Lecture Notes in Electrical Engineering 81

Massimo Conti
Simone Orcioni
Natividad Martínez Madrid
Ralf E.D. Seepold
Editors

Solutions on Embedded Systems

Springer

Lecture Notes in Electrical Engineering

For further volumes:
http://www.springer.com/series/7818

Massimo Conti · Simone Orcioni
Natividad Martínez Madrid
Ralf E. D. Seepold
Editors

Solutions on Embedded Systems

Springer

Editors
Prof. Dr. Massimo Conti
Dip. di Ingegneria Biomedica
Elettronica e Telecomunicazioni
DIBET
Università Politecnica delle Marche
Via brecce bianche 12
Ancona 60131
Italy
e-mail: m.conti@univpm.it

Prof. Dr. Simone Orcioni
Dip. di Ingegneria Biomedica
Elettronica e Telecomunicazioni
DIBET
Università Politecnica delle Marche
Via brecce bianche 12
Ancona 60131
Italy
e-mail: s.orcioni@univpm.it

Prof. Dr. Natividad Martínez Madrid
Computer Science
Reutlingen University
Alteburgstr. 150
Reutlingen 72762
Germany
e-mail: Natividad.martinez@
reutlingen-university.de

Prof. Dr. Ralf E. D. Seepold
Hochschule für Technik,
Wirtschaft und Gestaltung (HTWG)
Hochschule Konstanz
Brauneggerstrasse 55
Konstanz 78462
Germany
e-mail: ralf.seepold@htwg-konstanz.de

ISSN 1876-1100 e-ISSN 1876-1119

ISBN 978-94-007-0637-8 e-ISBN 978-94-007-0638-5

DOI 10.1007/978-94-007-0638-5

Springer Dordrecht Heidelberg London New York

Cover design: eStudio Calamar, Berlin/Figueres

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Today electronic computation is performed mainly not in personal computers, but in electronic systems integrated in devices that we use every day, like cars, mobile phones, household appliances and credit cards. Embedded computing gives a substantial added value to products. Innovation in many fields such as automotive, industrial automation, telecommunications, consumer electronics, entertainment and health equipment is mainly due to embedded computing.

Electronic systems give new features to the device, such as: energy management and power reduction, safety and security, comfort and ease to use.

The use of embedded systems in many different fields may help us to find a solution to problems that are strategic for the future of the world, such as:

- Energy production, management and delivery;
- Control and monitoring of the environment;
- Food production;
- Efficient and sustainable manufacturing;
- Traffic and mobility control and monitoring;
- Security and critical infrastructure protection;
- Home and building automation;
- Healthcare systems;
- Systems for integration of ageing and disabled people.

The book "Solutions on Embedded Systems" presents an overview on several fields of applied research, like sensor networks, network on chip and multicore systems, automotive applications, software design, system architectures, design of low power embedded systems. Each area is covered by a separate part of the book.

Contents

Chapter 1
Performance of Gossip Algorithms in Wireless Sensor Networks

Marco Baldi, Franco Chiaraluce and Elma Zanaj

1.1 Introduction

Ad-hoc wireless sensor networks are peer-to-peer systems formed by many small and simple devices, able to measure some quantities and to transmit their measured values to neighboring nodes. In such networks, nodes communicate in order to merge their single contributions into a common result. This also occurs in averaging problems, whose target is to calculate, in a distributed manner, the average value of a quantity of interest (e.g., temperature). Because of their features, these networks are suitable for many purposes, as environmental monitoring applications, allowing accurate control over large areas with favorable cost-to-benefit ratio [1]. Among these applications, however, hostile environments and scenarios of natural and man-made disasters represent great challenges, in which the network availability must be ensured, in spite of a number of possible impairments.

Among the several protocols that are available nowadays for sensors communication, an increasing attention has been devoted to simple decentralized procedures based on the gossip principle, through which the computational burden is distributed among all nodes.

M. Baldi and F. Chiaraluce (✉)
Dipartimento di Ingegneria Biomedica, Elettronica e Telecomunicazioni, Facoltà di Ingegneria, Università Politecnica delle Marche, Ancona, Italy
e-mail: f.chiaraluce@univpm.it

M. Baldi
e-mail: m.baldi@univpm.it

E. Zanaj
Departamenti i Elektronikes dhe Telekomunikacionit, Fakulteti i Teknologjise se Informacionit, Universiteti Politeknik i Tiranes, Tirana, Albania
e-mail: ezanaj@gmail.com

M. Conti et al. (eds.), *Solutions on Embedded Systems*,
Lecture Notes in Electrical Engineering, 81, DOI: 10.1007/978-94-007-0638-5_1,

The gossip algorithm was originally conceived for telephone networks [2, 3]. When gossip is applied in sensor networks, noting by x_i and x_j the local measures of the i-th and j-th nodes, an interaction among them updates one or both their values, that are then used for a subsequent interaction. The communication protocols can be managed either in a synchronous or in an asynchronous way, but the latter is more practical, because of its inherent simplicity. So, in this chapter, we will limit to consider an asynchronous time model, in which any node has a clock which ticks independently at the times of a rate 1 Poisson process. Therefore, the inter-tick times at any node are rate 1 exponentials, independent across nodes and over time.

Various implementations of gossip for averaging problems are possible; they all aim at estimating the mean value of the sensed quantity. More precisely, let us denote by N the number of nodes and by $\mathbf{x}(k) = [x_1(k), x_2(k), \ldots, x_N(k)]^T$ the vector of the estimates of all nodes after k clock ticks (superscript T denotes the transpose operation). The target of the algorithm is to find a reliable measure of the average value $x_{\text{ave}} = \sum_{i=1}^{N} x_i(0)/N$ in the shortest possible time, that is, maximizing the convergence speed.

In a first implementation, called "basic gossip" in the following, an interaction among the i-th and j-th nodes produces as output $x_i(k+1) = x_j(k+1) = x_i(k)/2 + x_j(k)/2$, that is used by both nodes for the subsequent interaction [4]. A variant of this proposal consists in the so-called "push-sum" algorithm [5]. According with such protocol, a node forwards a share of its values, properly defined, to one of its neighbors, randomly selected, while keeping the remaining part. The performance of the push-sum algorithm depends on the choice of the share, which therefore represents a degree of freedom to optimize.

Both the basic gossip and the push-sum algorithm are point-to-point protocols. However, in a wireless network, when a node transmits, all nodes in its coverage area can receive the transmitted data. This suggests implementing a "broadcast" algorithm to reduce the averaging time.

Although the fundamentals of the considered protocols are well known and a number of papers on these topics already appeared in previous literature, several issues are still open. Among them, we have mentioned above the problem of optimizing the share values in the push-sum algorithm. In [5], the authors limited to say that the choice of the shares may be deterministic or random, and may or may not depend on the time, without providing, however, a numerical evidence of the impact resulting from the different choices. The same was, at our best knowledge, in the subsequent literature. Only very recently, in [6], we presented a first set of numerical and theoretical results on this issue, focusing on ring and random geometric graph topologies.

Another relevant topic, rarely explored in the past, concerns the evaluation of the performance of gossip algorithms in the presence of link failures. Actually, when averaging algorithms are adopted in wireless sensor networks, the shadow fading or other kinds of radio impairments could prevent some links from being used, due to their poor quality in terms of signal-to-noise ratio.

