Energy Efficient Target Tracking in Wireless Sensor Networks: Sleep Scheduling, Particle Filtering, and Constrained Flooding

Bo Jiang

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Binoy Ravindran, Chair
Y. Thomas Hou
Mark T. Jones
Tom Martin
Anil Vullikanti

December 3, 2010
Blacksburg, Virginia

Keywords: Wireless Sensor Network, Target Tracking, Energy Efficiency, Sleep Scheduling, Particle Filters, Constrained Flooding

Energy Efficient Target Tracking in Wireless Sensor Networks: Sleep Scheduling, Particle Filtering, and Constrained Flooding

Bo Jiang

(ABSTRACT)

Energy efficiency is a critical feature of wireless sensor networks (WSNs), because sensor nodes run on batteries that are generally difficult to recharge once deployed. For target tracking—one of the most important WSN application types—energy efficiency needs to be considered in various forms and shapes, such as idle listening, trajectory estimation, and data propagation. In this dissertation, we study three correlated problems on energy efficient target tracking in WSNs: sleep scheduling, particle filtering, and constrained flooding.

We develop a Target Prediction and Sleep Scheduling protocol (TPSS) to improve energy efficiency for idle listening. We start with designing a target prediction method based on both kinematics and probability. Based on target prediction and proactive wake-up, TPSS precisely selects the nodes to awaken and reduces their active time, so as to enhance energy efficiency with limited tracking performance loss. In addition, we expand Sleep Scheduling to Multiple Target Tracking (SSMTT), and further reduce the energy consumption by leveraging the redundant alarm messages of interfering targets. Our simulation-based experimental studies show that compared to existing protocols such as Circle scheme and MCTA, TPSS and SSMTT introduce an improvement of $25\% \sim 45\%$ on energy efficiency, at the expense of only $5\% \sim 15\%$ increase on the detection delay.

Particle Filtering is one of the most widely used Bayesian estimation methods, when target tracking is considered as a dynamic state estimation problem for trajectory estimation. However, the significant computational and communication complexity prohibits its application in WSNs. We design two particle filters (PFs)—Vector space based Particle Filter (VPF) and Completely Distributed Particle Filter (CDPF)—to improve energy efficiency of PFs by reducing the number of particles and the communication cost. Our experimental evaluations show that even though VPF incurs 34% more estimation error than RPF, and CDPF incurs a similar estimation error to SDPF, they significantly improve the energy efficiency by as much as 68% and 90% respectively.

For data propagation, we present a Constrained Flooding protocol (CFlood) to enhance energy efficiency by increasing the deadline satisfaction ratio per unit energy consumption of time-sensitive packets. CFlood improves real-time performance by flooding, but effectively constrains energy consumption by controlling the scale of flooding—i.e., flooding only when necessary. If unicasting meets the distributed sub-deadline at a hop, CFlood aborts further flooding even after flooding has occurred in the current hop. Our simulation-based experimental studies show that CFlood achieves higher deadline satisfaction ratio per unit energy consumption by as much as 197%, 346%, and 20% than existing multipath forwarding protocols, namely, Mint Routing, MCMP and DFP respectively, especially in sparsely deployed or unreliable sensor network environments.

To verify the performance and efficiency of the dissertation's solutions, we developed a prototype implementation based on TelosB motes and TinyOS version 2.1.1. In the field experiments, we compared TPSS, VPF, CDPF, and CFlood algorithms/protocols to their respective competing efforts. Our implementation measurements not only verified the rationality and feasibility of the proposed solutions for target tracking in WSNs, but also

strengthened the observations on their efficiency from the simulation.

This work was supported by the Ministry of Knowledge Economy (MKE) of the Republic of Korea. [2008-F-052, Scalable/Mobile/Reliable Wireless Sensor Network Technology].

Dedication

This dissertation is dedicated to my family, who have been my companions throughout my long, Ph.D. journey, standing beside me all along.

Acknowledgments

In 2007, after working in the industry for three years, I found it very challenging to return to academia and resume research, especially when my project was just started and the problems were not well defined. Besides my hard work and commitment, I would not be able to obtain this Ph.D. degree without the great help and support from many people, to whom I would like to express my gratitude.

First, I would like to thank my advisor, Dr. Binoy Ravindran. Dr. Ravindran funded my education in the past three and a half years and made my Ph.D. dream come true. He not only provided me indispensable insights and invaluable advice on academic research, but also strengthened my professional skills in technical writing and presentations as well as result orientation. Most importantly, he was considerate to my work-life balance, providing me great flexibility in work. Thanks very much for his great support during my Ph.D. education!

