denoted as sensor nodes that are densely deployed over a region especially in harsh environments to gather data for some phenomena in this monitored region. These nodes can sense the surrounding environment and share the information with their neighboring nodes. They are gaining adoption on an increasing scale for tracking and monitoring purposes. Furthermore, sensor nodes are often used in control purposes. They are capable of performing simple processing.

In the near future, it is prospective that wireless sensor networks will offer and make conceivable a wide range of applications and emerge as an important area of computing. WSN technology is exciting with boundless potential for various application areas. They are now found in many industrial and civilian application areas, military and security applications, environmental monitoring, disaster prevention and health care applications, etc.

One of the most important issues in the design of WSNs is energy efficiency. Each node should be as energy efficient as possible. Processing a chunk of information is less costly than wireless communication; the ratio between them is commonly supposed to be much smaller than one [19]. There is a significant link between energy efficiency and superfluous data. The sensor node is going to consume unnecessary energy for the transmission of superfluous data to the central entity, which means minimizing the energy efficiency.

Furthermore, traditional WSN software systems do not apparently aim at efficient processing of continuous data or event streams. According to previous notions, we are looking for an approach that makes our system gains high performance and power saving via preventing the generation and transmission of needless data to the central entity. Therefore, it can dramatically reduce the communication burden between nodes and improve the lifetime of sensor networks. This approach takes into account the resource limitations in terms of computation power, memory, and communication. Sensor nodes can employ their processing capabilities to perform some computations. Therefore, an in-network complex event processing [2] based solution is proposed.

We have proposed to run a complex event processing engine inside the sensor nodes. CEP engine is implemented to transform the raw data into meaningful and beneficial events that are to be notified to the users after detecting them. It is responsible for combining primitive events to identify higher level complex events. This engine provides an efficient Non-deterministic Finite state Automata (NFA) [1] based implementation to lead the evaluation of the complex event queries where the automaton runs as an integral part of the in-network query plan. It also provides the theoretical basis of CEP as well as supports us with particular

[1]The terms data, events and tuples are used interchangeably.

[2]CEP is discussed in reference [15]

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

operators (conjunction, negation, disjunction and sequence operators, etc.).

Complex event processing over data stream has increasingly become an important field due to the increasing number of its applications for wireless sensor networks. There have been various event detection applications proposed in the WSNs, e.g. for detecting eruptions of volcanoes [18], forest fires, and for the habitat monitoring of animals [5]. An increasing number of applications in such networks is confronted with the necessity to process voluminous data streams in real time fashion.

The rest of the paper is organized as follows: section 2 provides an overview of the naive approaches for normal data and complex event processing in WSNs. Related works are briefly reviewed in section 3. Then we introduce the overall system architecture in order to perform complex event processing in sensor networks in section 4. Section 5 discusses how to create logical query plans to evaluate sensor portion queries. Section 6 explains our approach and how queries are implemented by automata. In section 7, the performance of our system is evaluated using a particular simulator. Finally, section 8 presents our concluding remarks and future works.

2. NAIVE APPROACHES IN WSNS

The ideas behind naive approaches which are definitely different from our approach lie in the processing of data as the central architectural concept. For **normal sensor data processing**, the centralized approach proceeds in two steps; the sensor data captured by all the sensor nodes is sent to the sink node and then routed to the central server (base station) where it is stored in centralized database. High volume data are arriving at the server. Subsequently, query processing takes place on this database by running queries against stored data. Each query executes one time and returns a set of results.

Another approach which adopts the idea of centralized architecture is the use of a central data stream management system (DSMS), which simply takes the sensor data stream as input source. Sending all sensor readings to DSMS is also an option for WSN data processing. DSMS is defined as a system that manages a data stream, executes a continuous query against a data stream and supports on-line analysis of rapidly changing data streams [10]. Traditional stream processing systems such as Aurora [2], NiagraCQ [7], and AnduIN [12] extend the relational query processing to work with stream data. Generally the select, project, join and aggregate operations are supported in these stream systems.

The naive approach for **Complex Event Processing** in WSNs is similar to the central architectural idea of normal data processing, but instead of using traditional database and data stream engine, CEP uses a dedicated engine for processing complex events such as Esper [8], SASE [11] and Cayuga [4], in which sensor data or events streams need to be filtered, aggregated, processed and analyzed to find the events of interest and identify some patterns among them, finally take actions if needed.