The study of networks with link failures could seem not different from that of non-fully-meshed networks, where, because of a limited coverage radius, each node can reach directly only a limited set of neighbors, being linked to the others only

through multiple hops (which means to pass through intermediate nodes). Really, the two situations are rather different; failures can be modeled as a stochastic phenomenon, and when a percentage x of links fail, malfunctions are generally distributed at random, without any specific correlation between distinct failures. Obviously, in some cases, failures may be due to mechanisms involving simultaneously a number of nodes that are close one each other; but this appears as a particular case, while the uncorrelation assumption seems more suitable to model practical situations. In this chapter, the convergence speed of the selected gossip algorithms, in presence of random link failures throughout the network, is investigated. Our analysis is mainly based on numerical simulations, but some theoretical issues are also discussed, particularly in regard to the share optimization when the push-sum approach is applied. We develop a number of comparisons, with the aim to show the limits and potentialities of the considered techniques.

In Sect. 1.2 we define the considered gossip versions. In Sect. 1.3 we introduce the simulation parameters and describe the graph whose performance in the presence of link failures will be investigated afterwards. In Sect. 1.4 we face, from a theoretical viewpoint, the problem of the share factor optimization in the push-sum algorithm; an analytical approach is developed, based on the computation of the potential function. In Sect. 1.5 we present a number of simulation results, first considering the various algorithms separately, and then in comparative terms. Most of the chapter contents were originally presented in [7].

1.2 The Considered Gossip Algorithms

1.2.1 Basic Gossip

The basic gossip algorithm is very simple, and has been briefly described in Sect. 1.1. Its main steps are as follows:

1. Node i chooses (at random) another node, j, inside its coverage area.
2. Nodes i and j split their information into two equal parts, $x_i(k)/2$ and $x_j(k)/2$, keeping one part and sending the other.
3. Nodes i and j calculate their new estimates by adding the received value to that already stored: $x_i(k+1) = x_j(k+1) = [x_i(k) + x_j(k)]/2$.

The choice of j is done according with a uniform distribution, conditioned on the value of the Euclidean distance D_{ij} between nodes i and j. In other words, the probability that node i contacts node j ($\neq i$) when it is selected for transmission is given by:

$$p_{ij} = \begin{cases} \frac{1}{d_i}, & D_{ij} \leq r, \\ 0, & D_{ij} > r, \end{cases} \tag{1.1}$$

where d_i is the number of nodes within its coverage area, that is delimited by a coverage radius r, assumed to be equal for all nodes. So, the coverage radius

represents the maximum distance at which a node can transmit reliably. Clearly, in order to recognize the nodes inside the coverage area, a query session is required, before interaction starts. We suppose that each node performs a very simple query aimed at knowing the number of reachable neighbors, d_i, in its coverage area. More sophisticated localization strategies [8] can be adopted, that permit to implement more efficient versions of averaging algorithms. In [9], for example, a geographic gossip has been proposed, based on greedy routing, that is potentially able to provide remarkable gains. But applicability of this kind of protocols, where each node must compute and compare a large number of distances from a prefixed target, seems difficult. For this reason, we have not included these gossip versions in our study. Alternatively, the selection probabilities could be optimized with the final goal to maximize the convergence speed [4] but, once again, this would make more involved the interaction while not providing, in many cases, substantial improvements [10].

The probabilities p_{ij} can be collected in a matrix $\mathbf{P}$, with $N \times N$ entries. This matrix is stochastic, i.e., each of its rows sums to 1. On the other hand, if the link between i and j fails, the corresponding p_{ij} is set equal to zero. In this case, matrix $\mathbf{P}$ is no longer stochastic, and the i-th node has a probability to communicate $\sum_j p_{ij} < 1$. In other words, if the link between i and j fails, at some clock tick the i-th node tries transmitting to the j-th node without success, thus wasting the communication attempt. Obviously, this reflects on the averaging time, which increases in a manner dependent on the number of faulty links and their distribution.

1.2.2 Push-Sum Algorithm

The push-sum protocol proceeds as follows. At the i-th node, with $i = 1, 2, \ldots, N$, two quantities are stored and updated through the interaction with the other nodes: they are named $s_i(k)$ and $w_i(k)$, respectively. These quantities satisfy the following mass conservation properties, for any k:

$$\sum_{i=1}^{N} s_i(k) = \sum_{i=1}^{N} x_i(0) = N x_{\text{ave}}, \quad \sum_{i=1}^{N} w_i(k) = N. \tag{1.2}$$

When the protocol starts, that is, once having acquired the sensed values, we have $s_i(0) = x_i(0)$ and $w_i(0) = 1$, $\forall i$. Later, if the clock of the i-th node ticks at the k-th time instant (let us remind that transmission is asynchronous in the considered system), it selects randomly one of its neighbors, say j, and sends to it a fraction $(1 - \alpha)$ of its parameters, while it retains the remaining fraction α. So, the parameters at nodes i and j are modified as follows:

$$\begin{aligned} s_i(k+1) &= \alpha s_i(k), \quad w_i(k+1) = \alpha w_i(k), \\ s_j(k+1) &= s_j(k) + (1-\alpha) s_i(k), \\ w_j(k+1) &= w_j(k) + (1-\alpha) w_i(k), \end{aligned} \tag{1.3}$$

while the parameters at all the other nodes remain unchanged. This way, conditions (1.2) are certainly satisfied. A new estimate at the interacted nodes is then derived as $x_m(k+1) = s_m(k+1)/w_m(k+1)$, with $m = i, j$.

In [5], where, besides point-to-point communications, also broadcast transmissions were considered, a more general mechanism was applied, where the share factor can be different for any node and even variable in time. This model, however, seems too involved for practical applications; so, we prefer to consider a single and constant α, whose value should be optimized in order to achieve the fastest convergence speed.

On the other hand, a bidirectional version of the push-sum algorithm could also be adopted where, every time node i contacts node j, sending to it a fraction of its message, node j does the same, sending to node i a share of its own message. It is possible to demonstrate (details are omitted for saving space) that, at least for a fully-meshed network, the optimum share for this case is 1/2. So, under this choice, such modified version of the push-sum algorithm practically becomes identical to the basic gossip.

1.2.3 Broadcast Algorithm

The idea to implement a broadcast algorithm originates from the observation that, when a node transmits some information, all the other nodes in its coverage area are able to receive the transmitted data. This suggests implementing a broadcast averaging algorithm that, at the expense of a slight increase in complexity, allows reducing significantly the averaging time. This broadcast algorithm is unidirectional, as the information flows from a transmitting node to a number of receiving nodes (depending on the coverage radius and the random nodes distribution) but not in the opposite sense. Similarly to the push-sum algorithm, the i-th node maintains a sum, $s_i(k)$, and a weight, $w_i(k)$. When the algorithm starts, that is for $k = 0$, we have $w_i(0) = 1$ and $s_i(0) = x_i(0)$, that coincides with the initial sensed value at node i. When the i-th node's clock ticks, say at step k, the node splits its information into a number of parts; it may keep the first, so that $[w_i(k+1), s_i(k+1)] = \alpha_i[w_i(k), s_i(k)]$, while it sends to each neighbor j one of the remaining parts: $\alpha_{ij}[w_i(k), s_i(k)]$. Node j receives the transmission and updates its values by adding the received ones, so that $[w_j(k+1), s_j(k+1)] = [w_j(k), s_j(k)] + \alpha_{ij}[w_i(k), s_i(k)]$. As stated in the expressions, this mechanism is ruled by the share parameters, α_i and α_{ij}, that can be collected in a matrix $\mathbf{A}$, having $\alpha_{ii} = \alpha_i$ along the main diagonal. The elements of $\mathbf{A}$, that satisfy the condition $\Sigma_j \alpha_{ij} = 1$, can be chosen at random or following some suitable deterministic rule. Although different laws [11] can be adopted, in [12] we showed that good results are obtained assuming $\alpha_i = 0$ and:

$$\alpha_{ij} = \begin{cases} \frac{1}{d_i} & D_{ij} \leq r, j \neq i, \\ 0 & D_{ij} > r. \end{cases} \tag{1.4}$$

