ABSTRACT

YOON, SUYOUNG. Power Management in Wireless Sensor Networks. (Under the direction of Assistant Professor Mihail L. Sichitiu.)

One of the unique characteristics of wireless sensor networks (WSNs) is that sensor nodes have very constrained resources. Typical sensor nodes have lower computing power, communication bandwidth, and smaller memory than other wireless devices, and operate on limited capacity batteries. Hence power efficiency is very important in WSNs because power failure of some sensor nodes may lead to total network failure. In many cases the WSNs have to operate in harsh environments without human intervention for expended period time. Thus, much research on reducing or minimizing the power consumption, and thereby increasing the network lifetime, has been performed at each layer of the network layers. In this dissertation we approach three important issues related power management in WSNs: routing, time synchronization, and medium access control (MAC). We first discuss the effect of selecting routing protocols on the lifetime of the WSNs. The maximum and minimum bounds of the lifetime with respect to the routing protocols are derived. The routing protocols corresponding to the bounds are also presented. The simulation results show that the choice of the routing protocol has very little impact on the lifetime of the network and that simple routing protocols such as shortest path routing perform very close to the the maximum bound of the lifetime of the network. Next, we propose a simple and accurate time synchronization protocol that can be used a a fundamental component of other synchronization-based protocols in WSNs. Analytical bounds on the synchronization errors of proposed protocol are discussed. The implementation results on Mica2 and Telos motes show that proposed time synchronization protocol outperforms existing ones in terms of the precision and required resources. Finally, we model the power consumption of WSN MAC protocols. We derive analytically the power consumption of well known MAC protocols for WSNs, and analyze and compare their performance. We validate the models by measuring the power consumption on Mica 2 motes and comparing those measured power consumption with the analytical results.

Power Management in Wireless Sensor Networks

by

Suyoung Yoon

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh, NC

2007

Approved By:

[illegible]

Dr. Rudra Dutta

[illegible]

Dr. Do Young Eun

[illegible]

Dr. Mihail L. Sichitiu
Chair of Advisory Committee

[illegible]

Dr. Arne A. Nilsson

For my parents, my parents-in-law, my dear wife Mijung, and my children Dahyoung, Haejung, and Junghwan.

Biography

Suyoung Yoon was born in South Korea in 1964. He received the Bachelor of Science degree in Department of Computer Science and Statistics from Seoul National University in 1987 and the Master of Science degree in Computer Science from the Korea Advanced Institute of Science and Technology in 1991. In 2002, he started his work as a doctoral student under the guidance of Dr. Sichitiu. Suyoung Yoon's research interest includes wireless networking including ad-hoc and wireless sensor networks.

Acknowledgements

This dissertation would not have been possible without the support of many people. First of all, I would like to thank my advisor Dr. Mihail L. Sichitiu for his thoughtful directions and affectionate encouragements. I would like to acknowledge my advisory committee members Dr. Arne A. Nilsson, Dr. Rudra Dutta, and Dr. Do Young Eun for their enlightening comments and constructive suggestions on my work. I would like to acknowledge Korea Telecom who supported my graduate study. And finally, I would like to thank my wife, children, parents, and numerous friends who endured this long process with me, always offering support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first present the characteristics and applications of wireless sensor networks. We then explain why the power management is important in these networks. Finally, we present our contributions in this area.

1.1 Wireless Sensor Network

With the technology development of micro electronic mechanical systems (MEMS), wireless communication, and embedded microprocessors, wireless sensor networks (WSNs) have been deployed in environment monitoring, military target detection, and commercial areas [2–4]. Wireless sensor networks usually consist of thousands of sensor nodes that are capable of sensing, processing, and communicating. In typical sensor network applications, each sensor node obtains data from the physical environment and transmits in wireless multihop fashion the data to the base station where the data will be delivered to users through infrastructures such as Internet.

Although more than 90% of sensors are still wired, *wired* sensor networks have been replaced by *wireless* sensor networks due to the cost and delay of deployment [5]. The dominant factor of constructing wired sensor networks is the wiring cost [6]. In addition, wired networks require considerable time to implement. In case of wireless sensor network, the deployment is rather simple, in many cases, just dropping off sensor nodes from the airplane into the target area instead of wiring from the target area to monitoring station.

Wireless sensor networks can be used in variety applications that allow existing information systems to obtain and process data from physical world. In environment monitoring, one of the earliest applications of wireless sensor network, wireless sensors are used to monitor animals and plants in wildlife habitat such as the Great Duck Island experiment [7]. Other related applications include monitoring of water pollution, wildfires, and earthquakes. In the military battlefield, where there is no infrastructure and it is very hard to access and deploy the sensor networks, the wireless sensor networks can be rapidly deployed to detect the enemy target and to track their movements in realtime. Commercial applications include the monitoring and tracking of assets, monitoring of the conditions of industrial equipment, automated meter reading, and warehouse management using RFID technologies. Wireless sensor networks can also be used in structural health monitoring of large structures such as airplanes, buildings, and bridges. With the dedicated short range communication (DSRC) standard, car-to-car networking with sensors can be available to provide emergency warning, traffic monitoring, and driver safety assistance. One of the most important area of wireless sensor network applications is health monitoring of patients in a hospital.

1.2 Power Management in Sensor Networks

Wireless sensor networks are similar to wireless ad hoc network in terms of networking topology and multi-hop routing. But, sensor networks are also different from other wireless ad hoc networks in that they consist of hundreds or thousands of autonomous nodes and the direction of most sensor traffic is from the sensor nodes to the base station. Another unique characteristics of wireless sensor networks is that the sensor nodes in WSNs are equipped with batteries of limited capacity and are expected to operate without human interaction for a long time. Therefore, the power management of each sensor node plays a very important role in increasing the lifetime of the sensor networks [8]. For example, when a Telos mote equipped with 2250mAh batteries idles without any power saving algorithm, the mote can operate for up to 52 days. With a 1% duty cycle, the lifetime of the node can be extended to 4076 days [9].

Recently, power management has been extensively studied in each area of sensor networks such as the transmission power expenditure, medium access control (MAC) layer techniques, power aware routing algorithms, network architecture, and sensor node deployment. The transmission power control methods try to reduce the power level of the transmitter while maintaining the network connectivity. MAC layer techniques aim to conserve battery power by turning the receiver off whenever it is not needed. Power aware routing algorithms seek to choose routes in such a way to maximize the lifetime of the network by minimizing radio power consumption. Scheduling sensor nodes where each node goes to power saving mode as long as possible can also save power consumption significantly by reducing idle-listening time. The choice of the network architecture design (homogeneous or heterogeneous; flat or hierarchical; the number of base stations; static or mobile access point) and node deployment policy (node connectivity) can also affect the power management schemes.

In this dissertation, we explore the power management in WSNs in the following areas: routing, time synchronization, and MAC protocols. The most energy consuming part among sensor node operations is the radio transmission and reception. Therefore, reducing the number of message transmissions and receptions for the functions that require message exchanges is imperative to minimize the power consumption a sensor node. Routing is one of the those functions since the routing algorithms exchange information with neighboring nodes to find the best route.

The nodes in sensor network are a distributed system. That is, each sensor node has its own processing unit, memory, transceiver, and clock. In such a system, time synchronization between nodes in the network is necessary in order for the sensor nodes to perform cooperative jobs such as target tracking and reporting of time-sensitive data. All time synchronization protocols are based on message exchanges between sensor nodes(either one or two ways). Thus, time synchronization is also an essential function for the power management of wireless sensor networks.

Without power efficient MAC protocols, sensor nodes would listen to the network at all time in order to receive messages from other nodes. This idle listening consumes a large percentage of the energy of sensor nodes (Tables 4.1 and 4.2). Therefore, in order to minimize the energy consumption, sensor nodes should be in sleep mode (or lower power mode) as long as possible and to be awake when only necessary.

Time synchronization, routing, and power efficient MAC protocols are closely related to each other as shown in Fig. 1.1. Many power efficient MAC protocols require accurate time synchronization. If the sensor nodes are appropriately scheduled, the MAC contention can be reduced, which leads to a lower number of message exchanges for routing and time synchronization. Global time synchronization also needs power efficient routes from sensor nodes to the base station. The number of messages that a sensor node hears depends on the routing protocol used, therefore, affecting the performance of power

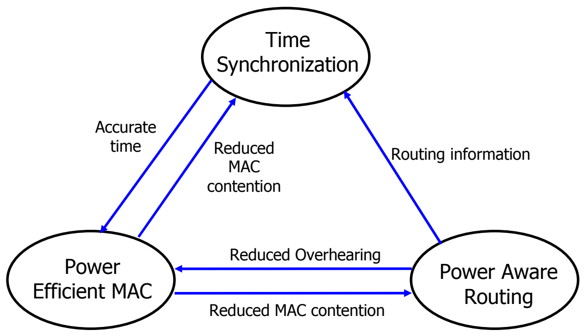


Figure 1.1: Interactions between the power management modules in this thesis.

efficient MAC protocols.

1.3 Contributions

Power management plays a very important role in wireless sensor networks because most sensor network applications operate on batteries and the deployment environments can be remote or hostile. Power management in sensor networks is performed by considering all aspects of these networks from the physical to the application layers, and even cross-layer. Our contributions in this field focus on the areas of power aware routing, time synchronization, and analysis of power consumption of WSN MAC protocols.

Power Aware Routing

In this area, we discuss the effect of power efficient routing algorithms on the lifetime of multihop wireless sensor networks. We assume that all data and control traffic in the network is flowing between the sensor nodes and the base station. This assumption results in a considerably simpler problem and solution than for the more general MANETs. We analytically derive the maximum and minimum bounds of the lifetime of the sensor network under specific routing algorithms. We also present the routing algorithms that result in the maximum and minimum lifetime. We observe that the choice of the routing algorithm has almost no consequence to the lifetime of the network. The simulation results show that the lifetime of proposed routing algorithms fall in the derived lifetime range.

Time Synchronization

In this area, we present two simple and accurate pairwise time synchronization protocols for wireless sensor networks. The two algorithms have the following features:

- *Deterministic bounds on the precision:* Most other algorithms provide best estimates for the offset and drift of the clocks, and possibly probabilistic bounds on these estimates. Our approach delivers tight, *deterministic* bounds on these estimates, such that absolute information can be deduced about ordering and simultaneous events.

- *Accuracy:* The experimental results show that pairwise time synchronization errors of proposed algorithms are less than 1.5 μs.

- *Low computation and storage complexity:* Wireless sensor nodes typically feature low computational power microprocessors with small amounts of RAM. Both algorithms have low computational and storage complexity.

- *Low sensitivity to communication errors:* Wireless communications are notoriously error prone, and thus one cannot rely on correct receipt of all messages. The presented approach works correctly even if a large percentage of the messages is lost.

We analytically derive deterministic bounds on offset and clock drift. The analytical results show that the bounds on offset and clock drift only depend on the round trip times of messages between two nodes.

Based on pairwise synchronization, we propose a method for synchronizing the entire network. Performing global synchronization does not require extra effort on sensor nodes, but is done implicitly by the pairwise synchronization. Proposed algorithms guarantee that the global synchronization errors only depends on the number of hops between two sensor nodes.

We implemented proposed algorithms on Mica2 and Telos motes, measured the time synchronization errors, and compared our results with other typical synchronization protocols.

- To minimize delay uncertainties, we adopted MAC layer time-stamping.

- To minimize the measurement errors, we measured the time synchronization errors by wiring the target sensor nodes into the measuring node and having the nodes report local times-tamps at the same time by wired interrupts.
- We showed how the synchronization intervals affect the performance of time synchronization protocols
- We showed that proposed algorithms are robust to unreliable wireless sensor networks.

Analyzing Power Consumption of WSN MAC protocols

We constructed a common power consumption model to be used heterogeneous MAC protocols. The power consumption model can represent the power consumption behavior of each MAC protocol at the very detail level. We determined analytically the power consumption of the following protocols:

- 802.11 basic mode
- 802.11 power-saving mode
- S-MAC
- T-MAC
- B-MAC
- X-MAC
- SCP-MAC

We performed the simulation using the analytical results and compare MAC protocols. With the simulation results, we conclude that B-MAC and X-MAC are appropriate for the applications that require short delay while S-MAC, T-MAC, or SCP-MAC should be used for monitoring applications that tolerate long delays. We validated the analytical results by measuring energy consumptions of the MAC protocols on a testbed using Mica 2 motes and compare the results. The experimental results show that the proposed power consumption models closely match the experimental results.

Chapter 2

Power Aware Routing

In this chapter, we discuss the effect of power efficient routing algorithms on the lifetime of wireless sensor networks. We calculate analytically lifetime bounds of the network under specific routing algorithms. We perform the simulations to verify the analytically derived lifetime bounds.

2.1 Introduction

The power management problem for wireless sensor networks has been studied intensively. Various approaches for reducing the energy expenditure have been presented in literature. In this work, we focus on routing strategies that maximize the lifetime of the wireless sensor networks

Several strategies are commonly employed for power aware routing in WSNs [10]:

- Minimizing the energy consumed for each message. This metric might unnecessarily overload some nodes causing them to die prematurely.
- Minimizing the variance in the power level of each node. This is based on the premise that it is useless to have battery power remaining at some nodes while others exhaust their battery, since all nodes are deemed to be equally important.
- Minimizing the cost/packet ratio In this approach, different costs can be assigned to different links, for example, incorporating the discharge curve of the battery, and thus postponing the moment of network partition.

- Minimizing the maximum energy drain of any node. The basis of this approach is that the network utility is first impacted when the first node exhausts its battery, and thus it is necessary to minimize the battery consumption at this node.

The above approaches focus on different metrics of energy efficiency. A common characteristic of these metrics is that they can lead to a disconnected network with a high residual power: once the critical nodes of the network have depleted their batteries, the network is essentially dead. Indeed we show that under our assumptions this is inevitable. For a practical sensing application, the network can be considered to have stopped working when it fails to deliver the sensed readings from a bulk of the sensors, and the important metric is the time when this occurs. In what follows, we will therefore use the network lifetime as our main performance measure, which we define in the next section.

While all the above approaches provide benefits in different classes of MANETs, the special case of WSNs merit closer evaluation since they are practically an important class of MANETs. Generally, the problem of computing the optimal lifetime of the MANETs is known to be hard due to node mobility. As a special case of MANETs, the WSNs are (in most sensing applications) stationary and have a base station sink, where all data traffic ends. In this work, we will derive bounds on the lifetime of WSNs. We show that the two characteristics mentioned above play a crucial role in these considerations. Somewhat surprisingly, we are able to show that network behavior under these conditions is quite specific, the maximum benefit obtainable from the batteries is very predictable, and achievable by rather simple routing strategies.

2.2 Related Work

Sensor Protocols for Information via Negotiation (SPIN) [11] makes good the deficiencies of classic flooding by negotiation and resource adaptation. Using SPIN routing algorithm, sensor nodes can conserve energy by sending the metadata that describes the sensor data instead of sending all the data. SPIN can reduce the power consumption of individual node, but it may decrease the lifetime of the whole network due to extra messages.

Low-Energy Adaptive Clustering Hierarchy (LEACH) is a clustering-based protocol that minimizes energy dissipation in sensor networks [12]. LEACH randomly selects sensor nodes as cluster-heads, so the high energy dissipation in communicating with the base station is spread to all sensor nodes in the sensor network. LEACH can suffer from the clustering overhead, which may result in extra power depletion.

The algorithm $max - min\ zP_{min}$ combines the benefits of selecting the path with the minimum power consumption and the path that maximizes the minimal residual power in the nodes of the network [13]. However, this algorithm has the disadvantage of being centralized and requiring knowledge of the power level of each node in the system. Its distributed version also has clock synchronization problem.

Mhatre et al. [14] obtained the minimum number of sensor nodes, cluster heads, and battery energy to ensure at least T unit of lifetime. They assume two types of sensor nodes: node 0 is sensor node and node 1 is cluster head. The main result is that the number of cluster heads should be the order of square root of the number of sensor nodes. They don't give exact formula for the maximum lifetime of the network. And, it is difficult for the assumption that cluster heads directly communicate with the base station to be applied to general applications. They observed that the nodes close to the cluster heads have high energy burden due to relaying of packets. But, one of their important observations, the sharp cutoff effect, to maximize the lifetime does not hold at all time.

Bhardwaj et al. [15] computed the upper bound of active lifetime in terms of the routing algorithms. But they did not consider that the first tier nodes determine the lifetime of the whole network. They measured the lifetime of the network as the time of first loss of the coverage. That is, they did not care the connectivity. [16] elaborated their work in [15] by taking into account the data aggregation and random topology.

Zhand and Hou [17] presented necessary and sufficient condition for k-coverage. They also derived the upper bound of the lifetime of all algorithms that select working nodes can achieve. They included data transmission, reception, idle listening time, sensing, and data processing as the power consumption of a node. They measured the lifetime of the network as -portion of coverage based on the assumption that if transmission range is as twice as sensing range, coverage implies the connectivity. Their work is different from ours because they derived the maximum lifetime of the network in terms of nodes selection. And they did not consider the properties of the first nodes.

Pan et al. [18] observed the property that the first tier nodes are important for the lifetime of the whole network. They provided approaches to maximize the topological lifetime of the network in terms of the base station placement for the two-tiered sensor networks where sensor nodes are deployed in clusters.

Alonso et al. [19] found explicit bounds on the minimal and maximal energy for routing algorithms and used the bounds to derive the lifetime of the network. But the maximum lifetime of the network derived by [19] is very similar to the minimum lifetime of the network derived by our approach.

The previous work on power aware routing algorithms focused on how to find the correct route efficiently or how to enhance the lifetime of the network. In this work, we derive the lower and upper bounds on the lifetime of the network in terms of the routing algorithms. That is, we give the answer for the question that what is the maximum lifetime of the WSNs that routing algorithms can achieve. We will also present the routing algorithms that show the maximum and minimum lifetime.

2.3 Definitions, Notations, and Assumptions

In this section, we define the lifetime of the network (the metric for determining the optimality of routing algorithms). We also present the assumptions and notations used in the following sections.

2.3.1 Definitions

The lifetime of the network L is the lifetime of the set of all its initial nodes.

2.3.2 Assumptions

We believe that the following assumptions apply to a large class of sensor network implementations and applications.

We assume that:

- A-1 all network nodes are stationary,
- A-2 all sensed data is sent to the base station (i.e. no filtering or other in-network processing is performed),
- A-3 all network nodes generate packets periodically with a common constant period,
- A-4 the transmission range and transmission power is constant for all transmissions from all nodes,
- A-5 all nodes have the same initial battery level,

A-6 there are more nodes in tier $i + 1$ than in tier i except for the last tier of nodes. (In terms of the notation we introduce in section 2.3.3 $N_{i+1} \geq N_i$, $1 \leq i \leq H - 2$.) If this assumption holds at deployment time, it will continue to hold for the lifetime of the network since the inner tiers carry more traffic that the outer tiers and, thus, more nodes die in the inner tiers than in the outer tiers.

A-7 the traffic forwarding load from nodes which are more than i hops from the base station is equally shared by all nodes which are i hops from the base station.

Of the above, the first two are the crucial ones we mentioned before. The next two assumptions merely represent a realistic case, and also simplify what follows, but do not reduce the scope of our results. A-5 also represents a quite realistic condition; the removal or relaxation of this assumption is not considered within the scope of this paper. A-6 is satisfied for most reasonable distribution of sensor nodes, for example approximately uniform distribution over a large area. The assumption of a uniform distribution is stronger than A-6 and is not needed for this paper. The main purpose of the minimal assumption A-6 is to eliminate pathological cases where the WSN becomes prematurely disconnected due to a bottleneck in the topology. Finally, A-7 is made for explanatory purposes and later we examine the consequences of removing this assumption.

2.3.3 Model and Notations

We model the power consumption P of a wireless node as:

$$P = P_a T + P_b, \tag{2.1}$$

where T is the number of flows transmitted by the node (comprising its own sensed data and data forwarded on behalf of other nodes), P_a is the power consumption used to forward the data in each flow, and P_b is the power consumption independent of the forwarded traffic. A sensor node that consumes the same power independent of the number of flows forwarded is likely a wasteful node. A power efficient sensor network has a very small P_b (mainly due to routing overhead, synchronization and other middleware services), practically all its power being expended in useful sensing and forwarding of information. In WSNs the traffic from the base station to the sensor nodes (queries, control information, etc.) is usually broadcast and hence contributes to P_b rather than to P_a. The choice of the MAC layer clearly influences the power efficiency of the network [20–29] – power efficient MAC layers result in reduced P_a and P_b. Beyond the particular values of P_a and P_b, the choice of the MAC layer is not relevant for the reminder of this paper.

Regardless of its value, for our purposes, P_b does not play a role in the contribution of routing to the network lifetime L: simply by offsetting the initial battery level by a constant quantity ($P_b L$) we can compute the same lifetime L by using a simplified model for the power consumption of a node:

$$P = P_a T. \tag{2.2}$$

We will use the following notation:

- β is the energy spent to transmit one packet once.
- p is the number of packets generated by each node in every second (thus, the energy spent every second by each node to generate or forward one flow is $\beta\, p$).
- b is the initial battery level of every node (as discussed only the battery expended for forwarding and sending its own data is relevant for the network lifetime).
- H is the maximum number of hops between the base station and any of the wireless nodes in the WSN.
- $\mathcal{N}$ is the set of all sensor nodes.
- $\mathcal{N}_i$ is the set of sensor nodes that are at a minimum of i hops away from the base station. We also call this set of nodes the i^{th} *tier* of nodes. For example, the first tier of nodes consists of the nodes that can directly reach the base station. With our assumptions, initially all nodes of in $\mathcal{N}_i$ will also be exactly i hops from the base station; however, as nodes in $\mathcal{N}_{i-1}$ die, some nodes in $\mathcal{N}_i$ may require more than i hops to reach the base station, and become part of the set $\mathcal{N}_{i+1}$. *However, note that nodes in $\mathcal{N}_1$ never migrate to other tiers.*
- N is the total number of sensor nodes; $N = |\mathcal{N}|$.
- N_i is the number of nodes in tier i; $N_i = |\mathcal{N}_i|$.
- $T_i^r(n)$ is the number of packets transmitted by node $n \in \mathcal{N}_i$ using the routing algorithm r.
- L^r is the lifetime of the network when using routing algorithm r

L_i^r is the lifetime of the nodes of $\mathcal{N}_i$ when using routing algorithm r.

$\mathcal{R}$ is the set of all *minimum hop* routing algorithms able to find a path between each sensor node and the base station if such a path exists. Usually, each node in the set $\mathcal{N}_i$ has multiple shortest hop neighbors in the set $\mathcal{N}_{i-1}$; The choice of one of these neighbors (e.g. randomly, or based on the residual power) differentiates among the algorithms in $\mathcal{R}$.

2.4 The effect of the routing algorithms on the lifetime of the WSN

When the traffic pattern in a network is such that all nodes transmit to an *egress* node such as a base station, the few nodes that can reach the base station directly will be responsible for the highest amount of traffic forwarding. We have examined this phenomenon in detail in [30], below we present the result that is relevant to us in the current context. Then we use this result to obtain lower and upper bounds for the lifetime of the network as a function of the routing protocol.

***Lemma* 1** *For any routing algorithm $r \in \mathcal{R}$, the lifetime of the nodes in $\mathcal{N}_1$ is equal to the lifetime of the nodes in other tiers ($\mathcal{N}_i,\ i > 1$). In other words, $L_1^r = L_i^r$ for all i and r such that $1 < i \leq H$ and $r \in \mathcal{R}$.*

Proof For all $r \in \mathcal{R}$ and $i > 1$,

$$\sum_{n \in \mathcal{N}_1} T_1^r(n) > \sum_{n \in \mathcal{N}_i} T_i^r(n) \tag{2.3}$$

because there are no loops in the paths through the nodes in tier i and, hence, the traffic in the first tier of nodes *includes* the traffic from any other tier (and adds its own traffic). Using either (2.1) or (2.2) this implies that the power consumption of nodes in the first tier is higher than that of the nodes in any other tier. Since all nodes have the same initial battery size (assumption A-5) and there are more nodes in tier i than in tier 1 (assumption A-6), the nodes in the first tier will deplete their battery strictly sooner than the nodes in any other tier. However, as soon as the first tier of nodes depletes its batteries, the entire network becomes disconnected (and by the definition of the lifetime in Section 2.3.1 all tiers reach their lifetimes).

***Theorem* 2** *For a WSN satisfying all assumptions in Section 2.3.2 and using a routing algorithm $r \in \mathcal{R}$ the lifetime of the network is*

$$L_{\min} = \frac{N_1 b}{N \beta p}. \tag{2.4}$$

Proof According to Lemma 1, the lifetime of the network is determined by the lifetime of the first tier of nodes. Considering assumption A-7, every node in the first tier will expend the battery at the same (constant - assumption A-3) rate. Further, each flow originating from outside of tier 1 is forwarded by exactly one first tier node: since tier 1 nodes are the only nodes that can transmit directly to the base station. Each first tier node also originates exactly one flow of its own. Finally, considering that all nodes have the same initial battery (assumption A-5), all nodes in the first tier will deplete their battery *at the same time*. The moment when the first (and last) battery is depleted coincides with the time of the death of the network. Thus, the battery expended on the first tier of nodes is used to forward data for all nodes in the network for the duration of the networks' lifetime:

$$N_1 b = L_{\min} N \beta p. \tag{2.5}$$

The above is valid if all nodes are alive until the lifetime expires, as will happen if the load balancing assumption A-7 strictly holds. However, this will not hold in practice because the node positions may have some asymmetry. We next examine the consequence of removing the assumption. To distinguish, we shall refer to the ideal routing situation where the assumption A-7 is perfectly met as *Load Balanced Shortest Path First* (LBSPF).

We focus on the first tier, since we know the lifetime is defined by these nodes. If assumption A-7 is not satisfied in the first tier, then all nodes of the first tier will not die at the same time. The lifetime of the network will be defined by the first tier node which dies last. However, before this time, the number of first tier nodes still alive has declined slowly. The number of nodes that remain alive in the first tier at any given time affects the total traffic generated by the first tier itself. Initially, the total battery amount of first tier nodes is $N_1 b$. For each period, the first tier consumes an energy equal to $N_{alive}\beta p$, where N_{alive} is the number of active nodes in the network. A routing algorithm can maximize the lifetime of the network if it can reduce N_{alive} as soon as possible. A practical way to quickly reduce the number of nodes that are alive is to overburden a node until its battery is depleted. Thus, the routing algorithm should select a node x_1 in the first tier and route *all* flows through node x_1 until it depletes its battery. After node x_1 dies, another node from the first tier, x_2, is selected to carry all the network flows, and so on until the last node in the first tier dies (at which time the network becomes disconnected). We shall refer to this rather curious routing approach as *Bottleneck Routing* (BR). While BR does not belong in the set $\mathcal{R}$ (not all nodes in tier 2 may be able to reach x_1 in one hop), it represents the extreme limit of unbalanced routing protocols in $\mathcal{R}$. Thus all protocols in $\mathcal{R}$ will result in a network lifetime bounded by those achieved by LBSPF and BR, from below and above respectively.

In LBSPF, the base station will receive readings from all nodes for the entire lifetime. This is no longer true for BR, some nodes will die and stop reporting before lifetime expires. While this may be a problem from the sensing application's perspective, we show below that it improves the lifetime of the network as we defined it earlier.

***Theorem* 3** *If we remove assumption A-7, the maximum lifetime of a WSN using a routing algorithm $r \in R$ is bounded by $L < L_{max}$, where*

$$L_{\max} = \frac{b}{\beta p}\left[1 - \left(1 - \frac{1}{N - N_1 + 1}\right)^{N_1}\right] \tag{2.6}$$

Proof As discussed above, the lifetime is composed of different periods when the different nodes of the first tier will take turns forwarding all traffic from outside the first tier. To compute the lifetime of the network we simply add the times it takes for all nodes in the first tier $x_1, x_2, \ldots, x_{n_1}$ to die:

- node x_1 will carry the flows on behalf of $N - N_1$ nodes and its own flow. Thus it will die after $t_1 = \frac{b}{(N-N_1+1)\beta p}$.
- node x_2 will carry only one flow for time t_1 and then the same number of flows as node x_1, and hence will die after t_2 seconds after the death of x_1: $t_2 = \frac{b-t_1\beta p}{(N-N_1+1)\beta p}$.

  $\vdots$
- node x_{N_1} will die t_{N_1} seconds after node x_{N_1-1} died, where $t_{N_1} = \frac{b-\sum_{i=1}^{N_1-1} t_i\beta p}{(N-N_1+1)\beta p}$.

Thus,

$$L_{\max} = \sum_{i=1}^{N_1} t_i. \tag{2.7}$$

Equation (2.7) can be further manipulated by noticing that $t_i = t_1(1 - \frac{1}{N-N_1+1})^{i-1}$ for all i such that $2 \le i \le N_1$. Then (2.6) follows immediately as the sum of a geometric distribution.