Reference [11] uses SASE in order to process RFID stream data for a real-world retail management scenario. Paper [3] demonstrates the use of Esper engine for object detection tracking in sensor networks. All the aforementioned engines use some variant of a NFA model to detect the complex event. Moreover, there are many CEP engines in the field of active databases. Most of the models in these engines are based on fixed data structures such as tree, graph, finite automaton or petri net. The authors of [6] used a tree based model. Paper [9] used petri net based model to detect complex events from active database. Reference [17] used Timed Petri-Net (TPN) to detect complex events from RFID stream.

3. RELATED WORKS

It is preferable to perform **In-Network Processing** inside sensor network to reduce the transmission cost between neighboring nodes. This concept is proposed by several systems such as TinyDB [16], and Cougar [19]. Cougar project applies a database system concept to sensor networks. It uses the declarative queries that are similar to SQL to query sensor nodes. Additionally, sensor data in cougar is considered like a “virtual” relational database. Cougar places on each node an additional query layer that lies between the network and application layers which has the responsibility of in-network processing. This system generates one plan for the leader node to perform aggregation and send the data to a sink node. Another plan is generated for non-leader nodes to measure the sensors status. The query plans are disseminated to the query layers of all sensor nodes. The query layer will register the plan inside the sensor node, enable desired sensors, and return results according to this plan.

TinyDB is an acquisitional query processing system for sensor networks which maintains a single, infinitely-long virtual database table. It uses an SQL-like interface to ask for data from the network. In this system, users specify the data they want and the rate at which the data should be refreshed, and the underlying system would decide the best plan to be executed. Several in-network aggregation techniques have been proposed in order to extend the life time of sensor network such as tree-based aggregation protocols i.e., directed diffusion.

Paper [13] proposes a framework to detect complex events in wireless sensor networks by transforming them into sub-events. In this case, the sub-events can easily be detected by sensor nodes. Reference [14] splits queries into server and node queries, where each query can be executed. The final results from both sides are combined by the results merger. In [20], symbolic aggregate approximation (SAX) is used to transform sensor data to symbolic representations. To detect complex events, a distance metric for string comparison is utilized. These papers are the closer works to our system.

Obviously, there is currently little work into how the idea of in-network processing can be extended and implemented to allow more complex event queries to be resolved within the network.

4. SYSTEM ARCHITECTURE

We have proposed a system architecture in which collected data at numerous, inexpensive sensor nodes are processed locally. The resulting information is transmitted to larger, more capable and more expensive nodes for further analysis and processing through specific node called sink node.

The architecture has three main parts that need to be modified or created to make our system better suited to queries over sensor nodes: 1- **Server side**: queries will be originated at server side and then forwarded to the nearest sink node. Additionally, this side mainly contains an application that runs on the user's PC (base station). Its main purpose is to collect the results stream over the sensor network and display them. Server side application can offer more functions i.e., further filtering for the collected data, perform joining on sensor data, extract, save, manage, and search the semantic information and apply further complex event processing on incoming events after processing them locally in sensor nodes. Because sensor data can be considered as a data stream, we proposed to use a data stream management system to play a role of server side, for that we selected AnduIN data stream engine. 2- **Sink side**: sink node (also known as root or gateway node) is one of the motes in the network which communicates with the base station directly, all the data collected by sensors is forwarded to a sink node and then to server side. This node will be in charge of disseminating the query down to all the sensor nodes in the network that comes from server side. 3- **Node side**: in this side, we have made huge changes to the traditional application which runs on the nodes themselves to enable database manner queries involving filters, aggregates, complex event processing operator (engine) and other operators to be slightly executed within sensor networks. These changes are done in order to reduce communication costs and get useful information instead of raw data.

When combining on-sensor portions of the query with the server side query, most of the pieces of the sensor data query are in place. This makes our system more advanced.

5. LOGICAL PLAN

Each and every sensor node of a network generates tuples. Every tuple may consist of information about the node id, and sensor readings. Query plan can specify the tuples flow between all necessary operators and a precise computation plan for each sensor node. Figure 1 (lower plan) illustrates how our query plan can be employed. It corresponds to an acyclic directed graph of operators. We assume the dataflow being upward. At the bottom, there is a homogeneous data source which generates data tuples that must be processed by operators belonging to query plans. Tuples are flowed through intermediate operators composed in the query graph. The operators perform the actual processing and eventually forward the data to the sink operator for transmitting the resulting information to the server side (base station). These operators adopt publish/subscribe mechanism to transfer tuples from one operator to next operator.