1.3 Simulation Parameters

The convergence speed of the considered protocols is evaluated through the computation of the normalized difference between the estimated average and the true average. More precisely, we determine in R simulations (with R sufficiently large) the random variable $e(k) = \|\mathbf{x}(k) - x_{\text{ave}}\mathbf{1}\|/\|\mathbf{x}(0)\|$, where $\|\mathbf{x}\|$ denotes the l_2 norm of vector $\mathbf{x}$ and $\mathbf{1}$ is the vector of all ones. A set of R curves $e^m(k)$ is obtained, $m = 1 \ldots R$, that are averaged in order to compute:

$$\langle e(k) \rangle = \frac{1}{R} \sum_{m=1}^{R} e^m(k) \tag{1.5}$$

which has the meaning of estimated mean curve. In order to average over possible different initial conditions, $\mathbf{x}(0)$ is randomly changed at the beginning of each simulation. According to the probability theory, it is known that $\lim_{R\to\infty} \langle e(k) \rangle = \widehat{e(k)}$, where $\widehat{e(k)}$ represents the true average of $e(k)$. Once having determined (1.5), the averaging time is defined as the number of clock ticks, say k^*, that permits to have a normalized difference smaller than, or equal to, a prefixed value, for example $e(k^*) \leq 10^{-10}$.

In this chapter, simulations are done over a random geometric graph (RGG), where nodes are randomly distributed in a unit square, according with a 2D homogeneous Poisson point process. The number of neighbors, d_i, the i-th node is linked to, gives its *nodal degree*. In the case of regular graphs (like the ring, for example) d_i is equal for all nodes, and it is a direct measure of the connectivity level of the network. On the other hand, in general, each node is characterized by the coverage radius r; for the RGG, even assuming that all nodes have the same r, the nodal degree is generally not unique. Moreover, for each value of r, the connectivity level can vary from graph to graph. So, an average nodal degree, $\langle d \rangle$, must be computed for the analysis purposes. The behavior of $\langle d \rangle$, as a function of r, is shown in Fig. 1.1; for each value of the coverage radius, 100 RGGs have been randomly generated, and their nodal degrees have been averaged.

As one of the objects of our study is to compare the impact of failures against that of a limited r, we suppose to start with a fully-meshed network (that implies to

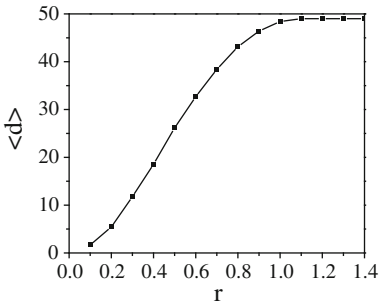


Fig. 1.1 Average value of d, computed over 100 random geometric graphs

have $r \geq \sqrt{2}$ on the unit square) and to eliminate, at random, a fraction x of its links. So, while in absence of failures the network connectivity is $N-1$ (see Fig. 1.1), in the new scenarios the average value of d becomes approximately:

$$\langle d \rangle = (N-1)(1-x). \tag{1.6}$$

The validity of (1.6) has been confirmed through simulation.

1.4 Share Factor Optimization

As mentioned in Sect. 1.2.2, for the push-sum algorithm an important issue concerns optimization of the share factor α that appears in (1.3). A useful analytical tool, in this sense, is provided by the *potential function* method.

Let us consider a vector $\mathbf{v}_i(k)$ whose components, $v_{ij}(k)$, are such that:

$$s_i(k) = \sum_{j=1}^{N} v_{ij}(k) x_j(0). \tag{1.7}$$

The following condition holds: $w_i(k) = \sum_j v_{ij}(k)$. So, if $v_i(k)$ is nearly proportional to the all-one vector, then $x_i(k) = s_i(k)/w_i(k)$ is close to the true average. The potential function at time k is defined as follows [5]:

$$\Phi(k) = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[v_{ij}(k) - \frac{w_i(k)}{N} \right]^2. \tag{1.8}$$

In the limit case of all nodes perfectly aware of the true average, the potential function is null. Therefore, evaluating the mean potential function, for any k, permits us to estimate the convergence speed of the algorithm.

More precisely, assuming that, at instant k, node l is selected as the transmitter and node m as the receiver, the following difference between the potential functions at time instants k and $k+1$ can be easily derived:

$$\begin{aligned} \delta\Phi = \Phi(k) - \Phi(k+1) &= 2\alpha(1-\alpha) \sum_{j=1}^{N} \left[v_{lj}(k) - \frac{w_l(k)}{N} \right]^2 \\ &- 2(1-\alpha) \sum_{j=1}^{N} \left[v_{lj}(k) - \frac{w_l(k)}{N} \right] \cdot \left[v_{mj}(k) - \frac{w_m(k)}{N} \right]. \end{aligned} \tag{1.9}$$

In the following of this section we will omit, for the sake of simplicity, the argument k. We wish to compute the average of (1.9) over all possible choices, uniformly distributed, of the transmitting and receiving nodes. For a fully-meshed network, through simple algebra, it is possible to find:

$$\langle \delta\Phi \rangle = \frac{2}{N}\left[\alpha(1-\alpha) + \frac{1-\alpha}{N-1}\right]\Phi, \tag{1.10}$$

where $\Phi = \Phi(k)$. A criterion for optimizing the value of α can consist in maximizing $\langle \delta\Phi \rangle / \Phi$. According with its own meaning, in fact, to have a large $\langle \delta\Phi \rangle$, for a given Φ, should reflect in a high convergence speed. Now, from (1.10), $\langle \delta\Phi \rangle / \Phi$ is maximum for:

$$\alpha_{\text{opt}} = \frac{N-2}{2(N-1)}, \tag{1.11}$$

and, for N sufficiently large, such value can be approximated by 0.5.

In the case of non-fully-meshed network, instead, that can occur because of a limited value of r and/or the appearance of link failures, Eq. 1.10 is no longer valid, and must be replaced as follows:

$$\begin{aligned}\langle \delta\Phi \rangle = {} & \frac{2\alpha(1-\alpha)}{N}\Phi + \frac{2(1-\alpha)}{N}\sum_{j=1}^{N}\sum_{l=1}^{N}\frac{1}{d_l}\left(v_{lj} - \frac{w_l}{N}\right)^2 \\ & - \frac{2(1-\alpha)}{N}\sum_{j=1}^{N}\sum_{l=1}^{N}\frac{1}{d_l}\sum_{m\in C_l}\left(v_{lj} - \frac{w_l}{N}\right)\left(v_{mj} - \frac{w_m}{N}\right),\end{aligned} \tag{1.12}$$

where C_l is the subset of nodes that includes node l and the nodes it is linked to. The higher complexity of (1.12), with respect to (1.10), is evident. First of all, a new contribution has been added, that is null in the case of a fully-meshed network, because of the mass conservation property (1.2). Secondly, it seems not possible to evidence, at the right side, the potential function Φ, that is a necessary step toward maximization of $\langle \delta\Phi \rangle / \Phi$.