Comments:

- LBSPF and BR are the two extreme approaches to routing in WSNs. LBSPF ensures that the time of the death of the first node is postponed as much as possible. On the other hand, BR postpones the time of the disconnection of the network as much as possible. Any minimum hop routing will result in routes that will fall between these two extremes, hence so will the lifetimes.
- Figure 2.1 depicts the difference in the network lifetimes of the two approaches as a function of the total number of nodes N over the number of nodes in the first tier N_1. For this figure N_1 was kept constant at 100 nodes while N increased from 200 nodes to 2500 nodes. It is interesting to see that the difference between the two extremes becomes very small as total number of nodes becomes large in comparison to the number of nodes in tier 1.
- Bottleneck Routing maximizes the lifetime of the network at the expense of purposely depleting some of the nodes relatively early. For most applications it is unlikely that this is desired, especially since, for large networks, the savings in the lifetime are insignificant (Fig. 2.1). This observation makes the definition of *optimal* WSN routing protocols that use only the lifetime as an optimization criteria questionable.

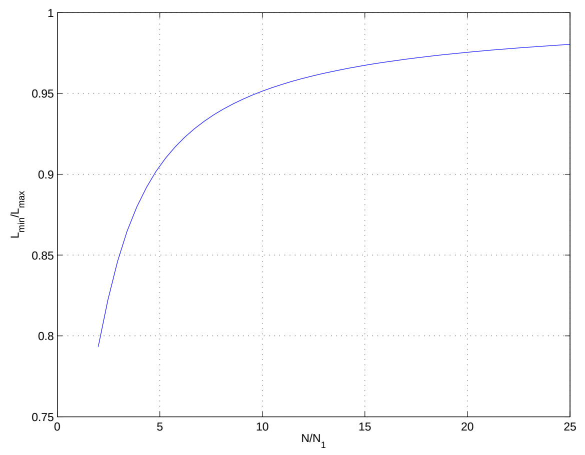


Figure 2.1: Lifetime ratios using LBSPF and BSPF as a function of the ratio between the total number of nodes and the nodes in tier one.

It is clear that for all possible routing algorithms in $\mathcal{R}$ the lifetime L of the network falls somewhere between the two extremes: $L_{\min} \leq L < L_{\max}$. Moreover, the two extremes are very close to each other especially for large networks. Therefore it can be claimed that the choice of the routing protocol does not make a significant difference in the lifetime of the network. For example, in a uniformly distributed WSN with five tiers the difference between the two lifetimes is less than 2%.

It is likely that a simple protocol will perform just as well as a more complex protocol. The only major differentiation between different routing protocols is in their overhead (included in P_b in (2.1)).

2.5 Simulation Results

To validate the results in Section 2.4 we simulated a WSN of variable size with two routing algorithms.

We have implemented two versions of Shortest Path First (SPF) algorithms to compare the lifetime of WSN using these algorithms with the theoretical limit:

MSPF The algorithm selects among the neighbors with the same number of minimum hops to the base station the one with the largest remaining power. Essentially this algorithm behaves very like Load Balancing SPF ensuring that all nodes in the first tier die at (almost) the same time.

RSPF The algorithm selects randomly among the neighbors with the same number of minimum hops to the base station.

We reroute (choose new routes for all nodes) periodically (every one time unit) or whenever a node dies.

We fixed the node density at ($0.01 nodes/m^2$) and the transmission radius of the nodes ($30m$). The transmission of the data generated by a node each time unit costs one unit of energy. The nodes initially have 1000 units of energy. All simulations were repeated thirty times with different random seeds; in what follows, the average of these results is presented.

Figures 2.2 and 2.3 depict the variation of the network lifetime with the network size (constant density) for uniform (we used a rectangular grid) and random placement respectively. The lifetimes of network using the two versions of SPF algorithms are very close together and between the theoretical values given by (2.4) and (2.6). The lifetime are so close that they are hard to tell apart from each other. There is also no significant difference between the strictly uniform and the random placements beyond

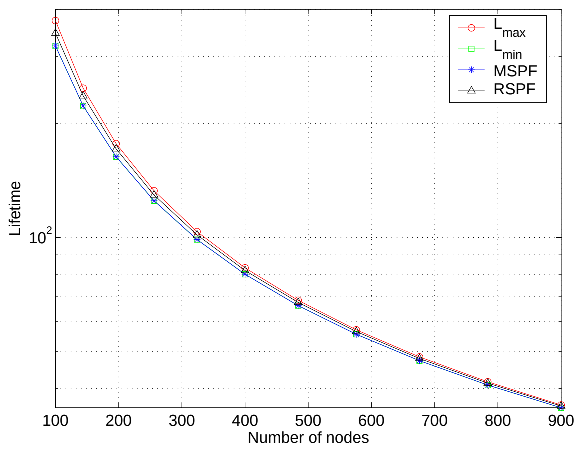


Figure 2.2: Variation of the lifetime of the network for a rectangular grid placement as a function of the networks size

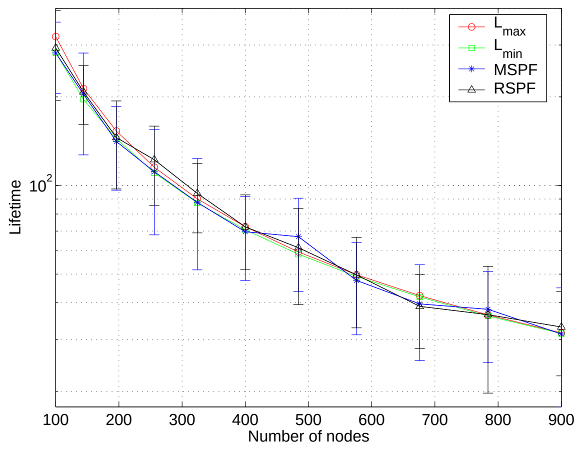


Figure 2.3: Variation of the lifetime of the network for random placement as a function of the networks size

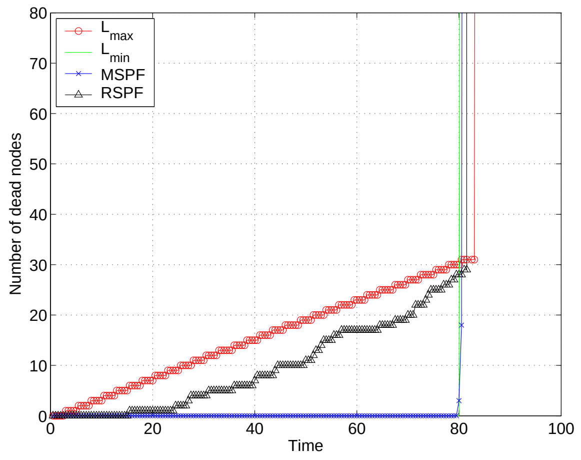


Figure 2.4: Time of death for each node for various routing strategies for an uniform rectangular grid placement.

the significant variation introduced by the random initial topology. Figure 2.3 also depicts the 95% confidence interval corresponding to the average lifetimes. The lifetime of the network decreases as the number of sensor nodes increases: a fixed number of tier one nodes carry increasingly more packets and, hence, naturally die sooner.

Figures 2.4 and 2.5 show the moment of death of each node in the first tier of the network. For the two limits corresponding to $L_{\min}$ and $L_{\max}$ we depicted the times when first tier nodes are expected to die following the LBSPF and BR algorithms. For this simulation we used $N = 400$ nodes in an area $190m \times 190m$. MSPF for the grid network works as expected: practically all nodes of the networks are alive for the entire lifetime of the network. For the random placement scenario, MSPF works reasonably well, but less so than in the case of the uniform grid. The main reason behind this behavior is that in the case of random placement there might not be possible to balance the load, and inevitably some nodes will die sooner than others. The number of disconnected nodes spikes abruptly when the network becomes disconnected, i.e. when the network reached its lifetime. As expected, for both placements, the lifetime of RSPF is slightly larger than for MSPF at the expense of the early deaths of some tier one nodes.

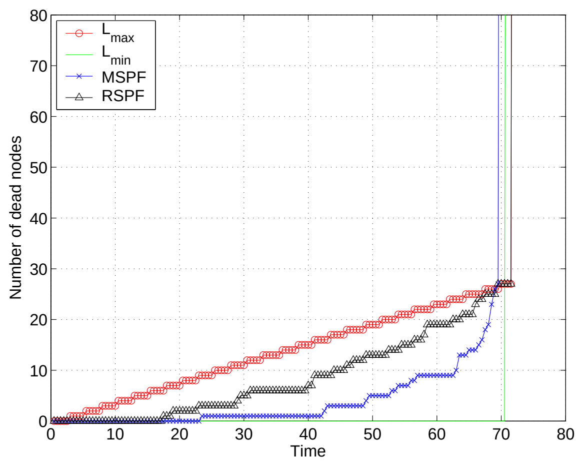


Figure 2.5: Time of death for each node for various routing strategies for a random placement.

2.6 Conclusion

In this chapter we presented an analysis of the lifetime of wireless sensor networks that employ periodic sensing. Lower and upper bounds on the network lifetime are derived, and corresponding routing algorithms leading to these bounds are presented. For large sensor networks the upper and the lower bounds on the network lifetime are relatively close (less than a few percents), leading thus to the conclusion that for such sensor networks the choice of the routing protocol is largely irrelevant for maximizing the network lifetime, as long as some form of shortest paths are followed. Simulations are used to validate the theoretical results.

While the set $\mathcal{R}$ may appear to be rather restrictive, in reality our results are likely to continue to hold for many sensible routing approaches. We are currently working on developing descriptions of such routing families, and on extending the concept of network lifetime.

Chapter 3

Time Synchronization

3.1 Introduction

Recent technological advances in low power radios, sensors and microcontrollers enable a new monitoring paradigm for large geographical areas. The vision involves a large number of inexpensive nodes equipped with one or more sensors, a small microcontroller and transceivers capable of short-range communications. Wireless sensor networks (WSNs) [2–4] have the potential to truly revolutionize the way we monitor and control our environment.

Many (if not most) WSN applications either benefit from, or require, time synchronization. Sensed data is typically forwarded over multiple hops to one or more base stations and encounters delays ranging from a few tens of milliseconds to several minutes [31]. Many energy-efficient algorithms trade power savings for increased delays. Therefore, without time synchronization, the time that sensed data reached the base station is often a poor approximation of the time the data was sensed. If the entire sensor network is time-synchronized (with respect to the base station), meaningful time-stamps can accompany sensed data. Target tracking, one of the most popular sensor network applications, requires tight time synchronization for beam-forming. Many WSN protocols (e.g., power-saving sleep schedules [32], TDMA schedules for maximizing the networks' capacity, security mechanisms for preventing replay attacks, etc.) also rely on time synchronization.

In this Chapter, we present two closely-related algorithms suitable for *pair-wise* time synchronization. In those algorithms, one node determines (tight) bounds of the relative offset and drift of its clock with respect to the one of the other node. We also present a scheme building on these algorithms, which is capable of synchronizing the clocks of an entire sensor field. To avoid confusion between the two algorithms, we will name them *mini-sync* and *tiny-sync* as they use limited and very limited resources, respectively. The two algorithms feature:

- *Drift awareness:* The algorithms not only take the drift of the clock into account, but also find tight bounds on the drift.
- *Deterministic bounds on the precision:* Most other algorithms provide best estimates for the offset and drift of the clocks, and possibly probabilistic bounds on these estimates. Our approach delivers tight, *deterministic* bounds on these estimates, such that absolute information can be deduced about ordering and simultaneous events.
- *Precision:* Given small uncertainty bounds on the delays exchanged messages undergo, the precision of the synchronization can be arbitrarily good.
- *Low computation and storage complexity:* Wireless sensor nodes typically feature low computational power microcontrollers with small amounts of RAM. Both algorithms have low computational and storage complexity.
- *Low sensitivity to communication errors:* Wireless communications are notoriously error prone, and thus one cannot rely on correct receipt of all messages. The presented approach works correctly even if a large percentage of the messages is lost.

The main contribution of the work is the development and performance evaluation of two *simple* algorithms that deliver accurate offset and drift information together with *tight* bounds on them.

The two algorithms presented in this chapter are not limited to wireless sensor networks. They can synchronize the nodes on any communication network which allows bidirectional data transmission. However, the algorithms provide very good precision (microsecond if crafted carefully) and bounds on the precision while using very limited resources, thus being especially well suited for wireless sensor networks.

3.2 Related Work

Time synchronization is a key service for many applications and operating systems in distributed computing environments. Many protocols have been proposed and used for time synchronization in wired and wireless networks. Mill's Network Time Protocol (NTP) [33,34] has been widely used in the Internet for decades. Nodes could also be equipped with a global positioning system (GPS) [35] to synchronize them. However, traditional synchronization schemes and GPS-equipped systems are not suitable for use in WSNs due to the specific requirements of those networks:

- *Precision:* Depending on the considered application, WSNs may require far better precision than traditional networks. For example, a precision of a few milliseconds is considered satisfactory for NTP, while in a WSN beam-forming application, microsecond precision can significantly improve the performance of the application [36]. Furthermore, when coordinating synchronized time schedules, the higher the precision of the synchronization algorithm, the smaller the guard times have to be (and, hence, the higher the efficiency of the scheduling approach).

- *Cost:* Cost is of primary concern in WSNs as, typically, nodes have limited batteries, computational and storage resources. Most of the protocols designed for wired environments exchange many messages for statistical processing. Furthermore, the protocols also need to store the messages to process them.

Recently, a significant amount of research on time synchronization for wireless sensor networks has been published [37,38]. An interesting approach called *post facto synchronization* was proposed by Elson and Estrin [39]. In this approach, each node's clock is normally unsynchronized with the rest of the network; a beacon node periodically broadcasts beacon messages to the sensor nodes in its wireless range. When an event is detected, each node records the time of the event (time-stamp with its own local clock). After the event (hence, the name), upon receiving the reference beacon message, nodes use it as time reference and adjust their event timestamps with respect to that reference.

The reference broadcast synchronization (RBS) protocol [40], uses a data collection mechanism similar to the one in the post facto synchronization: one node acts as a beacon by broadcasting a reference packet. All receivers record the packet arrival time. The receiver nodes then exchange their recorded timestamps and estimate their relative phase offsets. RBS also estimates the clock skew by using a least-squares linear regression. The interesting feature of RBS is that it records the timestamp only at the receivers. Thus, all timing uncertainties (including MAC medium access time) on the transmitter's side are eliminated. This characteristic makes it especially suitable for hardware that does not provide low-level access to the MAC layer (e.g., 802.11). Although RBS synchronization only involves one hop neighbors, the mechanism can be extended to synchronize a multi-hop network [41]. In such a system, timestamps in messages can be reconciled as they are being forwarded by the intermediate nodes (according to their next-hop destination and its time difference to the current node) [42]. The main disadvantage of RBS is that it does not synchronize the sender with the receiver directly and that, when the programmers have low-level access at the MAC layer, simpler methods (e.g., TPSN) can achieve a similar precision to RBS.

Römer [43] presented a synchronization protocol for ad hoc networks. The authors assume clocks with known upper-bounds on the clock drift. The basic idea of the algorithm is to compute timestamps using unsynchronized local clocks. When a local time-stamp is transferred between two nodes, the timestamp is transformed to the local time of the receiving node with guaranteed bounds based on the assumed maximum clock drift. These protocols focus on temporal relationships between the events such as ”event X happened before event Y” and ”event X and Y happened within a certain time interval.” [44] enhanced Römer's algorithm and achieved higher accuracy while reducing computation.

When implementing time synchronization protocols, significant challenge is minimizing timestamping uncertainties. RBS reduces these uncertainties by timestamping only at the receivers. The time synchronization protocol for sensor network (TPSN) [45] reduces the uncertainties by using timestamps at the medium access control (MAC) layer. This eliminates the (large) uncertainties introduced by the MAC layer, e.g., retransmissions, backoffs, medium access, etc. TPSN takes advantage of the availability of the MAC layer code in TinyOS [46]. For a single beacon, TPSN offers a two-fold increase in precision in comparison to RBS (although, asymptotically, as more beacons are sent, they achieve the same precision).

The Flooding Time Synchronization Protocol (FTSP) [47] was designed for a sniper localization application requiring very high precision [36]. FTSP achieves the required accuracy by utilizing a customized MAC layer time-stamping and by using calibration to eliminate unknown delays. FTSP is robust to network failures, as it uses flooding both for pair-wise and for global synchronization. Linear regression from multiple timestamps is used to estimate the clock drift and offset. The main drawback of FTSP is that it requires calibration on the hardware actually used in the deployment (and thus, it is not a purely software solution independent of the hardware). FTSP also *requires* intimate access to the MAC layer for multiple timestamps. However, if well calibrated, the FTSP's precision is impressive (less than $2\mu s$).

Lightweight Time Synchronization (LTS) [48] was proposed for applications where the required time accuracy is relatively low. The pair-wise synchronization on LTS is similar to TPSN except for the treatment of the uncertainties (LTS adopts a statistical model for handling the errors). The simulation results show that the accuracy of LTS is about 0.5 seconds.

Li and Rus [49] presented a high-level framework for global synchronization. The authors proposed three methods for global synchronization in WSNs. The first two methods, all-node-based and cluster-based synchronization, use global information and are, hence, not suitable for large WSNs. In the third method (diffusion), each node sets its clock to the average clock time of its neighbors. The authors showed that the diffusion method converges to a global average value. A drawback of this approach is the potentially large number of messages exchanged between neighbor nodes, especially in dense networks.

Dai and Han [50] proposed two time synchronization protocols, Hierarchy Referencing Time Synchronization (HRTS) and Individual-based Time Request (ITR). HRTS uses an idea similar to RBS except for the synchronization of the receivers after the beacon synchronization message was sent: instead of the receivers exchanging messages among themselves, a designated node sends its time to the beacon node that, in turn, broadcasts this message over the entire network, thus, significantly reducing the number of message exchanges. Similar to RBS, the beacon node has to be able to broadcast to the entire sensor network. The ITR protocol is based on NTP. Multichannel support is integrated both in ITR as well as HRTS to reduce the delay variations.

Delay Measurement Time Synchronization Protocol (DMTS) [51] reduces the number of message exchanges in RBS by accurately estimating the delay of the path from the sender to the receiver. DMTS is similar to RBS on the sender side and TPSN or FTSP on the receiver side.

Adaptive Clock Synchronization [52] is a probabilistic method for clock synchronization that uses the higher precision of receiver-to-receiver synchronization. The protocol extended the deterministic RBS protocol to provide a probabilistic bound on the accuracy of the clock synchronization. The bound allows a tradeoff between the accuracy and the resource requirement.

Su [53] proposed Time-Diffusion Synchronization Protocol (TDP) for network-wide time synchronization. TDP maintains global time synchronization within an adjustable bound (based on the application requirements). TDP achieves global synchronization by multi-hop flooding: the base station initiates the protocol by sending a special timing message to the entire network. Some of the nodes, upon receiving the message, become masters by using a leader election procedure (that uses a False Ticker Isolation Algorithm to discard outliers and a Load Distribution Algorithm to balance the energy consumption of the network). The master nodes start the time diffusion procedure involving electing diffused leaders (similar to the master election algorithm), multi-hop flooding and iterative weighted averaging of timing from different master nodes. TDP handles node mobility and failures by using a Peer Evaluation Procedure. The method achieves a precision of 0.1s.

3.3 Proposed Algorithms

In this section, we present the two proposed algorithms and their expected performance.

3.3.1 Data Collection

For the proposed algorithms, we will use a classical data collection algorithm [33, 43, 45, 54, 55]; however, we will process the data differently.

Consider a wireless node i with its hardware clock $t_i(t)$, where t is the Coordinated Universal Time (UTC). In general, the hardware clock of node i is a monotonically non-decreasing function of t. In practice, a quartz oscillator is often used to generate the real-time clock. The oscillator's frequency depends on the ambient conditions, but for relatively extended periods of time (minutes - hours), the hardware clock can be approximated with good accuracy by an oscillator with fixed frequency:

$$t_i(t) = a_i t + b_i, \tag{3.1}$$

where a_i and b_i are the drift and the offset of node i's clock. In general, a_i and b_i will be different for each node and approximately constant for an extended period of time.

Consider two wireless nodes 1 and 2 with their hardware clocks $t_1(t)$ and $t_2(t)$, respectively. From (3.1), it follows that t_1 and t_2 are linearly related:

$$t_1(t) = a_{12}t_2(t) + b_{12}. \tag{3.2}$$

The parameters a_{12} and b_{12} represent the *relative drift* and the *relative offset* between the two clocks, respectively. If the two clocks are perfectly synchronized, the relative drift is equal to one, and the relative offset is equal to zero.

Assume that node 1 would like to be able to determine the relationship between t_1 and t_2. Node 1 sends a probe message to node 2. The probe message is time-stamped just before it is sent with t_o. Upon receipt, node 2 time-stamps the probe t_b and returns it immediately (we will shortly relax this constraint) to node 1 which timestamps it upon receipt t_r. Figure 3.1 depicts such an exchange.

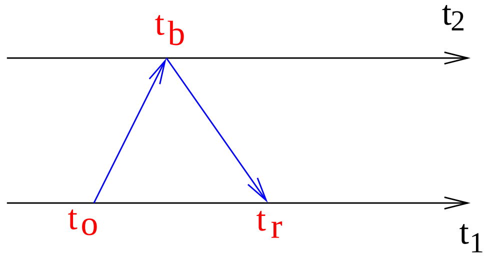


Figure 3.1: A probe message from node 1 is immediately returned by node 2 and time-stamped at each send/receive point resulting in the data-point (t_o, t_b, t_r).

The three time-stamps (t_o, t_b, t_r) form a data-point which effectively limits the possible values of parameters a_{12} and b_{12} in (3.2). Indeed, since t_o happened *before* t_b, and t_b happened *before* t_r, the following inequalities should hold:

$$t_o < a_{12}t_b + b_{12}, \quad \text{and} \tag{3.3}$$

$$t_r > a_{12}t_b + b_{12}. \tag{3.4}$$

The data collection procedure described above is repeated several times; and each probe that returns, provides a new data-point and, thus, new constraints on the admissible values of a_{12} and b_{12}.

The linear dependence between t_1 and t_2 and the constraints imposed by the data-points can be represented graphically as shown in Fig. 3.2. Each data-point can be represented by two constraints in the system of coordinates given by the local clocks of the two nodes t_2 and t_1. The double subscript of the timestamps t_{o_i}, t_{b_i} and t_{r_i} denotes the timestamps corresponding to data-point i. Inequality (3.3) imposes a constraint on the line representing the relationship (3.2): the line has to be over the point of

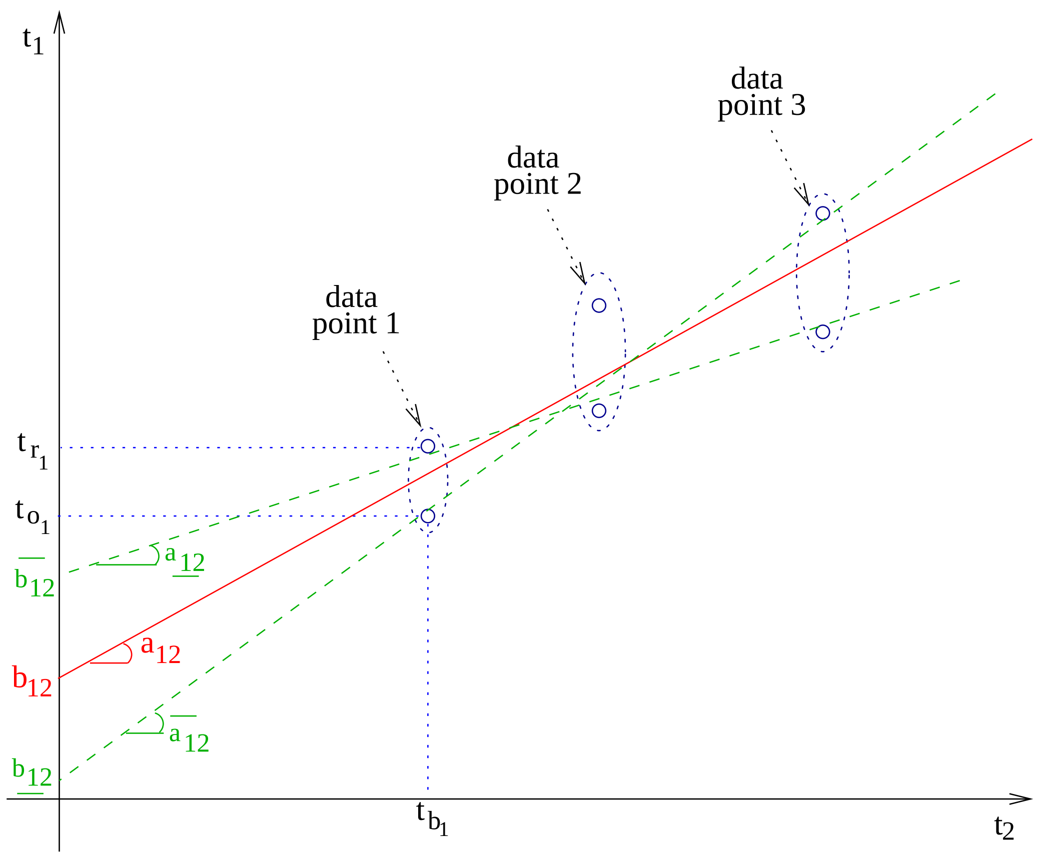


Figure 3.2: The linear dependence (3.2) and the constraints imposed on a_{12} and b_{12} by three data-points.

coordinates (t_b, t_o). Similarly, corresponding to inequality (3.4), the line has to be under the point of coordinates (t_b, t_r) (the second constraint). Satisfying both constraints, requires the line to be positioned *between* the two constraints determined by each data-point. The exact values of a_{12} and b_{12} cannot be accurately determined using this approach (or any other approach) as long as the message delays are unknown. But, a_{12} and b_{12} can be bounded by:

$$\underline{a_{12}} \leq \quad a_{12} \quad \leq \overline{a_{12}}, \quad \text{and} \tag{3.5}$$

$$\underline{b_{12}} \leq \quad b_{12} \quad \leq \overline{b_{12}}, \tag{3.6}$$

where $\overline{a_{12}}$ ($\underline{a_{12}}$) is the maximum (minimum) of the slopes of lines that satisfy the constraints, and $\underline{b_{12}}$ ($\overline{b_{12}}$) is the value on the y-axis at the intersection with the line corresponding to $\overline{a_{12}}$ ($\underline{a_{12}}$).

Not all combinations of a_{12} and b_{12} satisfying (3.5) and (3.6) are valid, but all valid combinations satisfy (3.5) and (3.6). The real values of a_{12} and b_{12} can be estimated as the midpoint of the range of possible values $\widehat{a_{12}}$ and $\widehat{b_{12}}$:

$$a_{12} \in \left[\widehat{a_{12}} - \frac{\Delta a_{12}}{2}; \widehat{a_{12}} + \frac{\Delta a_{12}}{2}\right], \quad \text{and} \tag{3.7}$$

$$b_{12} \in \left[\widehat{b_{12}} - \frac{\Delta b_{12}}{2}; \widehat{b_{12}} + \frac{\Delta b_{12}}{2}\right], \tag{3.8}$$

where

$$\widehat{a_{12}} = \frac{\overline{a_{12}} + \underline{a_{12}}}{2}, \tag{3.9}$$

$$\Delta a_{12} = \overline{a_{12}} - \underline{a_{12}}, \tag{3.10}$$

$$\widehat{b_{12}} = \frac{\overline{b_{12}} + \underline{b_{12}}}{2}, \quad \text{and} \tag{3.11}$$

$$\Delta b_{12} = \overline{b_{12}} - \underline{b_{12}}. \tag{3.12}$$

The goal of the algorithms is to determine $\overline{a_{12}}, \underline{a_{12}}, \overline{b_{12}}$ and $\underline{b_{12}}$ as tight as possible (such that it minimizes Δa_{12} and Δb_{12}). Once a_{12} and b_{12} are estimated, node 1 can always correct the reading of the local clock (using (3.2)) to have it match the readings of the clock at node 2.

To decrease the overhead of this data-gathering algorithm, the probes can be piggy-backed on data messages. Since most MAC protocols in wireless networks employ an acknowledgment (ACK) scheme, the probes can be piggy-backed on the data and the responses on the ACKs. Elaborate schemes with optional headers can be devised to reduce the length of the header when probes do not need to be sent. This way, synchronization can be achieved almost "for free" (i.e., with very little overhead in terms of communication bandwidth).

Relaxing the Immediate Reply Assumption

In Fig. 3.1, we assumed that node 2 replies immediately to node 1 when it receives a probe. The correctness of the presented approach is not affected in any way even if node 2 does not respond *immediately*. Node 2 can delay the reply as long as it wants; the relations (3.3) and (3.4), and, thus the rest of the analysis will still hold.

However, as the delay between t_o and t_r increases, the precision of the estimates will decrease. In practice, node 2 may have to delay the reply due to any number of reasons (e.g., it has something more important to send, it has to postpone its transmission due to medium access contention, etc.).

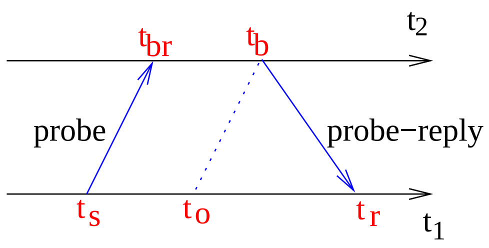


Figure 3.3: A probe message from node 1 may be returned by node 2 after being time-stamped at both the send and receive points.

