Technical Report

Estimating the Number of Nodes in Wireless Sensor Networks

Björn Andersson
Nuno Pereira
Eduardo Tovar

TR-060702
Version: 1.0
Date: July 2006

Estimating the Number of Nodes in Wireless Sensor Networks

Björn ANDERSSON, Nuno PEREIRA, Eduardo TOVAR

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {bandersson, npereira, emt}@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

We propose an efficient algorithm to estimate the number of live computer nodes in a network. This algorithm is fully distributed, and has a time-complexity which is independent of the number of computer nodes. The algorithm is designed to take advantage of a medium access control (MAC) protocol which is prioritized; that is, if two or more messages on different nodes contend for the medium, then the node contending with the highest priority will win, and all nodes will know the priority of the winner.

Estimating the Number of Nodes in Wireless Sensor Networks

Björn Andersson, Nuno Pereira and Eduardo Tovar

Department of Computer Engineering, School of Engineering,
Polytechnic Institute of Porto (ISEP-IPP),
Rua Dr. António Bernardino de Almeida 431,
4200-072 Porto, Portugal
{bandersson, npereira, emt}@dei.isep.ipp.pt

Abstract. We propose an efficient algorithm to estimate the number of live computer nodes in a network. This algorithm is fully distributed, and has a time-complexity which is independent of the number of computer nodes. The algorithm is designed to take advantage of a medium access control (MAC) protocol which is prioritized; that is, if two or more messages on different nodes contend for the medium, then the node contending with the highest priority will win, and all nodes will know the priority of the winner.

1 Introduction

Wireless sensor networks operate in unpredictable environments. When nodes are deployed, for example dropped from a helicopter, it is not known how many nodes will crash when they hit the ground. During operation, nodes stop working because their energy supply is exhausted or they suffer from physical faults. Hence, the number of live nodes is not the same as the number of nodes deployed and the number of live nodes changes with time.

Nonetheless, it is often necessary for a computer node to know the number of live computer nodes at a particular point in time, or to know the number of live nodes with certain characteristics; this is a related problem. Consider a sensor network that detects and reports important events, such as earthquakes [1]. It is often assumed that individual sensor readings cannot be trusted and hence an event should only be reported if many (more than a certain threshold) of sensors have detected the event or a certain fraction of all live nodes detect the event. This decision must be made quickly (typically with a delay less than a few seconds).

The problem of finding the number of nodes can be solved trivially by simply letting each node transmit a message with its identifier and let all nodes count the number of messages with unique identifiers. It has a time-complexity of $O(m)$, where m is the number of nodes. This is unfortunate because sensor networks are large; today, sensor networks with more than a thousand nodes [2] have been built and networks with a hundred thousand nodes are expected soon. Hence, it is important that the time-complexity does not depend on the number of nodes.

In this paper, a distributed algorithm which estimates the number of computer nodes is designed. The main idea of the algorithm is that all nodes generate random numbers and the minimum of these random numbers can be found efficiently by using a prioritized medium access control (MAC) protocol. An estimate of the number of nodes can be obtained from the minimum of the random numbers. The algorithm performs this k times (k is a design parameter). This gives our algorithm a time-complexity which is independent of m; it only needs the time to transmit k messages, and hence the time-complexity is $\mathrm{O}(k)$.

We advocate this algorithm to be significant, not only for estimating the number of nodes, but also because it serves as a useful building block for other calculations such as approximate majority voting, calculating the median of sensor readings and calculating the sum of all sensor readings.

The remainder of this paper is structured as follows. Section 2 presents the system model and properties of the MAC protocol used. The algorithm to estimate the number of nodes and the reasoning behind its design is presented in Section 3. A performance evaluation of the proposed algorithm is in Section 4, and Section 5 discusses related work and practical aspects of our work. Finally, Section 6, 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.

Computer nodes can make wireless broadcasts. These broadcasts can be a message of data bits or an unmodulated carrier wave. It is assumed that every signal transmitted (modulated data bits or unmodulated carriers) is received by all computer nodes. This implies that there are no hidden stations and the network provides reliable broadcast. A node can transmit an empty message; that is, a node can request to the MAC protocol to perform the contention for the medium, but not send any data. This is clarified later in this section.