To circumvent the problem, we introduce the position $1/d_l \approx \langle 1/d \rangle$, $\forall l = 1 \ldots N$. Based on this approximation, Eq. 1.12 can be rewritten as:

$$\begin{aligned}\langle \delta\Phi \rangle \approx {} & \frac{2}{N}\left[\alpha(1-\alpha) + \left\langle\frac{1}{d}\right\rangle(1-\alpha)\right]\Phi \\ & - \frac{2(1-\alpha)}{N}\left\langle\frac{1}{d}\right\rangle\sum_{j=1}^{N}\sum_{l=1}^{N}\sum_{m\in C_l}\left(v_{lj} - \frac{w_l}{N}\right)\left(v_{mj} - \frac{w_m}{N}\right).\end{aligned} \tag{1.13}$$

However, the problem of evaluating the last term remains. An estimation of such term can be obtained, based on the definition of Laplacian matrix [13], as reported next. The Laplacian matrix $\mathbf{Q}(G)$ of a graph $G(V, E)$, where V is the vertex set containing the N nodes and E is the edge set, is an $N \times N$ matrix whose elements are defined as follows:

$$Q_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and } (i,j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{1.14}$$

The eigenvalues of $\mathbf{Q}$ are called the Laplacian eigenvalues. They are all real and non-negative, and satisfy the condition: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$. λ_2 is also known as the algebraic connectivity, and is particularly important; it is equal to zero only if G is disconnected. Other properties of matrix $\mathbf{Q}$ and its eigenvalues can be found in the literature (see [14], for example).

Let $y_{ij} = v_{ij} - w_i/N$, $i = 1\ldots N$, be the components of a vector $\mathbf{y}_j$. Through simple algebra, Eq. 1.13 can be rewritten as follows:

$$\langle \delta\Phi \rangle = -\frac{2(1-\alpha)^2}{N}\Phi + \left\langle \frac{1}{d} \right\rangle \frac{2(1-\alpha)}{N} \sum_{j=1}^{N} \mathbf{y}_j^T \mathbf{Q} \mathbf{y_j}. \tag{1.15}$$

Let us define $\mathbf{z} = (\mathbf{y}_1^T, \mathbf{y}_2^T, \ldots, \mathbf{y}_N^T)^T$; it is evident that $\mathbf{z}^T\mathbf{z} = \Phi$. Moreover, let us consider a block matrix $\mathbf{L}$, with size $N^2 \times N^2$, having N repetitions of $\mathbf{Q}$ along the main diagonal and all the other blocks equal to the null matrix. Also $\mathbf{L}$ can be interpreted as a Laplacian matrix, whose eigenvalues coincide with those of $\mathbf{Q}$, but each appears with multiplicity N. Using these further definitions, Eq. 1.15 can be rewritten as:

$$\langle \delta\Phi \rangle = \left[-\frac{2(1-\alpha)^2}{N} + \left\langle \frac{1}{d} \right\rangle \frac{2(1-\alpha)}{N} \mathrm{RQ} \right] \Phi, \tag{1.16}$$

having denoted by $\mathrm{RQ} = \mathbf{z}^T\mathbf{L}\mathbf{z}/\mathbf{z}^T\mathbf{z}$ the so-called Rayleigh quotient. So, the value of α that maximizes $\langle \delta\Phi \rangle / \Phi$ results in:

$$\alpha_{\mathrm{opt}} = 1 - \left\langle \frac{1}{d} \right\rangle \frac{\mathrm{RQ}}{2}. \tag{1.17}$$

Because of the Courant–Fischer minimax theorem [15], we have $\lambda_2 \leq \mathrm{RQ} \leq \lambda_N$. As a confirmation of the correctness of (1.17), we can observe that, in the case of a fully-meshed network, all the eigenvalues λ_i, with $i \geq 2$, are equal to N and Eq. 1.17 becomes equal to Eq. 1.11. In general, the value of RQ is not easy to determine. So, we simplify the problem by approximating RQ with the average of the non-null eigenvalues, i.e.:

$$\mathrm{RQ} \approx \frac{\sum_{i=2}^{N} \lambda_i}{N-1}. \tag{1.18}$$

In [6] we verified that the value of α_{opt} obtainable from this assumption, in case of $r < 0.6$, can be rather different from the actual optimum value. On the contrary, because of the hypothesis of randomly distributed failures, approximation (1.18) is much more acceptable when the nodal degree reduction is due to faulty links. This remark will be confirmed in Sect. 1.5.

By employing the analytical approach, for an RGG with $N = 50$ and $10 < \langle d \rangle < 49$, we have found $0.39 < \alpha_{\mathrm{opt}} < 0.49$. As, from (1.6), such range of values of $\langle d \rangle$ corresponds to $x < 0.796$, we can expect that α_{opt} does not change significantly, even for very large failure rates. This conclusion will be confirmed,

Table 1.1 Coverage radius and failure rates producing nearly identical average connectivity

Limited r $(x = 0)$	$\langle d \rangle$	Failure rate x $\left(r \geq \sqrt{2}\right)$
0.3	11.8	0.76
0.4	18.6	0.63
0.5	26.2	0.48
0.6	32.7	0.35
0.7	38.4	0.23
0.8	43.1	0.14
0.9	46.4	0.07

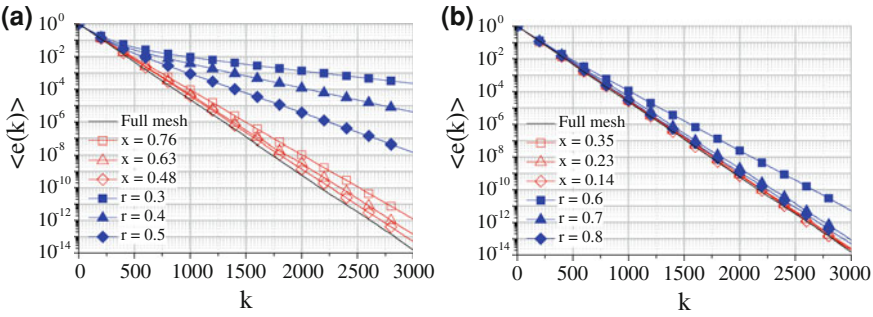


Fig. 1.2 $\langle e(k) \rangle$ for some values of radius and failure rate x (basic gossip algorithm)

in the following section, through numerical simulations, and is different from that occurring in the case of limited coverage radius, where, in the same range of $\langle d \rangle$, α_{opt} can become as low as 0.2.

1.5 Results

In this section we present a number of simulation results for the RGG with $N = 50$. Table 1.1 shows the values of $\langle d \rangle$, together with the coverage radius r (for a non-fully-meshed network with no failures) and the link failure rate x (for a fully-meshed network affected by failures). In the table, each row specifies r and x that determine (nearly) the same average connectivity level. So, we are able to compare the impact of the two different mechanisms that may be responsible for the reduction in the network connectivity. In the following, the pairs determining the same connectivity level will be denoted by r/x.