We differ between three different types of operators within a query graph [12]: 1- **Source operator**: produces tuples and transfers them to other operators. 2- **Sink operator**: receives incoming tuples from other operators. 3- **Inner operators**: receive incoming tuples from source operator, process them, and transfer the result to sink operator or other inner operators.

A query plan consists of one source at the bottom of a logical query graph, several inner operators, and one sink at the top and the tuples are flowing strictly upward. In our system, we have extended this plan to give the system the capability to perform the complex event processing and detecting by adding new operators. We have separated the mechanism for detecting complex events from the rest of normal processing side. We have a particular component working as an extra operator or engine within the main process, as we can see from figure 1 (upper plan). The detection

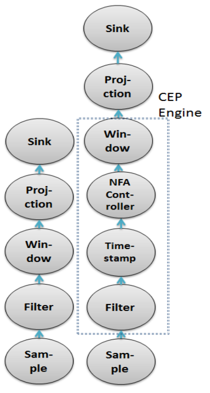

Figure 1: Logical query plan

mechanism takes as input primitive events from lower operators and detects occurrences of composite events which are used as an output to the rest of the system.

6. IN-NETWORK CEP SYSTEM

Various applications including WSNs require the ability to handle complex events among apparently unrelated events and find interesting and/or special patterns. Users want to be notified immediately as soon as these complex events are detected. Sensor node devices generate massive sensor data streams. These streams generate a variety of primitive events continuously. The continuous events form a sequence of primitive events, and recognition of the sequence supplies us a high level event, which the users are interested in.

Sensor event streams have to be automatically filtered, processed, and transformed into significative information. In non-centralized architecture, CEP has to be performed as close to real time as possible (inside the node). The task of identifying composite events from primitive ones is performed by the Complex Event Processing engine. CEP engine provides the runtime to perform complex event processing where they accept queries provided by the user, match those queries against continuous event streams, and trigger an event or an execution when the conditions specified in the queries have been satisfied. The idea of this concept is close to Event-Condition-Action (ECA) concept in conventional database systems where an action has to be carried out in response to an event and one or more conditions are satisfied.

Each data tuple from the sensor node is viewed as a primitive event and it has to be processed inside the node. We have proposed an event detection system that specifically targets applications with limited resources, such in our system. There are four phases for complex event processing in our in-network model: NFA creation, Filtering, Sequence scan and Response as shown in figure 2.

6.1 NFA Creation Phase

The first phase is NFA creation. NFA's structure is created by the translation from the sequence pattern through mapping the events to NFA states and edges, where the conditions of the events (generally called event types) are associated with edges. For pattern matching over sensor node streams, NFA is employed to represent the structure of an event sequence. For a concrete example, consider the query

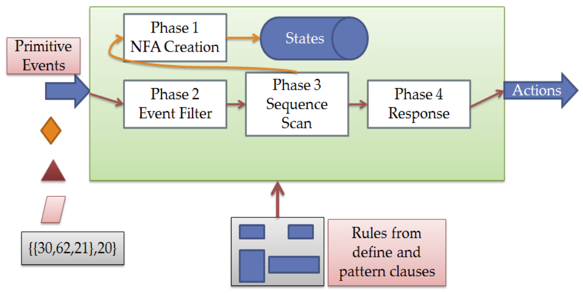

Figure 2: CEP Phases

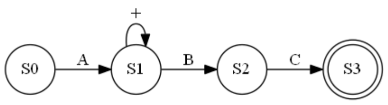

Figure 3: NFA for SEQ(A a, B+ b, C c)

pattern: **SEQ(A a, B+ b, C c)**[3]. Figure 3 shows the NFA created for the aforementioned pattern **(A, B+, C)**, where state $S0$ is the starting state, state $S1$ is for the successful detection of an A event, state $S2$ is for the detection of a B event after event A, also state $S3$ is for the detection of a C event after the B event. State $S1$ contains a self-loop with the condition of a B event. State $S3$ is the accepting state, reaching this state indicates that the sequence is detected.

6.2 Filtering Phase

The second phase is to filter primitive events at early stage, generated by sensor nodes. Sensor nodes cannot understand whether a particular event is necessary or not. When additional conditions are added to the system, possible event instances might be pruned at the first stage.