It is assumed that all messages sent by nodes are related to the estimation of the number of nodes; hence nodes do not transmit any other kind of messages. It is also assumed that nodes are requested to compute the number of nodes simultaneously. Both of these assumptions can be relaxed; this is done in Section 5.2.

Every node has an implementation of a prioritized MAC protocol [3, 4]. The fact that it is prioritized means that the MAC protocol assures that of all nodes requesting to transmit at a given moment, the ones with the highest priority will win; nodes that win will transmit their data bits (if they have any).

The MAC protocol performs a tournament as depicted in Figure 1. The nodes start by agreeing on an instant when the tournament starts. Then nodes transmit the priority bits starting with the most significant bit. A bit is assigned a time interval. If a node contends with a dominant bit ("0"), then a carrier wave is transmitted in this time interval; if the node contends with a recessive

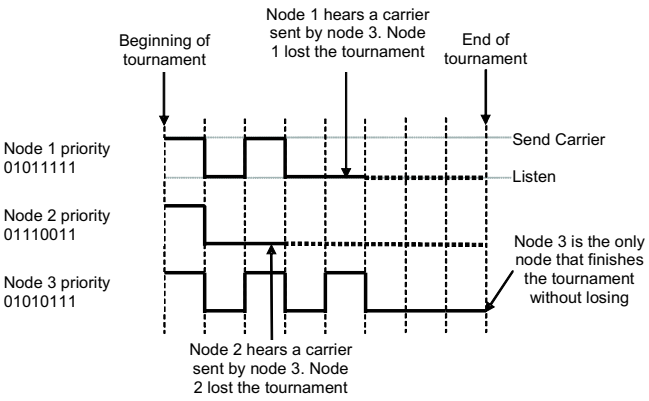


Fig. 1. The MAC protocol tournament.

bit ("1"), it transmits nothing but listens. At the beginning of the tournament, all nodes have the potential to win, but if a node contends with a recessive bit and perceives a dominant bit then it withdraws from the tournament and cannot win. If a node has lost the tournament then it continues to listen in order to know the priority of the winner. When a node finishes sending all priority bits without hearing a dominant bit, then it has won the tournament and clearly knows the priority of the winner. Hence, lower numbers represent higher priorities. This is similar to the CAN bus [5], but nodes in a CAN network are usually assigned unique priorities; that assumption is not made here.

Let $NPRIOBITS$ denote the number of priority bits. It is the same for all nodes. Since $NPRIOBITS$ are used to denote the priority, the priority represents a number from 0 to $2^{NPRIOBITS}$ - 1. Let $MAXV$ denote $2^{NPRIOBITS}$ - 1. A function which generates a uniformly distributed random integer variable in the range [0, $MAXV$] is denoted by `random`(0, $MAXV$).

The operating system offers systems 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 parameter and it is a priority. Interestingly, `send_empty` does not take any parameter describing the data. The system call `send_empty` also results in the MAC protocol performing the contention for the medium, but if the node 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 returns the priority of the winner. The `send_empty` system call will be used in environments where two nodes may have the same priority and hence there may be more than one node

Algorithm 1 Estimating the number of nodes

Require: All computer nodes start their execution simultaneously.

```
1: r : array[1..k] of integer
2: x : array[1..k] of integer
3: q : integer
4: for q ← 1 to k
5:     r[q] ← random(0, MAXV)
6:     x[q] ← send_empty(r[q])
7: end for
8: est_nodes ← ML_estimation(x[1], x[2], ..., x[k])
9: return est_nodes // the estimation of the number of nodes
```

that declares itself as a winner. This is acceptable since they do not send any data, so there is no collision of the data.

Let $P(E)$ denote the probability that event E occurs, and $P(E1 \mid E2)$ denote the conditional probability of event $E1$, given the occurrence of event $E2$.

The number of nodes m is unknown; m is a uniformly distributed random variable which is an integer. Hence, for the a priori probability it holds that:

$$\forall x \geq 1 : P\left(m = 1\right) = P\left(m = x\right) \tag{1}$$

It is also assumed that no faults occur during the execution of the algorithm.

3 Estimating the Number of Nodes

Section 3.1 presents an algorithm that estimates the number of nodes. Section 3.2 considers an interval and computes the a posteriori probability that the number of nodes is in this interval.