1.5.1 Basic Gossip

The simulated curves of mean normalized error are shown in Fig. 1.2. We can fix the attention on a specific error value and compare the k^* needed. A couple of

Table 1.2 Number of clock ticks required to reach $\langle e(k^*)\rangle = 10^{-10}$

r/x	k^* (limited radius)	k^* (random failures)
$\geq\sqrt{2}/0$	2,165	2,165
0.6/0.35	2,661	2,210
0.4/0.63	>3,000	2,347

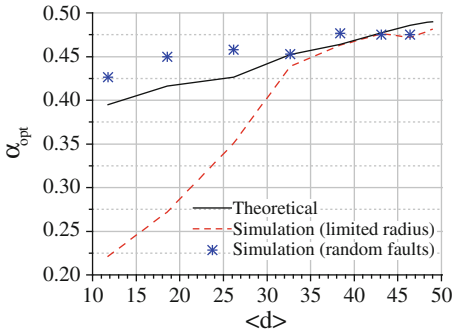


Fig. 1.3 Simulated α_{opt} for the push-sum algorithm

numerical examples are shown in Table 1.2; the k^* for the starting full-mesh network with no link failures is also reported as a benchmark. We notice that both mechanisms increase the convergence time, but the impact of the limited radius is stronger. In other words, for a given value of $\langle d\rangle$, to achieve the target requires a longer time (higher k^*) when the network is non-fully-meshed because of the limited coverage radius.

1.5.2 Push-Sum Algorithm

One problem for the push-sum algorithm is the optimization of the share factor α. The theoretical analysis developed in Sect. 1.4 gives a solid reference that, however, needs to be verified. For this purpose, we have considered $0.1 \leq x \leq 0.9$ and, for any value of the failure rate x, we have determined α_{opt} as the value of α minimizing the averaging time over a large number of repetitions of the random experiment. The result obtained is shown in Fig. 1.3, as a function of the average nodal degree. The curve is rather irregular but the optimal α is comprised between 0.43 and 0.48, which is in line with the results of the analysis in Sect. 1.4. The theoretical approach is not able to distinguish between the case of a limited radius and that of link failures. From the figure, we see that the actual network behavior is well predicted by the theory when missing links are distributed at random. When they follow from a limited coverage radius, instead, numerical simulations give results significantly different from theoretical expectations, particularly for low connectivity levels (see dashed line in Fig. 1.3).

Based on the simulation results, we can also say that the optimal value of α is close to 0.5 for the case of random faults, practically for any value of $\langle d\rangle$

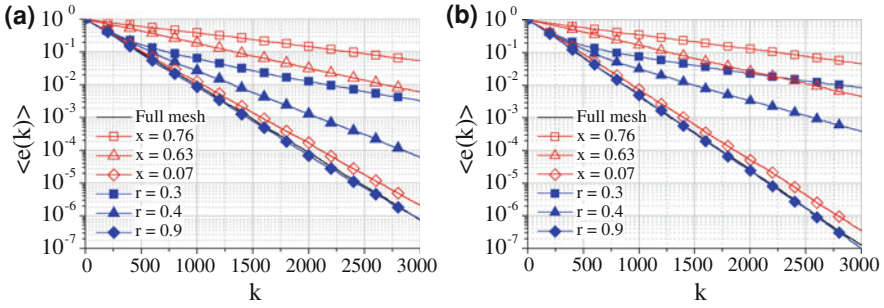
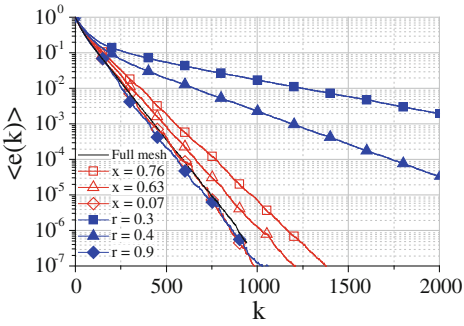


Fig. 1.4 $\langle e(k) \rangle$ for some values of coverage radius r and failure rate x by using the push-sum algorithm with **a** $\alpha = 0.3$ and **b** $\alpha = 0.5$

Fig. 1.5 $\langle e(k) \rangle$ for some values of radius r and failure rate x (broadcast algorithm)



(as observed, this is predicted by the theory), while smaller values should be adopted for α in the case of limited radius, particularly when $r < 0.6$. This statement is confirmed in Fig. 1.4, where $\alpha = 0.3$ and $\alpha = 0.5$ have been considered for both situations.

While in case of link failures the result for $\alpha = 0.5$ is better than that for $\alpha = 0.3$, if we focus on the results obtained with a significantly limited radius ($r < 0.6$), the opposite occurs for the non-fully-meshed network, where the smaller (although not necessarily optimum) value of α reduces the convergence time. We see that, for both values of α, the impact of faulty links is stronger than that of a limited coverage radius. This is an important difference between the behavior of the bidirectional algorithm (basic gossip) and the unidirectional one (push-sum). More will be said in Sect. 1.5.4 about the comparison between the two approaches.

1.5.3 Broadcast Algorithm

The analysis developed in the previous sections has been repeated for the broadcast algorithm. The results are shown in Fig. 1.5, for some values of $\langle d \rangle$. From the figure, we observe that the behavior of the broadcast algorithm is similar

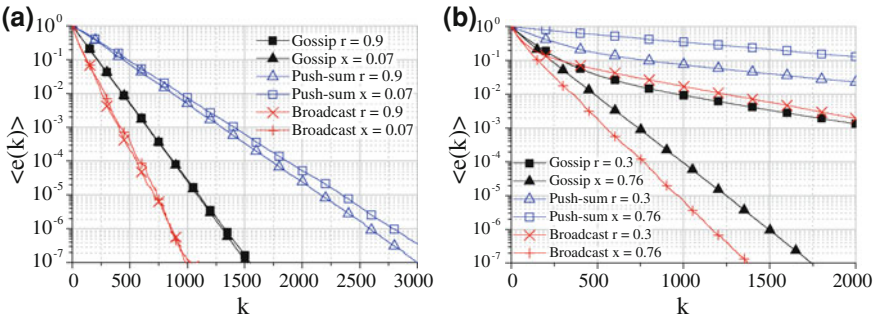


Fig. 1.6 $\langle e(k) \rangle$ for the considered averaging algorithms in the case of **a**$r = 0.9/x = 0.07$ and **b**$r = 0.3/x = 0.76$; optimum shares have been used for push-sum

to that of the basic gossip and, for the same $\langle d \rangle$, convergence of the faulty network is usually faster than that of the non-fully-meshed network.

1.5.4 Performance Comparison

For the sake of comparison, some results for the various algorithms and some pairs r/x are summarized in Fig. 1.6. As expected, the broadcast algorithm exhibits the fastest convergence to the average value, due to its point-to-multipoint nature. However, the basic gossip algorithm is also able to achieve good performance, though being simpler and requiring interaction only between couples of nodes. Its loss in terms of clock ticks, with respect to the broadcast algorithm, is usually limited within 30%. Moreover, there are situations, for very small coverage radius, where the basic gossip outperforms the broadcast algorithm (see the case $r = 0.3$ in Fig. 1.6b).

Performance of the push-sum algorithm is worse. For a network with good connectivity (see Fig. 1.6a), the push-sum algorithm requires approximately a doubled number of clock ticks with respect to the gossip algorithm to reach the same $\langle e(k) \rangle$. This can be justified by considering that, in push-sum, each interaction is unidirectional, while in the basic gossip it is bidirectional. If comparison is made on the number of transmissions, the efficiencies of basic gossip and push-sum become similar.

1.6 Conclusion

We have developed a numerical analysis of the averaging time for basic gossip, push-sum and broadcast algorithms, taking into account the impact of link failures, randomly distributed in the network. In spite of its practical importance, this topic has been rarely debated in previous literature.

Some important conclusions can be drawn from our analysis. First of all, we have demonstrated that the convergence of the algorithms is preserved, on average, regardless of the solution adopted, up to acceptably low values of the mean normalized error.