Secondly, I am very grateful to the professors serving in my committee, including Dr. Thomas Hou, Dr. Mark T. Jones, Dr. Tom Martin, and Dr. Anil Vullikanti. I very much appreciate their instructions and invaluable advice on my research. I would also like to thank Dr. Hyeonjoong Cho, who is an alumnus of our Real-time Systems Laboratory and now working in Korea University. He helped me warming up at the early stage of my education, and secured the financial support for my dissertation research from ETRI, Korea.

Thirdly, I would like to thank all my colleagues in the Real-time Systems Laboratory, including Jonathan Anderson, Kai Han, Shouwen Lai, Bo Zhang, Guanhong Pei, Fei Huang, Piyush Garyali, Junwhan Kim, and Peng Lv. They offered me great help in my research and experiments and in my life as well.

Last but not the least, my whole-hearted thankfulness goes to my family. My parents made me a man with integrity, and they always respect my choices in my education and career, giving me ready help if needed. My wife, Peng Gao, has always been very supportive in my education. She encouraged me when I was under stress, kept the house in order when I was fully occupied by work, and most importantly, she brought me a lovely son and a warm home. I found all words too pale to express my appreciation and love for her.

This dissertation is dedicated to all the people who helped and supported me all the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks (WSNs) are increasingly being envisioned for collecting data, such as physical or environmental properties, from a geographical region of interest. WSNs are composed of a large number of low cost sensor nodes, which are typically capable of sensing, computing, and communication. The applications of WSNs can be found in diverse areas such as military (e.g., battlefield surveillance), environmental protection (e.g., habitat monitoring), healthcare (e.g., telemonitoring of human physiological data), and home automation [1].

Sensor nodes in a WSN constitute a wireless ad-hoc network, with one or a few sink nodes as the collection point(s) and bridge(s) to the base station(s). Every node in the network may create data periodically, on demand of base stations, or triggered by events [2]. At the same time, every node may forward data that it receives toward sink nodes, which are often multiple hops away. A typical multihop architecture of WSNs is shown in Figure 1.1 [3].

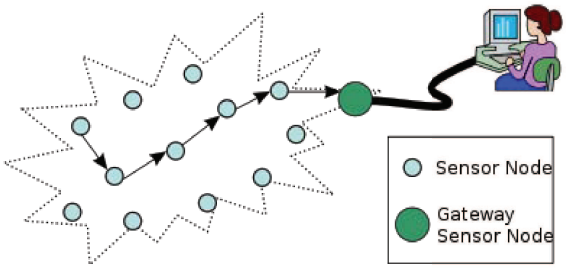


Figure 1.1: Typical multihop architecture of WSNs

A sensor node in WSNs usually consists of a microcontroller, memories, a radio transceiver, and sensing devices. All of these components are powered by portable power sources such as batteries [1]. For low cost purpose (so that they can be widely deployed), their power supply, and computing/communication capabilities are strictly constrained. Thus, resource constraints have to be considered carefully in the design of algorithms/protocols for WSNs.

1.1 Energy Efficient Target Tracking

Among the many diverse application domains of WSNs, tracking a mobile target (e.g., a human being or a vehicle) is one of the most significant ones. Tracking usually has more stringent performance requirements than detection: target detection studies discrete detection events at certain times and positions [4, 5]—i.e., it will succeed as long as there exists a single sensor node that can detect the target; however, a target tracking system is often required to ensure continuous monitoring and estimate the trajectory.

First, the continuity of monitoring mainly includes two aspects: 1) there always exist nodes that can detect the target along its trajectory, e.g., with low detection delay [6, 7] or high coverage level [8]; and 2) base stations can always receive the latest reports about the target in a timely manner, e.g., with high deadline satisfaction ratio for real-time data propagation [9].

Secondly, target tracking also needs to estimate the target trajectory based on detected samples, usually with small tracking error [10]. For this purpose, target tracking can be considered as a dynamic state estimation problem by modeling the state space as a stochastic process that evolves over time [11].

Therefore, the most stringent criterion of target tracking is to track with zero detection delay and 100% coverage, to propagate 100% of report messages to sink nodes within their deadlines, and to estimate the trajectory with zero error.

Since nodes often run on batteries that are generally difficult to be recharged once deployed, energy efficiency is a critical feature of WSNs for the purpose of extending the network lifetime. To date, many approaches have been explored to enhance energy efficiency, such as sleep scheduling [7], coverage or topology optimizing [12], and radio transmission power control [13]. Specifically for target tracking applications, energy efficiency needs to be enhanced with multiple approaches in various dimensions, to meet the requirements of a target tracking system. However, if energy efficiency is enhanced, the quality of service (QoS) of target tracking is highly likely to be negatively influenced. For example, forcing nodes to sleep can result in missing the passing target and lowering the tracking coverage. Therefore, energy efficient target tracking should improve the tradeoff between energy efficiency and tracking performance—e.g., by improving energy efficiency at the expense of a relatively small loss in tracking performance; achieving a better tracking performance per unit energy consumption than competing algorithms/protocols.