3.1 Estimating a Single Value

The pseudo code of the algorithm for estimating the number of nodes is shown in Algorithms 1, 2 and 3. The main algorithm (Algorithm 1) assumes that all computer nodes start their execution simultaneously and they do the following. First, on line 5, the algorithm generates a random number in the range [0, $MAXV$] and all nodes send their random number and find the minimum random number (line 6). This is performed k times. The line 8 computes the estimation of the number of nodes based on the minimum obtained on line 6. Line 8 uses a function, shown in Algorithm 2.

The design of the function in Algoritm 2 can be explained as follows. Consider m independent random variables which are integers and are uniformly distributed in the range [0, $MAXV$]. The random variables will be denoted r_1, r_2,...,r_m. It holds that (see Appendix A for details):

$$P\left(\min\left(r_1, r_2, ..., r_m\right) \leq x\right) = 1 - \left(1 - \frac{x}{MAXV}\right)^m \tag{2}$$

Algorithm 2 Function `ML_estimation`

Require: The division of two integers (as is done in line 6) returns a real number.

1: **function** *ML_estimation*(x : array[1..k] of integer) **return** an integer
2: v : array[1..k] of real
3: $sumv$, q, $h1$, $h2$: integer
4: $sumv \leftarrow 0$
5: **for** $q \leftarrow 1$ **to** k
6: $v[q] \leftarrow ln\left(\frac{1}{1-\frac{x[q]}{MAXV}}\right)$
7: $sumv \leftarrow sumv + v[q]$
8: **end for**
9: $h1 \leftarrow ceil(\ k\ /\ sumv\ )$
10: $h2 \leftarrow floor(\ k\ /\ sumv\ )$
11: **if** *numerator_probability*($\ h1,\ x\ $) $>$ *numerator_probability*($\ h2,\ x\ $)
12: **return** $h1$
13: **else**
14: **return** $h2$
15: **end if**
16: **end function**

Algorithm 3 Function `numerator_probability`

1: **function** *numerator_probability*(j : integer, x : array[1..k] of integer) **return** a real
2: **return** *compute (8)*
3: **end function**

Exploiting the fact that x is an integer variable gives us:

$$P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right) = \left(1 - \frac{x-1}{MAXV}\right)^m - \left(1 - \frac{x}{MAXV}\right)^m \quad (3)$$

Clearly (3) can be rewritten as a conditional probability as follows:

$$P\left(\min\left(r_1, r_2, ..., r_m\right) = x \,|m = j\,\right) = \left(1 - \frac{x-1}{MAXV}\right)^j - \left(1 - \frac{x}{MAXV}\right)^j \quad (4)$$

From Bayes' theorem it results that:

$$P\left(m = j \,|\min\left(r_1, r_2, ..., r_m\right) = x\,\right) = \frac{P(\min(r_1,r_2,...,r_m)=x|m=j\,)\times P(m=j)}{\sum_{l=1}^{\infty} P(\min(r_1,r_2,...,r_m)=x|m=l\,)\times P(m=l)} \quad (5)$$

Using the assumption stated by (1) on (5), and using (4):

$$P\left(m = j \,|\min\left(r_1, r_2, ..., r_m\right) = x\,\right) = \frac{\left(1-\frac{x-1}{MAXV}\right)^j - \left(1-\frac{x}{MAXV}\right)^j}{\sum_{l=1}^{\infty}\left(1-\frac{x-1}{MAXV}\right)^l - \left(1-\frac{x}{MAXV}\right)^l} \quad (6)$$

Observe from Algorithm 1 that each node will generate k random numbers. Let $r_1^1, r_2^1, \ldots, r_m^1$ denote the first random numbers, and let $r_1^2, r_2^2, \ldots, r_m^2$ denote

the second random numbers, and so on. Let x_q denote $min(r_1^q, r_2^q, \ldots, r_m^q)$. By rewriting (6), it follows that:

$$P\left(m = j \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\prod_{q=1}^{k}\left(\left(1 - \frac{x_q - 1}{MAXV}\right)^j - \left(1 - \frac{x_q}{MAXV}\right)^j\right)}{\sum_{l=1}^{\infty}\left(\prod_{q=1}^{k}\left(\left(1 - \frac{x_q - 1}{MAXV}\right)^l - \left(1 - \frac{x_q}{MAXV}\right)^l\right)\right)} \tag{7}$$

The goal is to make a maximum likelihood estimation of m. Hence, it is necessary to find the value of j such that (7) is maximized. Observe that the denominator remains constant for all values of j. Then, it is possible to state that the objective is to maximize:

$$\prod_{q=1}^{k}\left(\left(1 - \frac{x_q - 1}{MAXV}\right)^j - \left(1 - \frac{x_q}{MAXV}\right)^j\right) \tag{8}$$

This can be found by iterating values of j. However computing j in this manner is computationally expensive, and thus another method to find a value of j that maximizes (7) is devised next.