Then, we have verified that the share factor, when applicable, should be optimized for taking into account the network connectivity. However, starting from the results known for fully-meshed networks, the optimum share factor for the push-sum algorithm is less sensitive to the failure rate than to the limited coverage radius, which is another common reason for reduced connectivity. Moreover, the assumption of a random distribution for the link failures makes applicable an approximate, and inherently simple, analytical approach, based on the potential function, that instead does not provide equally accurate solutions for the case of limited coverage radius.

References

1. Barrenetxea G et al (2007) DemoAbstract: SensorScope, an urban environmental monitoring network. In: 4th European conference on wireless sensor networks (EWSN 2007), Delft, Netherlands, Jan 2007
2. Baker B, Shostak R (1972) Gossips and telephones. Discrete Math 2(3):191–193
3. Berman G (1973) The gossip problem. Discrete Math 4(1):91
4. Boyd S, Ghosh A, Prabhakar B, Shah D (2006) Randomized gossip algorithms. IEEE Trans Inf Theory 52(6):2508–2530
5. Kempe D, Dobra A, Gehrke J (2003) Gossip based computation of aggregate information. In: IEEE conference on foundation of computer science, Cambridge, MA, Oct 2003, pp 482–491
6. Zanaj E, Baldi M, Chiaraluce F (2009) Optimal share factors in the push-sum algorithm for ring and random geometric graph sensor networks. J Commun Softw Syst 5(1):9–18
7. Baldi M, Chiaraluce F, Zanaj E (2009) Fault tolerance in sensor networks: performance comparison of some gossip algorithms. In: 7th international workshop on intelligent solutions in embedded systems (WISES 2009), Ancona, Italy, June 2009, pp 11–20
8. Patwari N, Ash JN, Kyperountas S, Hero AO III, Moses RL, Correal NS (2005) Locating the nodes: cooperative localization in wireless sensor networks. IEEE Signal Process Mag 22(4):54–69
9. Dimakis AG, Sarwate AD, Wainwright MJ (2008) Geographic gossip: efficient averaging for sensor networks. IEEE Trans Signal Process 56(3):1205–1216
10. Zanaj E, Baldi M, Chiaraluce F (2007) Efficiency of the gossip algorithm for wireless sensor networks. In: 2007 international conference on software, telecommunications and computer networks (SoftCOM 2007), Split, Dubrovnik, Croatia, Sept 2007, Paper 7072
11. Zanaj E, Baldi M, Chiaraluce F (2008) Efficiency of unicast and broadcast gossip algorithms for wireless sensor networks. J Commun Softw Syst 4(2):105–112
12. Baldi M, Chiaraluce F, Zanaj E (2008) Comparison of averaging algorithms for wireless sensor networks. In: International conference on information and communication technologies (ICTTA'08), Damascus, Syria, Apr 2008, Paper TEL05_7
13. Merris R (1995) A survey of graph Laplacians. Linear Multilinear Algebra 39(1 and 2):19–31
14. Mohar B (1991) The Laplacian spectrum of graphs. In: Alavi Y, Chartrand G, Oellermann OR, Schwenk AJ (eds) Graph theory, combinatorics, and applications, vol 2. Wiley, New York, pp 871–898
15. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, London

Chapter 2
Using a Prioritized Medium Access Control Protocol for Incrementally Obtaining an Interpolation of Sensor Readings

Björn Andersson, Nuno Pereira, Eduardo Tovar and Ricardo Gomes

2.1 Introduction

A sensor network comprises a set of computer nodes each one equipped with a processor, memory, sensors and a transceiver for communications over a (wired or wireless) channel. The sensor network must obtain an accurate image of physical phenomena and do so with a high sampling rate in both time and space. A large number of computer nodes are needed in order to obtain a high sampling rate in space. But this generates a large number of sensor readings and since these sensor readings are located on different computer nodes, a significant amount of communication may be necessary forcing a reduction in the sampling rate in time. For systems with a very large number of computer nodes, it is therefore crucial to develop techniques that make it possible to obtain a snapshot, an approximate representation of all sensor readings, and achieve this with a time-complexity (as a function of the number of nodes) that is small.

A simple approach for obtaining an approximate representation of sensor readings would be to select a subset of the computer nodes at random and let the sensor readings at those computer nodes be used for obtaining an interpolation. Although this is fast, it has the drawback that some computer nodes with extreme sensor readings may have a significant impact on the interpolation if they would be selected but they may not be selected and this causes (as illustrated in [1]) the interpolation to be a poor representation of the physical phenomenon. And this can cause a sensor network to misperceive its physical environment.

A better approach for obtaining an approximate representation of sensor readings would be to select a subset of the computer nodes, carefully selected to be the ones that represent local extreme points and let the sensor readings at those

B. Andersson (✉) · N. Pereira · E. Tovar · R. Gomes
CISTER/IPP-Hurray Research Unit, Polytechnic Institute of Porto, Porto, Portugal
e-mail: bandersson@dei.isep.ipp.pt

M. Conti et al. (eds.), *Solutions on Embedded Systems*,
Lecture Notes in Electrical Engineering, 81, DOI: 10.1007/978-94-007-0638-5_2,

computer nodes be used for obtaining an interpolation. If one computer node had knowledge of all sensor readings then such a selection would be possible of course but in practice, a computer node only knows its own sensor reading (unless sensor readings are communicated) and therefore it has been non-obvious how to implement such an approach.

Recent work [1, 2] however have shown how to exploit a prioritized medium access control (MAC) protocol for selecting local extreme points and thereby it was shown how to quickly obtain an interpolation of sensor readings where sensor readings were taken by different computer nodes. This work assumes that the MAC protocol has a very large number of priority levels and that all sensor nodes know the priority of the node that was granted the channel. Such MAC protocols are common; the Controller Area Network (CAN) [3] bus is one such example for wired communication (with more than 300 million units sold) and a similar technology, WiDOM [2, 4] is available for wireless communication.

The algorithm [1, 2] which exploited a prioritized MAC protocol had a user-selectable parameter, k, which had the role that the k sensor nodes that contribute the most to the interpolation being a faithful representation of the physical reality are selected and the interpolation is based on those k sensor nodes. k is selected based on the number of local extrema of the signal as explained in [1]. With this approach it was possible to obtain the interpolation with a time-complexity that is $O(k)$, that is, the time-complexity is independent of the number of sensor nodes; yet the result of the interpolation was dependent on all sensor readings. The algorithm was implemented and tested both in wired systems (using CAN [3]) and in wireless systems (using WiDom [2, 4]).

The algorithm for obtaining an interpolation (i) had to run until completion and (ii) it was designed to have no prior knowledge of the physical environment. Unfortunately, these two facts bring two drawbacks:

1. There are situations where the delay from when the physical world changes until the computer system can react to this change is two times the duration required for obtaining the interpolation. (This situation occurs when the environment changed just after the algorithm for obtaining the interpolation had started; when this happens, the algorithm for obtaining the interpolation must finish execution and then take new sensor readings and finally obtain an interpolation of these new sensor readings.)
2. It is necessary that the sampling period of an application that uses the interpolation is $O(k)$ or greater. If the entire physical environment changes everywhere, it may really be necessary to obtain an interpolation from scratch. But one can expect that a change in the physical environment (such as a rapid fire, explosion or deformation) has only local effects initially (during the first milliseconds) and it changes the entire environment later. It would be desirable to use a sampling rate so high that the sampling period is independent of k and independent of the number of nodes, yet the system is able to detect extreme local changes with a duration of two sampling periods and obtain an image of the entire physical environment within k sampling periods.