1.2 Motivations and Problem Statement

The requirements of a target tracking system—continuous monitoring and trajectory estimation—provide us space for enhancing energy efficiency. The typical sequence of operations of a target tracking system is as follows: the tracking application first detects the target at a

node (or a set of nodes) and makes local measurements of the target's movement status. Target properties, such as its latest position and velocity etc., are then estimated. These properties are then reported to sink nodes in a timely manner. In this problem space, we identify and focus on three directions for improving energy efficiency. These constitute the dissertation problems and are discussed as follows.

1.2.1 Sleep Scheduling

For target detection and measurement, idle listening is a major source of energy waste [14]. To reduce the energy consumption during idle listening so as to extend the network lifetime, duty cycling [15] is one of the most commonly used approaches. The idea of duty cycling is to put nodes in the sleep state for most of the time, and only wake them up periodically. In certain cases, the sleep pattern of nodes may also be explicitly scheduled, i.e., forced to sleep or awakened on demand. This is usually called sleep scheduling [16].

Despite the energy conservation, forcing nodes to sleep will probably result in target missing, thereby impairing the tracking performance. As one of the sleep scheduling approaches, proactive wake-up has been studied for awakening nodes proactively to prepare for the approaching target [17, 18]. However, most existing efforts about proactive wake-up simply awaken all the neighbor nodes in the area that the target is expected to arrive without any differentiation [8, 13, 17]. In fact, it is sometimes unnecessary to awaken all the neighbor nodes. Based on target prediction [18–20], it is possible to sleep-schedule nodes precisely, so as to reduce the energy consumption for proactive wake-up. For example, if nodes know the exact route of a target, it will be sufficient to awaken those nodes that cover the route during the time when the target is expected to traverse their sensing areas.

Thus, the first dissertation problem that we study is prediction-based sleep scheduling for proactive wake-up. By precisely predicting the target movement based on kinematics and probability, we reduce the number of proactively awakened nodes as well as their active time, so as to decrease the energy consumption. Moreover, energy efficiency can be improved further for multiple target tracking by leveraging the redundant alarm messages of interfering targets: when nodes have already been awakened for one of two interfering targets, the proactive wake-up effort and the corresponding energy consumption for the other may be saved partially or even completely.

1.2.2 Particle Filtering

For trajectory estimation, target tracking is usually considered as a dynamic state estimation problem, for which, particle filtering is one of the most widely used Bayesian estimation methods that approximate the optimal solution [21]. Particle filters are sequential Monte Carlo methods that estimate nonlinear and/or non-Gaussian dynamic processes. As a nu-

merical method, the posterior probability density function (pdf) of Bayesian estimation is represented with discrete samples (or particles) with associated weights. Then, in each iteration, PFs draw particles from a proposal distribution (or importance density), assign them with corresponding weights, normalize the weights, possibly resample, and finally make the estimation as the weighted average of these particles. Despite their attractive tracking performance and flexibility [22], the application of PFs in WSNs is very challenging, due to the conflict between the significant computational/communication complexity of PFs and constrained resources of WSNs. The complexity of existing particle filters is embodied in many aspects, among which three main problems are identified as follows:

1) *Number of particles.* It is shown in [23] that the complexity of particle filters is linear to the number of particles. Though it is often necessary to draw a significant number of particles to achieve a suboptimal posterior density, it will result in substantial computational and communication costs.

2) *Sample impoverishment problem.* This problem is introduced by resampling, when particles with high weights are selected many times and the diversity is lost [21]. But existing solutions to this problem are usually costly [22, 24].

3) *Centralization.* For a centralized particle filter, weight normalization and resampling require collection of data from multiple nodes to a single computational center (either a cluster head [25, 26] or a global transceiver/sink node [27, 28]). This will lead to significant communication cost, yielding low energy efficiency. Besides, centralized particle filters also introduce the following problems: a) the convergecast communication introduces a long delay, as the computational center has to receive messages in a sequential order; and b) a centralized implementation is vulnerable as a single point of failure [29].

The second dissertation problem that we study is to decrease the computational and communication cost of particle filters, and improve their energy efficiency accordingly. The dissertation addresses all the three aforementioned problems associated with PFs:

1) Based on vector space theory, we reduce the number of particles, i.e., reduce the problem scale;

2) We present an efficient solution to the sample impoverishment problem; and

3) We design a completely distributed particle filter to simplify the solution process and reduce the communication cost.

With these integrated efforts, energy efficiency of trajectory estimation for target tracking is improved significantly.

