# Using a Prioritized MAC Protocol to Efficiently Compute Aggregated Quantities

Björn Andersson, Nuno Pereira and Eduardo Tovar
*IPP Hurray Research Group*
*Polytechnic Institute of Porto, Portugal*
*{bandersson,npereira,emt}@dei.isep.ipp.pt*

## Abstract

*Consider a distributed computer system such that every computer node can perform a wireless broadcast and when it does so, all other nodes receive this message. The computer nodes take sensor readings but individual sensor readings are not very important. It is important however to compute the aggregated quantities of these sensor readings. We show that a prioritized medium access control (MAC) protocol for wireless broadcast can compute simple aggregated quantities in a single transaction, and more complex quantities with many (but still a small number of) transactions. This leads to significant improvements in the time-complexity and as a consequence also similar reduction in energy "consumption".*

## 1. Introduction

It has been recently discussed [1] that sensor networks often take many sensor readings of the same type (for example, temperature readings), and instead of knowing each individual reading it is important to know aggregated quantities of these sensor readings. For example, each computer node senses the temperature at the node and we want to know the maximum temperature among all nodes at a particular moment.

This can be solved with a naïve algorithm; every node broadcasts its sensor reading and hence all nodes know all sensor readings and then they can compute the aggregated quantity. This has the drawback that in a network with $m$ nodes, it is required that $m$ broadcasts are made. Considering that sensor networks are designed for large scale (for example thousands or millions of nodes), the naïve approach can be inefficient with respect to energy and cause a large delay.

In this paper we show that a prioritized MAC protocol for wireless broadcast can significantly improve the time-complexity for computing certain aggregated quantities. In particular we show that the minimum value can be computed with a time complexity that does not depend on the number of nodes. Also the time complexity increases very slowly as the possible range of the value increases. The same technique can be used to compute the maximum value. We also show how to compute a more complex aggregated quantitiy: the median. This computation hinges on the ability to compute the number of nodes. We propose such a technique but it only gives estimation and hence the median function is only estimated.

We consider this result to be significant because (i) the problem of computing aggregated quantities is common in wireless sensor networks which is an area of increasing importance and (ii) the techniques that we use depend on the availability of prioritized MAC protocols that support a very large range of priority levels; such protocols have recently been proposed [2], implemented and tested [3].

The remainder of this paper is structured as follows. Section 2 presents the system model and properties of the MAC protocol that we use. Section 3 shows how to compute the aggregated quantities. Section 4 shows how to estimate the number of proposed elements. Section 5 evaluates the algorithm for computing the number of elements. Section 6 discusses related work and this work. Section 7 gives conclusions.

## 2. System model

Consider a computer system comprised of $m$ computing nodes that communicate over a wireless channel. Nodes do not have a shared memory; all data variables are local to each node. A computer node can make a wireless broadcast. This broadcast can be an unmodulated carrier wave or a message of data bits. We assume that all messages sent by nodes are related to computations of aggregate quantities. A node can transmit an empty message; that is, a message with no data. Every signal transmitted (unmodulated carriers or modulated data bits) is received by all computer nodes. This implies that there are no hidden stations and the network provides reliable broadcast.

Every node has an implementation of a MAC protocol. This MAC protocol is prioritized and collision-free. The fact that it is prioritized means that the MAC protocol assures that of all nodes that request to transmit at a moment, the one with the highest priority will transmit its data bits. The fact that it is collision-free implies that <u>if</u> priorities are unique then there is at most one node which transmits the data bits.

We assume that this MAC protocol is a dominance protocol. It operates as follows. The priority is encoded as a binary number with "0":s and "1":s. We say that a "0" is a dominant bit and a "1" is a recessive bit. We say that a low number represents a high priority. This is similar to the CAN bus [4]. Computer nodes agree on an instant when the tournament starts. Then nodes transmit the priority bits starting with the most significant bit. Priority bits are modulated using a variation of On-Off keying. A node sends an unmodulated carrier wave if it had a dominant bit and it sends nothing if it had a recessive bit. In the beginning of the tournament, all nodes have the potential to win but if it was recessive at a bit and perceived a dominant bit then it withdraws from the tournament and it cannot win. When a node has won the tournament, then it clearly knows the priority of the winner. If a node has lost the tournament then it continues to listen in order to know the priority of the winner.