Therefore, we presented, in a workshop paper [5], a new algorithm for obtaining an interpolation of sensor readings which eliminates the two above mentioned drawbacks. This chapter is an extension of that paper.

The main idea of the algorithm is that when the system starts-up, an interpolation is obtained using the previously known algorithm [1, 2]. This step of the algorithm has the time-complexity $O(k)$, which is larger than we desire but it is done only once. Each computer node now has the k sensor readings that can be used to form an interpolation of all sensor readings. All computer nodes will now periodically take sensor readings with a small period; this period is independent of k and it is independent of the number of computer nodes. All computer nodes take their sensor readings in parallel and then each computer node computes the interpolated value at itself and compares it to its own sensor reading. The computer node whose sensor reading contributes the least to the faithfulness of the interpolation is attempted to be deselected and the computer node whose sensor reading contributes the most to the faithfulness of the interpolation is selected.

We believe this algorithm to be useful for detecting deviations from the expected behavior in the physical world very quickly, for example detecting the deformation of mechanical elements (for example in a car or aircraft) in order to enact appropriate safety actions (such as deciding which airbag to inflate or which fuel pump to be stopped or which valve to be closed).

The remainder of this paper is organized as follows. Section 2.2 gives preliminaries, that is, the main idea of how a prioritized MAC protocol can be used for computations and also the system model we will use. Section 2.3 discusses how to obtain an interpolation; this discussion leads to the new interpolation scheme. Section 2.4 gives conclusions and future work.

2.2 Preliminaries and Motivation

The basic premise for this work is the use of a prioritized MAC protocol. This implies that the MAC protocol assures that out of all nodes contending for the medium at a given moment, the one(s) with the highest priority gain access to it. This is inspired by Dominance/Binary-Countdown protocols [6]. In such protocols, messages are assigned unique priorities, and before nodes try to transmit they perform a contention resolution phase named arbitration such that the node requesting to transmit the highest-priority message succeeds.

During the arbitration (depicted in Fig. 2.1), each node sends the message priority bit-by-bit, starting with the most significant one, while simultaneously monitoring the medium. The medium must be devised in such a way that nodes will only detect a “1” value if no other node is transmitting a “0”. Otherwise, every node detects a “0” value regardless of what the node itself is sending. For this reason, a “0” is said to be a dominant bit, while a “1” is said to be a recessive bit. Therefore, low numbers in the priority field of a message represent high priorities. If a node contends with a recessive bit but hears a dominant bit, then it

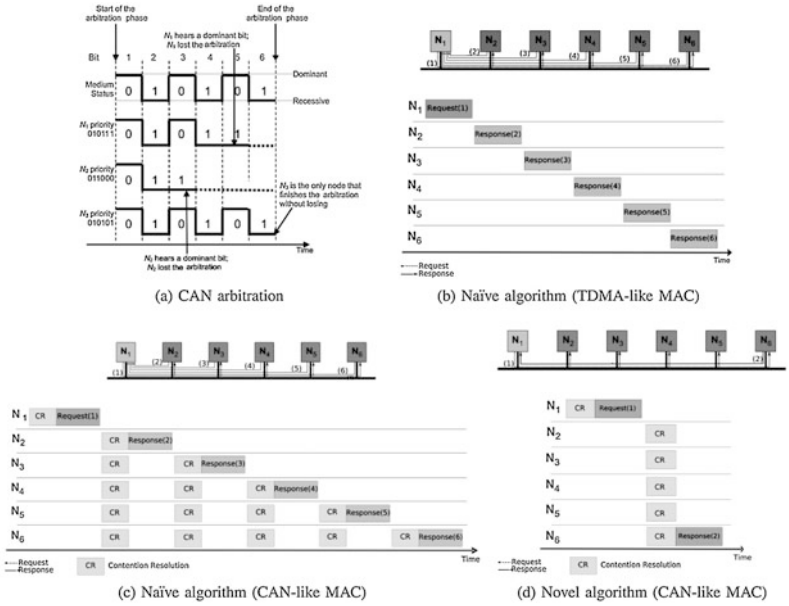


Fig. 2.1 Dominance/binary-countdown arbitration motivating examples. **a** Example of bitwise arbitration; **b** example application where N_1 needs to know the minimum (MIN) temperature reading among its neighbors (N_2–N_6); **c** possible solution for the example application using a CAN-like MAC, using fixed priorities for the messages; **d** possible solution for the example application exploiting the properties of a CAN-like MAC, where priorities are assigned at runtime according to the sensed values

will refrain from transmitting any further bits, and will proceed only monitoring the medium. Finally, exactly one node reaches the end of the arbitration phase, and this node (the winning node) proceeds with transmitting the data part of the message. As a result of the contention for the medium, all participating nodes will have knowledge of the winner's priority.

The CAN bus [3] is an example of a technology that offers such a MAC behavior. It is used in a wide range of applications, ranging from vehicles to factory automation (the reader is referred to [7] for more examples of application fields and figures about the use of CAN technologies). Its wide application fostered the development of robust error detection and fault confinement mechanisms, while at the same time maintaining its cost effectiveness. An interesting feature of CAN is that the maximum length of a bus can be traded-off for lower data rates. It is possible to have a CAN bus with a bit rate of 1 Mbit/s for a maximum bus length of 30 m, or a bus 1,000 m long (with no repeaters) using a bit rate of 50 Kbit/s. While the typical number of nodes in a CAN bus is usually smaller than 100, with careful design (selecting appropriate bus-line cross section, drop line length and quality of couplers, wires and transceivers) of the network it is possible to go well above this value. For example, CAN networks with more than a

thousand nodes have been deployed and they operate in a single broadcast domain (such networks have been built; see for example [8]).

The focus of this paper is on exploiting a prioritized MAC protocol for efficiently obtaining an interpolation function which approximates the sensor readings in a geographical area. A key idea in the design of such an algorithm is the use of a prioritized MAC protocol for performing computations—this is explained next.

2.2.1 The Main Idea

The problem of obtaining aggregated quantities in a single broadcast domain can be solved with a naïve algorithm: every node broadcasts its sensor reading sequentially. Hence, all nodes know all sensor readings and then they can obtain the aggregated quantity. This has the drawback that in a broadcast domain with m nodes, at least m broadcasts are required to be performed. Considering a network designed for $m \geq 100$, the naïve approach can be inefficient; it causes a large delay.

Let us consider the simple application scenario as depicted in Fig. 2.1b, where a node (node N_1) needs to know the minimum (MIN) temperature reading among its neighbors. Let us assume that no other node attempts to access the medium before this node. A naïve approach would imply that N_1 broadcasts a request to all its neighbors and then N_1 would wait for the corresponding replies from all of them. As a simplification, assume that nodes orderly access the medium in a time division multiple access (TDMA) fashion, and that the initiator node knows the number of neighbor nodes. Then, N_1 can derive a waiting timeout for replies based on this knowledge. Clearly, with this approach, the execution time depends on the number of neighbor nodes (m). Figure 2.1c depicts another naïve approach, but using a CAN-like MAC protocol.

Assume in that case that the priorities the nodes use to access the medium are ordered according to the nodes' ID, and are statically defined prior to runtime. Note that in order to send a message, nodes have to perform arbitration before accessing the medium. When a node wins it sends its response and stops trying to access the medium. It is clear that using a naïve approach with CAN brings no timing advantages as compared to the other naïve solution (Fig. 2.1b).