1.2.3 Constrained Flooding

For real-time data propagation, multipath forwarding is a commonly used redundant copy approach [30], which bypasses network congestion and node/link failures with multiple data copies transmitted along multiple paths. Many previous research efforts [31–36] have studied the efficiency of multipath forwarding protocols toward enhancing real-time performance while consuming optimized resources. Majority of these efforts have focused on reducing the number of flooding recipients at each hop so that the additional resource consumption can be minimized and QoS constraints such as real-time can also be satisfied. But even though the number of recipients is reduced, the redundancy introduced by this approach remains due to its probability-based recipient selection mechanism. In fact, it is possible that network congestion or disconnection is not significant enough throughout the entire path from source to sink to warrant multipath forwarding for each hop. Therefore, it is sometimes unnecessary to transmit redundant data copies generated by multipath forwarding protocols, which often consumes additional energy and bandwidth.

Furthermore, data aggregation has been widely used to reduce the redundancy during data propagation [37]. Whenever possible, combining multiple data packets into one will reduce the communication overhead, even if the aggregated data length is no less than the total length of data packets before the aggregation [38].

Thus, the third dissertation problem that we study is to improve energy efficiency of real-time data propagation with flooding. For those hops in which the connection status is good enough so that unicasting does work, redundant multipath flooding can be completely eliminated. The efficiency of flooding therefore can be further improved. For this purpose, we develop a constrained flooding protocol, which uses flooding to enhance deadline satisfaction ratio [9], and effectively improves energy efficiency by controlling the flooding scale and aggregating redundant data.

1.2.4 Prototype Implementation

Finally, we implement a prototype of the proposed algorithms and protocols designed for the three dissertation problems, and evaluate their efficiency with field experiments in a real deployment.

The implementation was motivated by the following three advantages of field experiments, compared to the simulation:

1) During the implementation, the rationality of the dissertation's algorithm/protocol design can be verified. For example, we need to guarantee each of the algorithms introduced in Section 4.4 can be executed on a single node, while this objective in itself can be achieved during the implementation.

2) With the implementation, we can verify that the complexity of the dissertation's solutions

is feasible for constrained resources of the actual hardware platform of motes, including computation, memory, and communication.

3) The implementation is much closer to the actual deployment of WSNs than the simulation. Thus, it can produce more convincing evaluation results than the simulation.

This prototype implementation used a network of 15 TelosB motes [39], with another two TelosB motes respectively as an emulated target and an interface node for data collection. The dissertation's algorithms and protocols were implemented using nesC programming language [40] on TinyOS version 2.1.1 [41].

1.3 Research Contributions

Our objectives in solving the three dissertation problems are as follows.

1) Reducing the energy consumption of proactive wake-up with limited increase in the detection delay;

2) Reducing the computational and communication complexity of particle filters with limited increase in the estimation error; and

3) Improving the deadline satisfaction ratio per unit energy consumption.

Given the above-mentioned objectives, the dissertation makes the following contributions.

First, we designed a target prediction and sleep scheduling protocol (TPSS) [42] to improve the energy efficiency with limited loss on the tracking performance. The target prediction scheme, which we developed based on both kinematics rules and theory of probability, not only predicts a target's next location, but also describes the probabilities with which it moves along all the directions. Unlike other physics-based prediction work [20], target prediction of TPSS provides a directional probability as the foundation of differentiated sleep scheduling in a geographical area. Based on the prediction results, TPSS enhances energy efficiency by reducing the number of awakened nodes and scheduling their sleep patterns in an integrated manner. The simulation-based experimental studies show that compared with existing protocols such as Circle scheme [8] and MCTA [13], TPSS introduces an improvement of 25% $\sim$ 45% in energy efficiency, at the expense of 5% $\sim$ 15% increase in detection delay. The implementation-based experimental results also show that compared to MCTA, TPSS achieves an improvement of 16.9% on energy efficiency, at the expense of only 4.1% increase on the escape distance percentage. In addition, we designed distributed algorithms for TPSS that can run on individual nodes. This enhances the scalability of TPSS for large-scale WSNs.

By extending TPSS to multiple targets, we developed an algorithm called sleep scheduling for multiple target tracking (SSMTT) [43]. Based on the efforts of TPSS, SSMTT enhances energy efficiency by leveraging the redundant alarm messages of interfering targets. In

addition, SSMTT provides a simple solution for the problem of missing alarm messages. The simulation-based experimental evaluations show that, beyond TPSS, SSMTT algorithm achieves $10\% \sim 15\%$ more improvement in energy efficiency during the interfering phase.