The operating system exposes three system calls for interacting with other nodes. The `send` system call takes two parameters, one describing the priority of the message and one describing the data bits to be transmitted. If `send` loses the tournament then it waits until a new tournament starts. The program making this system call blocks until a message is successfully transmitted. The function `send_empty` takes only one parameters and it is a priority. Interestingly, `send_empty` does not take any parameter describing the data. The system call `send_empty` works like the function `send` but if it wins it does not send anything. In addition, when the tournament is over (regardless of whether the node wins or loses), the function `send_empty` gives the control back to the application and the function `send_empty` returns the priority of the winner. There is also a function `just_listen` which works like `send_empty` but it loses even before the first bit, so `just_listen` will only return the priority of the winner.

We assume that a computer node proposes a value. This value may be a sensor reading such as a temperature. Computer node $N_i$ proposes the value $v_i$. The range of the value $v_i$ is known; it is $[MINV..MAXV]$, we assume $0 \leq MINV$. For example it could be a 12 bit non-negative integer. Then the range is [0..4095]. All $v_i$ have the same range for all proposed values. We assume that computer nodes do not know $m$.

We consider the problem of computing $f(v_1,v_2,\ldots,v_n)$ efficiently. We say that $f$ is an *aggregated* quantity. We assume that there is one or many nodes that initiate the computation of $f$. When a node $i$ has heard from one of these nodes that initiate the computation then node $i$ proposes its value $v_i$. Every node has the potential to initiate a computation.

## 3. Computing aggregated quantities

We will first compute two simple quantities exactly in Section 3.1 and Section 3.2 and then, Section 3.3 shows how to compute a more complex quantity.

### 3.1. Computing the minimum value

Consider the case where the quantity that we want to compute $f(v_1,v_2,\ldots,v_m)$ is $min(v_1,v_2,\ldots,v_m)$. This can be performed as follows:

**Algorithm 1. Calculating Min**

```
When a node requests that min should be computed:
   Broadcast a message INITIATE_MIN
end
When a message INITIATE_MIN is received:
   Node i calculates value vi that it proposes.
   minv := calcmin( vi )
end
subroutine calcmin( vi )
   return send_empty( priority = vi )
end
```

### 3.2. Computing the maximum value

Let us consider the computation of $f(v_1,v_2,\ldots,v_m)$ is $max(v_1,v_2,\ldots,v_m)$. This can be performed as follows:

**Algorithm 2. Calculating Max**

```
When a node requests that max should be computed:
   Broadcast a message INITIATE_MAX
end
When a message INITIATE_MAX is received:
   Node i calculates value vi that it proposes.
   maxv := calcmax( vi )
end
subroutine calcmax( vi )
   return MAXV-send_empty( priority = MAXV - vi )
end
```

### 3.3. Computing the median value

We now consider the case where the function that we want to compute is the median of $v_1,v_2,\ldots,v_m$. We will find it convenient to introduce the notation $V_{less}(q)$ and $V_{greater}(q)$ as:

$$V_{less}(q) = \{v_j : v_j \leq q\} \quad (1)$$

$$V_{greater}(q) = \{v_j : v_j \geq q\} \quad (2)$$

With these definitions our goal is to find $q$ such that $||V_{greater}(q)|-|V_{less}(q)||$ is minimized. We assume the existence of the function `get_n_elements_in( LB, UB, active)`. It will be described in Section 4 and it returns the number of computer nodes that proposed a value which is greater than or equal to LB and less than or equal to UB.

**Algorithm 3. Calculating median value**

```
When a node requests that median should be
  computed:
   Broadcast a message INITIATE_MEDIAN
end
When a message INITIATE_MEDIAN is received:
   Node i calculates value vi that it proposes.
   median := calcmedianvalues( vi )
end
subroutine calcmedianvalue( vi )
   LB := MINV
   UB := MAXV
   for j:=1..to log2(MAXV-MINV) do
      mid := ( LB + UB ) / 2
      active    :=vi<=mid
      nVless    :=get_n_elements_in(LB,mid,active)
      active    :=vi>=mid
      nVgreater:=get_n_elements_in(mid,UB,active)
      if nVless<=nVgreater then
        LB := mid
      else
        UB := mid
      end if
   endfor
   return mid
end
```

## 4. Computing the number of proposed elements

Computing the number of proposed nodes is equivalent to computing the number of nodes. However, computing this is non-trivial. Consider a node $i$ that proposes a value $v_i$. All nodes will receive a value $R$ from `send_empty`. If $R = v_i$ then node $i$ cannot know if it is the only node (and hence $m = 1$) or there are many other nodes with $v_i=R$ as well. In fact, with the use of our MAC protocol this is impossible to achieve for an algorithm where all nodes makes a single call to `send_empty` at the same time. Based on this impossibility, we will focus on algorithms that do not find the exact value of $m$, but try to find an estimate of $m$. The intuition is that each computer node generates a random number and if there is a large number of nodes then the minimum random number is very small. We repeat this $k$ times. Hence a large value of $k$ gives a good accuracy of the estimate whereas a low value of $k$ has low time-complexity. We think $k$=5 is a reasonable compromise (which will be discussed later). Algorithm 4 describes this.