Maximizing an expression *expr* is equivalent to maximizing the logarithm of *expr*. Then, the goal is to maximize:

$$\sum_{q=1}^{k}\ln\left(\left(1 - \frac{x_q - 1}{MAXV}\right)^j - \left(1 - \frac{x_q}{MAXV}\right)^j\right) \tag{9}$$

where *ln* means natural logarithm.

If $1 << MAXV$, then (9) can be approximated by:

$$\sum_{q=1}^{k}\ln\left(\left(1 - \frac{x_q}{MAXV}\right)^{j-1} \times \frac{j}{MAXV}\right) \tag{10}$$

By applying the rules of logarithms to (10), it holds that:

$$\sum_{q=1}^{k}\left(\left(-(j-1) \times \ln\frac{1}{1 - \frac{x_q}{MAXV}}\right) + \ln\left(\frac{j}{MAXV}\right)\right) \tag{11}$$

The variable j in (11) is an integer. Let us replace j in (11) by a real valued variable j':

$$\sum_{q=1}^{k}\left(\left(-(j'-1) \times \ln\frac{1}{1 - \frac{x_q}{MAXV}}\right) + \ln\left(\frac{j'}{MAXV}\right)\right) \tag{12}$$

The 1^{st} derivative of (12) with respect to j' is:

$$\sum_{q=1}^{k}\left(\left(-\ln\frac{1}{1 - \frac{x_q}{MAXV}}\right) + \frac{1}{j'}\right) \tag{13}$$

and the 2^{nd} derivative of (12) with respect to j' is:

$$\sum_{q=1}^{k} -\frac{1}{j'^2} \tag{14}$$

Since (14) is negative, finding the $j^{'}$ that sets (13) equal to 0 is a maximizer of (12). Doing this gives:

$$\sum_{q=1}^{k} \frac{1}{j'} = \sum_{q=1}^{k} \ln \frac{1}{1 - \frac{x_q}{MAXV}} \tag{15}$$

and simplifying (15) leads to:

$$j' = \frac{k}{\sum\limits_{q=1}^{k} \ln \frac{1}{1-\frac{x_q}{MAXV}}} \tag{16}$$

Since (14) is negative, the integer j' that maximizes (12) is either the ceiling or the floor of (16). This results in a j which is very close to the j that maximizes (7). This is what the function in Algorithm 2 does.

3.2 Estimating an Interval

It is sometimes necessary to know the probability that the number of nodes is less than or equal to j_2. On the other hand, it is sometimes necessary to know if the number of nodes is greater than or equal to j_1. And we want to know this with a certain confidence. Since there is diversity in what application developers might want, we will only design a simple generic function. We will design a function that computes the probability that: $j_1 \ \leq m \ \leq j_2$, where j_1 and j_2 are parameters selected by the designer. If the probability is not large enough, then it is up to the application program to decrease j_1 or increase j_2, or perform an estimation with a larger k.

Let us now present the reasoning for such probability estimation. From (7):

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\sum\limits_{l=j_1}^{j_2} \left(\prod_{q=1}^{k} \left(\left(1-\frac{x_q-1}{MAXV}\right)^l - \left(1-\frac{x_q}{MAXV}\right)^l \right) \right)}{\sum\limits_{l=1}^{\infty} \left(\prod_{q=1}^{k} \left(\left(1-\frac{x_q-1}{MAXV}\right)^l - \left(1-\frac{x_q}{MAXV}\right)^l \right) \right)} \tag{17}$$

If $1 << MAXV$, then (17) can be approximated as:

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\sum\limits_{l=j_1}^{j_2} \left(\prod_{q=1}^{k} \left(\frac{l}{MAXV} \times \left(1-\frac{x_q}{MAXV}\right)^{l-1} \right) \right)}{\sum\limits_{l=1}^{\infty} \left(\prod_{q=1}^{k} \left(\frac{l}{MAXV} \times \left(1-\frac{x_q}{MAXV}\right)^{l-1} \right) \right)} \tag{18}$$

Let Q be defined as:

$$Q = \prod_{q=1}^{k} \left(1 - \frac{x_q}{MAXV}\right) \tag{19}$$

Let us rewrite (18) as:

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\sum_{l=j_1}^{j_2} \left(\left(\frac{l}{MAXV}\right)^k \times \left(\prod_{q=1}^{k}\left(1 - \frac{x_q}{MAXV}\right)^{l-1}\right)\right)}{\sum_{l=1}^{\infty} \left(\left(\frac{l}{MAXV}\right)^k \times \left(\prod_{q=1}^{k}\left(1 - \frac{x_q}{MAXV}\right)^{l-1}\right)\right)} \tag{20}$$

Now, applying (19) on (20) leads to:

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\sum_{l=j_1}^{j_2} \left(\left(\frac{l}{MAXV}\right)^k \times Q^{l-1}\right)}{\sum_{l=1}^{\infty} \left(\left(\frac{l}{MAXV}\right)^k \times Q^{l-1}\right)} \tag{21}$$

Given that (21) can be approximated by:

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\int_{j_1}^{j_2} \left(\frac{j}{MAXV}\right)^k \times Q^{j-1} dj}{\int_{1}^{\infty} \left(\frac{j}{MAXV}\right)^k \times Q^{j-1} dj} \tag{22}$$

and that, from Appendix C:

$$\int j^k \times Q^{j-1} dj = \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q}} \times Q^{j-1} \times (-1)^{k-q} + CONST \tag{23}$$

applying (23) on (22) leads to:

$$P\left(j_1 \leq m \leq j_2 \,|rmin_1 = x_1 \wedge rmin_2 = x_2 \wedge ... \wedge rmin_m = x_m\right) = \frac{\left[\frac{1}{MAXV^k} \times \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q}} \times Q^{j-1} \times (-1)^{k-q}\right]_{j=j_1}^{j=j_2}}{\left[\frac{1}{MAXV^k} \times \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q}} \times Q^{j-1} \times (-1)^{k-q}\right]_{j=1}^{j \to \infty}} \tag{24}$$

Observe that the sum in (24) does not need to iterate over the interval $j_1..j_2$. It is only k iterations, and k is typically small. Also observe that many terms in (24) change in a similar way. Hence, applying the idea of Horner's rule (normally used for polynomial evaluation) enables a more efficient computation of (24). Algorithms 4 and 5 do this. Now we have an algorithm that can compute the probability that the number of nodes is within $j_1 \leq m \ \leq j_2$.

Algorithm 4 The a posteriori probability that m satisfies $j_1 \leq m \leq j_2$

Require: Lines 1-7 in Algorithm 1 have been executed.

```
1: Q, fj1, fj2, denom : real
2: Q ← compute Q according to (19) using x
3: fj1 ← computef( j1, Q )
4: fj2 ← computef( j2, Q )
5: denom ← computef( 1, Q )
6: return (fj1 − fj2) / denom   // a real number with the a posteriori probability
```

Algorithm 5 Function `computef`

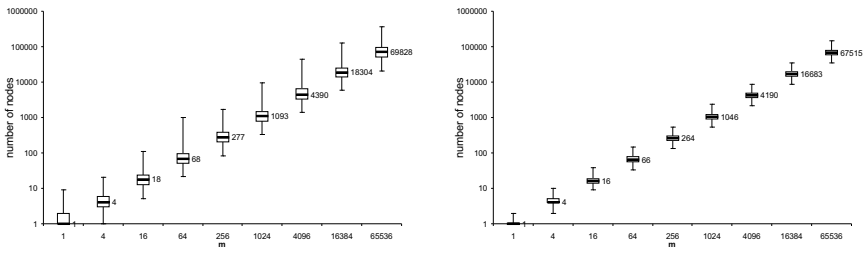
Require: $pow(\ r,\ n\ )$ computes r^n.

```
1: function computef(j : integer, x : array[1..k] of integer, Q : real) return an integer
2:     sum, term : real
3:     sum ← 0
4:     term ← pow(j/MAXV, k) × pow(Q, j − 1)
5:     sum ← sum + term
6:     for q ← (k − 1) downto 0
7:         term ← term × (q + 1) × 1/j × 1/ln Q × (−1)
8:         sum ← sum + term
9:     end for
10:    return sum
11: end function
```