Then, based on vector space theory, we designed a Vector space-based Particle Filter (VPF) to reduce the problem scale of particle filters, and accordingly improve their energy efficiency. Based on the given tolerance for the tracking error, VPF reduces the number of particles and consequently reduces the computation overhead by partitioning the state space into cells and folding redundant particles in the same cell. This reduces the problem scale based on the density of cell partition. For the sample impoverishment problem, we introduced an efficient solution by distributing particles with high weights to neighbor cells. Since all the operations are built upon vector space theory, both particle number reduction and expansion involve less computational workload. This makes it feasible for the constrained resources of WSNs. In addition, vector space theory is independent from the dynamic system model, which provides a significant generality to particle filters. Our simulation-based experimental evaluations show that although VPF presents about 34% more estimation error than regularized particle filter (RPF) [22], it is significantly more efficient than RPF by as much as 68%, in terms of the execution time per iteration. Moreover, the implementation-based experimental results also show that compared to RPF, VPF achieves a reduction of 72.2% on the computation time per iteration, at the expense of an increase of 50.6% on the estimation error. Since the computation workload of VPF is less than that of RPF, VPF's energy efficiency on the computation is higher than RPF's.

For the particle filtering problem, we designed another improved method, namely completely distributed particle filter (CDPF), and further improved it with neighborhood estimation toward minimizing the communication cost. CDPF algorithm replaces centralization with a completely distributed implementation, by adjusting the order of four steps of generic particle filters, and leveraging the data aggregation during particle propagation. Moreover, we developed a neighborhood estimation method to replace the measurement broadcasting and the calculation of likelihood functions, so that the communication cost of DPFs can be minimized. Our simulation-based experimental evaluations show that even though CDPF incurs a similar estimation error to semi-distributed particle filter (SDPF) [27], its communication cost is lower than that of SDPF by as much as 90%. Moreover, the implementation-based experimental results also show that compared to SDPF, CDPF achieves a reduction of 89.0% on the communication cost, at the expense of an increase of 13.5% on the estimation error.

For the constrained flooding problem, we designed a constrained flooding protocol, called CFlood [44], to improve the energy efficiency of flooding by enhancing the deadline satisfaction ratio per unit energy consumption. CFlood improves real-time performance by flooding, but effectively constrains energy consumption by controlling the scale of flooding, i.e., flooding only when necessary. If unicasting meets the distributed sub-deadline of a hop, CFlood aborts further flooding even after flooding has occurred in the current hop. This is achieved by adding a plug-in block to the RTS/CTS handshaking mechanism of the CSMA/CA MAC protocol. In addition, CFlood aggregates data during the waiting period

for ABORT messages, and thus avoids increasing the end-to-end delay by specifically waiting for data packets to be aggregated. Our simulation-based experimental studies show that CFlood achieves higher deadline satisfaction ratio per unit energy consumption by as much as 197%, 346%, and 20% compared to existing multipath forwarding protocols, namely, Mint Routing [45], MCMP [30] and DFP [46] respectively, especially in sparsely deployed or unreliable sensor network environments. In addition, the implementation-based experimental results show that compared to Mint Routing and DFP, CFlood and its extension with data aggregation effort enable an increase of $48.6\% \sim 66.2\%$ and $7.85\% \sim 20.6\%$ on the deadline satisfaction ratio per unit energy consumption respectively.

Finally, we implemented a prototype system based on TelosB motes [39] and TinyOS version 2.1.1 [41] to evaluate the dissertation's solutions, including TPSS, VPF, CDPF, and CFlood. The experimental system consists of 15 motes in a 3×5 grid, an emulated target with a specifically designed mote, and a data collection system with a Java client as the user interface. We compared the dissertation's algorithms and protocols to the closest and latest competing efforts: TPSS was compared to Circle [8] and MCTA [13], VPF was compared to RPF [22], CDPF was compared to SDPF [27], and CFlood was compared to Mint Routing [45] and DFP [46]. The implementation results confirmed that the dissertation's solutions have similar performance and efficiency trends as those in the simulation, and that they are effective in a real WSN platform.

In summary, the dissertation's research contributions include:

- *Sleep scheduling.* We presented a prediction-based sleep scheduling protocol (i.e., TPSS) and its expansion for multiple target tracking (i.e., SSMTT). The target prediction scheme, which is based on both kinematics and probability, describes the probabilities with which it moves along different directions. Based on the prediction results, we improve the energy efficiency of proactive wake-up by reducing the number of awakened nodes and scheduling their sleep patterns to shorten the active time in an integrated manner. In addition, SSMTT leverages the redundant alarm messages of interfering targets to further improve the energy efficiency.

- *Particle filtering.* We designed two particle filters (i.e., VPF and CDPF) to improve the energy efficiency and facilitate the application of particle filters in sensor networks. VPF reduces the problem size of particle filtering by reducing the number of particles, and CDPF provides a completely distributed implementation of PFs. Both methods enhance energy efficiency significantly, by simplifying the computation and reducing the communication cost respectively.