To counteract the possible loss in precision, node 2 can time-stamp the probe message upon receipt (t_{br}) as well as upon reply (t_b) (as depicted in Fig. 3.3). In this case, node 1 can adjust t_o as follows:

$$t_o = t_s + \widehat{a_{12}} \left(t_b - t_{br} \right), \tag{3.13}$$

where t_o is the latest time at node 1 that is known to have occurred before t_b. The same inequalities (3.3) and (3.4) hold for this case as well.

Increasing Precision by Considering Minimum Delay

If no information about delays encountered by the probe messages is available, nothing else can be done to increase the precision. However, if the minimum delay a probe encounters between the nodes is known, the data-points can be adjusted for an increase in the precision of the results.

To determine the minimum delay, one can take into account the minimum length of such a probe and the time it takes to transmit such a probe (at the transmission rate of the sensor node), and, eventually, other operations that have to be completed before the probe is sent or upon receiving such a probe (e.g., encryption/decryption, CRC calculation, etc.).

Assume that we are able to determine the minimum delay δ_{12} that the probe encounters between t_o (t_s) at node 1 and t_b (t_{br}) at node 2. Also, denote with δ_{21}, the minimum delay between the moment t_b at node 2 and the moment t_r at node 1. Then, $(t_o + \delta_{12},\ t_b,\ t_r - \delta_{21})$ should be used as a data-point as this will offer increased accuracy over the data-point (t_o, t_b, t_r).

If, for two probes, both minimums δ_{12} and δ_{21} are reached, the method presented in this paper can achieve *perfect* synchronization (i.e., there will be no uncertainties for relative offset and relative drift: $\Delta a_{12} = \Delta b_{12} = 0$).

3.3.2 Tiny-sync and Mini-sync - Processing the Data

After acquiring a few (at least two) data-points, the offset and the drift can be estimated using inequalities (3.3) and (3.4). An existing solution for finding the optimal bounds on the drift and offset involves solving two linear programming problems with twice as many inequalities as data-points [55]. By optimal bounds (and the corresponding optimal solution) we mean the bounds that minimize the difference between the bounds.

The disadvantage of this approach is that as more and more data samples are collected, the computational and storage requirements increase (potentially unbounded). Also, one should not limit the number of collected samples to a fixed window as the best drift estimates are obtained when a large number of samples are available. The approach in [55] is clearly not suitable for systems with limited memory and computing resources such as wireless sensor nodes. In this paper, we will pursue another avenue.

The two proposed algorithms spring from the observation that not all data-points are useful. In Fig. 3.2, the bounds on the estimates $[\underline{a_{12}}, \overline{a_{12}}]$ and $[\underline{b_{12}}, \overline{b_{12}}]$ are constrained only by data-points 1 and 3. Therefore, we do not need data-point 2, and we can discard it, as data-point 3 produces better estimates than data-point 2.

It seems that only four constraints (the ones which define the best bounds on the estimates) have to be stored at any time. Upon the arrival of a new data-point, the two new constraints are compared with the existing four constraints and two of the six are discarded (i.e., the four constraints which result in the best estimates are kept). The comparison operation to decide which four constraints to be kept is very simple, computationally (only 8 additions, 4 divisions and 4 comparisons). At any one time, only the information for the best four constraints needs to be stored. We will name the algorithm described in this paragraph "tiny-sync." The four constraints that are stored at any one time instant may belong to two, three or four different data-points. With the four constraints, the bounds of the clock offset and drift are easily computed from the two lines that can be constructed with the two lower bound and two upper bound constraints: the first line is determined by the first lower bound constraint and the second upper bound constraint and the other line is determined by the first upper bound constraint and second lower bound constraint. The values of $\overline{a_{12}}$, $\underline{b_{12}}$, $\underline{a_{12}}$ and $\overline{b_{12}}$ are the slope and offset of those two lines respectively.

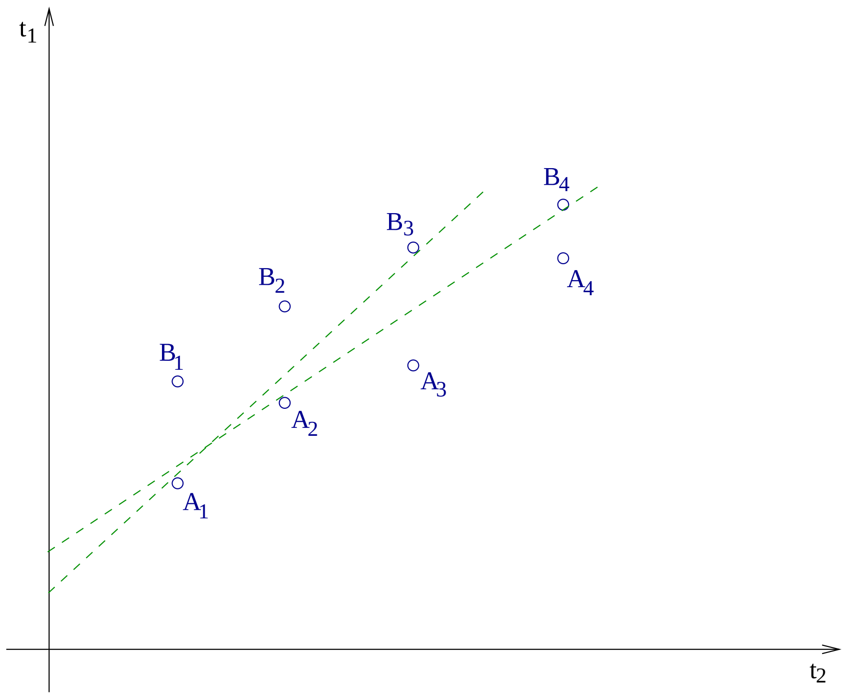


Figure 3.4: In this situation, after receiving data-point $A_3 - B_3$, tiny-sync will discard constraints A_2 and B_2. However, after receiving $A_4 - B_4$, it turns out that the most constraining constraint would have been $A_2 - B_4$.

Unfortunately, while tiny-sync is very efficient, it does not always produce the optimal solution. Consider the situation depicted in Fig. 3.4. For clarity, we labeled each constraint individually, that is, for data-point i, A_i represents constraint (t_{b_i}, t_{o_i}), and B_i represents constraint (t_{b_i}, t_{r_i}). After the first two data-points (A_1-B_1) and (A_2-B_2) are received, the first estimates for the drift and offset may be computed. After the third data-point (A_3-B_3) is received, the bounds on the estimates improve (i.e., Δa_{12} and Δb_{12} are smaller); so, the constraints A_1,B_1,A_3 and B_3 are stored, while A_2 and B_2 are discarded. The next data point (A_4-B_4) could have used constraint A_2 to construct a better estimate. Unfortunately, A_2 was already discarded at this point; and thus, a less than ideal estimate for $\underline{b_{12}}$ will now be imposed by A_1 and A_4. Thus, while producing correct results, tiny-sync might miss the optimum result. We will compare the performance of tiny-sync with the optimal solution in Section 3.5.

In Fig. 3.4, the constraint A_2 was discarded by tiny-sync because it was not immediately useful, but rather only potentially useful in the future. By a potentially useful constraint we mean a constraint that, with the current data points, is not one of the constraints that is determining the values of $\underline{a_{12}}, \overline{a_{12}}, \underline{b_{12}}$ and $\overline{b_{12}}$, but which may do so given future data points. This does not mean that all the constraints are potentially useful. In fact, only the constraints A_j (e.g., A_2) that satisfy the condition

$$m(A_i, A_j) > m(A_j, A_k) \tag{3.14}$$

for some integers $1 \leq i < j < k$ are potentially useful in the future (by $m(X, Y)$, we denote the slope of the line going through the points X and Y).

***Theorem* 4** *Any constraint A_j which satisfies*

$$m(A_i, A_j) \leq m(A_j, A_k) \tag{3.15}$$

for at least one set of integers $1 \leq i < j < k$ can be safely discarded as it will never constrain the bounds $\underline{a_{12}}, \overline{a_{12}}, \underline{b_{12}}$ and $\overline{b_{12}}$ more than any existing constraints.

The proof is presented in the Appendix. Similar conditions for discarding upper-bound constraints (B_i) exist.

The resulting algorithm (called “mini-sync”), upon the receipt of a new-data point, will check if the new constraints can eliminate any of the old constraints. Potentially, many old constraints can be eliminated with one new data-point. We use all the remaining constraints to obtain the (still optimal) solution using the same procedure as in [55]. Since we only eliminate constraints (inequalities) that are irrelevant, we still obtain the optimal solution with only a few points (solving the set of all inequalities is shown to result in the optimal solution [55]).

Storing only four points, as in tiny-sync, does not produce the optimal solution. How many points have to be stored for the optimal solution? Theoretically, a potentially large number. If the delay between nodes 1 and 2 is monotonically increasing, (3.14) can hold for all of the constraints A_j. In practice, the delays do not continuously increase monotonically; therefore, only several constraints need to be stored to obtain the optimal result.

3.3.3 Analysis of the Proposed Algorithms

In this section, we will analyze the expected performance of tiny-sync as a function of the system parameters (round-trip time, probing period, etc.).

For the analysis, we make the simplifying assumption that the difference between t_o and t_r is constant and equal to RTT:

$$t_{r_i} - t_{o_i} = RTT \qquad \forall i \geq 1. \tag{3.16}$$

In other words, we assume that the sum of all unknown delays between the moment the probe is sent and the moment the reply is received is always constant and equal to RTT.

We justify this simplifying assumption by considering the following:

- Tiny-sync only stores and uses in its computation the *best* two data-points collected so far (in terms of constraining the uncertain relative drift and offset). Data points with small round-trip times result in the tightest constraints, and, hence, most often, the two stored data-points have their round-trip times equal to the minimum round trip-time of the connection;

- The *variation* of the round-trip delays is typically very small (microseconds) when compared to the sampling intervals (seconds to tens of seconds); and

- As shown in Section 3.5, the theoretical analysis matches the experimental results exceedingly well.

With assumption (3.16), the data points that define the best approximation are always the first and last data-points collected. All of the other data-points can be safely discarded. Thus, with this assumption, tiny-sync's performance is identical to the performance of mini-sync (the optimal algorithm). Denote with $(t_{o_1}, t_{b_1}, t_{r_1})$ and $(t_{o_2}, t_{b_2}, t_{r_2})$ the first and last collected data-points, respectively.

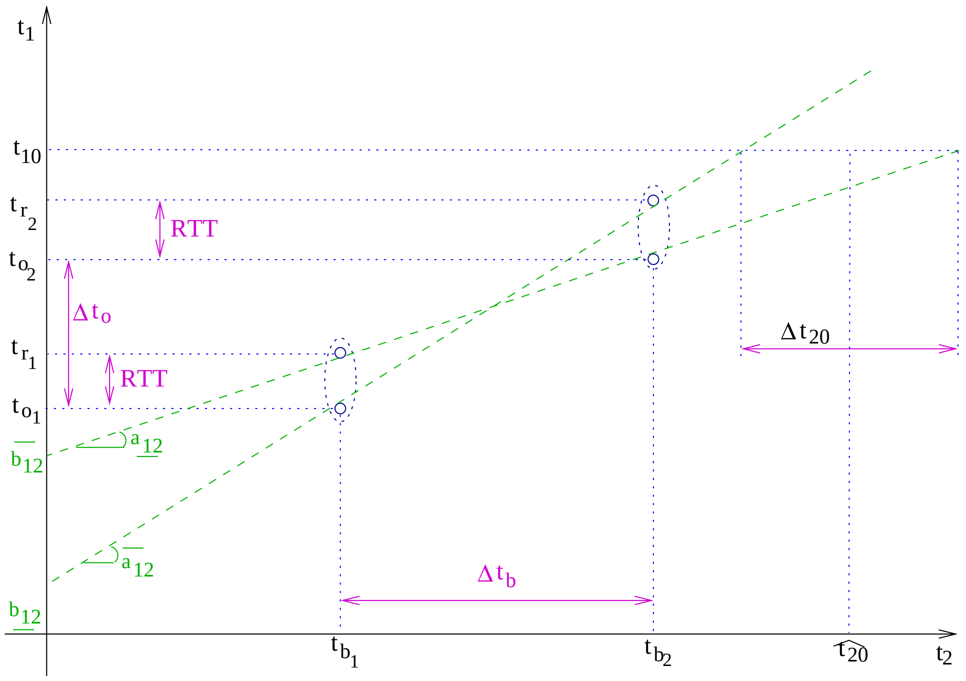


Figure 3.5: Setup for the simplified analysis of the algorithms.

We use the following notation:

$$\Delta t_o = t_{o_2} - t_{o_1} \quad \text{and} \tag{3.17}$$

$$\Delta t_b = t_{b_2} - t_{b_1}. \tag{3.18}$$

Solving the linear equations for the unknowns $\underline{a_{12}}$, $\overline{a_{12}}$, $\underline{b_{12}}$ and $\overline{b_{12}}$, we obtain:

$$\underline{a_{12}} = \frac{\Delta t_o - RTT}{\Delta t_b}, \tag{3.19}$$

$$\overline{a_{12}} = \frac{\Delta t_o + RTT}{\Delta t_b}, \tag{3.20}$$

$$\underline{b_{12}} = t_{o_1} - t_{b_1}\overline{a_{12}}, \quad \text{and} \tag{3.21}$$

$$\overline{b_{12}} = t_{o_1} + RTT - t_{b_1}\underline{a_{12}}. \tag{3.22}$$

Using (3.10) and (3.12), we obtain the following bounds on Δa_{12} and Δb_{12}:

$$\Delta a_{12} = \frac{2RTT}{\Delta t_b}, \quad \text{and} \tag{3.23}$$

$$\Delta b_{12} = RTT + t_{b_1}\Delta a_{12}. \tag{3.24}$$

Equations (3.23) and (3.24) capture the essential behavior of the presented algorithms. Equation (3.23) implies that the uncertainty bound on the drift decreases continuously as the distance between the first and the last collected data-points increases:

$$\lim_{\Delta t_b \to \infty} \Delta a_{12} = 0. \tag{3.25}$$

Figure 3.6 depicts the evolution of the uncertainty bound on the relative clock drifts Δa_{12}, which remains constant between data-points, and improves upon the receipt of a new data-point.

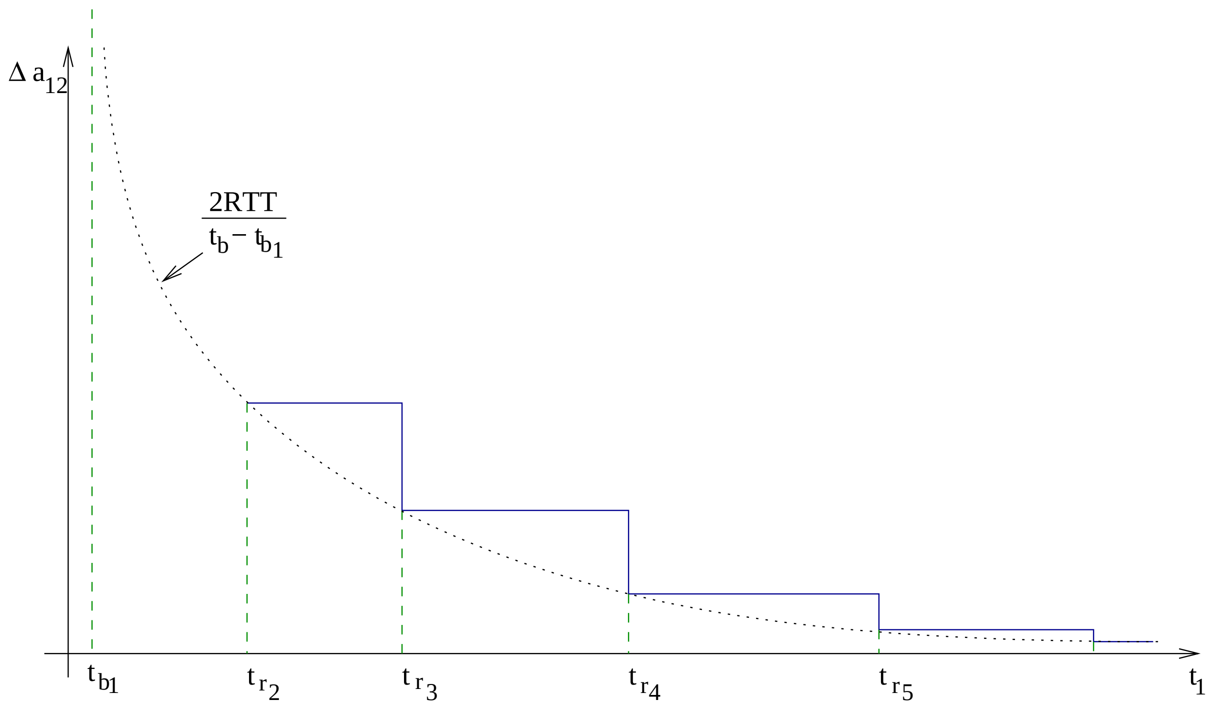


Figure 3.6: Evolution of the uncertainty bound on the relative clock drifts Δa_{12} as new data-points are received.

Theoretically, the precision on the relative drift for the two clocks Δa_{12} can be made arbitrarily small. In practice, assumption (3.1) only holds for limited time T_0, and, thus, the achievable precision is on the order of $\frac{2RTT}{T_0}$.

Equation (3.24) implies that the precision on the relative offset Δb_{12} is limited by the round-trip time RTT:

$$\lim_{\Delta t_b \to \infty} \Delta b_{12} = RTT. \tag{3.26}$$

In practice, if assumption (3.16) is eliminated, Δb_{12} is slightly higher than the minimum

RTT. If MAC layer timestamping is possible and/or the minimum deterministic RTT delay is known (as described in Section 3.3.1), then the value of the RTT in (3.26) reduces accordingly (to account only for the non-deterministic component of the delay). In turn, this corresponds to an increase in the precision of the approach.

The second implication of (3.24) is that the performance of the algorithm depends on the choice of origin: if t_{b_1} is large, the precision of the offset will initially be poor (it will improve as $\Delta a_{12} \to 0$). To avoid the initial loss in precision, the origin can be shifted:

$$t_{b_1} = 0. \tag{3.27}$$

This shift can be easily achieved by subtracting t_{b_1} from every component of all data-points and keeping the rest of the algorithms unchanged.

The goal of the synchronization is to be able to estimate the clock of the remote machine t_2 given the local clock t_1 after the two clocks have been synchronized. In Fig. 3.5, node 1 estimates the value of node 2's clock t_{2_m} at the same real time that node 1's clock is reading t_{1_m}.

Using (3.2) and writing t_{2_m} as:

$$t_{2_m} = \widehat{t_{2_m}} \pm \frac{\Delta t_{2_m}}{2}, \tag{3.28}$$

we can find:

$$\Delta t_{2_m}(t_{1_m}) = t_{1_m} \frac{\Delta a_{12}}{\overline{a_{12}}\,\underline{a_{12}}} + \frac{\underline{a_{12} b_{12}} - \overline{a_{12}}\,\overline{b_{12}}}{\overline{a_{12}}\,\underline{a_{12}}}. \tag{3.29}$$

Using (3.19)-(3.22), (3.29) becomes:

$$\Delta t_{2_m}(t_{1_m}) = \frac{\Delta a_{12}(t_{1_m} - t_{o_1})}{\overline{a_{12}}\,\underline{a_{12}}} + \frac{RTT}{\underline{a_{12}}}. \tag{3.30}$$

Fig. 3.7 depicts the qualitative evolution of the uncertainty bound Δt_{2_m} as more data-points are received, (i.e. a sketch of Δt_{2_m} as a function of t_{1_m} as in (3.30)).

Equation (3.30) implies that the uncertainty in computing t_{2_m} increases linearly with the increase of t_{1_m}. Intuitively, this can be observed in Fig. 3.5: the later the moment t_{1_m} is chosen, the larger Δt_{2_m} gets (i.e., the precision degrades). Since the uncertainty of the relative clock drift Δa_{12} decreases inversely proportionally with Δt_b, the uncertainty in estimating t_{2_m}, Δt_{2_m} returns to an (almost) fixed base-line with each new data point:

$$\Delta t_{2_m}(t_{r_i}) = RTT\left(\frac{2\Delta t_o}{\overline{a_{12}}\,\underline{a_{12}}\Delta t_b} - \frac{1}{\underline{a_{12}}}\right) \approx const, \quad \forall i \geq 2 \tag{3.31}$$

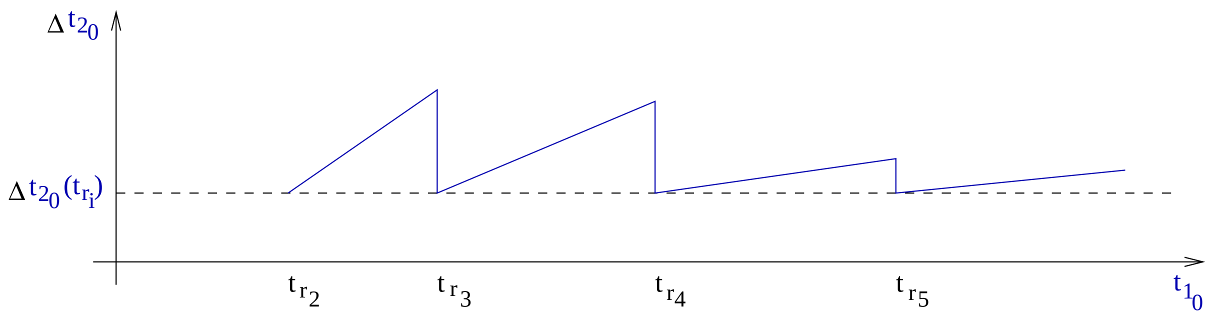


Figure 3.7: Evolution of the precision of Δt_{2_m} as more data-points are received.

(because $\frac{\Delta t_o}{\Delta t_b} \approx 1,\ \underline{a_{12}} \approx 1,\ \overline{a_{12}} \approx 1$).

After each new data-point, the precision on the relative clock drift improves (i.e., Δa_{12} decreases), and, therefore, the slope of the increase of Δt_{2_m} decreases.

Enforcing the Linear Clock Drift Assumption

The assumption of perfectly linear clock drifts (3.1) only holds for limited time, as a number of factors (temperature, humidity, power source voltage, etc.) may change the oscillator's frequency over time. Thus, it is important to *detect* when assumption (3.1) no longer holds since tiny-sync assumes linear drifts.

The analysis presented in this section can be used to determine when the linear drift assumption (3.1) no longer holds: the expected bound on the relative drift Δa_{12} should decrease as $\frac{2RTT}{\Delta t_b}$ (according to (3.23)). If the linearity assumption on the clock drifts does not hold, Δa_{12} will decrease faster than $\frac{2RTT}{\Delta t_b}$ (and, eventually, if nothing is made to correct the situation, it will become negative as $\overline{a_{12}}$ and $\underline{a_{12}}$ intersect).

The reason for this decrease in Δa_{12} can be understood if one imagines the set of constraints imposed by all data points as a flexible pipe and the two lines that determine $\overline{a_{12}}$, $\underline{a_{12}}$, $\overline{b_{12}}$ and $\underline{b_{12}}$ as two thin rods inside the pipe, spanning the diagonals of the pipe. If the flexible pipe bends (i.e., the clock drift changes), the two rods will get closer to a parallel position, corresponding to a reduction in Δa_{12} (with respect to Δa_{12} for the perfectly straight pipe).

Thus, in practical terms, in the algorithm, after each probe, the computed Δa_{12} is compared with $\frac{2RTT}{\Delta t_b}$, where RTT is the minimum RTT of the stored data-points, and Δt_b is the difference between the t_b timestamps of the stored data-points. If

$$\Delta a_{12} < \frac{2RTT}{\Delta t_b}, \tag{3.32}$$

the oldest constraint is discarded and a new constraint is acquired. In essence, the algorithm restarts upon detection of the failure of assumption (3.1).

3.3.4 Synchronizing an Entire Network

Extending a *pair-wise* synchronization scheme to global network synchronization is relatively straightforward and has been previously explored [45, 47]. The simplest approach is to construct a tree using a leader election algorithm [47] and iteratively perform pair-wise synchronization between each parent node and its children. In this section, we will only present the bounds on the performance of multi-hop synchronization.

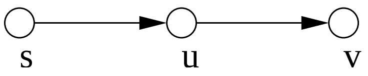


Figure 3.8: Synchronization transitivity: if s is synchronized with u, and u is synchronized with v, then s is synchronized with v.

Consider the situation depicted in Fig. 3.8 where node s synchronizes with node u that, in turn, synchronizes with node v. Node s is able to determine the bounds

$$\underline{a_{su}} \leq \quad a_{su} \quad \leq \overline{a_{su}}, \quad \text{and} \tag{3.33}$$

$$\underline{b_{su}} \leq \quad b_{su} \quad \leq \overline{b_{su}}, \tag{3.34}$$

and node u is able to determine the bounds

$$\underline{a_{uv}} \leq \quad a_{uv} \quad \leq \overline{a_{uv}}, \quad \text{and} \tag{3.35}$$

$$\underline{b_{uv}} \leq \quad b_{uv} \quad \leq \overline{b_{uv}}. \tag{3.36}$$

If node u sends its bounds $\underline{a_{uv}}$, $\overline{a_{uv}}$, $\underline{b_{uv}}$, and $\overline{b_{uv}}$ to node s, then s can compute the bounds

$$\underline{a_{sv}} \leq \quad a_{sv} \quad \leq \overline{a_{sv}}, \quad \text{and} \tag{3.37}$$

$$\underline{b_{sv}} \leq \quad b_{sv} \quad \leq \overline{b_{sv}}, \tag{3.38}$$

where

$$\begin{aligned}
\underline{a_{sv}} &= \underline{a_{su}}\ \underline{a_{uv}}, && (3.39)\\
\overline{a_{sv}} &= \overline{a_{su}}\ \overline{a_{uv}}, && (3.40)\\
\underline{b_{sv}} &= \min\left\{\underline{a_{su}}\ \underline{b_{uv}} + \underline{b_{su}}, \overline{a_{su}}\underline{b_{uv}} + \underline{b_{su}}\right\}, \quad \text{and} && (3.41)\\
\overline{b_{sv}} &= \max\left\{\overline{a_{su}}\overline{b_{uv}} + \overline{b_{su}}, \underline{a_{su}}\overline{b_{uv}} + \overline{b_{su}}\right\}. && (3.42)
\end{aligned}$$

In general, for k nodes in a chain it can be shown that:

$$t_1 = \prod_{i=1}^{k-1} a_{i(i+1)} t_k + \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \tag{3.43}$$

The corresponding bounds are:

$$\begin{aligned}
\underline{a_{1k}} &= \prod_{i=1}^{k-1} \underline{a_{i(i+1)}}, && (3.44)\\
\overline{a_{1k}} &= \prod_{i=1}^{k-1} \overline{a_{i(i+1)}}, && (3.45)\\
\underline{b_{1k}} &= \min_{\substack{a_{i(i+1)} \in \{\underline{a_{i(i+1)}}, \overline{a_{i(i+1)}}\} \\ b_{i(i+1)} \in \{\underline{b_{i(i+1)}}, \overline{b_{i(i+1)}}\}}} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \right\}, \quad \text{and} && (3.46)\\
\overline{b_{1k}} &= \max_{\substack{a_{i(i+1)} \in \{\underline{a_{i(i+1)}}, \overline{a_{i(i+1)}}\} \\ b_{i(i+1)} \in \{\underline{b_{i(i+1)}}, \overline{b_{i(i+1)}}\}}} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \right\}. && (3.47)
\end{aligned}$$

Since $\underline{a_{i(i+1)}} \approx \overline{a_{i(i+1)}} \approx 1 \quad (\forall) i = 1 \ldots k-1$, from (3.46) and (3.47) it can be inferred that the precision bounds of the offset degrade linearly with the increase in the number of hops (a somewhat intuitive result).

3.4 Experimental Setup

In this section, we present the experimental environment we used to measure the performance of the proposed time synchronization protocol.

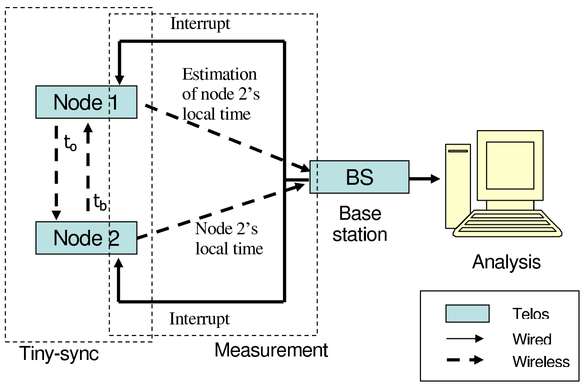


Figure 3.9: Experimental setup.

For reasons explained in Section 3.3.3, we expect tiny-sync to perform very close to the optimal mini-sync, while using fewer resources. The results presented in Section 3.5 confirm our expectation. Thus, in the performance evaluation, we focus on the performance of tiny-sync, which is more likely to be implemented in practice.