4 Performance Evaluation

Several experiments were developed to show different characteristics of the algorithm. First, to show how the execution time of the algorithm compares with a naïve algorithm (as mentioned in the introduction), the time to execute the algorithm was acquired by using data from previous research [4], and running the proposed algorithm on a cycle accurate simulator for a mote platform, called Avrora [6]. From [4], it is possible to know that the time to run the tournament C_{trt} is 45 ms. Running the algorithm in Avrora, provided a measurement on the time to generate a random number ($\approx$ 0.003 ms, which together with all computations other than the estimation itself, was considered negligible) and the time to compute the function `ML_estimation`, as depicted in Algorithm 2 ($C_{est} \approx 86$ ms, for $k = 5$). Therefore, for $k = 5$, the time to perform the algorithm is $k \times C_{trt} + C_{est} = 5 \times 45 + 86 = 316$ ms.

Let us assume a very simplified model for assessing the overhead of the naïve algorithm. Only the time to transmit messages was considered and everything else is regarded as negligible. Considering a radio transmitting at 38.4 Kbps (a typical value for mote platforms [7]), a message with 2 bytes of data and 3 bytes for header/preamble would take $C_{msg} = \frac{(2+3)\times 8}{38400} \approx 1$ ms. Let us assume that nodes have set up a scheme to orderly access the medium and thus there are no collisions. The time to run a naïve protocol is then $m \times C_{msg}$.



(a) $k = 5$ (b) $k = 20$

Fig. 2. Estimation of the number of nodes for different values of m and k.

From this, it is obvious that, with $k = 5$, our algorithm always runs faster when $m > 316$, and, more importantly, our protocol has a time complexity which only depends on k. We emphasize that this comparison is greatly biased towards favouring the naïve algorithm, since assuming that there is no delay for messages to access the medium is a very optimistic assumption.

Next was studied how the error of the protocol varies as a function of m and k. A simulation of the algorithm proposed in this paper[1] was run with $k = 5$ and $k = 20$, for different numbers of nodes ($m = 1, 4, 16, ..., 2^{16}$). The boxplots in Figure 2 are presented in a logarithmic scale and depict the distribution of 1000 estimations for the different numbers of nodes. In these boxplots, the box stretches from 25^{th} percentile to the 75^{th} percentile. The value of the median is shown and depicted as a line across the box. The minimum values are depicted below the box and the maximum is above.

From the box plots in Figure 2, it is possible to observe that the quality of the estimation improves significantly by increasing k.

5 Discussion and Previous work

5.1 Previous work

The problem of estimating the number of nodes in a network can be viewed from different perspectives. Gossip, rumour spreading and infectious algorithms, all have in common that they use randomized local computations repeatedly to achieve a global computation. Originally these algorithms were developed to propagate data, but recently they have been reworked to calculate aggregated quantities. These algorithms are robust in face of node and link failures and they can operate in multihop networks. Such algorithms are available for a large number of distributed calculations, such as MIN, MAX, SUM, AVERAGE (see for

[1] The source code for the simulation can be found at `http://www.hurray.isep.ipp.pt/widom/`

example [8, 9]). These calculations can be used to calculate/estimate the number of nodes as well. Two techniques are known. The first technique populates the value of zero on all nodes but one; this special node is populated the value of one. Then the protocol calculates the average and taking the inverse of the average gives the number of nodes. The second technique calculates the minimum value and applies maximum likelihood estimation to obtain the number of nodes.

Deterministic algorithms for unstructured environments have also been proposed. The algorithm in [10] performs repeated local operations to compute an average and it works in multihop environments. The algorithm in [11] computes the average in a single-hop network. It is designed to perform well against an adversary that injects faults but unfortunately, its time complexity is high. These techniques that compute averages could (as mentioned above) be used to compute the number of nodes.

Data aggregation protocols for WSN can compute the number of nodes, typically using a convergecast tree [12, 13]. The same problem has been addressed by researchers in data communications with the goal of estimating the size of the audience of a multicast [14, 15].