After filtering, timestamp operator will add the occurrence time of the event t. A new operator is designed for adding a timestamp t to the events (tuples) before entering the complex event processing operator. We can notice that from figure 1. The timestamp attribute value of an event t records the reading of a clock in the system in which the event was created, in this case it can reflect the true order of the occurrences of primitive events.

6.3 Sequence Scan Phase

The third phase is sequence scan to detect a pattern match. We have three modes state the way in which events may contribute to scan a sequence: UNRESTRICTED, RECENT and FIRST. Every mode has a different behavior. The selection between them depends on the users and the application domain. These modes have advantages and disadvantages. We will illustrate them below.

In the UNRESTRICTED mode, each start event e, which allows a sequence to move from the initial state to the next state, starts a separate sequence detection. In this case any event occurrence combination that matches the definition of the sequence can be considered as an output. By using this mode, we can get all the possibilities of event combination which satisfy the sequence. When the sequence is created, it

[3]Notice: In this paper, we are going to only focus on sequence operator **SEQ** because of the limited number of pages.

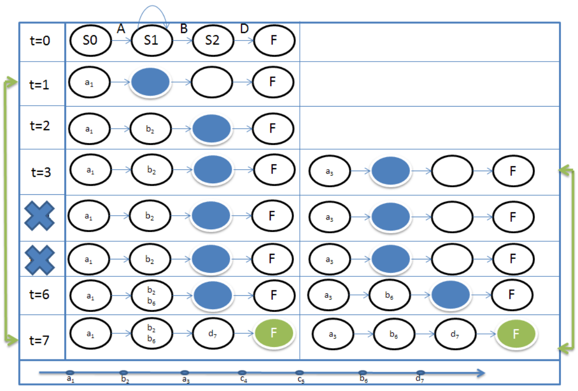

Figure 4: Sequence Scan for SEQ (A, B+, D) within 6 Time Unit Using UNRESTRICTED Mode

is waiting for the arrival of events in its starting state. Once a new instance event e arrives, the sequence scan responds as follows: 1- It checks whether the type of instance (from attributes) and occurrence time of e satisfy a transition for one of the logical existing sequences. If not, the event is directly rejected. 2- If yes, e is registered in the system (the registration is done in the sliding window) and the sequence advances to next state. 3- If e allows for a sequence to move from the starting state to next state, the engine will create other logical sequence to process further incoming events while keeping the original sequence in its current state to receive new event. Therefore, multiple sequences work on the events at the same time. 4- Delete some sequences when their last received items are not within a time limit. It becomes impossible for them to proceed to the next state since the time limits for future transitions have already expired.

Next, we use an example to illustrate how UNRESTRICTED sequence scan works. Suppose we have the following pattern[4] **SEQ (A, B+, D)** and sequence of events (tuples) presented as **[a1, b2, a3, c4, c5, b6, d7 ...]** within 6 time unit. Figure 4 shows, step by step, how the aforementioned events are processed. Once the sequence has reached the accepting state (F), the occurrences of **SEQ (A, B+, D)** will be established at : **{{a1, b2, d7 }, {a1, b6, d7 }, {a3, b6, d7 }}**.

The drawback of this mode is the use of high storage to accumulate all the events that participate in the combinations in addition to computation overhead for the detection. It consumes more energy. On other hand, it gives us all the possibilities of event combination which can be used (e.g. for further analysis). In our system, we only output one of these possibilities to reduce transmission cost overhead. All registered events are stored in a sliding window. Once the overflow has occurred, the candidate events would be the newest registered ones from the first sequence. The engine will continue to replace the events from the first sequence as long as there is no space. When the initial event (first event in the first sequence combination) is replaced, the engine starts the replacement from the second sequence and so on. The engine applies this replacement policy to ensure that the system still has several sequences to detect a composite event, because replacing the initial events would destroy the

[4]The terms complex event, composite event, pattern and sequence are used interchangeably.

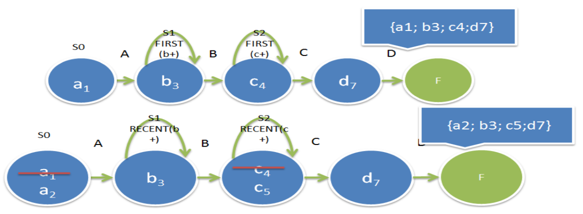

Figure 5: First and Recent Modes

whole sequence.