We implemented tiny-sync on the Telos platform [9], using TinyOS [46] version 1.1.12 (that allows MAC layer time-stamping and byte alignment correction, thus reducing the uncertainties in the delays). In the experiment, we used the external 32kHz crystal clock to obtain current time-stamps. We also implemented tiny-sync on the Mica2 platform [56] (using the 7.4 MHz internal clock).

Sensor Nodes Configuration

Consistent with the goal of the proposed synchronization protocols, the experimental setup is primarily designed to test the performance of pair-wise synchronization.

Three Telos motes were used to implement the algorithms and measure the synchronization errors. Figure 3.9 depicts the experimental setup: nodes 1 and 2 run the tiny-sync algorithm. In our experiment, node 1 estimates the time at node 2. The third mote (the base station (BS)) is responsible for querying and collecting clock readings. The base station generates an interrupt periodically (with a period of four seconds). The interrupt is *simultaneously* transmitted to nodes 1 and 2 (through two wires). Upon receiving the interrupt, node 2 sends its local time, and node 1 sends its estimation of the time at node 2 to the base station. After collecting time readings from two nodes, the BS sends them to the host machine for analysis.

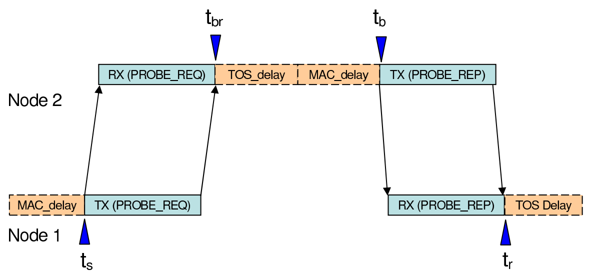


Figure 3.10: Time-stamping in tiny-sync.

Tiny-sync Implementation

Figure 3.10 illustrates the moments we record the four time-stamps in Section 3.3.1. We used the hooks provided by TinyOS to time-stamp the packet just before the first byte of the packet is transmitted and immediately after the last byte of the packet is received. Thus, all of the non-deterministic delays (RTT in Section 3.3.3) are due to the transceiver and scheduling of TinyOS. Our measurements showed that the one-way delay is approximately 1.419 ms (2.838 ms round-trip time).

Synchronization Errors Measurements

We define the time synchronization error as the time difference between the real clock reading of a sensor node and the estimated clock value by the other sensor node. There are two distinct sources of errors when measuring the synchronization error:

- the synchronization algorithm and
- the measurement procedure itself.

As previously mentioned, in the experimental setup, the base station triggers simultaneous interrupts at the two nodes running tiny-sync. This measurement method eliminates the radio transceiver delay uncertainties from the measurement process. However, it introduces other sources of non-deterministic delays that may result in erroneous measurements. Figure 3.11 shows the possible sources of delays affecting the measurements. When an interrupt occurs, unless interrupts are suspended, the processor first completes the current executing instruction, saves the program counter (PC) and the status register (SR), then jumps to the corresponding interrupt service routine (ISR). In our case, the ISR reads the current clock. Unfortunately, if the processor is in the middle of a non-interruptible operation at the time the interrupt arrives (i.e., the interrupts were suspended), or there are other pending interrupts, the measurement interrupt will be delayed for an unknown time (in our experiments, we found it to be up to 61 μs).

There can be several ways to define the measurement error. Assume we measure the synchronization error for each interrupt j, where $j = 1, 2, 3, ...$. We define the measurement error E_j of the synchronization error between node n_1 and n_2 at measurement j as

$$E_j = x_{n_1,j} - x_{n_1,j-1} - \widehat{a_{12}}(x_{n_2,j} - x_{n_2,j-1}), \tag{3.48}$$

where node n_1 estimates the clock of node n_2, and $x_{n_i,j}$ is the time-stamp of node n_i at measurement j, and $\widehat{a_{12}}$ is defined in (3.9). That is, the measurement error is the inter-measurement time difference between participating nodes. The measurement errors can be reduced by discarding the data-points whose measurement errors are greater than a threshold value τ. In this experiment, we set the parameter τ to $33\mu s$ (i.e., we discarded all measurements with an error higher than $33\mu s$). We observed that 94.42% of the measurements have their errors of smaller than the threshold value.

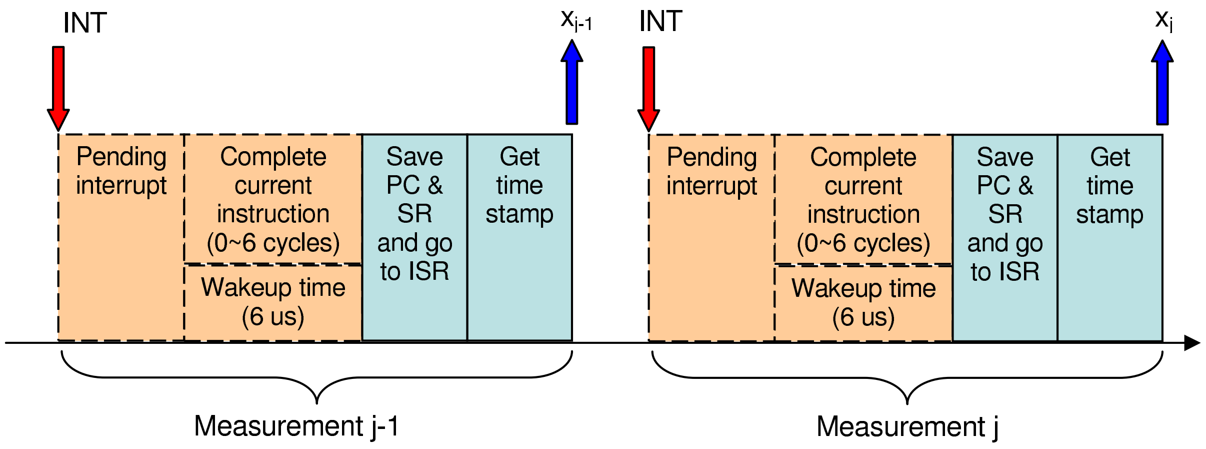


Figure 3.11: Sources of delay in the measurement process error.

Global Synchronization

In order to evaluate global synchronization errors, we added three more Telos motes to the existing two, thus forming a chain of five nodes. Each node $i + 1$ estimates the clock of its parent (node i). The query node periodically generates a measurement interrupt. Upon the receipt of each interrupt, each node i reports the value of its local clock as well as its estimate of the clock at node $i-1$ (obviously, with the exception of node 1 that only sends its local clock). The multihop configuration was enforced by static routing.

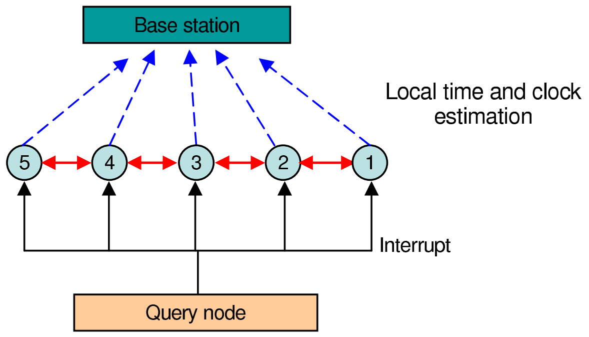


Figure 3.12: Experimental setup for global synchronization.

3.5 Experimental Results

In this section, we present the results of the experiments. We focused on the performance of tiny-sync as well as on comparing its performance with mini-sync and other existing time synchronization protocols.

Time Synchronization Error of Tiny-sync

Using the setup described in Section 3.4, we ran the first experiment for 2.1 hours. Figure 3.13 shows the bound on the relative drift of the two clocks Δa_{12} and the theoretical bound $\frac{2RTT}{\Delta t_b}$ in (3.23). It is clear that the relation (3.23) holds exceedingly well, implying that the assumption of linear drift (3.1) holds within the bounds of our accuracy for this first experiment. In Section 3.5, we present the results of another experiment where this assumption is purposely broken.

Figure 3.14 shows the synchronization error of tiny-sync as a function of time. The average synchronization error achieved by tiny-sync is 12.075 μs. The synchronization error is smaller than the quantization error: for the 32 kHz external clock oscillator, each unit of time is equal to 31.21 μs.

The algorithm converges quickly to the nominal precision: it achieves the average precision with the first few probes. Considering that the entire network can be synchronized as fast as a pair of

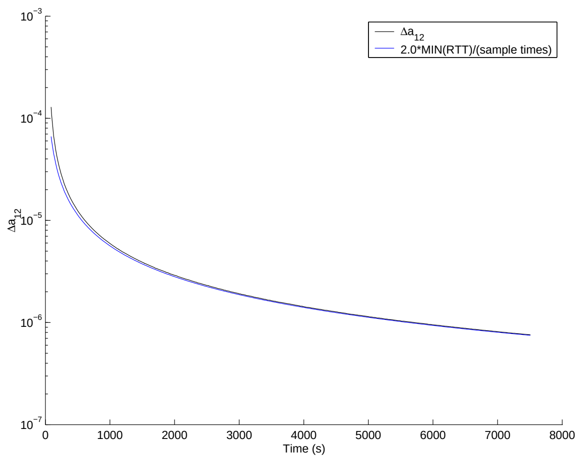


Figure 3.13: The evolution of the bound on the relative drift Δa_{12} and the predicted evolution (3.23) for the first experiment.

nodes (Section 3.3.4), only a few seconds are necessary to synchronize an entire network.

Figure 3.15 shows the distribution of the synchronization errors. The graph indicates that most (about 99.48%) of the time synchronization errors are smaller than 30 μs. Thus, tiny-sync is suitable for applications that require good accuracy and consistency.

Figure 3.16 shows the synchronization errors of tiny-sync algorithm as a function of the number of hops as well as the synchronization interval. The synchronization interval is the time between two consecutive probe messages. The synchronization error increases both with the number of hops between two nodes as well as with the synchronization interval.

Comparison with Other Approaches

In this section, we compare the performance of tiny-sync with two other time synchronization approaches for WSN: TPSN [45] and FTSP [47] as well as a simple linear fit on the collected data. In order to make the comparison as fair as possible, we did not filter out the measurement errors for any synchronization protocol (i.e., all measurement data were included in comparison).

TPSN uses a similar probing method as tiny-sync to estimate the clock of the target sensor node. It also exploits the MAC layer time-stamping to reduce the timing uncertainties. Thus, we used the same data collection to determine the time synchronization error of tiny-sync and TPSN. For FTSP, we downloaded the latest version of FTSP source code from [57] and implemented it on the same Telos

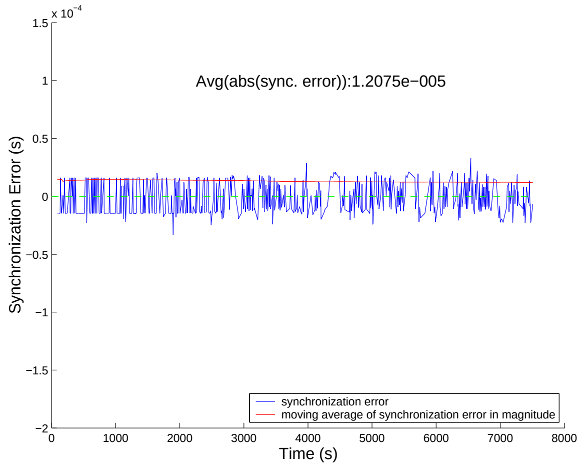


Figure 3.14: Tiny-sync pairwise synchronization error.

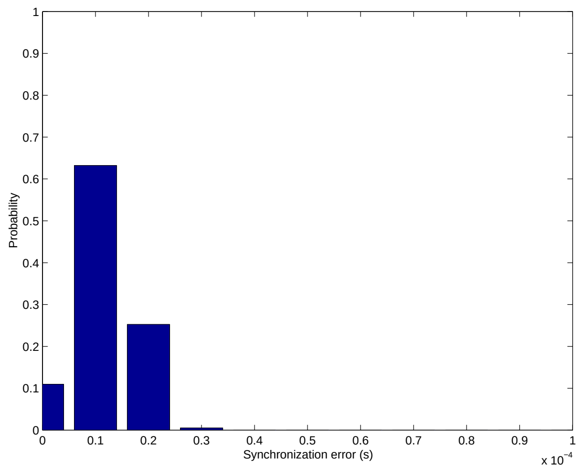


Figure 3.15: Distribution of the synchronization errors.

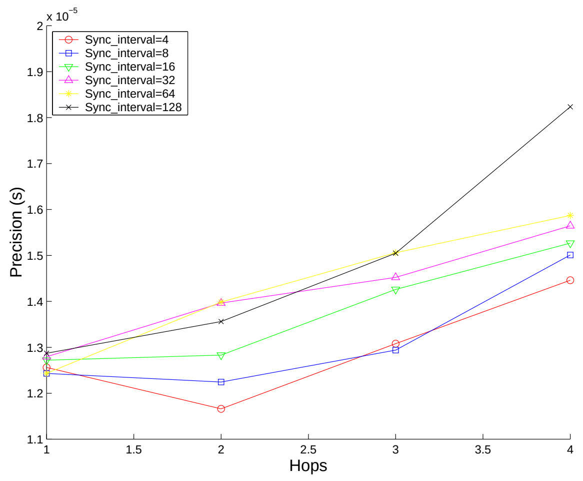


Figure 3.16: Tiny-sync global synchronization error.

motes. We used a test program provided by FTSP to evaluate time synchronization errors of the sensor nodes running FTSP.

For the linear fit we used least square to minimize the error between a linear fit and the data points (both (t_o, t_b) and (t_b, t_r)). We also tested the performance of a one-way linear fit (that only considers forward probes (t_o, t_b)) and the results were nearly identical. However, while a one-way linear fit only requires one probe packet, it also requires hardware-dependent calibration. Thus, in what follows, we present the results of the two-way fit (again, practically identical with the one-way fit), as the two way fit match the properties of the proposed scheme (in terms of need for calibration).

Since the results of the linear fit are sensitive to the number of data points considered, we plotted synchronization error of the linear fit as a function of the number of data points considered for the fit. The results are shown in Fig. 3.17. The graph shows that the linear fit has a an optimum number of points - both a lower and a higher number of points lead to a decrease in accuracy. Intuitively, if only a small number of points are used, the accuracy of the linear fit is limited by the quality of the data provided by these points. However, if the drift is non-linear, as the number of data-points considered for the linear fit increases, the fit necessarily becomes an increasingly poor approximation of the current relationship between the two clocks (as it minimizes the error to *all* considered points). In the following comparisons we use both 128 points that optimizes the performance of the fit for our experimental setup as well as a 2 point fit that has a comparable computational and storage complexity with tiny-sync.

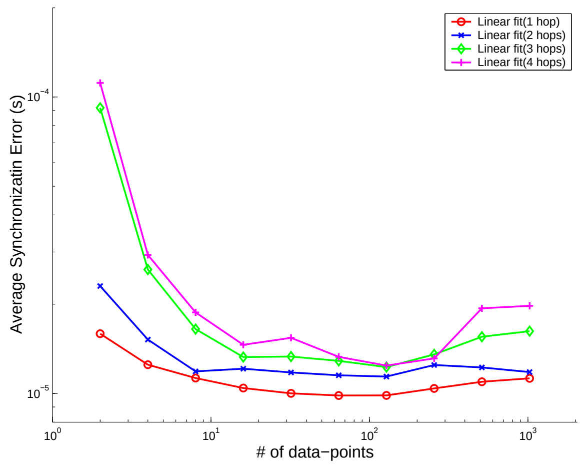


Figure 3.17: Synchronization errors as a function of the number of points considered for the linear fit and tiny-sync.

Figure 3.18 shows the comparison of the synchronization errors as a function of the number of hops between sensor nodes. When the synchronization interval is 4 s, tiny-sync and TPSN show similar results. However, when the synchronization interval is 128 s, tiny-sync clearly outperforms TPSN and FTSP. Furthermore, the synchronization error of FTSP increases faster than tiny-sync and TPSN with the number of hops. The linear fit that uses 128 points (optimal for this data-set) at four second synchronization interval, slightly outperforms tiny-sync for all number of hops. This slight improvement is attained at a price of higher computational and storage resources as well as at a loss of generality: the linear fit has to be optimized for the particular degree of the non-linearity of the clock drift. When only two points are used for the linear fit or when the synchronization interval is higher (128 seconds), tiny-sync outperforms the linear fit.

Figure 3.19 shows the comparison of the synchronization errors as a function of the synchronization intervals. Tiny-sync outperforms TPSN and FTSP in terms of synchronization error, and it shows reduced variation with the increase in the synchronization interval. Synchronization error of TPSN increases faster than others as the synchronization interval increases because TPSN does not compensate for the clock drift. The synchronization error of FTSP increases slowly as the number of hops increases, as FTSP uses the linear regression to compensate for the drift. The accuracy of the linear fit decreases slightly with the increase in the synchronization interval, as the number of points is also correspondingly reduced. The 128 points linear fit outperforms all synchronization algorithms for

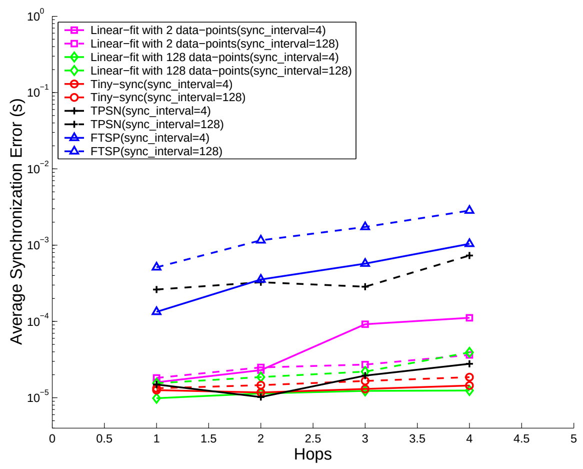


Figure 3.18: Comparison of synchronization errors as a function of the number of hops.

a single hop, and a small synchronization interval.

Robustness Evaluation

In this section, we present results of the experiment that tested the robustness of tiny-sync with respect to large variations in the relative drift of the clocks. Essentially, we break the assumption (3.1) and study its effect on the precision of the algorithm. To induce a variation in the drift, we drastically changed the temperature of node 2, while keeping node 1 at a constant room temperature. One hour into the experiment, we placed node 2 into a box filled with ice and closed the lid. One hour later, we removed the node from the ice box and continued the experiment for a total duration of 4.48 hours.

The change in temperature, as expected, negatively affected the bounds on the relative drift of the clocks (Figure 3.20(a)). We used the procedure detailed in Section 3.3.3 to detect the change in the drift of the clocks and restart the algorithm. Figure 3.20(b) shows the moments when the algorithm detects the change in the relative drift of the clocks and restarts.

Figure 3.21 shows the synchronization error for the second experiment. Despite the significant change in the relative drift while node 2 was in the ice box, the average synchronization error during that period did not change. The average precision of this experiment (10.78 μs) was actually better than that of the initial experiment (12.07 μs in Fig. 3.14). This experiment validates that the restart detection algorithm shows that tiny-sync is robust with respect to changes in the clock drifts.

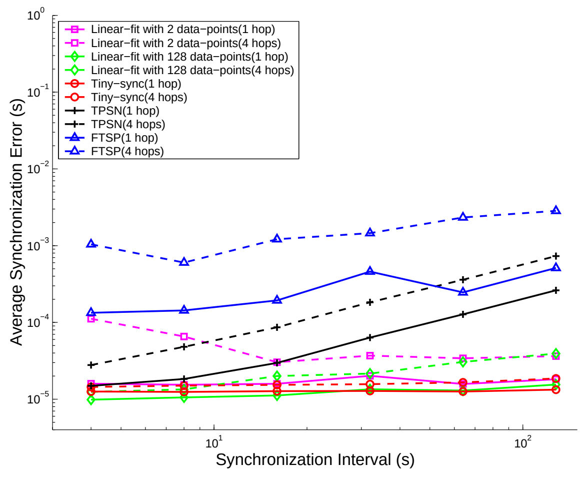
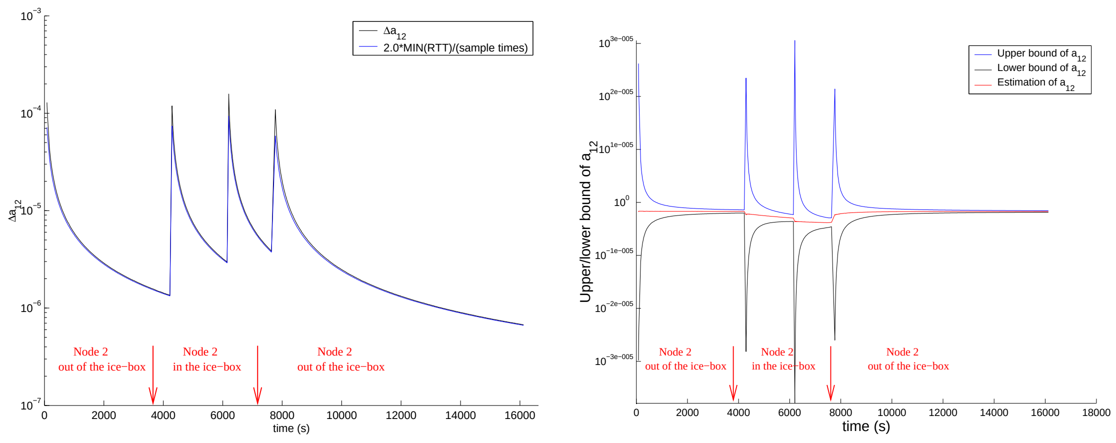


Figure 3.19: Comparison of synchronization errors as a function of the synchronization interval.



(a) (b)

Figure 3.20: Behavior of tiny-sync when the clocks are forced to change their relative clock drift by sudden changes in the temperature of one of them (by temporary placing it in an ice-box). During the cool-down and warm-up period, tiny-sync restarts several times. (a) the evolution of the bound on the relative drift Δa_{12} and the predicted evolution (3.23); (b) the evolution of the bounds on the drift.

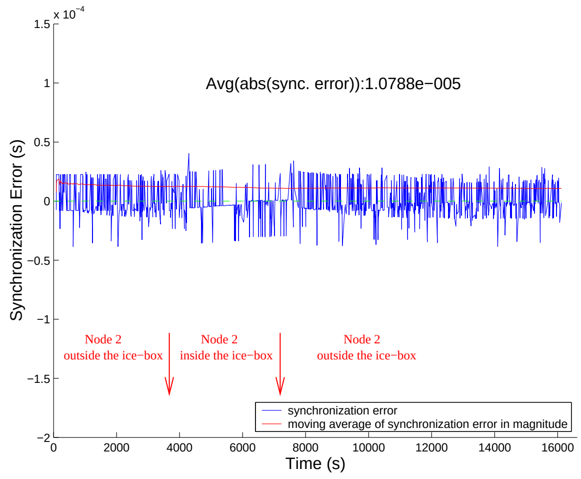


Figure 3.21: The synchronization error when nodes were subjected to variation of the relative drift.

Figure 3.22 shows the results of the robustness of tiny-sync with respect to packet loss. In this experiment, we simulated the environment in which synchronization packets experience random loss. Using the data collected from the first experiment, we varied the packet loss rate (for both the probe request and reply) from 0 to 90%. Except for cases with large packet loss rate and large synchronization intervals, the precision was not significantly affected.

Comparison of Tiny-sync and Mini-sync

In this section, we compare the performance of tiny-sync and mini-sync, which is optimal (Section 3.3.2). Figures 3.23 and 3.24 show the synchronization errors of tiny-sync and mini-sync as a function of the number of hops and synchronization interval, respectively. As expected, there is no significant difference in performance between tiny-sync and mini-sync. Given the fact that tiny-sync requires less resources (computational and storage) than mini-sync, this result suggests that using tiny-sync is preferred to mini-sync.

Although one would expect mini-sync to always outperform tiny-sync, the results show that, in practice, sometimes tiny-sync can show better precision than mini-sync. It turns out that the selection of the first data-points to be used in the drift estimation can affect the overall performance of the algorithm. Details on this effect are presented in Section 3.5.

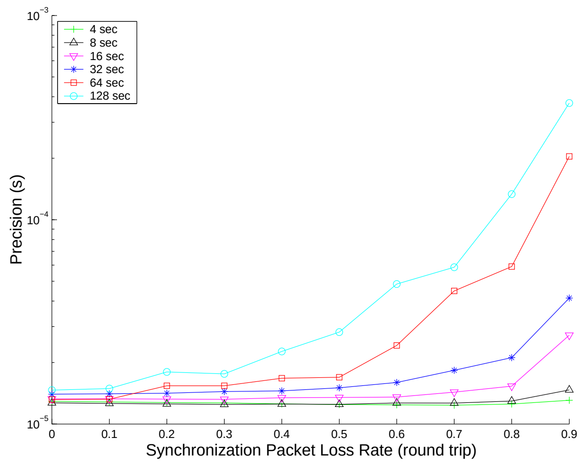


Figure 3.22: The synchronization error with packet loss.

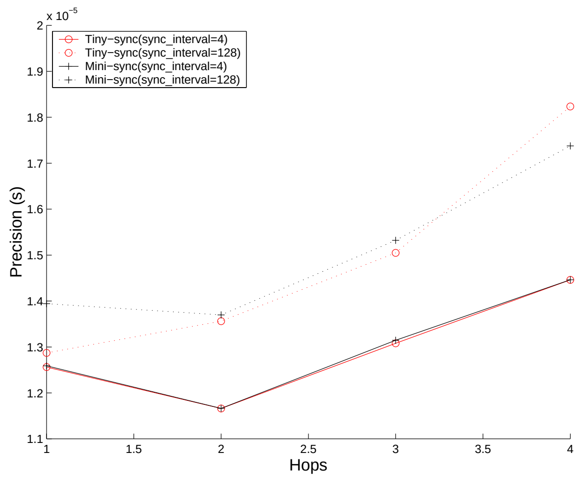


Figure 3.23: Synchronization error of tiny-sync and mini-sync as a function of the number of hops.

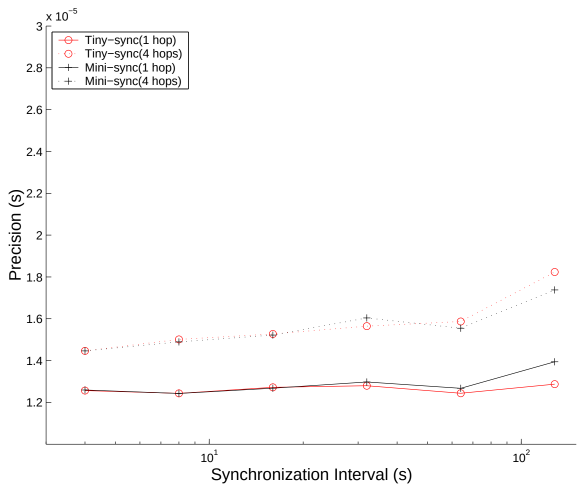


Figure 3.24: Synchronization error of tiny-sync and mini-sync as a function of the synchronization interval.

Experimental Results on Mica2 Platform

We also implemented tiny-sync on the Mica2 platform [56] using TinyOS. In this experiment, we used the far more accurate 7.4 MHz internal clock to obtain time-stamps. The average pair-wise synchronization error achieved by tiny-sync on Mica2 was 1.3868 μs. According to published data [45, 47], the pair-wise accuracies of RBS, TPSN and FTSP on the Mica platform are 7 μs, 16.9 μs and 1.48 μs, respectively. The seemingly poor result for TPSN is for a single time-stamp exchange on a Mica 1 platform (that has a slower clock and radio transceiver). We expect that an averaging implementation on the Mica2 platform would result in a precision similar to FTSP and tiny-sync. The accuracy of tiny-sync is very close to the accuracy of FTSP (tiny-sync is slightly better). It is likely that both approaches reached the lower bound of achievable accuracy on the Mica2 platform (due to non-deterministic delays, most likely introduced by the transceiver and/or clock inaccuracies).

Figure 3.25 shows the pair-wise synchronization errors for tiny-sync using Telos and Mica2 motes. In this experiment, we varied the synchronization interval from 4 s to 1024 s. When the synchronization interval is short, the synchronization error on the Mica2 platform is better than that on Telos due to the clock resolution on the Mica2 platform; however, the external crystal clock of the Telos platform is more stable than the internal clock of Mica2 in the long run. Thus, for short synchronization intervals, the synchronization errors on Mica2 are smaller than on the Telos motes; however, the situation is

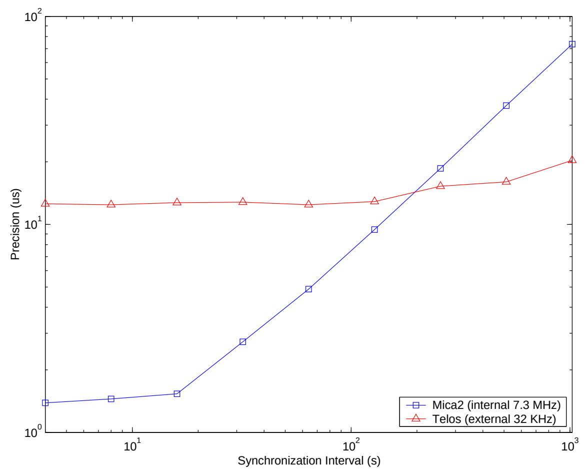


Figure 3.25: Synchronization error of tiny-sync on Mica2 and Telos platforms.

reversed when the synchronization interval is longer than 128 s.