Common to all these works [8–15] is that their time complexity is $O(m)$ or more whereas our techique has a time complexity which is independent of m.

5.2 Practical issues

So far, it was assumed that all messages transmitted are used to find the number of nodes. This assumption can be easily relaxed. The priority field can be divided into two subfields. The most significant bits are called service identifier and the least significant bits are called data bits. For example, for 20 priority bits; the 2 most significant bits could be the service identifiers and the remaining 18 bits are priority bits. The MAC protocol runs the tournament based on all 20 bits. If the 2 services bits are 00 then the following 18 bits denote the priority of a normal message and these 18 bit number represents a unique priority and it has normal payload and it is collision free. If the 2 bits are 01 it means that the 18 remaining contains data that should be used to compute the number of nodes. An application can make a function call `send_empty`(01, 20) which proposes the value 20 and returns the minimum value.

We have also assumed that all nodes start the execution of the protocol simultaneously. This can be dealt easily by letting a node broadcast a message containing a request to compute the number of nodes. All nodes receive this at approximately the same time. There are small differences in time when nodes start the protocol, but the MAC protocol (see [4]) synchronizes so that the tournament on all nodes executes simultaneously, so this poses no problem.

The overhead introduced by the MAC protocol is to a large extent due to the transition time between transmission and reception. The platform used to implement the MAC protocol in [4] had a switching time of 192μs. But this is a technological parameter that can be improved with better radio hardware, as witnessed by the fact that the Hiperlan standard [16] required a switching time of 2μs.

6 Conclusions and Future work

A technique for efficiently estimating the number of nodes was proposed. It has a time-complexity which does not depend on the number of nodes. The algorithm depends on a prioritized MAC protocol. This has already been implemented and tested. Our protocol requires that no faults occur; in general this is difficult to achieve in practice in wireless networks. However, it was observed that in short distance communication, using a spread spectrum transceiver, it is possible to achieve good reliability [4].

We belive that it is in scenarios with a very large number of nodes (say 100 000) in a small area that our protocol and our estimation technique is the most useful.

The efficiency of this technique greatly depends on the overhead of the MAC protocol. If a transceiver with faster transmit/receive switching times were available, and it offered the flexibility to design the MAC protocol in software then the overhead of the protocol could be reduced greatly.

References

1. J. Elson and D. Estrin. Sensor networks: a bridge to the physical world. *Wireless Sensor Networks*, pages 3–20, 2004.
2. A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, M. Gouda, Y. Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extreme scale wireless sensor network. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 102–108, Washington, DC, USA, 2005. IEEE Computer Society.
3. B. Andersson and E. Tovar. Static-priority scheduling of sporadic messages on a wireless channel. In *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS'05)*, Pisa, Italy, 2005.
4. N. Pereira, B. Andersson, and E. Tovar. Implementation of a dominance protocol for wireless medium access. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06), to appear*, Sydney, Australia, 2006.
5. Bosch. *CAN Specification, ver. 2.0, Robert Bosch GmbH, Stuttgart*, 1991.
6. AVRORA - the AVR simulation and analysis framework, 2005.
7. Crossbow. MICA2 - wireless measurement system product datasheet, 2005.
8. D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 482–491, Washington, DC, USA, 2003. IEEE Computer Society.
9. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.
10. D.S. Scherber and H.C. Papadopoulos. Distributed computation of averages over ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 23(4):776– 787, 2005.
11. M. Kutylwski and D. Letkiewicz. Computing average value in ad hoc networks. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, pages 511–520. Springer, 2003.
12. Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR'03)*, 2003.
13. S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI'02)*, 2002.
14. K. Horowitz and D. Malkhi. Estimating network size from local information. *Information Processing Letters*, 88(5):237–243, 2003.
15. M. Nekovee, A. Soppera, and T. Burbridge. An adaptive method for dynamic audience size estimation in multicast. In *Lecture Notes in Computer Science*, volume 2816, pages 23–33, 2003.
16. ETSI (European Telecommunications Standards Institute). *Broadband Radio Access Networks(BRAN);HIPERACCESS; PHY protocol specification.*