**Algorithm 4. Calculating nelements**

```
When a node requests that number of elements
  should be computed:
    Broadcast a message INITIATE_NELEMENTS
When a message INITIATE_NELEMENTS is received:
    nnodes :=get_n_elements_in(MINV, MAXV, TRUE)
end

subroutine get_n_elements_in( LB, UB, active)
    for q:=1 to k do
      if active then
        R[q] := send_empty(priority = random(LB,UB) )
      else
        R[q] := just_listen
      end if
    end
    return ML_estimation( R[1],…,R[k], LB, UB )
end
subroutine ML_estimation( R, LB, UB )
    for q:=1 to k do
      u[q] := (UB-R[q])/(UB-LB)
    endfor
    loginvsum := 0
    for q := 1 to k do
      loginvsum := logsinvsum + ln( 1/u[q] )
    endfor
    return k/loginvsum
end
```

In Algorithm 4, we conveniently ignore the possibility of an interval with no nodes. We can understand the function `ML_estimation` by considering the following analysis. Let $A_j$ denote the event that there were $j$ nodes. Let $B(R^k)$ denote the event that the minimum of the proposed values is $R^l$ when we generated random numbers the $l$:th time.
Let $B(R)=B(R^1)\cap B(R^2) \cap\ldots B(R^k)$. Let $A_j$ denote the event that there are $j$ nodes. When we have the minimum of the proposed values (in Algorithm 4) we wish to compute P( $A_j | B( R )$ ) for all values of $j$ and select the value of $j$ that maximizes

$$P\left(A_j \middle| B(R)\right) \qquad (3)$$

We will do so now. We know from Bayes´s formula that:

$$P\left(A_j \middle| B(R)\right) = \frac{P\left(B(R) \middle| A_j\right) \times P\left(A_j\right)}{\sum_{i=1}^{\infty} \left(P\left(B(R) \middle| A_i\right) \times P\left(A_i\right)\right)} \qquad (4)$$

Let us assume that:

$$\forall j : P(A_1) = P(A_j) \qquad (5)$$

Applying (5) in (4) gives us:

$$P\left(A_j \middle| B(R)\right) = \frac{P\left(B(R) \middle| A_j\right)}{\sum_{i=1}^{\infty} P\left(B(R) \middle| A_i\right)} \qquad (6)$$

Let us now compute P($B(R)|A_j$). We know that:

$$P\left(B(R) \middle| A_i\right) = P\left(B(R^1) \middle| A_i\right) \times \ldots \times P\left(B(R^k) \middle| A_i\right) \qquad (6b)$$

We obtain.

$$P\left(B(R^k) \middle| A_i\right) = \frac{d}{dR} CDF\left(B(R^k) \middle| A_i\right) \qquad (7)$$

where $CDF$ is the probability that the minimum is less than or equal to $R$. We compute it as follows. The probability that a random number is greater than or equal to $R$ is

$$P(random \quad number \geq R) = \frac{MAXV - R}{MAXV - MINV} \qquad (8)$$

The probability that the minimum of the $i$ randomly generated numbers is greater than or equal to $R$ is

$$P(\min imum \quad number \geq R) = \left(\frac{MAXV - R}{MAXV - MINV}\right)^i \qquad (9)$$

Hence, we obtain:

$$CDF\left(B(R^k) \middle| A_i\right) = 1 - \left(\frac{MAXV - R}{MAXV - MINV}\right)^i \qquad (10)$$

Combining (10) with (7) gives us:

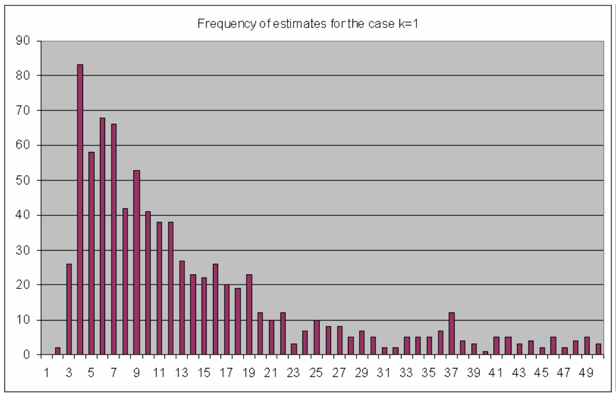
$$P\left(B(R^k) \middle| A_i\right) = i \times \left(\frac{MAXV - R}{MAXV - MINV}\right)^{i-1} \qquad (11)$$

Inserting (11) in (6) gives us:

$$P\left(A_j \middle| B(R^k)\right) = \frac{j \times \left(\frac{MAXV - R}{MAXV - MINV}\right)^{j-1}}{\sum_{i=1}^{\infty} \left(i \times \left(\frac{MAXV - R}{MAXV - MINV}\right)^{i-1}\right)} \qquad (12)$$

We wish to find the $j$ that maximizes P( $A_j | B( R )$ ). We observe that this depends only on the numerator. Hence, we want to find the value of $j_{solution}$ that maximizes:

$$\prod_{q=1}^{k} \left( j_{solution} \times \left(\frac{MAXV - R^q}{MAXV - MINV}\right)^{j_{solution}-1} \right) \qquad (13)$$



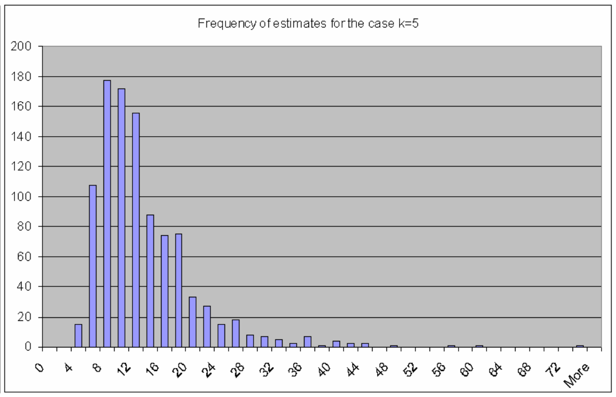


**Figure 1. The frequency of the estimates for different values of *k*.**

We can simplify (13) further. Let us use the notation:

$$u_q = \frac{MAXV - R^q}{MAXV - MINV} \tag{14}$$

and rewrite (13) we obtain that we want to maximize:

$$\prod_{q=1}^{k} \left( j_{solution} \times u_q^{\,j_{solution}-1} \right) \tag{15}$$

Observe that maximizing (15) is equivalent to maximizing the natural logarithm of (15). We know that the logarithm of a product is the sum of the logarithm of the factors. Hence, we want to maximize:

$$\sum_{q=1}^{k} \ln\left( j_{solution} \times u_q^{\,j_{solution}-1} \right) \tag{16}$$

We can rewrite (16) into the problem we want to maximize:

$$\sum_{q=1}^{k} \left( \ln j_{solution} + (j_{solution}) \times \ln u_q \right) \tag{17}$$

We have that the first derivative of (17) with respect to $j_{solution}$ is:

$$\sum_{q=1}^{k} \left( \frac{1}{j_{solution}} + \ln u_q \right) \tag{18}$$

And the second derivative of (17) with respect to $j_{solution}$ is:

$$-\sum_{q=1}^{k} \frac{1}{j_{solution}^{\;2}} \tag{19}$$

We can see from (18) and (19) that finding the $j_{solution}$ such that (18) is equal to 0 gives us the maximum likelihood estimate. Hence, we should select $j_{solution}$ such that:

$$\sum_{q=1}^{k} \left( \frac{1}{j_{solution}} + \ln u_q \right) = 0 \tag{20}$$

We can rewrite (20) to:

$$\frac{k}{j_{solution}} = -\sum_{q=1}^{k} \ln u_q \tag{21}$$

Rewriting yields:

$$\frac{1}{j_{solution}} = \frac{\sum_{q=1}^{k} \ln \frac{1}{u_q}}{k} \tag{22}$$

Further rewriting yields:

$$j_{solution} = \frac{k}{\sum_{q=1}^{k} \ln \frac{1}{u_q}} \tag{23}$$

This is a simple way to compute our estimate and we can see that `ML_estimation` in Algorithm 5 is based on this equation. We think it is simple enough to be used in a mote, although motes have very low processor speed.