Tiny-sync Initialization

As mentioned in Section 3.5, the selection of data-points in the beginning of the algorithm may affect the synchronization precision in the initial phase (especially for long synchronization intervals).

Figure 3.26 shows an example of a data set with high synchronization error for the beginning of the experiment. The synchronization interval is 128s. The reason for the poor performance is in the choice of the initial data-points. If the first few data samples are of poor quality (i.e., with large non-deterministic delays), the initial precision will suffer (until at least two reasonably "good" data-points are received). A simple method of forcing tiny-sync to start with "good" data-points is to start with the best two data-points out of the first N. Figure 3.27 shows the synchronization error when N is 10.

3.6 Conclusion

In this chapter, we proposed and evaluated the performance of two simple time synchronization algorithms suitable for wireless sensor networks. The algorithms perform pair-wise synchronization and can be used as the basic building block for synchronizing an entire network. While the performance of the two algorithms may vary in theory, in practice, they yield very similar results. Thus, the simplest one, tiny-sync, is likely to be practically implemented. The main advantage of tiny-sync is its simplic-

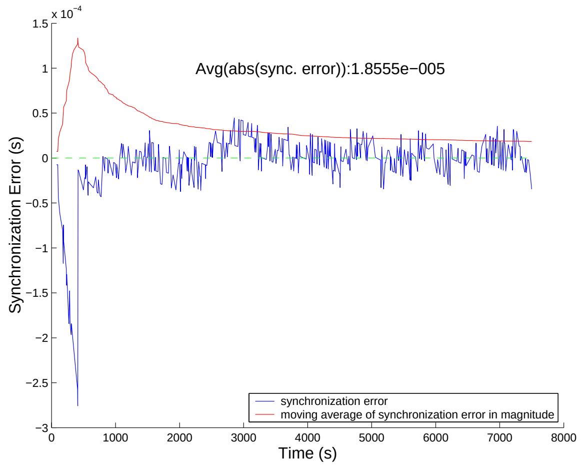


Figure 3.26: Synchronization errors of tiny-sync without poor initial data.

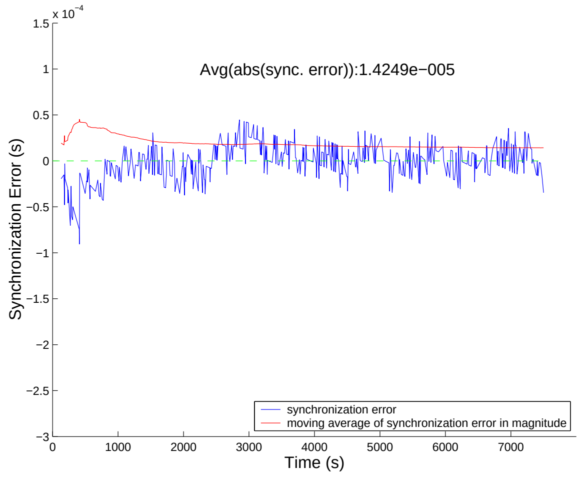


Figure 3.27: Synchronization errors of tiny-sync when the best two out of the first ten data-points are used to initialize the algorithm.

ity and parsimonious resources requirements. Other advantages include robustness to large variations in clock drift and its ability to achieve fast synchronization of an entire network. Experimental results show that it performs as well or better than existing time synchronization approaches for wireless sensor networks.

Chapter 4

Analyzing Power Consumption of Wireless Sensor Networks MAC protocols

4.1 Introduction

With the development of micro electronic mechanical system (MEMS), wireless networking, and embedded microprocessor, wireless sensor networks (WSNs) have been deployed in environmental, military, and commercial areas [2–4]. WSNs consists of thousands of small nodes capable of sensing, processing, and communicating. In many sensor network applications, each node periodically samples its sensors and transmits the data (in multihop fashion) to the base station, where it will be further delivered to the end users through long-haul links (e.g., the Internet).

An important characteristics of WSNs is that the nodes are equipped with batteries with limited capacity and, often, changing the batteries is either expensive or impossible. Therefore, power management plays a very important role in increasing the lifetime of the sensor networks [8]. For example, when a Telos mote equipped with 2250mAh batteries idles without any power saving algorithm, the mote can operate for up to 52 days. With a 1% duty cycle, the lifetime of the node can be extended to 4076 days [9].

The power management in WSNs has been extensively studied at each layer of the networking stack (e.g., power saving medium access control (MAC) protocols, power aware routing algorithms, topology control, deployment strategies, etc.) In this paper, we study the power efficiency of the MAC layer of WSNs. A significant (in some nodes the largest) source of power consumption in sensor nodes is the radio transceiver. There are several sources of power wastage in the MAC layer, such as collision, overhearing, protocol overhead, and idle listening. Among those, for many applications, idle listening is often the major source or energy inefficiency. In many MAC protocols for WSNs nodes save energy by going to sleep if they do not have data to sense, receive, or transmit. When sensor nodes sleep, they turn off the microcontroller and the radio. In this paper, we determine analytically the power consumption of several well-known MAC protocols for WSNs by calculating how much energy is consumed in each mode of radio operations. Some of existing WSN MAC protocols provide the analysis of power consumption. However, the existing results are based on the protocol specific assumptions (e.g., piggybacking). Moreover, most existing models are based on a single-hop network model. The power consumption models of this paper are based on a multi-hop environment. As much as possible, we made the same assumptions as all compared protocols. To validate the analysis, we measure energy consumptions of the MAC protocols on a testbed using the Mica 2 motes and compare the results.

4.2 Analysis of power consumption

In this section, we determine power consumptions of several common MAC protocols for WSNs including S-MAC [21, 58], T-MAC [59], B-MAC [60], X-MAC [1], and SCP-MAC [61]. For the comparison, we also consider the power consumptions of 802.11 [62] both with and without power saving in ad-hoc mode as well as that of an ideal MAC protocol in terms of power consumption.

We classify energy efficient MAC protocols for WSNs into two groups; synchronization-based and preamble-based.

- In synchronization-based MAC protocols, the sampling period is divided into fixed size *frames*. Sensor nodes wake up and go to sleep once during each frame. To send and receive frames, all nodes in a neighborhood wake up and go to sleep at the same time. Therefore, synchronization-based MAC protocols provide (and use) synchronization mechanisms. Synchronization-based MAC protocols include 802.11 power saving mode, S-MAC, T-MAC, and SCP-MAC.

- In preamble-based MAC protocols, the nodes are not synchronized with each other. Instead, senders use a long preamble to wake up receivers. In these protocols the receiver sleeps for a long time and wakes up for a very short time to check for the existence of a preamble. If the receiver does not detect any preamble, it goes back to sleep immediately. If it detects the preamble, the node goes back to sleep after performing the protocol specific radio operations. We define a *frame* in the preamble-based protocols as the time combining the sleep time and the time to check the preamble. Unlike in synchronized-based protocols, the number of frames during each sampling period in preamble-based protocols varies depending on the traffic load during the sampling period. Preamble-based protocols include B-MAC and X-MAC.

4.2.1 Assumptions

Following are the assumptions used to derive the models for the power consumption of MAC protocols:

- each sensor node takes one sample and sends it to the base station during each sampling period T (i.e., we assume a continuous monitoring application);
- all nodes follow the same power saving schedule. We assume that for all MAC protocols that require it, time synchronization is provided by exchanging the beacons or $SYNC$ messages;
- there are no transmission errors;
- the network has sufficient capacity to transport the data;
- there are no collisions. This assumption is effectively a reasonable approximation of reality in systems with very low duty cycles and low contention;
- the power consumption during the contention period is the same for all MAC protocols. Thus we omit the power consumption during the contention period;
- all traffic is forwarded to a single base station through a shortest path routing *tree*.

4.2.2 Sources of Power Consumptions

In this section we present all power consumption sources used in our models.

- **Collisions** When two or more packets arrive at the receiver at overlapping times, those packets collide with each other. In this case, the senders have to retransmit the packets. This increases the number of packet transmissions and the energy consumption. MAC protocols provide different methods to reduce the collisions such as collision avoidance backoffs and RTS/CTS schemes. The MAC protocols we analyze use similar methods to reduce collisions. Thus, we assume that the same amount of energy due to collisions is used for all MAC protocols.

- **Transmissions/Receptions** Radio transmissions and receptions are necessary and unavoidable for the actual data transfer.

- **Overhearing** This source of power consumption is rooted in the broadcast characteristics of wireless communication. That is, even if a packet is destined to a specific receiver, all nodes within the transmission range of the sender can hear the packet. Thus, the neighboring nodes that are not the intended receiver still receive and process the packet before discarding it. The amount of overhearing increases with the density of the nodes.

- **Idle listening** Idle listening occurs when nodes wait to receive packets by listening idly to the channel. Measurements show that for most transceivers the energy needed to listen to the wireless channel is almost as high as that for packet receptions [25]. In many applications sensor nodes are in idle listening state for long period of time. Thus, idle listening is the dominant component of power consumption of MAC protocols in WSNs.

- **Sensing** The power consumption for sensing the environment differs from application to application. In some applications, it may be a large percentage of the total power consumption. However, this component is not related to the MAC protocol, and therefore we do not consider it in our analysis of MAC protocols.

- **Sleep** When nodes go to sleep, the radio is turned off. Furthermore, we assume that the CPU is also at stand-by mode to minimize the power consumption when it is in sleep state.

4.2.3 Notation

In this section we present the notation we use for the analysis. We denote the variables starting with t to represent the time for a specific operation or a time period and the variables starting with T to represent the total time of a operation during one sampling period.

T The sampling period;

t_f The frame size. The frame is a fixed time period, which is used in synchronization-based MAC protocols such as 802.11, S-MAC, and T-MAC. Within the frame, nodes stay awake at least a portion of the frame to send/receive data, or go to sleep to save the energy according to the schedule of the MAC protocol;

K The number of frames that can be sent during one sampling period;

$$T = t_f K. \tag{4.1}$$

t_{listen} The active listening period within the frame when each node receives and/or transmits data;

t_{sleep} The sleeping period within the frame where each node goes to sleep if it does not have data to transmit (or receive):

$$t_f = t_{listen} + t_{sleep}; \tag{4.2}$$

t_{beacon} **(**t_{sync}**)** The beacon interval (t_{beacon}) of 802.11 protocols and the synchronization interval (t_{sync}) of MAC protocols that require the time synchronization. For fairness we assume $t_{beacon} = t_{sync}$;

t_X The time for sending/receiving a packet of type X, where X is RTS, CTS, $DATA$, ACK_{DATA}, P (for preamble), ACK_P (for ACK of the preamble), B (for BEACON), or $SYNC$;

t_{WU} The time for radio transition from sleep to a fully awake state;

t_A The timeout interval for T-MAC;

t_{CCA} The time for clear channel assessment in B-MAC and X-MAC;

T_Y The time spent for Y during the *sampling period*, where $Y \in \{tx, rx, oh, idl\_lsn, startup, slp\}$. T_{tx} is the time spent on transmitting packets during the sampling period. T_{rx} is the time spent on receiving packets during the sampling period. T_{oh} and T_{idl_lsn} are the overhearing and idle listening time during the sampling period. $T_{startup}$ is the sum of t_{WU} during the sampling period:

$$T = T_{tx} + T_{rx} + T_{oh} + T_{idl\_lsn} + T_{startup} + T_{slp}; \tag{4.3}$$

P_Y The power consumed for Y, where $X \in \{tx, rx, oh, idl\_lsn, startup, slp\}$;

E_Y The energy consumed for Y during the sampling period, where $Y \in \{tx, rx, oh, idl\_lsn, startup, slp\}$;

E The energy consumed during each sampling period:

$$E = T_{tx}P_{tx} + T_{rx}P_{rx} + T_{oh}P_{oh} + T_{idl\_lsn}P_{idl\_lsn} + T_{startup}P_{startup} + T_{slp}P_{slp}; \tag{4.4}$$

$N_{children}(i)$ The number of descendent nodes of node i in the routing tree, i.e., node i needs to forward $N_{children(i)}$ data packets during one sampling period. If node i is a leaf node in the routing tree, $N_{children(i)} = 0$;

Υ_i The set of nodes whose transmissions can be received by node i ;

$N_{neighbor}(i)$ The cardinality of Υ_i;

$N_h(i)$ The total number of $DATA$ packets that node i hears from its neighbors during one sampling period:

$$N_h(i) = \sum_{k \in \Upsilon_i} (N_{children}(k) + 1); \tag{4.5}$$

$N_{tx}(i)$ The number of packets that node i transmits during each sampling period T, $N_{tx}(i) = N_{children}(i) + 1$;

$N_{rx}(i)$ The number of packets that node i receives during each sampling period T, $N_{rx}(i) = N_{children}(i)$;

$N_{oh,X}(i)$ The number of packets of type X that node i overhears during each sampling period T, where $X \in \{DATA, ATIM, RTS, ACK_{DATA}, ACK_{ATIM}, CTS\}$:

$$N_{oh,DATA}(i) = N_h(i) - N_{rx}(i). \tag{4.6}$$

Equation (4.6) also holds for ATIM and RTS packets.

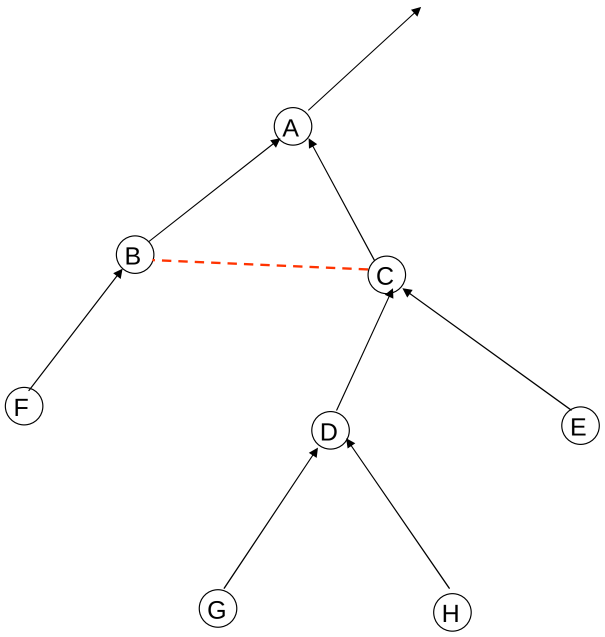


Figure 4.1: An example of a WSN topology.

Example

Let us illustrate the main variables for node C in Fig. 4.1. Solid lines in the figure represent routing paths and two nodes connected by dotted lines are within transmission range of each other (but they do not route packets to each other). Node C can hear packet transmissions from nodes A, B, D, and E even if they are not all sent through node C. Thus $N_{neighbor}(C) = 4$, $N_{children}(C) = 4$, $\Upsilon_C = \{A, B, D, E\}$, and $N_h(C) = (N_{children}(A) + 1) + (N_{children}(B) + 1) + (N_{children}(D) + 1) + (N_{children}(E) + 1) = 14$.

***Theorem* 5** *We assume a typical sensor network application where each sensor node periodically reads sensor data and transmits the data to the base station. When a sensor node receives a DATA packet, it replies with an ACK packet to the sender. Assume each node generates one packet during the sampling period and transmits the packet to the base station using the shortest hop routing. Let $N_A(i)$ be the number of ACK packets that node i hears from its neighbors. Let $N_D(i)$ be the number of DATA packets that node i hears from its neighbors. Then the following relationship is satisfied:*

$$N_A(i) = N_D(i) - N_{neighbor}(i). \tag{4.7}$$

Proof Node i hears only the packets that its neighbors send. Let j be one of node i's neighbors. Then, the number of $DATA$ packets that node i hears from node j is $N_{tx}(j)$ and the number of ACK packets that node i hears from node j is $N_{rx}(j)$. We know $N_{tx}(j) = N_{rx}(j) + 1$. That is, for each neighbor j of node i, the number of ACK packets that node i hears from node j is one less than that of $DATA$ packets. Summation for all neighbors provide (4.7). ■

***Corollary* 6** *Let $N_{oh,ACK_{DATA}}(i)$ be the number of ACK_{DATA} packets that node i overhears during each sampling period T. Let $N_h(i)$] be the total number of $DATA$ packets that node i hears from its neighbors during the sampling period. Then the following relationship is satisfied:*

$$N_{oh,ACK_{DATA}}(i) = N_h(i) - N_{neighbor}(i) - N_{rx}(i). \tag{4.8}$$

Proof Node i receives an ACK_{DATA} packet for each of its DATA packet transmission. By Theorem 5, the total number of ACK_{DATA} packets that node i hears is $N_h(i) - N_{neighbor}(i)$. Among $N_h(i) - N_{neighbor}(i)$, only $N_{tx}(i)$ are ACK packets that node i should receive. Therefore, Corollary 6 holds. ■

Note the Corollary 1 holds for ACK_{ATIM} and CTS packets.

4.2.4 Power Consumption Analysis of 802.11 basic mode (ad-hoc)

Protocol overview

For comparison purposes, we examine the power consumption of the 802.11 protocol in ad-hoc mode [62]. The initial design of this protocol assumes that all nodes are within transmission range each other. All nodes attempt to send $BEACON$ packets to synchronize with each other. Only one $BEACON$ packet is sent among all nodes in a neighborhood during a beacon interval. If a node has a packet to transmit and the medium is free for the duration of a (DIFS), the node starts sending the packet as we assume no collisions and no collision avoidance. As shown in Fig. 4.2, nodes are always in receive mode unless they are transmitting.

Power consumption analysis

Sensor nodes are always on (either transmitting, receiving, or idle listening) through the entire sampling period. Thus, T_{slp} = $T_{startup}$ = 0. The energy consumption in each sampling period is:

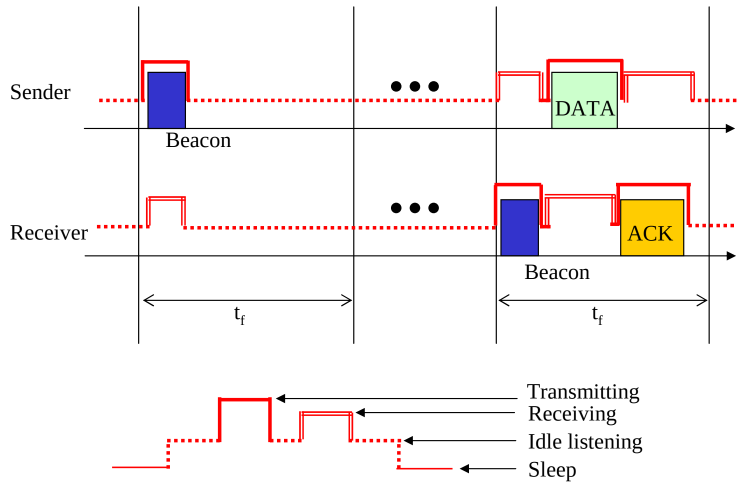


Figure 4.2: Example of data exchange in 802.11 in ad-hoc mode without power saving.

$$E = T_{tx}P_{tx} + T_{rx}P_{rx} + T_{oh}P_{oh} + T_{idl\_lsn}P_{idl\_lsn}, \tag{4.9}$$

where we omit the node index from the variables of time and energy consumption to simplify the expressions.

Both the sender and the receiver spend $t_{DATA}+t_{ACK_{DATA}}$ seconds for transmitting or receiving one $DATA$ packet. In 802.11 specification, the default $BEACON$ interval T_f is around $100ms$. However, in sensor network MAC protocols such S-MAC [58] and SCP-MAC [61] that require time synchronization, the synchronization interval is much longer than $100ms$. To make the comparison as fair as possible, the beacon interval in 802.11 is set to the same values as the synchronization interval in SCP-MAC. Thus, each sensor node, on the average, transmits $\frac{T}{t_{beacon}}\frac{1}{N_{neighbor}(i)+1}$ and receives $N_h\frac{T}{t_{beacon}}\frac{N_{neighbor}(i)}{N_{neighbor}(i)+1}$ $BEACON$ messages during one sampling period T. The energy consumptions in each sampling period for transmitting and receiving packets are given by (4.10) and (4.11), respectively:

$$E_{tx} = \left(N_{tx}(i)t_{DATA} + N_{rx}(i)t_{ACK}\right)P_{tx} + t_B\frac{T}{t_{beacon}}\frac{1}{N_{neighbor}(i)+1}P_{tx}, \tag{4.10}$$

$$\begin{aligned} E_{rx} &= \left(N_{rx}(i)t_{DATA} + N_{tx}(i)t_{ACK}\right) P_{rx} \\ &+ t_B \frac{T}{t_{beacon}} \frac{N_{neighbor}(i)}{N_{neighbor}(i)+1} P_{rx}. \end{aligned} \tag{4.11}$$

Overhearing time T_{oh} for a sensor node i is the time spent on receiving packets meant for other nodes. To compute T_{oh} we first compute the total time spent on receiving packets from its neighbors including $DATA$ and ACK_{DATA}. Then, we subtract the time spent on receiving packets for node i from the total time. Thus, using (4.6) and Corollary 1, we can compute the time a node spends on overhearing in each sampling period (4.12). The same expression applies to all other protocols.

$$T_{oh} = N_{oh,DATA}(i)(t_{DATA}) + N_{oh,ACK_{DATA}}(i)(t_{ACK_{DATA}}). \tag{4.12}$$

The idle-listening time is the difference between the sampling period T and the sum of all other times:

$$\begin{aligned} T_{idl\_lsn} &= T - (N_{tx}(i) + N_{rx}(i))(t_{DATA} + t_{ACK}) \\ &- T_{oh} - t_B \frac{T}{t_{beacon}}. \end{aligned} \tag{4.13}$$

4.2.5 Power Consumption Analysis of 802.11 PS mode (ad-hoc)

Protocol overview

The 802.11 power saving (PS) mode also assumes that all nodes are within the transmission range of each other [62]. In this mode, the beacon interval is divided into active (listen) and sleep periods. The time durations of the active and sleep periods are fixed and determined by the *duty cycle*. Nodes are awake when they are in the active period. During the active period, nodes are following the same synchronization algorithm (using $BEACON$ messages) as in the basic 802.11 mode. When a node has packets to transmit, it first sends an Ad hoc Traffic Indication Map ($ATIM$) where the sender specifies the receiver of the $DATA$ packet. Only the intended receiver replies with an acknowledgment. After this exchange, both the sender and the receiver stay awake for the duration of the beacon period to send and, respectively, receive the $DATA$ packets and the associated acknowledgments. If a node does not have any packet to send and it is not an intended receiver, the node goes to sleep for the rest of the sleep period (as shown in Fig. 4.3).

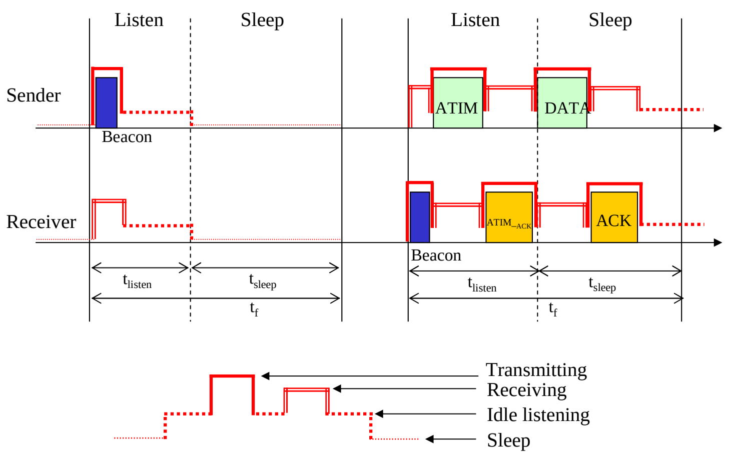


Figure 4.3: Example of data exchange in 802.11 power saving mode (ad-hoc).

Power Consumption Analysis

Each frame period t_f has a listen t_{listen} and a sleep period t_{sleep}:

$$T_f = t_{sleep} + t_{listen}. \tag{4.14}$$

When the sensor node is awake, it consumes energy by transmitting, receiving, and idle listening. When the sensor node is in sleep mode, it consumes comparatively very little energy. However, it also takes time and energy to wake up the radio:

$$T = T_{tx} + T_{rx} + T_{oh} + T_{idl\_lsn} + T_{slp} + T_{startup}. \tag{4.15}$$

The energy per sampling period spent on sending and receiving packets in 802.11 PS mode is identical with the one for 802.11 basic mode except for the additional exchange of $ATIM$ and $ATIM_{ACK}$ packets (4.16), (4.17):

$$\begin{aligned} E_{tx} = {} & \left[N_{tx}(i)\left(t_{DATA} + t_{ATIM}\right) + N_{rx}(i)\left(t_{ACK} + t_{ATIM_{ACK}}\right)\right] P_{tx} \\ & + t_B \frac{T}{t_{beacon}} \frac{1}{N_{neighbor}(i) + 1} P_{tx}, \end{aligned} \tag{4.16}$$

$$\begin{aligned} E_{rx} &= \left[N_{rx}(i)\left(t_{DATA} + t_{ATIM}\right) + N_{tx}(i)\left(t_{ACK} + t_{ATIM_{ACK}}\right)\right] P_{rx} \\ &+ t_B \frac{T}{t_{beacon}} \frac{N_{neighbor}(i)}{N_{neighbor}(i) + 1} P_{rx}. \end{aligned} \tag{4.17}$$

We divide the overhearing time of 802.11 PS mode into two parts: overhearing time during the listen period T_{oh_a} and overhearing time during the sleep period T_{oh_s}:

$$T_{oh} = T_{oh_a} + T_{oh_s}. \tag{4.18}$$

The overhearing time of 802.11 PS mode is reduced because nodes go to sleep when they it neither have packets to send nor to receive. Thus, overhearing does not occur when the node is in sleep mode. Nodes are always awake in listen (active) state in all frames of the sampling period. Thus, the overhearing time in the listen period can be computed using the same logic as in 802.11 basic mode. The difference between the overhearing time in listen period of 802.11 PS mode and that in 802.11 basic mode is that the listen period of 802.11 PS mode has t_{ATIM} and $t_{ACK_{ATIM}}$ while 802.11 basic mode has t_{DATA} and t_{ACK}:

$$T_{oh_a} = N_{oh,ATIM}(i) t_{ATIM} + N_{oh,ACK_{ATIM}}(i) t_{ACK_{ATIM}}. \tag{4.19}$$

Nodes use the radio in sleep period when they have data to transmit or receive. Thus, the overhearing time in sleep period T_{oh_s} is computed as the portion of frames for transmitting and receiving over the whole frames, $\frac{N_{tx}(i)+N_{rx}(i)}{K}$, multiplied by the overhearing time in 802.11 non power saving mode:

$$T_{oh_s} = \frac{N_{tx}(i) + N_{rx}(i)}{K} \left[N_{oh,DATA}(i)(t_{DATA}) + N_{oh,ACK_{DATA}}(i)(t_{ACK_{DATA}})\right]. \tag{4.20}$$

To compute the idle listening time T_{idl_lsn} in listening period, we subtract the radio operation time from total listening time:

$$\begin{aligned} T_{idl\_lsn_a} &= t_{listen} K - N_{tx}(i)(t_{ATIM} + t_{ACK_{ATIM}}) \\ &- N_{rx}(i)(t_{ATIM} + t_{ACK_{ATIM}}) \\ &- T_{oh_a} - t_B \frac{T}{t_{beacon}}. \end{aligned} \tag{4.21}$$

For the idle listening time of sleep period, we subtract the radio operation time from the time where the node is awake in sleep the period:

$$\begin{aligned} T_{idl\_lsn_s} = t_{sleep}&(N_{tx} + N_{rx}) \\ &- N_{tx}(i)(t_{DATA} + t_{ACK_{DATA}}) \\ &- N_{rx}(i)(t_{DATA} + t_{ACK_{DATA}}) - T_{oh_s}. \end{aligned} \tag{4.22}$$

Nodes turn on the radio once per each frame, therefore $t_{WU}K$ is needed for radio startup in each period T:

$$T_{startup} = t_{WU}K. \tag{4.23}$$

The total sleep time T_{slp} during the sampling period is computed by subtracting the time for ratio operations from the sampling period (4.15). This can be applied to sleep time computation of other protocols.

4.2.6 Power Consumption Analysis of S-MAC

Protocol overview

The basic scheme of S-MAC [21, 58] is shown in Fig. 4.4. Each node sleeps for some time, and then wakes up and listens to determine if it needs to receive a packet. The sleep schedules of all nodes in a neighborhood are synchronized and nodes use $SYNC$ packets to maintain synchronization. The power model of S-MAC is similar to the one of 802.11 in power saving mode. The main difference in terms of power consumption is that, for S-MAC, the nodes go to sleep after transmitting and receiving a $DATA$ packet rather than remaining awake for the rest of the beacon period. S-MAC also provides a mechanism that handles the situation where there exist nodes with different sleep schedules (thus being suitable for multihop networks). In this case, the border nodes maintain multiple schedules to adopt the different schedules. However, to make the comparison with other protocols fair, we only consider the power consumption with a single schedule.