- *Constrained flooding.* We developed a constrained flooding protocol (i.e., CFlood) to improve the flooding efficiency by aborting the flooding that has already occurred and utilizing unicasting whenever possible. Based on a simple modification to RTS/CTS handshaking used in MAC protocols, we enable nodes to know at an early stage if

unicasting can satisfy the real-time constraint or not. Data aggregation was also used as an auxiliary effort to enhance the flooding efficiency further.

- *Prototype implementation.* We implemented a prototype for most of the dissertation solutions and conducted field experiments. By successfully implementing the designs and running them on actual hardware platforms of motes, we verified the rationality of the dissertation's solutions, and the feasibility of applying our designs onto the constrained resources of actual motes. Most importantly, our implementation-based experiment results, which are more convincing than the simulation-based experiments, showed similar results as that in the simulation. This significantly strengthened the dissertation's contributions.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows. Related work is discussed in Chapter 2. In Chapter 3, we outline the basic preliminaries, including assumptions, models, and performance metrics. Then, we present TPSS protocol for single target tracking in Chapter 4, and SSMTT algorithm for multiple target tracking in Chapter 5. The design details of VPF and CDPF are described in Chapter 6 and Chapter 7, respectively. In Chapter 8, we present CFlood protocol. These chapters, i.e. Chapters $4 \sim 8$, also discuss simulation-based experiments that compare the proposed techniques against their state-of-the-art competitors. The prototype implementation and implementation measurements are detailed in Chapter 9. In Chapter 10, we conclude the dissertation and discuss the future work.

Chapter 2

Related Work

Focusing on the three dissertation problems, we mainly review related work on energy efficiency, target tracking and prediction, sleep scheduling, particle filtering, multipath forwarding and flooding.

2.1 Energy Efficiency

Since energy efficiency is a critical feature of WSNs, it has been extensively studied either independently or jointly with other features. Besides these efforts, energy efficiency is also considered as a constraint when designing algorithms or protocols for WSNs.

In [47], the authors proposed, analyzed and evaluated the energy consumption models in WSNs with probabilistic distance distributions to optimize grid size and minimize energy consumption accurately. The models were also used to study variable-size grids, which can further improve the energy efficiency by balancing the relayed traffic in wireless sensor networks.

Jung *et al.* presented two lifetime models that describe two of the most common working modes of sensor nodes: trigger-driven and duty-cycle driven [48]. The models use a set of hardware parameters such as power consumption per task, state transition overheads, and communication cost to compute a node's average lifetime for a given event arrival rate. Through comparison of the two models and a case study from a real camera sensor node design, the authors showed how the models can be applied to drive architectural decisions, compute energy budgets and duty-cycles, and to perform side-by-side comparison of different platforms.

[49] is an experimental effort for energy conservation based on real implementation. The authors specified the energy consumption in each stage of target tracking in a surveillance application. The corresponding conservation techniques are described in fine-grained details.

In [50], Sengul *et al.* explored the energy-latency-reliability tradeoff for broadcast in WSNs by presenting a new protocol called PBBF. Essentially, for a given reliability level, energy and latency were found to be inversely related and their study quantified this relationship at the reliability boundary. Therefore, PBBF offers an application designer considerable flexibility in the choice of desired operation points. Furthermore, the authors proposed an extension to dynamically adjust the PBBF parameters to minimize the input required from the designer.

Yu *et al.* studied the problem of scheduling packet transmissions for data gathering in wireless sensor networks [51]. The focus is to explore the energy-latency tradeoffs in wireless communication using techniques such as modulation scaling. The authors presented algorithms to minimize the overall energy dissipation of the sensor nodes in the aggregation tree subject to the latency constraint.

In [31], the authors proposed a distributed, scalable and localized multipath search protocol to discover multiple node-disjoint paths between the sink and source nodes. Moreover, a load balancing algorithm was also proposed to distribute the traffic over the multiple paths discovered. In the paper, energy is considered as a constraint so that the design is feasible for the limited resources of WSNs.

2.2 Target Tracking and Prediction

2.2.1 Target Tracking

As one of the most important applications of WSNs, target tracking was widely studied from many perspectives. First, tracking was studied as a series of continuous localization operations in many existing efforts [52, 53]. Secondly, target tracking was sometimes considered as a dynamic state estimation problem on the trajectory, and Bayesian estimation methods, e.g., particle filtering, were used to obtain optimal or approximately optimal solutions [21]. The related work for this part will be detailed in Section 2.4. Thirdly, in some cases, target tracking was considered as an objective application when corresponding performance metrics, e.g., energy efficiency [8] or real-time feature [6], were the focus. Fourthly, a few efforts were conducted based on real implementation, and emphasized the actual measurement for a tracking application [6]. Finally, a few target tracking efforts did not explicitly distinguish tracking from similar efforts, such as detection [8] and classification [54]. A few examples are identified as follows.