## 5. Performance evaluation of nodes estimation

We have already mentioned that the calculation of the complex function median depends on the estimation of the number of nodes that propose a value. Hence, it is important that this estimation is accurate. For this purpose, we evaluate the accuracy using simulation experiments. Figure 1 shows the experimental results.

We ran 1000 experiments. For every experiment, 10 nodes generate random numbers and estimate the number of nodes. The estimation was made using (23). We can see that using five random numbers gives a significant improvement in the accuracy of the estimation as compared to one random number.

## 6. Related work and Discussion

### 6.1. Related work

A prioritized MAC protocol is useful to schedule real-time traffic [2, 3] and it can support data dissemination when topology is unknown [5]. In this paper we have discussed how to efficiently compute aggregated quantities using a prioritized MAC protocol.

Distributed calculations have been performed in previous research. It has been observed that nodes often [6, 7] detect an event and then needs to spread the knowledge of this event to its neighbours. This is called [6] one-to-*k* communication because only *k* neighbours need to receive the message. After that, the neighbour nodes perform local computations and reports back to the node that made the request for 1-to-*k* communication. This reporting back is called *k*-to-1 communication. Algorithms for both 1-to-*k* and *k*-to-1 communication are shown to be faster than naïve algorithm but unfortunately, the time-complexity increases as *k* increases. Our algorithms computes a function *f* and takes parameters from different nodes; this is similar to the average calculations in [8] . However our algorithms are different from [6, 7]; our algorithms have a time-complexity that does not depend on the number of nodes. We think our new algorithms are also useful building blocks for leader election and clock synchronization.

In this paper, nodes are permitted to use duplicated priorities, so any message transmitted after the tournament could collide and, for this reason, we use a send_empty primitive. However, it would be easy to code the priority in such a way that it would be unique by concatenating the node identifier to the priority. In this way, nodes could send a valid data message after winning the tournament. This is useful to because we may want to know not only the maximum value (for example the maximum temperature) but also other related values (for example the position of the node that detected the maximum temperature).

One way to use these algorithms is to encapsulate them in a query processor for database queries. Query processors for sensor networks have been studied in previous work [9, 10] but they are different in that they operate in multhop environment, do not compute aggregated quantities as efficiently as we do. They assume one single sink node and that the other nodes should report an aggregated quantity to this sink node. The sink node floods its interest in the data it wants into the network and this also makes nodes to discover the topology. When a node has new data it, broadcasts this data; other nodes hear it and it is routed and combined so that the sink node receives the aggregated. These works exploit the broadcast characteristics of the wireless medium (like we do) but they do not make any assumption on the MAC protocol (and hence they do not take advantage of the MAC protocol). One important aspect of these protocols is to create a spanning tree. It is known that computing an optimal spanning tree for the case when only a subset of nodes can generate data is equivalent to finding a Steiner-tree, a problem known to be NP-hard (the decision problem is NP-complete, see page 208 in [11]). For this reason, approximation algorithms have been proposed [12, 13]. However, in the average case, very simple randomized algorithms perform well [14]. Since a node will forward its data to the sink using a path which is not necessarily the shortest path to the sink, these protocols cause an extra delay. Hence, there is a trade-off between delay and energy-efficiency. To make this trade-off, a framework based on feedback was developed [15] for computing aggregated quantities. Techniques to aggregate data in the network such that the user at the base station can detect whether one node gives faked data has been addressed as well [16].

It has been observed that computing the median is especially difficult in multihop networks because combining two medians from different subnetworks is requires large amount of memory. Researchers in [17] observed that it is necessary for packets forwarded to be bigger and bigger the closer they get to the base station. Several algorithms for computing the exact median in O(*m*) time complexity are available (the earliest one is [18]). Our algorithm is faster; it has the time complexity O(log (MAXV-MINV)) but at the expensive of the accuracy of the result.

Computing averages has been done under the assumption that an adversary generates faults [19]. Unfortunately, it has a time-complexity which is larger than our algorithm and also larger than the algorithm proposed by [18] .

### 6.2. Practical issues

It is beyond the scope of this paper to describe all the details of the MAC protocol (see [2, 3] for details); it is important to observe however that the MAC protocol has the following properties. First, a priority bit has a duration adapted to time-of-flight, Rx/Tx switching time and time to detect a carrier and the duration of this bit can be quite large whereas a bit in the data packet has normal duration (for example on the CC2420 transceiver with a speed of 250kbps, a bit takes 4us). Hence, unlike CAN, in our protocol, the bit rate of the data transmission has the potential to be high even on long distances. Second, before the tournament in the protocol starts, the tournament waits for a long time of silence and synchronizes. This implies that even if nodes start the execution of the algorithms at slightly different times then the priority bits will be compared properly. This scheme only works if the different in time when message transmit messages "simultaneously" is not too big. We believe this assumption can easily be true however, by letting the algorithm start when it receives a message from a master node ordering the other nodes to start the execution of the algorithm.