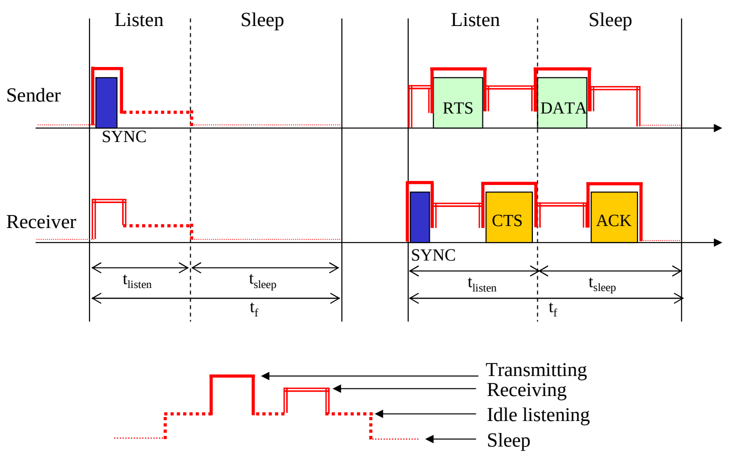


Figure 4.4: Example of data exchange in S-MAC.

Power Consumption Analysis

The time spent sending and receiving packets in S-MAC is identical with the one for 802.11 PS mode (4.16), (4.17) except that the beacon interval, t_{beacon}, should be replaced by the synchronization interval, t_{sync}.

When the data transmission is completed, S-MAC nodes return to sleep state. This reduces not only the idle listening time but also the overhearing time of nodes. Since, in S-MAC, the overhearing occurs during the listen period, the power consumption due to overhearing is similar to the power consumption of overhearing during the listen period of 802.11 PS mode (4.19):

$$T_{oh} = N_{oh,RTS}(i)t_{RTS} + N_{oh,CTS}(i)t_{CTS}. \tag{4.24}$$

The idle listening also occurs only during the listen period. We compute the idle listening time by subtracting the radio operation time from the total listen period time of the sampling period:

$$\begin{aligned} T_{idl\_lsn} &= t_{listen}K - N_{tx}(i)(t_{RTS} + t_{CTS}) \\ &\quad - N_{rx}(i)(t_{RTS} + t_{CTS}) - T_{oh} - t_{SYNC}\frac{T}{t_{sync}}. \end{aligned} \tag{4.25}$$

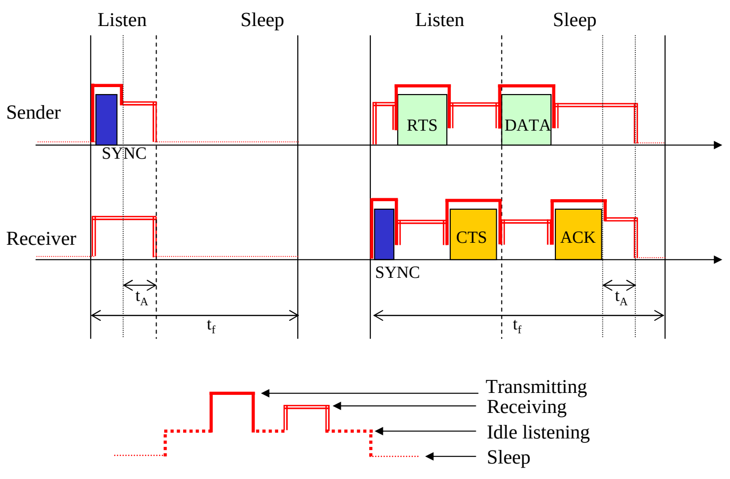


Figure 4.5: Example of data exchange in T-MAC.

4.2.7 Power Consumption Analysis of T-MAC

Protocol overview

Figure 4.5 shows the basic scheme of the T-MAC protocol [59]. Similarly to S-MAC, every node periodically wakes up to communicate with its neighbors, and then goes to back sleep again until next frame. In the meantime, all new messages are queued. Nodes communicate with each other using a RTS, CTS, ACK scheme, which provides both collision avoidance and reliable transmission. To handle load variations in time and location T-MAC introduces an adaptive duty cycle by dynamically adjusting the length of the active period. A node will continue to listen and potentially transmit, as long as it is in an active period. An active period ends when no activation event has occurred for a timeout period t_A. A node will sleep if it is not in an active period. Consequently, t_A determines the minimal amount of idle listening per frame. This adaptive scheme reduces the amount of energy wasted on idle listening, in which nodes wait for potentially incoming messages, while still maintaining a reasonable throughput.

Power Consumption Analysis

Unlike S-MAC that has a fixed active listening period, T-MAC maintains a dynamic active listening period. Thus, T-MAC reduces the idle listening and overhearing times during the active period and it introduces a time-out period t_A after a packet transmission. We assume that the packet transmissions and receptions uniformly occur throughout the frame period T_f. Since S-MAC and T-MAC follow the same message exchange sequence to transmit or receive packets, the energy consumptions spent on sending and receiving packets in T-MAC, T_{tx} and T_{rx}, respectively, are the same as those of S-MAC, which are also same as those of 802.11 PS mode (4.16, 4.17).

Let T_{oht} be the overhearing time during the sampling period when the node stays awake until the next frame begins:

$$T_{oht} = N_{oh,RTS}(i)t_{RTS} + N_{oh,CTS}(i)t_{CTS} + N_{oh,DATA}(i)t_{DATA} + N_{oh,ACK_{DATA}}(i)t_{ACK}. \quad (4.26)$$

Then, the overhearing time T_{oh} during the sampling period is given by the portion of total timeout time Kt_A over the sum of the remaining time until the end of each frame multiplied by T_{oht}:

$$T_{oh} = T_{oht}\frac{Kt_A}{T - T_{oht} - \frac{T}{t_{sync}}t_{SYNC}}. \quad (4.27)$$

Idle listening occurs only during t_A in T-MAC. If there is no channel activity during t_A after the synchronization, the idle listening time is t_A during one frame T_f. Nodes also listen to the channel for t_A after the packet exchanges. We ignore the idle listening time between two back to back packets within one frame T_f Thus, the idle listening time during a sampling period is simply:

$$T_{idl\_lsn} = t_A K. \quad (4.28)$$

4.2.8 Power Consumption Analysis of B-MAC

Protocol overview

Figure 4.6 shows the basic operation of B-MAC [60]. B-MAC uses a preamble to implement a low power listening (LPL) scheme. Nodes sleep (t_{sleep}) and wake up (t_{CCA}) periodically. When a node wakes up, it turns on its radio and checks for activity on the channel. If the node does not detect any activity on the channel, it goes immediately back to sleep. If it detects the activity, the node keeps the radio on and receives the packet. After receiving the packet, the node goes back to sleep. Senders transmit a long preamble before each payload, such that receivers can detect the channel activity and remain active. To reliably receive the data, the preamble length should be greater than or equal to the interval that the channel is checked for activity.

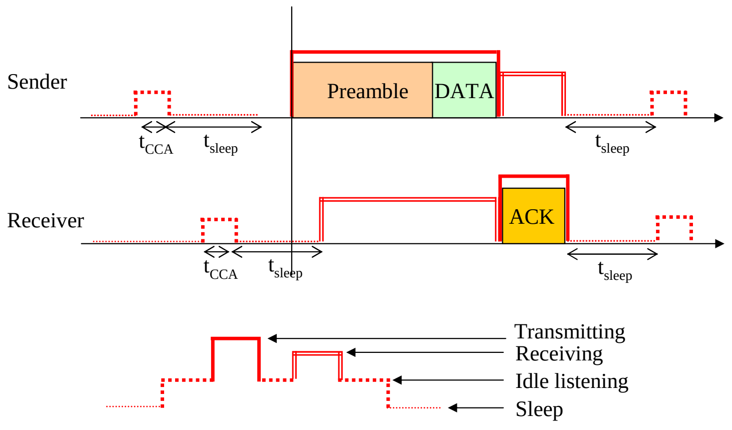


Figure 4.6: Example of data exchange in B-MAC.

Power Consumption Analysis

The power consumption of B-MAC was analyzed in [60, 61], based on a single-hop network model. We follow their approach for the analysis, however, we provide the power consumption formula for a multihop network. We also obtain more detailed power consumption than those of [60, 61].

When it has a packet to transmit, B-MAC first sends a long preamble (of length t_P) followed by the payload. For a meaningful comparison, similar to the other protocols, we assume that B-MAC enables acknowledgments, (which are optional in B-MAC):

$$T_{tx} = N_{tx}(i)(t_P + t_{DATA}) + N_{rx}(i)t_{ACK_{DATA}}. \tag{4.29}$$

Sensor nodes, on average, receive half of entire preamble for every packet destined to them:

$$T_{rx} = N_{rx}(i)(0.5t_P + t_{DATA}) + N_{tx}(i)t_{ACK_{DATA}}. \tag{4.30}$$

When the packets are destined to other nodes (overhearing), sensor nodes only receive the preamble and the data, but they do not send ACK:

$$T_{oh} = N_{oh,DATA}(i)(0.5t_P + t_{DATA}). \tag{4.31}$$

Unlike the other protocols such as 802.11 PSM, S-MAC, and T-MAC where the number of frames per sampling interval is constant and therefore the startup time ($T_{startup}$) is also constant for each sampling period (4.23), the number of frames in B-MAC, K_B, varies depending on the number of data to send or receive (4.32). The K_B is calculated as the number of wake-ups that result in packet transmissions or receptions (including overhearing) during the sampling period plus the number of wake-ups without sensing activities:

$$\begin{aligned} K_B = & (N_{tx(i)} + N_h)(i) \\ & + \frac{1}{T_P + t_{CCA}}\{T \\ & - N_{tx}(i)(t_{CCA} + t_P + t_{DATA} + t_{ACK_{DATA}}) \\ & - N_{rx}(i)(t_{CCA} + 0.5t_P + t_{DATA} + t_{ACK_{DATA}}) \\ & - (N_h(i) - N_{rx}(i))(t_{CCA} + 0.5t_P + t_{DATA})\}. \end{aligned} \tag{4.32}$$

B-MAC senders and receivers turn on the radio and perform the CCA operation once per each frame. Thus, the CCA operation includes the radio startup operation, which means that we can assume that the startup time $T_{startup}$ is zero and the only idle listening time in B-MAC is the time for CCA.

$$T_{startup} = 0, \tag{4.33}$$

$$T_{idl\_lsn} = K_B t_{CCA}. \tag{4.34}$$

4.2.9 Power Consumption Analysis of X-MAC

Protocol overview

The LPL (low power listening) in B-MAC using the long preamble enables the receivers to sleep for long periods and wake up for short time intervals to receive packets. However, this long preamble introduces a long transmission time for the senders and a large overhearing time [1, 61]. One method to solve the drawback of long preambles in BMAC is to divide the long preamble into a series of short preamble packets [1].

Figure 4.7 shows the basic operation of X-MAC. The operation of X-MAC is similar to B-MAC. The nodes sleep and wake up repeatedly. When a node has data to send, it broadcast a series of short preambles that include the receiver's address. When a node wakes up and receives a short

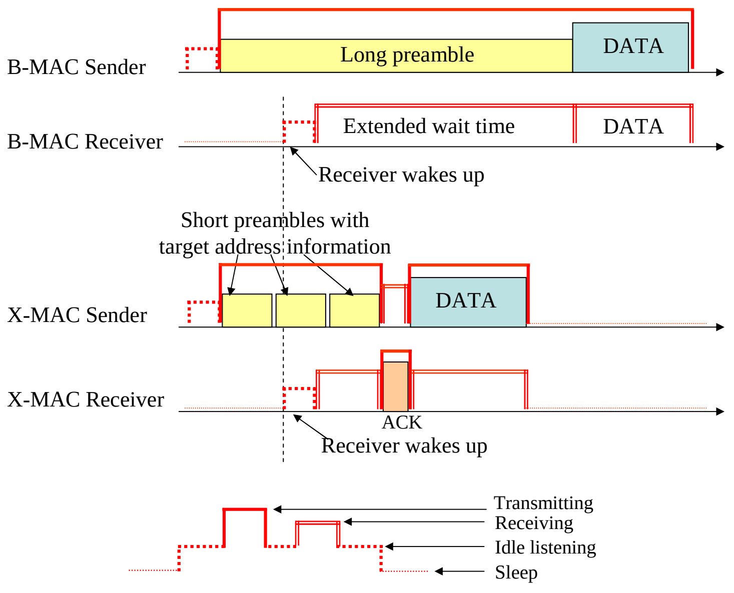


Figure 4.7: Comparison of the timelines between B-MAC and X-MAC [1].

preamble packet, it examine the receiver ID field. If the target address does not match its address, the node goes back to sleep. If the node is the intended receiver (i.e., the address in the short preamble is the address of the receiver), it sends ACK to the sender and receives the DATA part of the packet. The sender waits for the ACK from the intended receiver for a short period after it send a short preamble to the nodes. After receiving the ACK, the sender transmits the DATA part of the packet.

Power Consumption Analysis

The power consumption analysis of X-MAC is presented in [1]. However, our analysis is different from X-MAX's approach in the following ways:

- we assume the multi-hop environment while X-MAC only analyzes the power consumption of single hop environment;
- in order to handle effectively the case where multiple transmitters are trying to send packets to a receiver, in X-MAC, the receiver remains awake for a short period of time. However, we do not consider this post packet reception delay;
- for fairness of the comparison, while the X-MAC protocol does not explicitly specify the use of

acknowledgements, we assume that the receiver sends an ACK packet after it receives a DATA packet.

The power consumption of X-MAC differs from that of B-MAC in sending/receiving the preambles.

In X-MAC, authors does not specify the actual length of the short preamble. Instead, authors specify the time to send a short preamble for Telos mote, which is almost the same as the time to send an ACK. Denote by t_{P_x} the time to send a short preamble. Then, the number of short preambles, N_{P_x}, to transmit during t_P in B-MAC is:

$$N_{P_x} = \frac{t_P}{t_{P_x}}. \tag{4.35}$$

When the X-MAC sender starts to transmit short preambles, the receiver is, on average, in the middle of its sleep period. Thus, the sender transmits $\frac{N_{P_x}}{2} + 1$ short frames until the receiver wakes up to receive the short preamble and replies with ACK to the sender:

$$T_{tx} = N_{tx}(i) \left[\left(\frac{N_{P_x}}{2} + 1 \right) t_{P_x} + t_{DATA} \right] + N_{rx}(i) \left(t_{ACK_{PREAMBLE}} + t_{ACK_{DATA}} \right). \tag{4.36}$$

When the receiver wakes up, it will receive on average $\frac{3}{2}$ short preambles before it receives the DATA part of the packet:

$$T_{rx} = N_{rx}(i) \left(\frac{3}{2} t_{P_x} + t_{DATA} \right) + N_{tx}(i) \left(t_{ACK_{PREAMBLE}} + t_{ACK_{DATA}} \right). \tag{4.37}$$

The number of packets that the X-MAC receiver overhears is the same as that of in B-MAC. However, after the receiver receives only one short preamble, it can determine if the packet is destined to itself or not:

$$T_{oh} = [N_h(i) - N_{rx}(i)] \left(\frac{3}{2} t_{P_x} \right). \tag{4.38}$$

Since we assume that the check intervals in B-MAC and X-MAC are the same and the radio remain awake between short preambles, the $T_{startup}$ and T_{idl_lsn} of X-MAC are the same as those of B-MAC (4.33, 4.34).

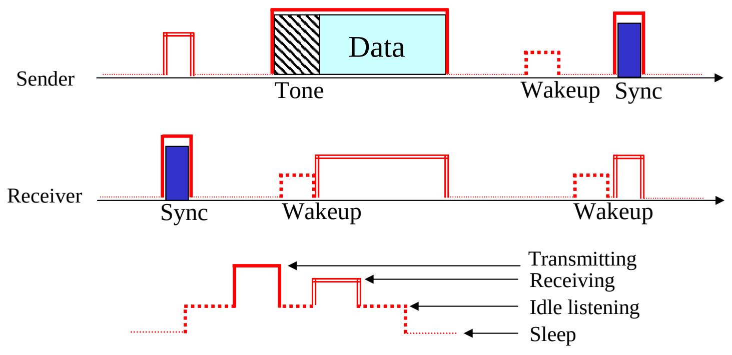


Figure 4.8: SCP-MAC synchronization scheme.

4.2.10 Power Consumption Analysis of SCP-MAC

Protocol overview

As explained in Section 4.2.9, B-MAC transmissions waste energy with long preambles. SCP (Scheduled Channel Polling) MAC reduces the preamble size by waking up the senders and the receiver at the same time and exchanging packets. All nodes, in SCP-MAC [61], sleep and wake up repeatedly. When a sender wakes up, it sends a short preamble called the tone to the neighbors as shown in Figure 4.8. The receiver also wakes up at approximately the same time, detects the tone, and receives the data. Since the nodes are synchronized, the duration of the tone can be very small as long as it is detected at the receiver. Therefore, most of the overhead in SCP-MAC stems from maintaining the synchronization. The SCP-MAC authors use the synchronization scheme used in S-MAC [58]. The authors determine the optimal synchronization period, which ranges between $7\times$ *packet generation interval* and $16\times$ *packet generation interval*. The authors also derive the wakeup tone length that minimizes the overall power consumption, which is about from 10 ms to 30 ms depending on the packet generation interval.

Power Consumption Analysis

The power consumption of SCP-MAC was analyzed in [61]. The differences between our power consumption analysis and the one in [61] are as follows.

- the authors of SCP-MAC provide the formulas for two power consumptions, the best and the worst case. In the best case, all synchronization information is piggybacked on data packets, therefore, there is no extra overhead for synchronization. In the worst case there is no piggybacking. For the fairness of the comparison, we do not assume piggybacking;

- the analysis of SCP-MAC is also based on a single-hop network model. We assume a multi-hop network model;
- while SCP-MAC protocol does not explicitly specify acknowledgements for the data, for fairness we assume the existence of an ACK packet for each DATA packet.

Let $t_{P_{SCP}}$ and t_{sync} be the optimal duration of the tone and optimal synchronization interval, respectively. The synchronization packet also requires a short preamble. Thus, it will take $t_{P_{SCP}} + t_{SYNC}$ to send a synchronization packet during one sampling period. Then, the time for transmitting packets during each sampling period is given as:

$$T_{tx} = N_{tx}(i)(t_{P_{SCP}} + t_{DATA}) + N_{rx}(i)t_{ACK_{DATA}} + t_{SYNC}\frac{T}{t_{sync}}\frac{1}{N_{neighbor}(i)+1}. \qquad (4.39)$$

The receiving and overhearing times during the sampling period are also similar to those of B-MAC except for the preamble size and the synchronization overhead:

$$T_{rx} = N_{rx}(i)(0.5t_{P_{SCP}} + t_{DATA}) + N_{tx}(i)t_{ACK_{DATA}} + t_{SYNC}\frac{T}{t_{sync}}\frac{N_{neighbor}(i)}{N_{neighbor}(i)+1}, \qquad (4.40)$$

$$T_{oh} = (N_h(i) - N_{rx}(i))(0.5t_{P_{SCP}} + t_{DATA}). \qquad (4.41)$$

4.2.11 Power Consumption Analysis of an Ideal MAC

Protocol overview

Although it is practically impossible to implement an ideal MAC, we present the I-MAC (the Ideal MAC) for comparison purposes. In I-MAC, nodes sleep all the time except when they transmit or receive packets. That is, we assume that senders and receivers have exact schedule information of each other and are perfectly synchronized with no overhead. When the sender transmits data packets, the receiver can receive the packet without any collisions. Thus, sensor nodes consume the energy only for

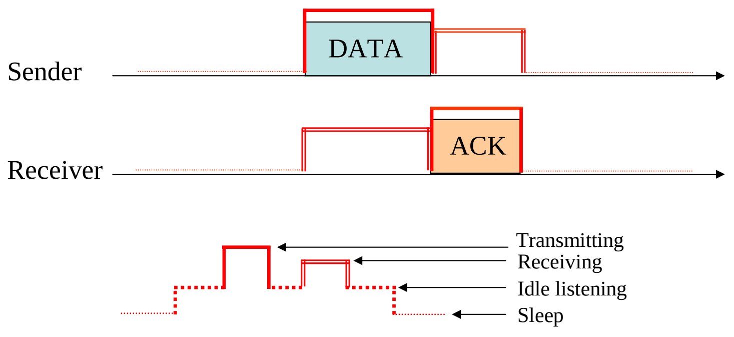


Figure 4.9: Example of data exchange in I-MAC.

data packet transmissions and receptions. For fairness, we also assume the existence of an ACK packet for each DATA packet. Figure 4.9 shows the basic operation of I-MAC.

Power Consumption Analysis

In I-MAC, nodes wake up only if there are packets to transmit or receive. Therefore, there is no overhearing or idle listening in I-MAC:

$$E = E_{tx} + E_{rx} + E_{startup}. \tag{4.42}$$

When senders transmit packets, they consume the energy for transmitting DATA packets and receiving ACK packets:

$$E_{tx} = (N_{tx}(i)t_{DATA} + N_{rx}(i)t_{ACK_{DATA}})P_{tx}. \tag{4.43}$$

When receivers receive packets, they consume the energy for receiving DATA packets and transmitting ACK packets:

$$E_{rx} = (N_{rx}(i)t_{DATA} + N_{tx}(i)t_{ACK_{DATA}})P_{rx}. \tag{4.44}$$

For each packet transmission and reception, nodes need to start up the radio:

$$E_{startup} = (N_{tx}(i) + N_{rx}(i))t_{WU}P_{startup}. \tag{4.45}$$

4.3 Numerical Results

In this section, we compare the energy consumption of the MAC protocols that we presented in the previous section.

4.3.1 Simulation Setup

Since current ad hoc network simulators are not suitable for modeling the power consumption of large scale wireless sensor networks with a variety of MAC protocols, we implemented our own simulator that idealizes the physical and MAC layers [63]. We simulated N nodes randomly deployed in a 60m x 60m rectangular area. In the middle of the area there is a single base station. The transmission range is assumed circular with a radius of 10m. The nodes are assumed to be powered by two AA alkaline batteries with a capacity of 2000 mAh. Since the focus of the paper is not on the routing algorithm, a simple shortest path algorithm was employed. If multiple shortest paths with different amounts of power left are available, the one with the largest power available is preferred. We assumed that no in-network processing is done, and therefore, all data is forwarded (in a multihop fashion) to the base station. We assume that the base station has enough power to outlast any of the sensor nodes. We use the time until 50% of the nodes can no longer forward data to the base station (either because their batteries are depleted, or because there are no routes to the base station) as the main performance measure.

We used two set of parameters one for Mica 2 [60] and one for Telos mote [9], shown in Tables 4.1 and 4.2, respectively. Mica 2 and Telos use the CC1000 and CC2420 radio chips, respectively. The power consumptions for data transmission and reception of CC2420 are similar to those of CC1000; however, the time to transmit/receive a byte with CC2420 is more than 10 times shorter than with CC1000. There are also significant differences in the startup time and power consumption during sleeping. We were not able to find a reference on the average current consumption for starting up the radio for CC2420. Based on the similarity of the power consumption in transmit and receive modes, we use the same value as that of CC1000, i.e., 5 mA [60].

We assume that the sizes of all control and data packets are 10 bytes and 20 bytes, respectively. We also assume that the sampling period (T) is 600 s and the frame size is $100ms$. Thus, there are 6000 frames in one sampling period. Table 4.3 shows the parameters used in the experiments.

Table 4.1: Current consumption and time for Mica2 radio operations

Operation	**Current(mA)**	**Time(s)**
Idle listening	15.1	
Receive 1 byte	15.1	416E-6
Transmit 1 byte (0 dBm)	25.4	416E-6
Sleep	0.019	
Start up	5	1.8E-3

Table 4.2: Current consumption and time for Telos radio operations

Operation	**Current(mA)**	**Time(s)**
Idle listening	21.8	
Receive 1 byte	21.8	32E-6
Transmit 1 byte (0 dBm)	19.5	32E-6
Sleep	0.0051	
Start up	5	0.58E-3

Table 4.3: Parameters used in the experiments

Parameter	**Default**	**Range**
Deployment area	60m $\times$ 60m	fixed
Battery size	2 $\times$ 2000 mAh	fixed
Sampling period	600s	fixed
Number of nodes	227	227 $\sim$ 629
Wake up interval	$100ms$	$100ms \sim 10100ms$
Duty cycle	15%	$0.15\% \sim 15\%$

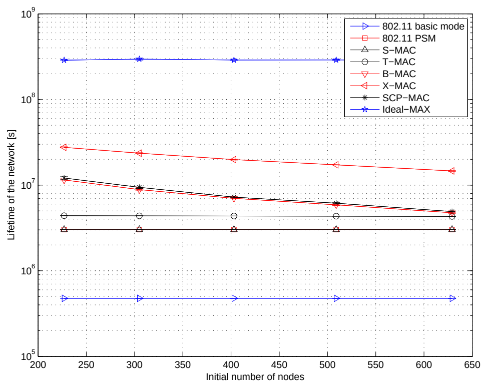
4.3.2 Simulation Results

Dependency on the density of sensor nodes

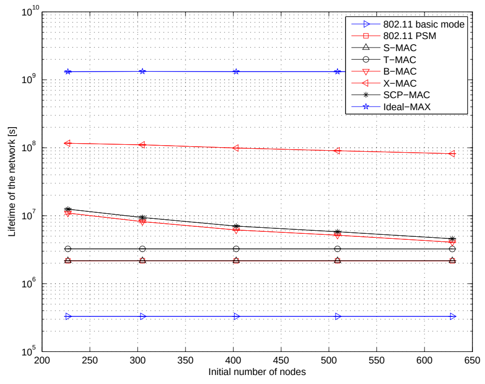
Figures 4.10 (a) and (b) show the lifetime of the networks using as a function of the initial number of nodes for Mica 2 and Telos motes, respectively while maintaining the deployment area fixed. In this simulation, we varied the initial number of sensor nodes from 227 to 629, while the duty cycle was fixed to 15%, i.e., the active time within a frame is $15ms$ for 802.11 PSM, S-MAC, and T-MAC. For Mica 2 mote, we set the check interval of B-MAC to $100ms$, which results in a preamble size of 240 bytes. The time to wake up from sleep mode and sense the channel T_{CCA} is set to $1.8ms$. For X-MAC, we used 10 byte short preambles. For SCP-MAC, we used the tone length of $30ms$, which is recommended as an optimal value in [61] for the sampling interval $600s$. It is impossible to implement B-MAC on Telos mote because Telos uses packet-based CC2420 radio chip, thus, it cannot utilize the long preamble of B-MAC. However, we depict the B-MAC lifetime on Telos assuming that the long preamble is implemented using back-to-back packets like in X-MAC [1].

In Fig. 4.10, two groups of protocols show similar results as the number of initial sensor nodes increases. The lifetimes of preamble-based protocols such as B-MAC, X-MAC, and SCP-MAC that use preambles to notify the receivers of the data frames are longer than those of frame-based protocols such as S-MAC and T-MAC that use designated sleeping/listening intervals. However, the lifetime of preamble-based protocols decreases as the number of nodes increases, while the performance of frame-based MACs does not change with the number of initial nodes. The reason is that the overhearing time of a sensor node in frame-based protocols is less affected by the number of neighbors of the node than those in preamble-based protocols (4.24), (4.27), (4.31), (4.38), (4.41). The overhearing times of preamble-based protocols are primarily determined by the preamble size and the number of neighbors while the overhearing times of frame-based protocols are affected by control packets (CTS and $ATIM_{ACK}$) that a node hears and the number of control packets is less than the number of preambles that a node hears by the number of neighbors (4.7).

For 802.11 and S-MAC, as the density increases, the overhearing time also increases but the idle listening time decreases by the same amount. That is, the sum of idle listening time and overhearing time is constant regardless of the density. For T-MAC, the lifetime of the network decreases only slightly as the density increases. That is because T-MAC wastes a minimal amount of idle listening time(t_A)



(a) Mica 2



(b) Telos

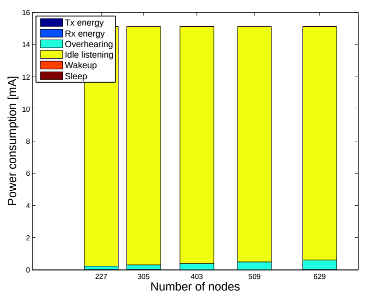
Figure 4.10: Lifetime of the network as a function of the number of sensor nodes for Mica 2 and Telos motes (at constant network area).

each frame. Figures 4.11 and 4.12 depict the components of the power consumption of the nodes in the first tier (one hop from the base station) during one sampling period as the function of initial number of sensor nodes. The first tier nodes determine the lifetime of the network as, upon depletion of their batteries, the network becomes disconnected. We observe from those figures that the idle listening time dominates the total power consumption of frame-based protocols while overhearing time takes large part of the energy consumption of preamble-based protocols.