In [52], Aslam *et al.* developed a particle filtering style algorithm for target tracking using minimalist sensors. The authors proposed a binary sensor model, where each sensors value is converted reliably to one bit of information only: whether the object is moving toward the sensor or away from the sensor. It was shown that a network of binary sensors has geometric properties that can be used to develop a solution for tracking with binary sensors

and present resulting algorithms and simulation experiments.

Eickstedt *et al.* described a framework for adaptive and cooperative control of the autonomous sensor platforms in such a network [53]. This framework has two major components, an intelligent sensor that provides high-level state information to a behavior-based autonomous vehicle control system and a new approach to behavior-based control of autonomous vehicles using multiple objective functions that allows reactive control in complex environments with multiple constraints.

In [8], Gui *et al.* studied the power saving operations in surveillance state and tracking state of network operations. During surveillance state, a set of novel metrics for quality of surveillance was proposed specifically for detecting moving objects. In the tracking state, a collaborative messaging scheme was proposed to wake up and shut down the sensor nodes with spatial and temporal preciseness. Though the paper studies network operations in the tracking state, it focuses more on detection, i.e., single occurrences that nodes detect the target.

In [6], He *et al.* presented the real-time design and analysis of VigilNet, a large-scale sensor network system which tracks, detects and classifies targets in a timely and energy efficient manner. The whole tracking process is partitioned into six phases, and the deadline satisfaction efforts for each phase are detailed based on real implementation. Based on a deadline partition method and theoretical derivations to guarantee each sub-deadline, the end-to-end tracking deadline will be satisfied.

In [54], Arora *et al.* studied the application of sensor networks to the intrusion detection problem and the related problems of classifying and tracking targets. The authors explored the design space of sensors, signal processing algorithms, communications, networking, and middleware services. Moreover, they introduced the influence field, which can be estimated from a network of binary sensors, as the basis for a novel classifier. Though tracking is one of the main purposes, the authors emphasized more on classification than on tracking.

2.2.2 Multiple Target Tracking

In fact, the tracking techniques for multiple targets are based on those for single target tracking. When extending the research efforts for single target tracking to multiple target case, more emphasis is put on differentiating multiple targets from each other (i.e., classification) [55, 56], or data association [57, 58] than tracking itself.

Liu *et al.* provided a survey of techniques for tracking multiple targets in distributed sensor networks and introduce some recent developments in [55]. The primary focus is the association problem of multiple target trajectories. In addition, the authors surveyed the compensation for interfering sensing of targets at close proximity, with a higher-dimensional joint space.

In [58], Chen *et al.* proposed techniques based on graphical models to efficiently solve the problem of associating sensor measurements with target tracks. It takes advantage of the sparsity inherent in the problem structure resulting from the fact that each target can be observed by only a small number of sensors and makes use of efficient message-passing algorithms for graphical models to infer the maximum *a posteriori* association configuration.

There are a few other efforts that discuss problems related to multiple target tracking. For example, in [56], the emphasis is the resource constraints of nodes when tracking multiple targets simultaneously. In [59], Liu *et al.* utilized multiple targets as tracking objects in their simulation studies.

2.2.3 Target Prediction

Typical techniques for target prediction include kinematics-based prediction [13,20], dynamics-based prediction [60], and Bayesian estimation methods [21, 61].

Kinematics and dynamics are two branches of the classical mechanics. Kinematics describes the motion of objects without considering the circumstances that cause the motion, while dynamics studies the relationship between the object motion and its causes [62]. In fact, most of past work about target prediction uses kinematics rules as the foundation, even for those that use Bayesian estimation methods.

MCTA algorithm presented in [13] is just an example of kinematics-based prediction. By estimating the possible moving area of vehicles with vehicular kinematics, MCTA predicts the contour of tracking areas.

Another example of the kinematics-based prediction is the Prediction-based Energy Saving scheme (PES) introduced in [20]. It only uses simple models to predict a specific location without considering the detailed moving probabilities.

In [60], Taqi *et al.* discussed a dynamics-based prediction protocol named as A-YAP. They leveraged the physics research results on the yaw rate and the side force. However, these results depend on the target mass, which requires the surveillance system to recognize the target with target classification techniques. In many cases, target classification is difficult especially when the real-time tracking constraint is applied. Moreover, A-YAP also predicts an exact location that the target is probably moving to, instead of considering all the possibilities.

In fact, kinematics-based prediction can be represented by a linear state transition model, which will be detailed in Chapter 3. Therefore, particle filters provide a more generic solution to target prediction than these physics based approaches.