So far we have assumed that all messages transmitted deal with aggregated quantities and we have assumed that there is only one type of aggregated quantity that we want to compute. This can be solved easily. We can subdivide the priority field into two subfield. The most significant bits are called service identifier and the least significant bits are called data bits. For example, we have 10 priority bits; the 4 most significant bits could be the service identifiers and the remaining 6 bits are priority bits. The MAC protocol runs the tournament base on all 10 bits. If the 4 services bits are 0000 then the following 6 bits denotes the priority of a normal message and these 6 bits number represent a unique priority and is normal payload and it is collision free. If the 4 bits are 0001 it means that the 6 remaining contains data that should be used to compute the maximum temperature. An application can make a function call `send_empty` (0001, 20) which proposes the value 20 and returns the maximum temperature.

## 7. Conclusions

We have shown how to use a prioritized protocol to compute aggregated quantities efficiently. The computational complexity for min and max is O(log2(MAXV-MINV)), that is they do not depend on the number of nodes. Our estimation of the median can be computed efficiently as well, its time complexity is $O(k*[\log2(MAXV-MINV)]^2)$.

## References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393-422, 2002.

[2] B. Andersson and E. Tovar, "Static-Priority Scheduling of Sporadic Messages on a Wireless Channel," presented at International Conference on Principles of Distributed Systems (OPODIS´05), Pisa, Italy, 2005.

[3] N. Pereira, B. Andersson, and E. Tovar, "Implementation of a Dominance Protocol for Wireless Medium Access," presented at IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Sydney, Australia, 2006.

[4] Bosch, "CAN Specification, ver. 2.0, Robert Bosch GmbH, Stuttgart," 1991.

[5] B. Andersson, N. Pereira, and E. Tovar, "Disseminating Data Using Broadcast when Topology is Unknown," presented at Proceedings of the 26th IEEE Real-Time Systems Symposium, Work-in-Progress Session, Miami Beach, Florida, 2005.

[6] R. Zheng and L. Sha, "MAC Layer Support for Group Communication in Wireless Sensor Networks," Department of Computer Science, University of Houston UH-CS-05-14, July 21 2005.

[7] K. Jamieson, H. Balakrishnan, and Y. C. Tay, "Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks," presented at Third European Workshop on Sensor Networks, Zurich, Switzerland, 2006.

[8] D. S. Scherber and H. C. Papadopoulos, "Distributed computation of averages over ad-hoc networks," *IEEE J. Select. Areas Commun*, vol. 23, pp. 776-787, 2005.

[9] Y. Yao and J. E. Gehrke, "Query Processing in Sensor Networks," presented at Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003), Asilomar, California, 2003.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," presented at Proceedings of OSDI, Boston, MA., 2002.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability A guide to the Theory of NP-Completeness* New York: W. H. Freeman and Company, 1979.

[12] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," presented at 22nd International Conference on Distributed Computing Systems, 2002.

[13] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of Network Density on Data Aggregation in Wireless Sensor Networks," presented at Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2002.

[14] M. Enachescu, A. Goel, R. Govindan, and R. Motwani, "Scale Free Aggregation in Sensor Networks," presented at First International Workshop on Algorithmic Aspects of Wireless Sensor Networks, 2004.

[15] T. Abdelzaher, T. He, and J. Stankovic, "Feedback Control of Data Aggregation in Sensor Networks," presented at 43rd IEEE Conference on Decision and Control, Paradise Island, Bahamas, 2004.

[16] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks," presented at ACM SenSys (Conference on Embedded Networked Sensor Systems)], 2003.

[17] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks. SenSys 2004: 239-249," presented at SenSys, Baltimore, Maryland, USA, 2004.

[18] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *J. Cornput. System Sci*, vol. 7, pp. 448-461., 1973.

[19] M. Kutylwski and D. Letkiewicz, "Computing Average Value in ad hoc Networks," presented at MFCS 2003 28th International Symposium on Mathematical Foundations of Computer Science, Bratislava, Slovak Republic, Europe, 2003.