In the FIRST mode, the earliest occurrence of each contributing event type is used to form the composite event output. Only the first event from a group of events which have the same type advances the sequence to the next state. In this mode, we have just one sequence in the system. The automaton engine will examine every incoming instance e, whether the type of it and occurrence time of e satisfy a transition from the current state to next state. If it is, the sequence will register the event in the current state and advance to next state. If not, the event is directly rejected. Suppose we have the following pattern **SEQ (A, B+, C+, D)** and sequence of tuples presented as **[a1, a2, b3, c4, c5, b6, d7 ...]** within 6 time unit. The result as shown in the upper part of figure 5 .

In the RECENT mode (as the lower part of figure 5 which has FIRST pattern and the same sequence of tuples), the most recent event occurrences of contributing event types are used to form the composite event. In RECENT mode, once an instance satisfies the condition and timing constraint to jump from a state to next state, the engine will stay in the current state unlike FIRST mode. This mode tries to find the most recent instance from consecutive instances for that state before moving to next state. When $a1$ enters the engine. It satisfies the condition to move from $S0$ to $S1$. The engine registers it, stays in $S0$ and does not jump to the next state. Perhaps the new incoming instance is more recent from the last one in the current state.

The advantages of FIRST and RECENT modes are the use of less storage to accumulate all the events that participate in the combinations. Only a few events will be registered in the system in addition to low computation overhead for the detection. They consume less energy. Unlike UNRESTRICTED, they do not give all possible matches.

6.4 Response Phase

Once an accepting state F is reached by the engine, the engine should immediately output the event sequence. This phase is responsible for preparing the output sequence to pass it to the sink operator. The output sequence depends on the mode of the scan. This phase will start to create the response by reading the sliding window contents. In case of FIRST and RECENT modes, the sliding window contains only the events which contribute in sequence detection. In UNRESTRICTED mode, the engine randomly selects a combination of events which matches the pattern in order to reduce transmission cost.

7. EVALUATION

We have completed an initial in-network complex event processing implementation. All the source code, implementing the in-network complex event processing techniques as well as base station functionality, is written in TinyOS. Our code runs successfully on both real motes and the TinyOS Avrora simulator. The aim of the proposed work is to compare the performance of our system, in-network processor which includes complex event engine in comparison with centralized approach in wireless sensor networks and to assess the suitability of our approach in an environment where resources are limited. The comparison would be done in terms of energy efficiency (amount of energy consumed) and the number of messages transmitted per particular interval, in the entire network. The experiment was run for varying the SEQ length. We started with length 2 then 3 and finally 5. Simulations were run for 60 seconds with one event per second. The performance for different SEQ lengths and different modes with a network of 75 nodes is shown in figure 6. The centralized architecture led to higher energy consumption because sensor nodes transmitted events to the sink node at regular periods. In our system, we used in-network complex event processing to decrease the number of transmissions of needless events at each sensor node. What we can notice from figure 6 is summarized as: 1- By increasing the SEQ length in our approach, the RAM size is increased while energy consumption is reduced. The reason is: the transmission will not occur until the sequence reaches the accepting state, few events (tuples) will be relatively satisfied. Hence, the number of transmissions after detections will be decreased. 2- FIRST is a little bit better than RECENT, and both of them are better than UNRESTRICTED in energy consumption. The gap between them is resulting from processing energy consumption, that is because UNRESTRICTED needs more processing power while the other needs less, as shown in figure 6.

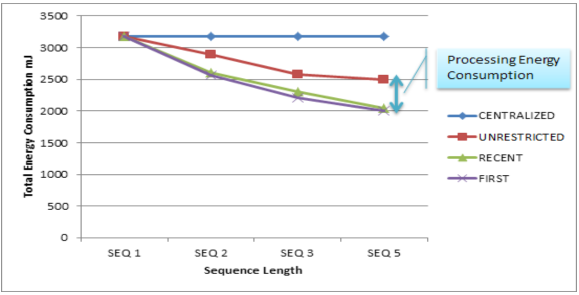

Figure 6: Total Energy Consumption

Figure 7 shows the radio energy consumption for each sensor node and the total number of messages when SEQ length was 3. The nodes in the centralized architecture sent more messages than our approach (nearly three times more). Hence, it consumed more radio energy. Additionally, the gateway nodes consumed more radio energy due to receiving and processing the messages from other sensor nodes. In a 25 nodes network, the centralized approach consumed energy nearly 4203mJ in sink side, while our approach consumed around 2811mJ. Thus, our system conserved nearly 1392mJ (33% of the centralized approach) of the energy. In our architecture, the number of transmissions was reduced. Therefore, the radio energy consumption is reduced not only at the sensor nodes but also at the sink nodes.