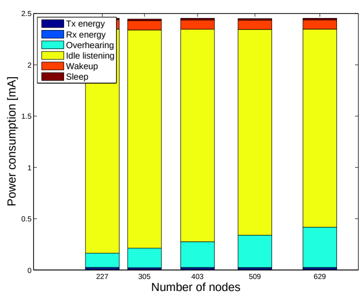
Dependency on the wake-up interval

In this simulation, we fixed the initial number of nodes to 227 while varying the wake-up interval from $100ms$ to $10100ms$ in steps of $500ms$. All other parameters are the same as in Section 4.3.2. A variation in the wake-up interval corresponds to a variation in the frame period. Thus, when the wake-up interval increases, the sleep period also increases while the listening period remains constant. The listening period is always $15ms$. Therefore, increasing the wake-up interval means decreasing radio duty cycle as well as increasing the packet transmission delay. For preamble-based protocols, we vary the check interval at which the nodes wake up periodically.

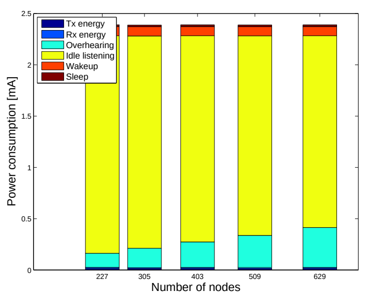
Figures 4.13 (a) and (b) show the lifetime of the networks using as a function of wake-up interval for Mica 2 and Telos motes, respectively. There are no significant differences in the trends of the lifetime of the network on the two hardware platforms. Frame-based protocols and preamble-based protocols show different behaviors as the wake-up interval increase. As the wake-up interval



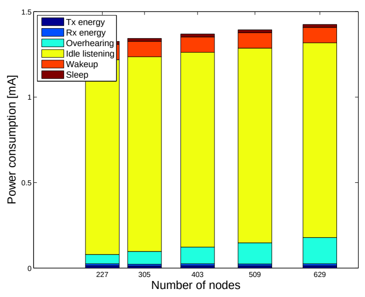
(a) 801.11 basic mode



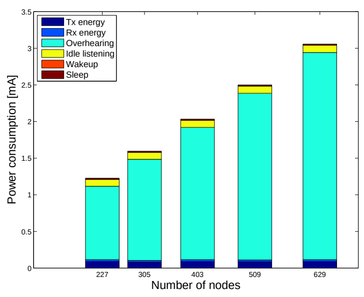
(b) 802.11 PSM



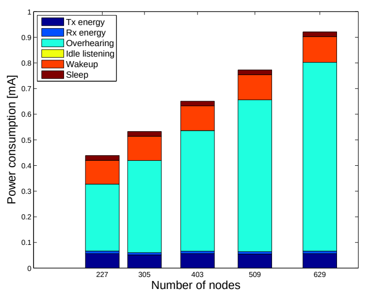
(c) S-MAC



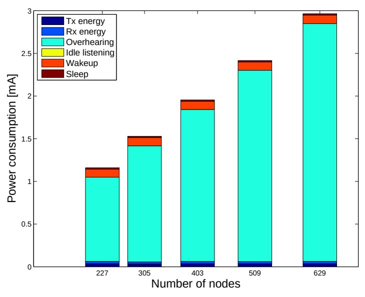
(d) T-MAC



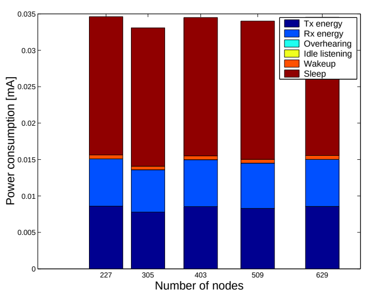
(e) B-MAC



(f) X-MAC

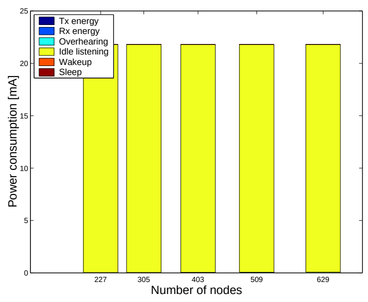


(g) SCP-MAC

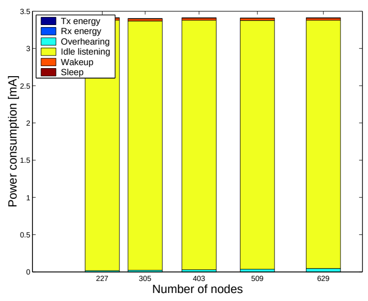


(h) Ideal MAC

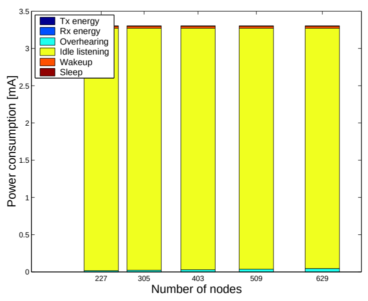
Figure 4.11: Components of the power consumption of the first-tier of nodes as a function of the number of sensor nodes with constant deployment area for Mica2.



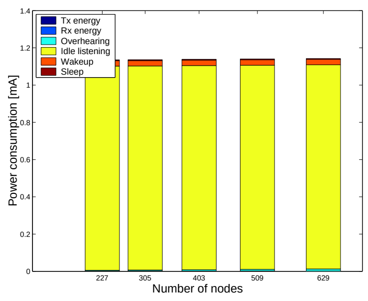
(a) 801.11 basic mode



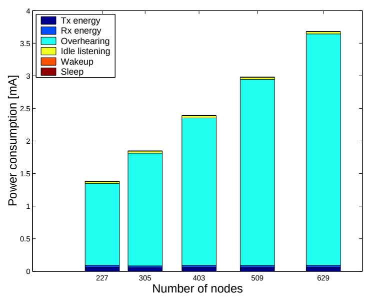
(b) 802.11 PSM



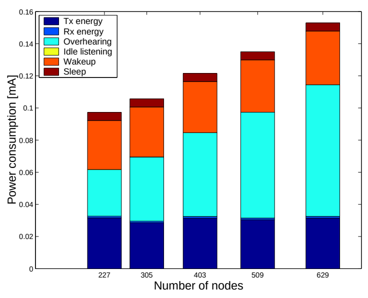
(c) S-MAC



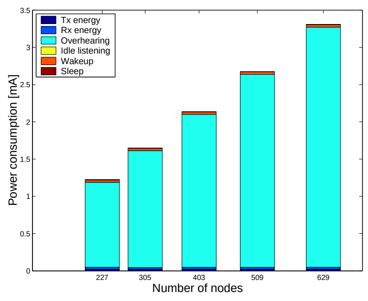
(d) T-MAC



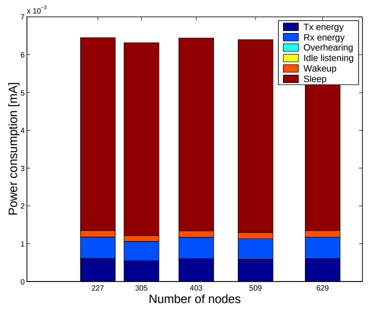
(e) B-MAC



(f) X-MAC

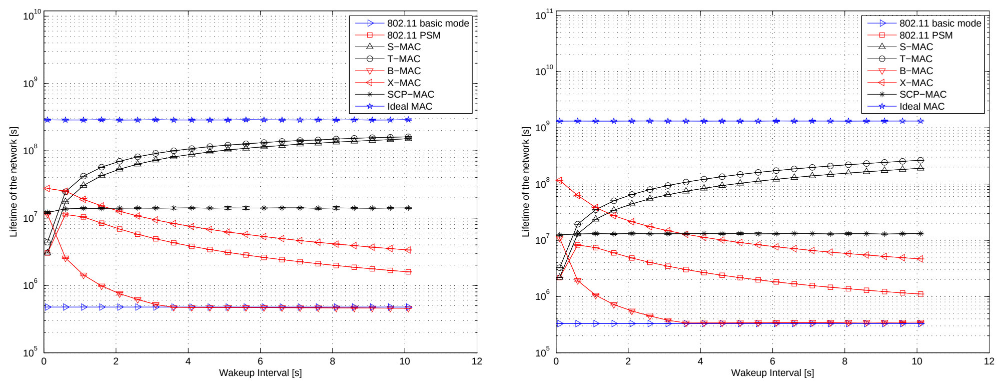


(g) SCP-MAC



(h) Ideal MAC

Figure 4.12: Components of the power consumption of the first-tier of nodes as a function of the number of sensor nodes with constant deployment area for Telos motes.

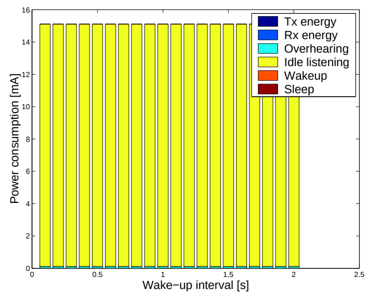


(a) Mica 2 (b) Telos

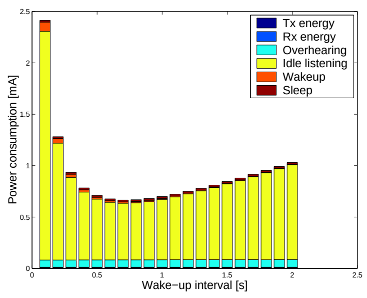
Figure 4.13: Lifetime of the network as a function of wake-up (check) interval for Mica 2 and Telos motes.

increases, the lifetime of frame-based protocols except 802.11 PS mode increases while the lifetime of preamble-based protocols decreases. The lifetimes of S-MAC and T-MAC approach that of Ideal MAC as the wake-up interval increases (at the expense of an increase in delay). B-MAC shows the worst performance as the wake-up interval increases since the overhearing time caused by the long preamble becomes very large (see Figures 4.14 and 4.15). These results suggest that when the applications require short delays (less than $1s$), X-MAC or SCP-MAC should be used; S-MAC or T-MAC should be used for monitoring applications that tolerate the long delays.

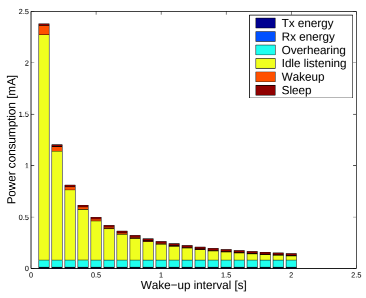
One interesting result is that for 802.11 PS mode, the lifetime of the network increases until the wake-up interval reaches around $0.7s$ and decreases after that point. Figures 4.14 and 4.15, which depict the components of the power consumption of the nodes in the first tier as the function of wake up interval, provide the explanation for this result. The idle listening time during the listen period is constant regardless of the wake-up interval, thus, the idle listening time decreases with the number of frames when the sampling period decreases (i.e., as the wake-up interval increases). However, as the wake-up interval increases, the idle listening time during the sleep period also increases. The decreasing rate of idle listening time during the listen period is faster than that in the sleep period; however the idle listening time during the listen period has a lower bound while the increase of the idle listening time during the sleep period continues with the increase in the wake-up time.



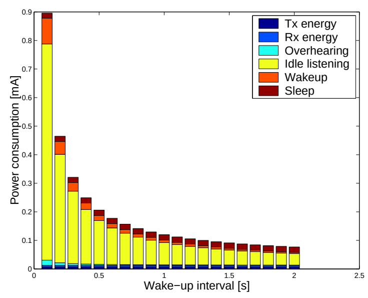
(a) 801.11 basic mode



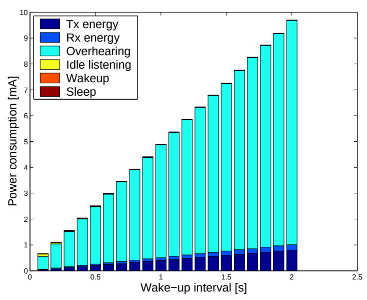
(b) 802.11 PSM



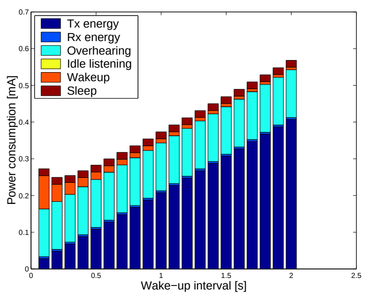
(c) S-MAC



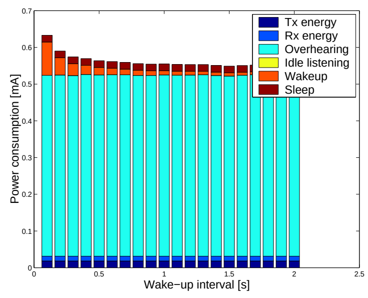
(c) T-MAC



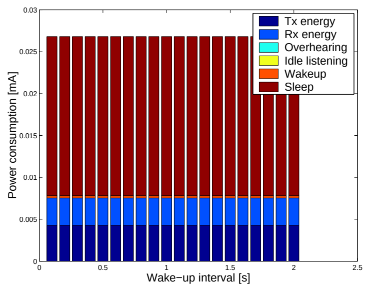
(d) B-MAC



(f) X-MAC

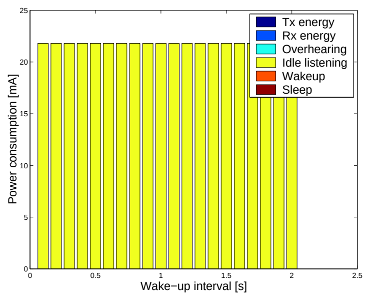


(g) SCP-MAC

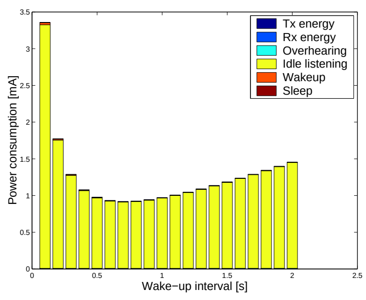


(h) Ideal MAC

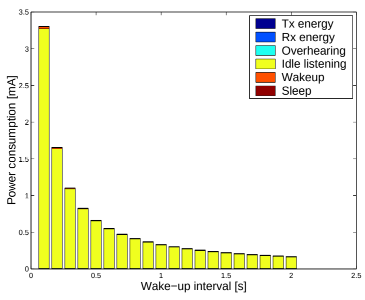
Figure 4.14: Components of the power consumption of the first-tier of nodes as a function of the wake-up interval for Mica 2.



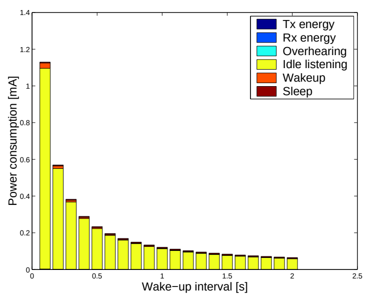
(a) 801.11 basic mode



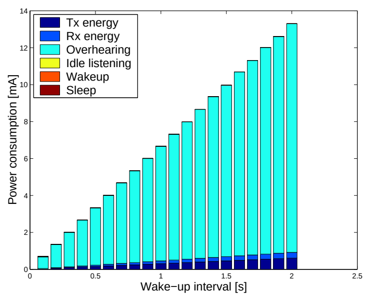
(b) 802.11 PSM



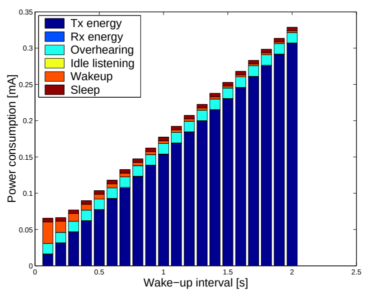
(c) S-MAC



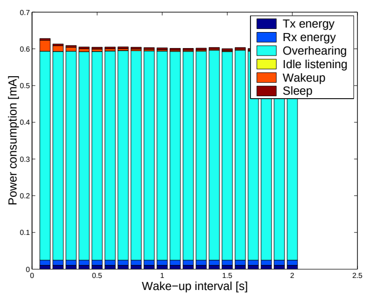
(c) T-MAC



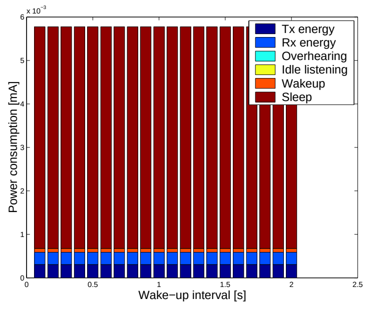
(d) B-MAC



(f) X-MAC



(g) SCP-MAC



(h) Ideal MAC

Figure 4.15: Components of the power consumption of the first-tier of nodes as a function of the wake-up interval for Telos 2.

4.4 Experimental results

In this section, we measure the energy consumptions of the MAC protocols implementation on the Mica 2 platform using trace-based method to validate the analytical results that we derived in Section 4.2. We chose the Mica2 platform as most power saving MACs that were implemented used this platform (either due to its flexibility or lack of an alternative at the time).

4.4.1 Experiment setup

The trace-based measurement of energy consumption is a well known method [64–66] that tracks the energy consumption in each state of the transceiver during the operation of the MAC protocols by generating timestamped log messages when changing the radio state. After collecting the log messages from the mote, the energy consumption is computed offline. For the power consumption in each radio state, we use the results from PowerTOSSIM [66] (also shown in Table 4.1), a TinyOS simulator with support for power consumption.

We trace the following radio operations:

- RadioStart: Turn on the radio
- RadioStop: Turn off the radio
- Rxmode: Start of Rx mode
- Txmode: Start of Tx mode
- SetPower: Set the transmission power

Figure 4.16 shows the simplified radio state diagram. The idea used in trace-based measurement like in PowerTOSSIM is as follows: when the mote is working on its normal operation, the radio should be in one of three state: sleeping (*Sleep*), receiving (*Idle, Rx, or PreTx*), or transmitting (*Tx*). When the radio state changes, we generate the corresponding log message with current timestamp. For example, when the mote goes into sleep state, the *RadioStop* message is generated. When it wakes up and goes into idle (*Idle/Rx*) state, the *RadioStart* and *RxMode* messages are generated. With those messages and the timestamp we can determine how long the mote stayed in the corresponding radio state.

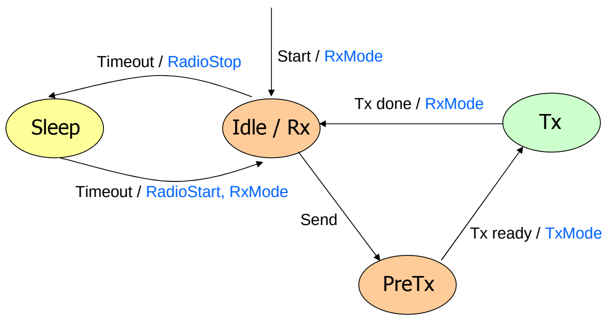


Figure 4.16: The radio state diagram and log messages.

However, the trace method we employ is slightly different from the one in PowerTOSSIM. The main difference is that our method determines the energy consumption by the mote during the experiments while PowerTOSSIM generates the log messages form the simulation and calculate the the power consumption off-line. The reason is that the mote does not have enough memory space to store all log messages generated during the entire experiment period. When the mote changes its radio state changes, instead of generating log messages, we compute the radio operation times online after saving the current timestamp. For example, when the radio state changes from Tx to Idle, we can compute the time duration that the radio stayed in the Tx state. With two timestamps (current and previous), the event type, and previous radio state, we can determine time spent in the previous state. The mote transmits the cumulative radio operation times in DATA packets at each sampling time.

Figure 4.17 shows the network topology used during the experiment. The solid black lines represent the routing paths and dotted red lines indicate a neighboring relation between nodes. In this topology, node 1, 2, and 4 can hear packets from all nodes while node 3, 5, 6, and 7 only can hear packets from node 1, 2, and 4.

4.4.2 Experimental results

We measure the energy consumption from Mica 2 mote for 802.11 basic, 802.11 PS mode, S-MAC, and B-MAC and compare the experimental results with the analytical results.

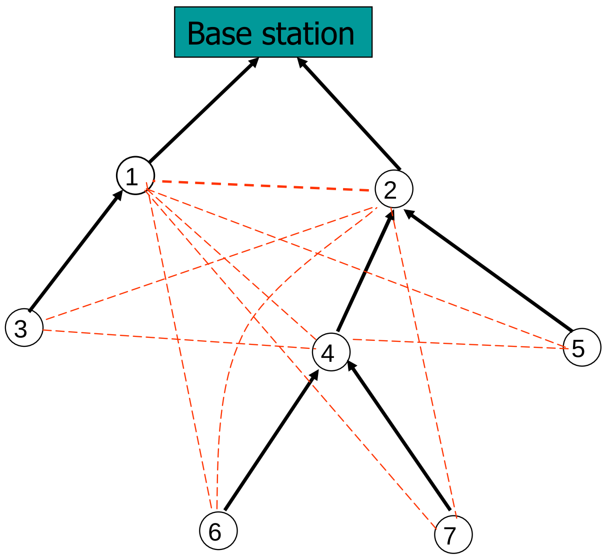


Figure 4.17: The experimental topology.

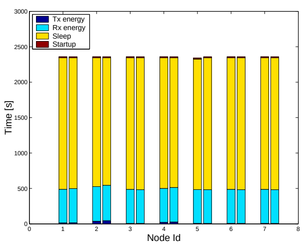
Experimental results of S-MAC, 802.11 power-saving, and 802.11 basic mode

We used the latest version of S-MAC from [67] to measure the power consumption. In addition to the basic functionality described in Section 4.2.6, S-MAC adopts the adaptive listening technique to reduce the end-to-end delay in multihop networks. S-MAC also uses the NAV to minimize the power consumption caused by overhearings. In this experiment we disable those features.

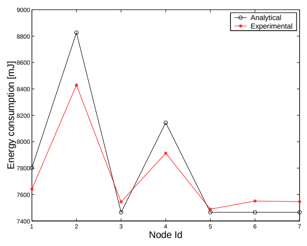
The S-MAC implementation uses $115ms$ listening period. We set the duty cycle and the sampling period to 16.7% and $20s$, respectively. Accordingly, the frame length was set to $690ms$. Each node sends total 120 packets to the base station, therefore, total experiment time is $2400s$.

In the analytical model, we did not consider the energy consumption spent on accessing the channel such as contention window, DIFS (distributed (coordination function) interframe space), and SIFS (short interframe space). For the comparison with the experimental results, we add those energy consumptions to the analytical model, for a total of about $31ms$ for each packet transmission. S-MAC uses a guard time at the end of each listen interval. We also add the guard time ($8ms$ in S-MAC) to the analytical model in the comparison.

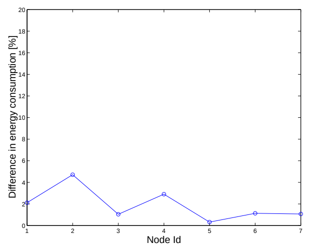
Figures 4.18 (a) and (b) shows the comparison of the time spent on each radio operation and total energy consumption of S-MAC using analytical modeling and experiments. We observe that the



(a) Time spent on radio operations for experiment (left) and analysis (right)

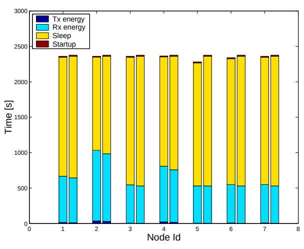


(b) Total energy consumption

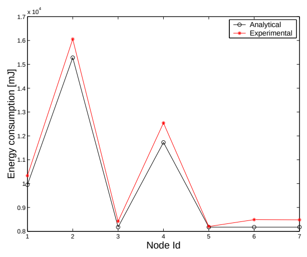


(c) Total energy difference in percentage

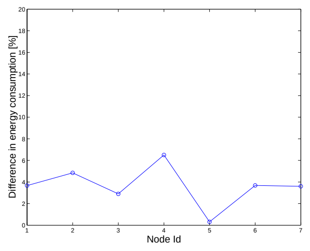
Figure 4.18: Comparison of time and energy consumption of S-MAC between the analytical and experimental results.



(a) Time spent on radio operations for experiment (left) and analysis (right)



(b) Total energy consumption



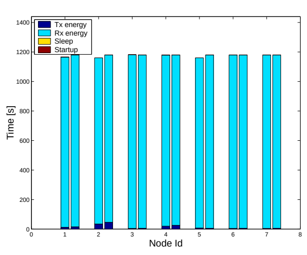
(c) Total energy difference in percentage

Figure 4.19: Comparison of time and energy consumption of 802.11 power-saving mode between the analytical and experimental results.

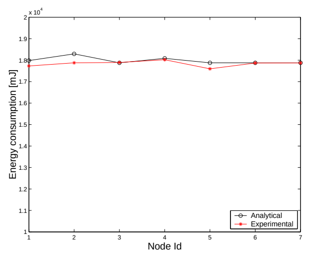
experimental result are very similar to the analytical modeling. The difference between the experimental and analytical results are less than 5% (Figure 4.18 (c)).

The basic operation of 802.11 power-saving mode is the same as S-MAC except that the radio stays awake after the packet transmission in 802.11 power-saving mode (Figure 4.3). Thus, we modified the S-MAC implementation accordingly for the experiment. We also use the same parameters as those in S-MAC experiment. The comparison results in Figures 4.19 (a), (b), and (c) show that the energy consumption of the analytical model is very close to that of the experiment.

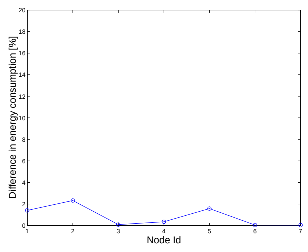
For the experiment of 802.11 basic mode, we use the S-MAC implementation with NO_SLEEP_CYCLE option since the basic operation of 802.11 basic mode is similar to that of S-MAC except that the radio always stays awake when there are no transmission or reception operations. From the Fig. 4.20 (a), (b), and (c) we can conclude that there are no significant differences between analytical model and the



(a) Time spent on radio operations for experiment (left) and analysis (right)

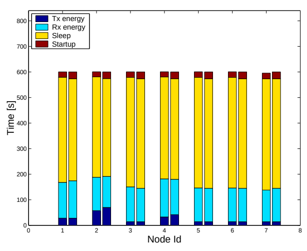


(b) Total energy consumption

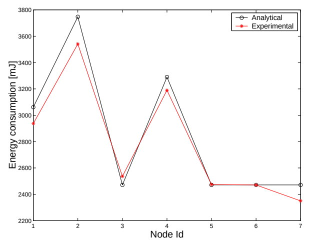


(c) Total energy difference in percentage

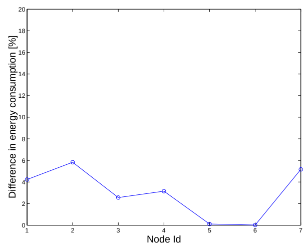
Figure 4.20: Comparison of time and energy consumption of 802.11 basic mode between the analytical and experimental results.



(a) Time spent on radio operations for experiment (left) and analysis (right)



(b) Total energy consumption



(c) Total energy difference in percentage

Figure 4.21: Comparison of time and energy consumption of B-MAC between the analytical and experimental results.

experiment for energy consumption of 802.11 basic mode (less than 3%).

Experimental results of B-MAC

We used the TinyOS 1.1.15 for B-MAC experiment. The preamble length and the time for clear channel assessment t_{CCA} were set to 250 bytes and $8ms$, respectively. The sampling period is $5ms$. Each node transmits 120 packets to the base station during the experiment.

Figures 4.21 (a), (b), and (c) show the comparison of energy consumption for analytical model and the experiment. Like in other experimental results, the energy consumption of the analytical model of B-MAC is very close to the experimental one.

4.5 Related work

It is difficult to measure the energy consumption of an MAC protocol. The most realistic method to measure the energy consumption is to deploy the motes with a specific MAC protocol in real environment and determine the lifetime of the network. However, this method is not practical because it may take a very long time (i.e., more than a year), for the network to expire. The measurements can be expedited by performing the measurement for a short period of time and determining the remaining energy of the batteries. Another drawback is that the environment where the motes are deployed may affect energy consumption. Finally, this method cannot differentiate between different sources of power consumption, measuring the total energy consumption of the mote including CPU, memory access, sensing, and radio operations.

Another method of measuring power consumption is to directly measure current consumption of the mote in each state. This method accurately measures the total power consumption of the mote in a controlled environment. By selectively powering different hardware components we can determine the contribution of each of them.

Trace-based modeling and offline processing are also widely used to measure the power consumption of hardware components [66]. This approach tracks the energy consumption of each hardware component of the simulated motes by generating the corresponding messages.

Finally, analytical models can be used to estimate the energy consumptions of MAC protocols. The main advantage of analytical models is their flexibility and insight they offer. Given the power consumption of each radio state, we can easily determine the power consumption of each MAC protocol. Some of proposed MAC protocols proposed provide analytical models for their power consumptions. However, each analysis is performed in isolation based on different assumptions making the comparison of the protocols difficult. To the best of our knowledge, our work is the first attempt to a comprehensive, fair comparison of modeling power consumptions of existing MAC protocols in WSNs.