Appendix A

For r_i it holds that:

$$P\left(r_i \leq x\right) = \frac{x}{MAXV}$$

Exploiting the fact that r_i is an integer:

$$P\left(x+1 \leq r_i\right) = 1 - \frac{x}{MAXV}$$

Since r_1, r_2,...,r_m are independent random variables:

$$P\left(x+1 \leq \min\left(r_1, r_2, ..., r_m\right)\right) = \left(1 - \frac{x}{MAXV}\right)^m$$

Since r_1, r_2,...,r_m are independent random variables:

$$P\left(\min\left(r_1, r_2, ..., r_m\right) \leq x\right) = 1 - \left(1 - \frac{x}{MAXV}\right)^m$$

Appendix B

We can reason about conditional probabilities as follows:

$$P\left(m = j \,|\min\left(r_1, r_2, ..., r_m\right) = x\,\right) = \frac{P\left(m = j \wedge \min\left(r_1, r_2, ..., r_m\right) = x\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)}$$

and like this:

$$\frac{P\left(m = j \wedge \min\left(r_1, r_2, ..., r_m\right) = x\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)} = \frac{P\left(\min\left(r_1, r_2, ..., r_m\right) = x \wedge m = j\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)}$$

and like this:

$$\frac{P\left(\min\left(r_1, r_2, ..., r_m\right) = x \wedge m = j\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)} = \frac{P\left(\min\left(r_1, r_2, ..., r_m\right) = x \,|m = j\,\right) \times P\left(m = j\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)}$$

Now, combining the three previous equalities:

$$P\left(m = j \,|\min\left(r_1, r_2, ..., r_m\right) = x\,\right) = \frac{P\left(\min\left(r_1, r_2, ..., r_m\right) = x \,|m = j\,\right) \times P\left(m = j\right)}{P\left(\min\left(r_1, r_2, ..., r_m\right) = x\right)}$$

The previous expression can be rewritten as:

$$P\left(m = j \,|\min\left(r_1, r_2, ..., r_m\right) = x\,\right) = \frac{P\left(\min\left(r_1, r_2, ..., r_m\right) = x \,|m = j\,\right) \times P\left(m = j\right)}{\sum_{l=1}^{\infty} P\left(\min\left(r_1, r_2, ..., r_m\right) = x \,|m = l\,\right) \times P\left(m = l\right)}$$

Appendix C

Let us define $(a)_b$ for integers a and b as:

$$(a)_b = \frac{a!}{(a-b)!}$$

Let f be defined as follows (observe that ln means natural logarithm):

$$f(j) = \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q-1}} \times \frac{Q^{j-1}}{\ln Q} \times (-1)^{k-q}$$

Let us take the derivative of f with respect to j:

$$\frac{df}{dj} = \left(\sum_{q=0}^{k} (k)_{k-q} \times q \times j^{q-1} \times \frac{1}{(\ln Q)^{k-q-1}} \times \frac{Q^{j-1}}{\ln Q} \times (-1)^{k-q} + \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q-1}} \times Q^{j-1} \times (-1)^{k-q} \right)$$

Let us insert $q = p$ in the upper term of $\frac{df}{dj}$ and study that. We have:

$$(k)_{k-p} \times p \times j^{p-1} \times \frac{1}{(\ln Q)^{k-p-1}} \times \frac{Q^{j-1}}{\ln Q} \times (-1)^{k-p}$$

Now, inserting $q = p$-1 in the lower term of $\frac{df}{dj}$ and study that. We have:

$$(k)_{k-q+1} \times j^{p-1} \times \frac{1}{(\ln Q)^{k-p}} \times Q^{j-1} \times (-1)^{k-p+1}$$

It can be seen that these terms have the same magnitude but with opposite sign. Hence, they cancel out. Also observe that the upper term of $\frac{df}{dj}$ becomes zero for q=0. Hence:

$$\frac{df}{dj} = j^k \times Q^{j-1}$$

Naturally this results that:

$$\int j^k \times Q^{j-1} dj = f + CONST$$

Using the definition of f yields:

$$\int j^k \times Q^{j-1} dj = \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q-1}} \times \frac{Q^{j-1}}{\ln Q} \times (-1)^{k-q} + CONST$$

Finally, this can be rewritten to:

$$\int j^k \times Q^{j-1} dj = \sum_{q=0}^{k} (k)_{k-q} \times j^q \times \frac{1}{(\ln Q)^{k-q}} \times Q^{j-1} \times (-1)^{k-q} + CONST$$