8. CONCLUSIONS

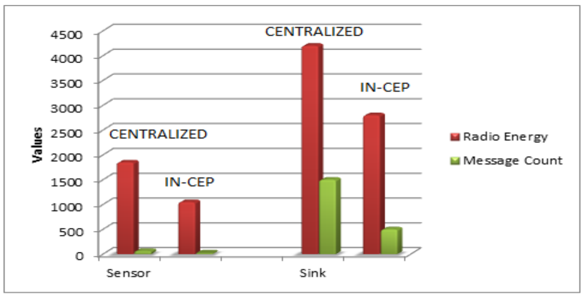

Figure 7: Energy Consumption vs. Radio Message

Sensor networks provide a considerably challenging programming and computing environment. They require advanced paradigms for software design, due to their characteristics such as limited computational power, limited memory and battery power which WSNs suffer from. In this paper, we presented our system, an in-network complex event processing, a system that efficiently carries out complex event queries inside network nodes.

We have proposed an engine to allow the system to detect complex events and valuable information from primitive events.

We developed a query plan based approach to implement the system. We provided the architecture to collect the events from sensor network, this architecture includes three sides; sensor side to perform in-network complex event processing, sink side to deliver the events from the network to AnduIN server side which has the responsibility to display them and perform further analysis.

We demonstrated the effectiveness of our system in a detailed performance study. Results obtained from a comparison between centralized approach and our approach confirms that our in-network complex event processing in small-scale and large-scale sensor networks has shown to increase the lifetime of the network. We plan to continue our research to build distributed in-network complex event processing, in which each sensor node has a different complex event processing plan and can communicate directly between them to detect complex events.

9. REFERENCES

[1] Nondeterministic finite automaton. `http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton`.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J.-h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *ACM SIGMOD Conference*, page 666, 2003.

[3] R. Bhargavi, V. Vaidehi, P. T. V. Bhuvaneswari, P. Balamuralidhar, and M. G. Chandra. Complex event processing for object tracking and intrusion detection in wireless sensor networks. In *ICARCV*, pages 848–853. IEEE, 2010.

[4] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White. Cayuga: a high-performance event processing engine. In *ACM SIGMOD*, pages 1100–1102, New York, NY, USA, 2007. ACM.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. *SIGCOMM Comput. Commun. Rev.*, 31(2 supplement):20–41, Apr. 2001.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: semantics, contexts and detection. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *ACM SIGMOD*, pages 379–390, New York, NY, USA, 2000. ACM.

[8] EsperTech. Event stream intelligence: Esper & NEsper. `http://www.esper.codehaus.org/`.

[9] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system, 1993.

[10] V. Goebel and T. Plagemann. Data stream management systems - a technology for network monitoring and traffic analysis? In *ConTEL 2005*, volume 2, pages 685–686, June 2005.

[11] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: complex event processing over streams (Demo). In *CIDR*, pages 407–411, 2007.

[12] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K. Sattler. Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIN, distributed and parallel databases. *Distributed and Parallel Databases*, 29(1):151–183, Jan. 2011.

[13] Y. Lai, W. Zeng, Z. Lin, and G. Li. LAMF: framework for complex event processing in wireless sensor networks. In *2nd International Conference on (ICISE)*, pages 2155–2158, Dec. 2010.

[14] P. Li and W. Bingwen. Design of complex event processing system for wireless sensor networks. In *NSWCTC*, volume 1, pages 354–357, Apr. 2010.

[15] D. C. Luckham. *The power of events*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[16] 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.*, 30(1):122–173, Mar. 2005.

[17] J. Xingyi, L. Xiaodong, K. Ning, and Y. Baoping. Efficient complex event processing over RFID data stream. In *IEEE/ACIS*, pages 75–81, May 2008.

[18] X. Yang, H. B. Lim, T. M. Özsu, and K. L. Tan. In-network execution of monitoring queries in sensor networks. In *ACM SIGMOD*, pages 521–532, New York, NY, USA, 2007. ACM.

[19] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, Sept. 2002.

[20] M. Zoumboulakis and G. Roussos. Escalation: complex event detection in wireless sensor networks. In *EuroSSC*, pages 270–285, 2007.