The main disadvantages of analytical modes are inevitable simplifying assumptions. To asses the impact of these assumptions we validate our modes on a real hardware platform and found that the models closely match reality.

4.6 Energy Efficient Scheduling for Wireless Sensor Networks

The best way to minimize the power consumption of sensor nodes is to have the nodes sleep as long as possible and awake them only for doing their required work, such as sensing and data forwarding. In this section, we analyze a situation when collisions may occur in our node scheduling protocol (cross-layer scheduling) [63] and provide several solutions. Simulation results show that the power consumption of cross-layer scheduling is very close to that of the Ideal MAC protocol.

4.6.1 Overview of cross-layer scheduling

In our previous work, we proposed a cross-layer node scheduling for periodic traffic where sensor nodes dynamically create on-off schedules in such a way that the nodes will be awake only when needed and asleep the rest of the time [63]. The scheduling consists of two distinct phases for each flow in the network:

- **The setup and reconfiguration phase** is relatively short in comparison to the steady state phase. Its goal is to set up the schedules that will be used during the steady state phase.
- **The steady state phase** takes place between consecutive setup and reconfiguration phases. It utilizes the schedule established in the setup and reconfiguration phase to forward the data to the base station.

During the route setup, a special route-setup packet is sent along the route from the source node of the flow to the base station. Intermediate nodes along the path of route setup packet find a time slot when a data packet can be scheduled without colliding with other nearby transmissions. The scheduling scheme utilizes RTS/CTS protocol to ensure that only non-contending transmissions are scheduled at the same time.

The simulation results showed that the network lifetime using the approach is order of magnitude longer than that of the case without scheduling. The approach consumed only about one tenth of 802.11 power saving mode.

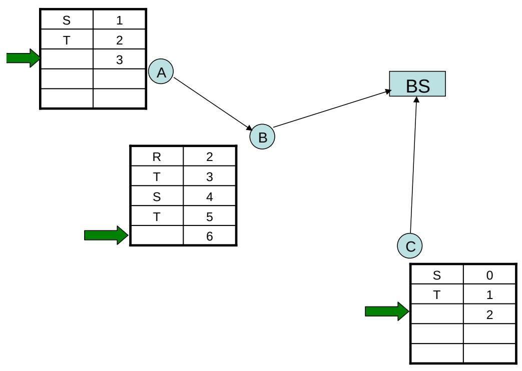
4.6.2 Recovery from Scheduling Collisions

As pointed out in [68], in some cases, the presented protocol may result in schedules that consistently result in DATA packet collisions. At the hearts of this conflict is the hidden terminal problem. Figure 4.22(a) shows such a situation when the proposed scheduling protocol can fail. In Fig. 4.22(a), notes A, B and C are in steady state. At the end of their scheduled jobs, nodes B and C wait for RSETUP packets from other nodes during time periods 6 and 2, respectively. In Fig. 4.22(b), a new node D joins the network and tries to establish a new schedule. Node D broadcasts a RSETUP packet at time 2 and node C correctly receives it, ACKs it, then forwards it to the base station. In Fig. 4.22(c), after the RACK reply from the base station nodes A and D have a conflicting sending schedule, as they are both scheduled to send at time 2, their packets colliding at node B (although D's packet is correctly received at node C). At the core of this problem is the fact that node A's transmission is hidden from the new node D.

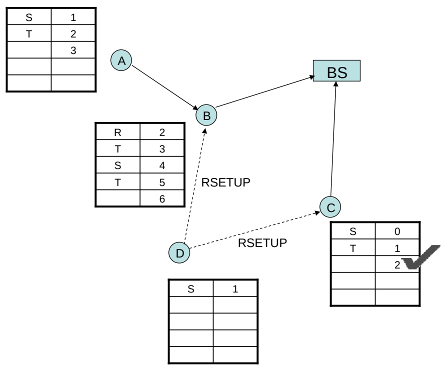
There are at least three different methods we can use to solve this problem. The first, and simplest, is to detect the disturbance of a data flow and trigger a new schedule discovery. Thus, if several consecutive data packets are lost on an otherwise reliable route, another route schedule should be sought (through RSETUP). Notice that in Fig. 4.22, if node A joins after node D, it cannot schedule its data packets in slot 2, as node B hears the transmission from D and would not reply to a request in that slot. This is likely the best solution, especially for sensor networks with low duty cycle, and, thus, with sparse schedule tables. In those networks, it is unlikely that the newly schedule will conflict with another flow.

A second solution is to use a MAC protocol that prevents the hidden terminal problem in the steady state phase as well. An RTS/CTS mechanism would certainly work, although it may be inefficient due to the extra control information. A better solution is the use of a MACA-BI [69] type of MAC protocol, that foregoes the use of RTS. In this version, the receivers send a CTS before every data transfer. Thus, in every transmission slot, the receiver sends a CTS (that effectively will protect the receiver from hidden terminal collisions) and the sender follows immediately (in the same slot) with the DATA packet. The main disadvantage of this solution is the increase in control overhead introduced by the extra CTS packet that has the be sent for every DATA packet.

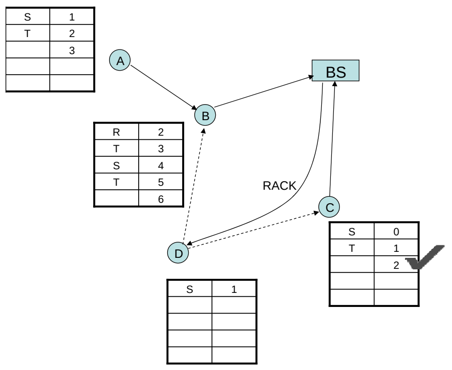
Finally, a slightly more comples solution involves broadcasting the schedules of all nodes to



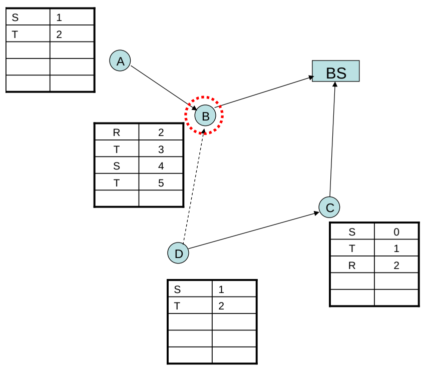
(a) The initial topology



(b) A new node D joins

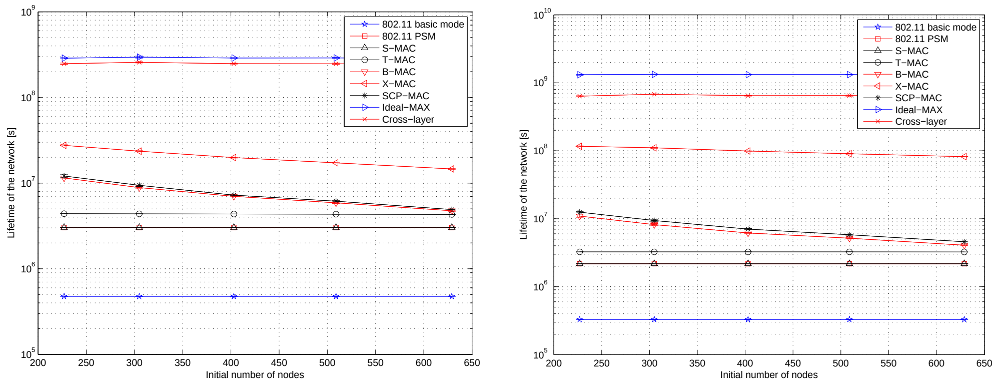


(c) Route setup through node C



(d) Collision at node B

Figure 4.22: An example of DATA packet collision.



(a) Mica 2 (b) Telos

Figure 4.23: Lifetime of the network as a function of the number of sensor nodes for Mica 2 and Telos motes (at constant network area).

all their neighbors. Thus, incoming nodes, before attempting to send RSETUP packets, will first make sure that none of their neighbors are receiving DATA packets on the slots they try to setup flows. This solution is a bit more involved than the other two, but it has the advantage to work even in networks with high duty cycles and have a relatively low overhead (the schedules do not have to be broadcasted in every cycle).

4.6.3 Simulation results

We setup the same simulation environment as in Section 4.3.1. One of important parameter in the cross-layer scheduling is the time synchronization error [63]. We set the time synchronization error to $10ms$ while most time synchronization protocols including the Tiny-sync in Chapter 3 can provide the time synchronization errors of less than $1ms$. Figures 4.23 (a) and (b) show the lifetime of the network with cross-layer scheduling in comparison with WSN MAC protocols. The performance of the cross-layer scheduling approach is much higher than that of other MAC protocols and close to the one of the Ideal MAC.

4.7 Conclusion

In this chapter, we analyze the power efficiency of several well known MAC protocols for WSNs. Not surprisingly, the MAC protocols specifically designed for WSNs perform significantly better than 802.11 (both with and without power savings enabled). Among the WSN MACs, preamble-based protocols such as B-MAC and X-MAC, show better performance in terms of energy consumption when the wake-up interval is relatively small, while synchronization-based protocols such S-MAC and T-MAC outperform preamble-based protocols when the wake-up interval is large. Thus, preamble-based MAC protocols can be used for applications where the delay, in addition to the energy consumption, is considered a major performance factor. The synchronization-based protocols are appropriate for applications that do not have strict delay constraints.

To validate the analytically derived formulae, we also measured the energy consumption for several MAC protocols using Mica 2 and Telos motes. The experimental results closely match the analytical models.

Chapter 5

Conclusion

In this dissertation, we presented an analysis of the lifetime of wireless sensor networks that employ periodic sensing. Lower and upper bounds on the network lifetime are derived, and the corresponding routing algorithms achieving these bounds are presented. For large sensor networks the upper and the lower bounds on the network lifetime are relatively close (less than a few percents), leading thus to the conclusion that for such sensor networks the choice of the routing protocol is largely irrelevant for the network lifetime, as long as some form of shortest paths are followed. Simulations are used to validate the theoretical results. While the set of routing algorithms may seem to be restricted, our results are likely to hold for all sensible routing approaches.

We also proposed and evaluated the performance of two simple time synchronization algorithms suitable for wireless sensor networks. The algorithms perform pair-wise synchronization and can be used as the basic building block for synchronizing an entire network. While the performance of the two algorithms may vary in theory, in practice, they yield very similar results. Thus, the simplest one, tiny-sync, is likely to be practically implemented. The main advantage of tiny-sync is its simplicity and parsimonious resources requirements. Other advantages include robustness to large variations in clock drift and its ability to achieve fast synchronization of an entire network. Experimental results show that it performs as well or better than existing time synchronization approaches for wireless sensor networks.

Finally, we constructed power consumption models at MAC layer of WSNs. We derived analytically the power consumption of well known WSN MAC protocols and analyzed and compared their performance. Validation results using Mica 2 motes show that the analytical models closely match the experimental results.

Bibliography

[1] M. Buettner, G. V. Yee, E. Anderson, and R. Han, “X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks,” in *Proc. of the 4th ACM Conference on Embedded Network Sensor Systems (Sensys’06)*, 2006.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” *IEEE Communication Magazine*, vol. 40, pp. 102–116, Aug. 2002.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” *IEEE Computer*, vol. 38, pp. 393–422, Mar. 2002.

[4] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy, “Wireless integrated network sensors: Low power systems on a chip,” in *Proc. of the 24th European Solid-State Circuits Conference*, (The Hague, Netherlands), 1998.

[5] F. Zhao and L. Guibas, “Wireless sensor networks: an information processing approach,” *Elsivier*, 2004.

[6] V. A. Kottapalli, A. S. Kiremidjiana, J. P. Lyncha, E. Carryerb, T. W. Kennyb, K. H. Lawa, and Y. Leia, “Two-tiered wireless sensor network architecture for structural health monitoring,” in *Proc. 10th Annual International Symposium on Smart Structures and Materials*, (San Diego, CA), March 2003.

[7] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler, “Wireless sensor networks for habitat monitoring,” in *Proc. of the First ACM International Workshop on Wireless Sensor Networks and Applications*, (Atlanta, GA), Sept. 2002.

[8] B. M. Sadler, “Fundamentals of energy-constrained sensor network systems,” *IEEE A&E Systems Magazine*, vol. 20, pp. 17–35, Aug. 2005.

[9] J. Polastre, R. Szewczyk, and D. Culler, “Telos: Enabling ultra-low power wireless research,” in *In Proceedings of IPSN/SPOTS*, (Los Angeles, CA), Apr. 2005.

[10] S. Singh, M. Woo, and C. S. Raghavendra, “Power-aware routing in mobile ad hoc networks,” in *Mobile Computing and Networking*, pp. 181–190, 1998.

[11] W. Heinzelman, J. Kulik, and H. Balakrishnan, “Adaptive protocols for information dissemination in wireless sensor networks,” in *Proc. MOBICOM*, pp. 174–185, 1999.

[12] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy-efficient communication protocol for wireless microsensor networks,” in *HICSS*, 2000.

[13] J. Aslam, Q. Li, and D. Rus, “Three power-aware routing algorithms for sensor networks,” *Wireless Communications and Mobile Computing*, vol. 2, pp. 187–208, Mar. 2003.

[14] V. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff, “A minimum cost heterogeneous sensor network with a lifetime constraint,” *IEEE TMC*, vol. 4, pp. 4–15, Jan. 2005.

[15] M. Bhardwaj, T. Garnett, and A. P. Chandrakasan, “Upper bounds on the lifetime of sensor networks,” in *Proc. of ICC*, pp. 785–790, 2001.

[16] M. Bhardwaj and A. P. Chandrakasan, “Bounding the lifetime of sensor networks via optimal role assignments,” in *Proc. of INFOCOM*, pp. 1587–1596, 2002.

[17] H. Zhang and J. Hou, “On deriving the upper bound of -lifetime for large sensor networks,” in *Proc. of MOBIHOC*, pp. 121–132, May 2004.

[18] J. Pan, Y. T. Hou, L. Cai, Y. Shi, and S. X. Shen, “Topology control for wireless sensor networks,” in *Proc. of MobiCom*, Sept. 2003.

[19] J. Alonso, A. Dunkels, and T. Voigt, “Bounds on the energy consumption of routings in wireless sensor networks,” in *Proc. of WIOPT*, 2004.

[20] A. Woo and D. E. Culler, "A transmission control scheme for media access in sensor networks," in *Proc. ACM MOBICOM*, July 2001.

[21] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. of Infocom 2002*, pp. 1567–1576, USC/Information Sciences Institute, 2002.

[22] S. Singh and C. Raghavendra, "Power efficient MAC protocol for multihop radio networks," in *The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 153–157, 1998.

[23] R. Rozovsky and P. R. Kumar, "SEEDEX: A MAC protocol for ad hoc networks," in *Proc. of The ACM Symposium on Mobile Ad Hoc Networking & Computing, (MobiHoc)*, (Long Beach, CA), pp. 67–75, Oct. 2001.

[24] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks," in *Proc. of INFOCOM*, vol. 1, pp. 200–209, 2002.

[25] M. Stemm and R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," *IEICE Transactions on Communications*, vol. E80-B, no. 8, pp. 1125–1131, 1997.

[26] L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proc. of IEEE INFOCOM*, pp. 1548–1557, 2001.

[27] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A network-centric approach to embedded software for tiny devices," in *Proc. of the First International Workshop on Embedded Software (EMSOFT)*, Oct. 2001.

[28] S. Singh and C. Raghavendra, "PAMAS: Power aware multi-access protocol with signalling for ad hoc networks," *ACM Computer Communications Review*, 1999.

[29] E.-S. Jung and N. Vaidya, "A power saving MAC protocol for wireless networks." Technical Report, UIUC, July 2002.

[30] M. L. Sichitiu and R. Dutta, "Benefits of multiple battery levels for the lifetime of large wireless sensor networks," in *Proc. of Networking 2005*, (Waterloo, Canada), May 2005.

[31] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, “A wireless sensor network for structural monitoring,” in *SenSys ’04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 13–24, ACM Press, 2004.

[32] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, “Delay efficient sleep scheduling in wireless sensor networks,” in *Proc. of Infocom 2005*, Mar. 2005.

[33] D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Trans. Communications*, vol. 39, pp. 1482–1493, Oct. 1991.

[34] D. L. Mills, “Improved algorithms for synchronizing computer network clocks,” in *Proc. of ACM Conference on Communication Architectures (ACM SIGCOMM’94)*, (London, UK), Aug. 1994.

[35] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*. Springer-Verlag, 4th ed., 1997.

[36] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, “Sensor network-based countersniper system,” in *SenSys ’04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, ACM Press, 2004.

[37] F. Sivrikaya and B. Yener, “Time synchronization in sensor networks: A survey,” *IEEE Network*, vol. 18, pp. 45–50, July 2004.

[38] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, “Clock synchronization in wireless sensor networks: A survey,” *Ad-Hoc Networks*, vol. 3, pp. 281–323, May 2005.

[39] J. Elson and D. Estrin, “Time synchronization for wireless sensor networks,” in *Proc. of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, (San Francisco, CA), Apr. 2001.

[40] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in *UCLA Technical Report 020008*, Feb. 2002.

[41] D. E. Richard Karp, Jeremy Elson and S. Shenker, “Optimal and global time synchronization in sensornets,” Tech. Rep. 0012, CENS, 2003.

[42] L. D. Girod, *A Self-Calibrating System of Distributed Acoustic Arrays*. PhD thesis, University of California Los Angeles, 2005.

[43] K. Römer, “Time synchronization in ad hoc networks,” in *Proc. of ACM Mobihoc*, (Long Beach, CA), 2001.

[44] L. Meier, P. Blum, and L. Thiele, “Internal synchronization of drift-constraint clocks in ad-hoc sensor networks,” in *MobiHoc*, 2004.

[45] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in *Proc. of the First ACM Conference on Embedded Networked Sensor System (SenSys)*, pp. 138–149, Nov. 2003.

[46] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, “System architecture directions for networked sensors,” in *Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, 2000.

[47] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, “The flooding time synchronization protocol,” in *SenSys ’04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, ACM Press, 2004.

[48] J. V. Greunen and J. Rabaey, “Lightweight time synchronization for sensor networks,” in *WSNA’03*, 2003.

[49] Q. Li and D. Rus, “Global clock synchronization in sensor network,” in *INFOCOM*, 2004.

[50] H. Dai and R. Han, “Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 125–139, 2004.

[51] S. Ping, “Delay measurement time synchronization for wireless sensor networks.” In Intel Research, IRB-TR-03-013, June 2003.

[52] S. PalChaudhuri, A. Saha, and D. B. Johnson, “Adaptive clock synchronization in sensor networks,” in *ISPN ’04: Proceeding of The Third International Symposium on Information Processing in Sensor Networks*, pp. 340–348, Apr. 2004.

[53] W. Su and I. F. Akyildiz, “Time-diffusion synchronization protocol for wireless sensor networks,” *IEEE/ACM Transactions on Networking*, vol. 13, pp. 384–397, Apr. 2005.

[54] F. Cristian, “Probabilistic clock synchronization,” *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.

[55] M. Lemmon, J. Ganguly, and L. Xia, “Model-based clock synchronization in networks with drifting clocks,” in *Proc. of the 2000 Pacific Rim International Symposium on Dependable Computing*, (Los Angeles, CA), pp. 177–185, Dec. 2000.

[56] Crossbow, “Mica2 wireless measurement system.” http://www.xbow.com/.

[57] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, “Implementation of flooding time synchronization protocol,” 2005.

[58] W. Ye, J. Heidemann, and D. Estrin, “Medium access control with coordinated adaptive sleeping for wireless sensor networks,” *IEEE/ACM Transactions on Networking*, vol. 12, June 2004.

[59] T. van Dam and K. Langendoen, “An adaptive energy-efficient MAC protocol for wireless sensor networks,” in *Proc. of of the 1st ACM Conference on Embedded Network Sensor Systems (Sensys’03)*, 2003.

[60] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks,” in *Proc. of the 2nd ACM Conference on Embedded Network Sensor Systems (Sensys’04)*, 2004.

[61] W. Ye, F. Silva, and J. Heidemann, “Ultra-low duty cycle mac with scheduled channel polling,” in *Proc. of the 4th ACM Conference on Embedded Network Sensor Systems (Sensys’06)*, 2006.

[62] IEEE, “Wireless LAN medium access control (MAC) and physical layer (PHY) specification.” IEEE Std. 802.11, June 1999.

[63] M. L. Sichitiu, “Cross-layer scheduling for power efficiency in wireless sensor networks,” in *Proc. of Infocom 2004*, vol. 3, (Hong Kong, PRC), pp. 1740–1750, Mar. 2004.

[64] V. T. D. Brooks and M. Martonosi, “Wattch: a framework for architectural-level power analysis and optimizations,” in *ISCA*, pp. 83–94, 2000.

[65] J. Flinn and M. Satyanarayanan, “Powerscope: a tool for profiling the energy usage of mobile applications,” in *Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2–10, Feb. 1999.

[66] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, “Simulating the power consumption of largescale sensor network applications,” in *In Proceedings of SenSys’04*, (Baltimore, Maryland), Nov. 2004.

[67] “S-mac software: Information and source code,”

[68] M. J. Miller and N. H. Vaidya, “On-demand tdma scheduling for energy conservation in sensor networks.” Technical Report, UIUC, 2004.

[69] F. Talucci, M. Gerla, and L. Fratta, “MACA–BI (MACA by invitation): A receiver oriented access protocol for wireless multiple networks,” in *PIMRC ’97*, (Helsinki, Finland), Sept. 1997.

Appendix

Appendix A

Proof of Theorem 4

In this section we provide the proof for Theorem 4. Let us denote with (A, B) the line determined by the points A and B and with $m(A, B)$ the slope of the line (A, B). We start by proving the following lemma:

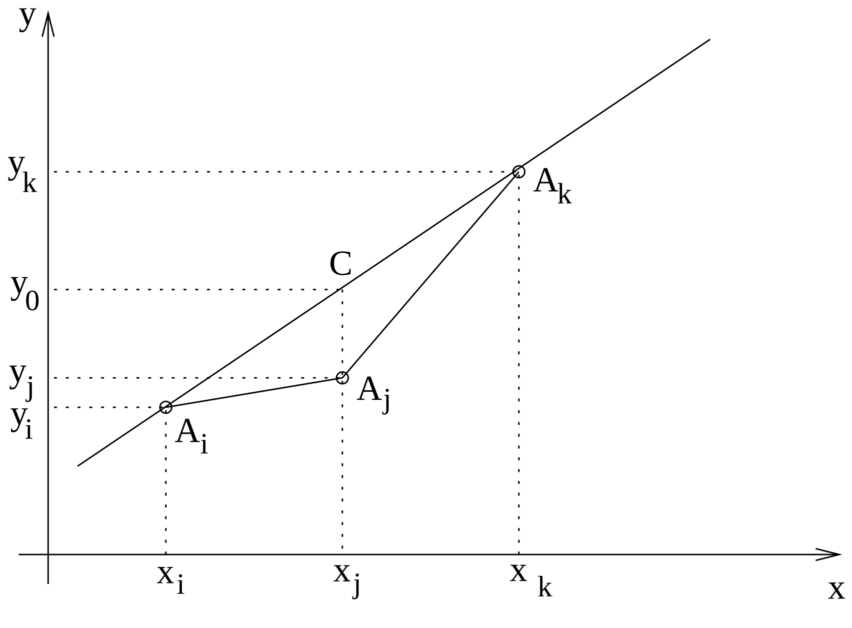


Figure A.1: Illustration for Lemma 7.

***Lemma* 7** *Consider three points A_i, A_j, A_k of coordinates (x_i, y_i), (x_j, y_j) and (x_k, y_k) respectively* *with $x_i < x_j < x_k$. Then*

$$m(A_i, A_j) < m(A_i, A_k) < m(A_j, A_k), \tag{A.1}$$

if and only if A_j is below the line (A_i, A_k). Similarly,

$$m(A_i, A_j) > m(A_i, A_k) > m(A_j, A_k), \tag{A.2}$$

if and only if A_j is above the line (A_i, A_k).

Proof Denote with y_0 the y-coordinate of point C where the vertical line from A_j intersects the line (A_i, A_k):

$$y_0 = \frac{(y_k - y_i)x_j + y_i x_k - y_k x_i}{x_k - x_i}. \tag{A.3}$$

The expression "A_j is below the line (A_i, A_k)" is shorthand for $y_j < y_0$. With this notation, if and only if $y_j < y_0$, then

$$m(A_i, A_j) = \frac{y_j - y_i}{x_j - x_i} < \frac{y_0 - y_i}{x_j - x_i} = m(A_i, A_k) = \frac{y_k - y_0}{x_k - x_j} < \frac{y_k - y_j}{x_k - x_j} = m(A_j, A_k), \tag{A.4}$$

which is (A.1). By a similar argument it can be shown that (A.2) holds if and only if $y_j > y_0$. ■

Theorem 4 *Any constraint A_j which satisfies*

$$m(A_i, A_j) \leq m(A_j, A_k) \tag{A.5}$$

for at least one set of integers $1 \leq i < j < k$ can be safely discarded as it will never constrain the bounds $\underline{a_{12}}, \overline{a_{12}}, \underline{b_{12}}$ and $\overline{b_{12}}$ more than any existing constraints.

Proof Consider the situation depicted in Fig. A.2. The constraints A_i, A_j and A_k satisfy condition (A.5). Consider a constraint B_x of coordinates (t_{2_x}, t_{1_x}) such that $t_{1_x} > t_{1_k}$ and $t_{2_x} > t_{2_k}$. Constraint B_x and one of the constraints A_i, A_j or A_k may determine the upper bound $\overline{a_{12}}$). Denote with $\overline{a_{12}}(A_y, B_x)$ the upper bound on a_{12} determined by the constraints A_y and B_x. We can distinguish two cases:

1. When B_x is *below* the line (A_i, A_k) (e.g., position B_{x_1} in Fig. A.2). In this case, because both A_j and B_{x_1} are under the line (A_i, A_k), A_k is above the line (A_j, B_{x_1}). From lemma 7 it follows that

$$m(A_k, B_{x_1}) < m(A_j, B_{x_1}), \tag{A.6}$$

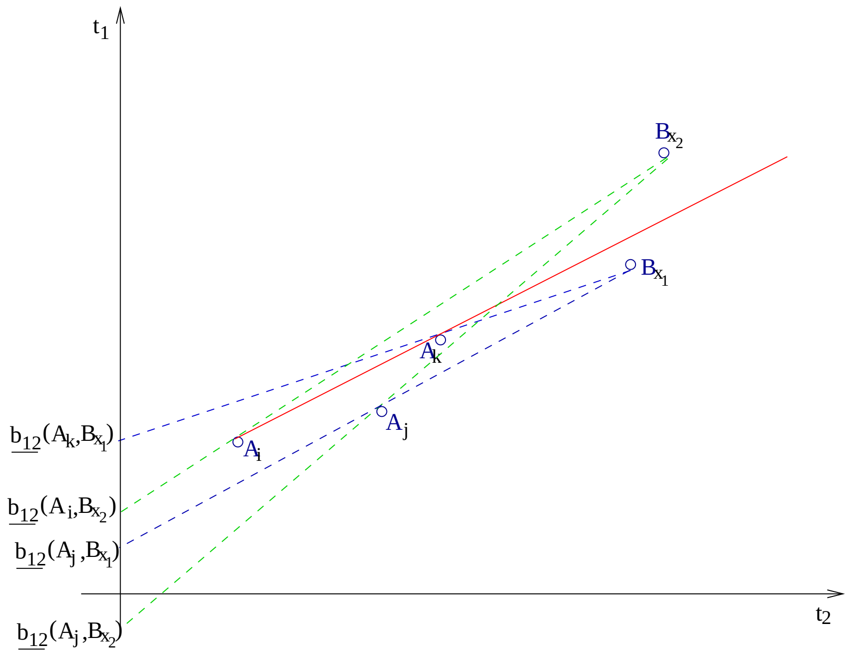


Figure A.2: A_j can be safely discarded as A_i or A_k will result in better estimates in all cases.

which is equivalent with

$$\overline{a_{12}}(A_k, B_{x_1}) < \overline{a_{12}}(A_j, B_{x_1}). \quad \text{(A.7)}$$

Thus, the constraint (A_k, B_x) is tighter than the constraint (A_j, B_x) for the case when B_x is below the line (A_i, A_k).

2. When B_x is *above* the line (A_i, A_k) (position B_{x_2} in Fig. A.2). In this case, by a similar logic, it can be shown that

$$\overline{a_{12}}(A_i, B_{x_2}) < \overline{a_{12}}(A_j, B_{x_2}), \quad \text{(A.8)}$$

i.e., the constraint (A_i, B_x) is tighter than the constraint (A_j, B_x) for the case when B_x is above the line (A_i, A_k)

Thus, in both cases, one of the upper bounds $\overline{a_{12}}(A_i, B_{x_2})$ or $\overline{a_{12}}(A_k, B_{x_1})$ is tighter than the upper bound introduced by A_j, $\overline{a_{12}}(A_j, B_x)$, showing that constraint A_j will always results in a looser upper bound than A_i or A_k, and, thus, enabling its disposal without any possible reduction in optimality of the solution. $\blacksquare$