Consider now that instead of using their priorities to access the medium, nodes use the value of its sensor reading as priority. Assume that the range of the analog to digital converters (ADC) on the nodes is known, and that the MAC protocol can, at least, represent as many priority levels. This assumption typically holds since ADC tend to have a data width of 8, 10, 12 or 16-bit while the CAN bus offers up to 29 priority bits. This alternative would allow an approach as depicted in Fig. 2.1d. With such an approach, to obtain the minimum temperature among its neighbors, node N_1 needs to perform a broadcast request that will trigger all its neighbors to contend for the medium using the prioritized MAC protocol. If neighbors access the medium using the value of their temperature reading as the

priority, the priority winning the contention for the medium will be the minimum temperature reading. With this scheme, more than one node can win the contention for the medium. But, considering that at the end of the arbitration the priority of the winner is known to all nodes, no more information needs to be transmitted by the winning node. In this scenario, the time to obtain the minimum temperature reading only depends on the time to perform the contention for the medium, not on m. If, for example, one wishes that the winning node transmits information (such as its location) in the data packet, then one can code the priority of the nodes by adding a unique number (for example, the node ID) in the least significant bits, such that priorities will be unique.

A similar approach can be used to obtain the maximum (MAX) temperature reading. In that case, instead of directly coding the priority with the temperature reading, nodes will use the bitwise negation of the temperature reading as the priority. Upon completion of the medium access contention, given the winning priority, nodes perform bitwise negation again to know the maximum temperature value.

MIN and MAX are just two simple and pretty much obvious examples of how aggregate quantities can be obtained with a minimum message complexity (and therefore time complexity) if message priorities are dynamically assigned at runtime upon the values of the sensed quantity. In Sect. 2.3 we will show how this technique of using a prioritized MAC protocol for computations can be used for obtaining an interpolation of sensor readings.

2.2.2 System Model

The network consists of m nodes that take sensor readings where a node is given a unique identifier in the range $1\ldots m$. MAXNNODES denotes an upper bound on m and we assume that MAXNNODES is known by the designer of the system before run-time. Nodes do not have a shared memory and all data variables are local to each node.

Each node has a transceiver and is able to transmit to or receive from a single channel. Every node has an implementation of a prioritized MAC protocol with the characteristics as described earlier. Nodes perform requests to transmit, and each transmission request has an associated priority. Priorities are integers in the range [0, MAXP], where lower numbers correspond to higher priorities. Let *NPRIOBITS* denote the number of priority bits. This parameter has the same value for all nodes. Since *NPRIOBITS* is used to denote the number of bits used to represent the priority, the priority is a number in the range of $0–2^{\text{NPRIOBITS}} - 1$. Clearly, $\text{MAXP} = 2^{\text{NPRIOBITS}} - 1$.

A node can request to transmit an *empty packet*; 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. All nodes share a single reliable broadcast domain.

A program on a node can access the communication system via the following interface. The `send` system call takes two parameters, one describing the priority of the packet and another one describing the data to be transmitted. If a node calling `send` wins the contention, then it transmits its packet and the program making the call unblocks. If a node calling `send` loses the contention, then it waits until the contention resolution phase has finished and the winner has transmitted its packet (assuming that the winner did not send an empty packet). Then, the node contends for the channel again. The system call `send` blocks until it has won the contention and transmitted a packet. The function `send_empty` takes only one parameter, which is a priority and causes the node only to perform the contention but not to send any data after the contention. In addition, when the contention 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 system call `send_and_rcv` takes two parameters, priority and data to be transmitted. The contention is performed with the given priority and then the data is transmitted if the node wins. Regardless of whether the node wins or loses, the system call returns the priority and data transmitted by the winner and then unblocks the application.

A node N_i takes a sensor reading s_i. It is an integer in the range [0, MAXS] and it is assumed that MAXS $\leq$ MAXP.

2.3 Interpolation of Sensor Data with Location

Having seen the main idea of how to take advantage of a prioritized MAC protocol, we are now in position to present our approach for obtaining an interpolation of sensor readings. We will do so formally with pseudo-code; this pseudo-code returns an upper bound on the error of the interpolation. We will first (in Sect. 2.3.1) present the main idea of the previously known interpolation scheme.

This will lead us (in Sect. 2.3.2) to the new incremental interpolation scheme. This new incremental interpolation scheme needs to, as an intermediate result, evaluate the interpolation at certain geographical points. Therefore, we will (in Sect. 2.3.3) modify this algorithm to perform calculations at those specific points with additional speed and this results in an improvement of the new incremental interpolation scheme.

2.3.1 Previously Known Algorithm

We assume that nodes take sensor readings, but we will also assume that a node N_i knows its location given by two coordinates (x_i, y_i). With this knowledge, it is possible to obtain an interpolation of sensor data over space. This offers a compact representation of the sensor data and it can be used to compute virtually anything.

We let $f(x, y)$ denote the function that interpolates the sensor data. Also let e_j denote the magnitude of the error at node N_j; that is:

$$e_j = \left| s_j - f\left(x_j, y_j\right) \right| \tag{2.1}$$

and let e denote the global error; that is:

$$e = \max_{j=1\ldots m} e_j \tag{2.2}$$

The goal is to find $f(x, y)$ that minimizes e subject to the following constraints: (i) the time required for computing f at a specific point should be low; and (ii) the time required to obtain the function $f(x, y)$ from sensor readings should be low. The latter is motivated by the fact that it is interesting to track physical quantities that change quickly; it may be necessary to update the interpolation periodically in order to track, for example, how the concentration of hazardous gases move. For this reason, we will use weighted-average interpolation (WAI) [9–11]. WAI is defined as follows:

$$f(x,y) = \begin{cases} 0 & \text{if } S = \emptyset \\ s_j & \text{if } \exists N_j \in S : x_j = x \ \wedge \ y_j = y \\ \frac{\sum_{j \in S} s_j * w_j(x,y)}{\sum_{j \in S} w_j(x,y)} & \text{otherwise} \end{cases} \tag{2.3}$$

where S is a set of nodes used for interpolation. The weights $w_j(x, y)$ are given by:

$$w_j(x,y) = \frac{1}{\left(x_j - x\right)^2 + \left(y_j - y\right)^2} \tag{2.4}$$

Algorithm 1 Finding a subset of nodes to be used inWAI

Require: All nodes start Algorithm 1 simultaneously.
Require: k denotes the desired number of interpolation points.
Require: A node N_i knows x_i,y_i and s_i.
Require: The code below is executed by every node. A node can read the variable `i` and obtain its node index.
Require: (MAXS+1) × (MAXNNODES+1) + MAXNNODES ≤ MAXP.

```
1:  function find_nodes() return a set of packets
2:      S ← ∅
3:      for q ← 1 to k do
4:          Calculate f(x_i,y_i) in Equation 3 and assign
              it to the variable "myinterpolatedvalue"
5:          error ← abs( s_i - to_integer(myinterpolatedvalue) )
6:          temp_prio ← error × (MAXNNODES + 1) + i
7:          prio ← (MAXP+1) - temp_prio
8:          snd_pack ←< s_i,x_i,y_i >
9:          <winning_prio, rcv_pack> ← send_and_rcv( prio, snd_pack)
10:         S ← S ⋃ { rcv_pack }
11:     end for
12:     return S
13: end function
```

Intuitively, Eqs. 2.3 and 2.4 state that the interpolated value is a weighted average of all data points in S and the weight is the inverse of the square of the distance. There are many possible choices on how the weight should be computed