2.3 Sleep Scheduling

In the past, there were many existing research efforts about sleep scheduling to prolong the network lifetime of WSNs. But since the first dissertation problem focuses on sleep scheduling for energy efficient target tracking, we mainly review those efforts that integrate sleep scheduling and target prediction [13, 19].

In terms of target motion engagement, the sleep scheduling efforts can be classified as having no engagement [16], engaged without prediction [8], and engaged with prediction [13]. Most of them do not have target motion engaged. Even those which consider target motion do not fully leverage the prediction results to enhance energy efficiency. For example, some work schedules the sleep pattern based only on a node's distance from a target's current location: the further a node is away from the target, the deeper its sleep level would be. In other words, they only consider a target's velocity magnitude. Such a sleep scheduling protocol is often called "circle-based scheme" (Circle) [8]. In this legacy Circle scheme, all the nodes in a circle follow the same sleep pattern. For example, they will be awakened at the same time and usually kept active all the time during an expected period, without distinguishing among different possible directions and speeds of the target. On the contrary, if we can predict a target's potential motion and leverage this prediction result for sleep scheduling, the energy consumption could potentially be reduced. For example, if the target's next location can be predicted, the surveillance system would not have to awaken all the nodes in a circle for tracking, or keep all the awakened nodes always active.

In fact, the motion of objects in the real world follows certain physics rules, at the same time it is also subject to uncertainty. This apparent contradiction is because: 1) during a short time period, there is no significant change on the motion state, therefore the target will approximately follow physics rules; 2) however, a target's long term behavior is uncertain, e.g., a harsh brake or a sharp turn for shunning an obstacle is hard to precisely predict. This contradiction aggravates the complexity of target prediction.

In [13], Jeong *et al.* presented a Minimal Contour Tracking Algorithm (MCTA) to enhance energy efficiency by solely reducing the number of awakened nodes. MCTA uses the concept of "contour" to integrate the precise prediction and the uncertainty of real mobile targets. In addition, MCTA keeps all the nodes in the contour active without any differentiated sleep scheduling.

Wang *et al.* discussed sleep scheduling based on target prediction in [19]. But it does not consider the uncertainty, either.

2.4 Particle Filtering

2.4.1 Centralized Particle Filters

Bayesian estimation methods estimate the states in a dynamic system in an iterative manner, by incorporating new measures to filter the prior distribution to the posterior one. When certain constrains (including Gaussian process and measurement noises, and linear state transition and measurement functions) hold, Kalman filter [63] serves as the optimal solution by minimizing the estimated error covariance. If otherwise the dynamic system is nonlinear and/or non-Gaussian, which is usually true in real applications, particle filters [21] are usually used to approximate the optimal solutions.

Particle filters were leveraged by many actual applications, such as target tracking, sensor selection, and prediction-based sleep scheduling. In [64], Singh *et al.* applied PF to multiple target tracking with binary proximity sensors. Information-driven sensor querying (IDSQ) [65] optimizes the sensor selection to maximize the information gain while minimizing the communication and resource usage. Wang *et al.* adopts the prediction result of PFs for sleep scheduling and energy optimization in [28].

Given the high computation/communication cost of PFs, it is often difficult to apply PFs to WSNs. Many research efforts were conducted to either reduce the number of particles or compress the data amount of communication. In [66], the author applied KLD-sampling to adapt the number of particles dynamically, so that the estimation error is bounded at a given probability. Kwak *et al.* introduced a heuristic algorithm based on a back-propagation neural network to adapt the sample size in [67].

2.4.2 Distributed Particle Filters

Compared with centralized particle filters (CPFs) [21], the research for DPFs is much less mature. One of the most important reasons for this is that PFs are easy to be defined for centralized architectures, but difficult to be extended to distributed systems [68].

[23] is an important work about DPFs, in which Coates presented the achievable compression on particles either by training parametric models or with adaptive encoding. The compressed data, instead of the raw data, is propagated and aggregated throughout the network. This compressed data may be either parameters trained from a certain parametric model of the factorized likelihood, or quantized data by encoding the measurements. This work was the first one to complete the data aggregation step by step along with the propagation. Ing and Coates further improved the idea of adaptive encoding with Huffman tree in [69].

Although the training of a parametric model was proposed in [23], the author did not present a specific parametric model. Sheng *et al.* provided one, i.e., Gaussian mixture model (GMM), in [25]. The distributed algorithms are run over a set of uncorrelated sensor cliques, which

are dynamically constructed according to the moving trajectories of the target. With GMM, the measurement data may be compressed and aggregated efficiently.

Unlike [25], Liu *et al.* introduced a non-parametric method named support vector machine (SVM) in [29]. The raw data can also be compressed to reduce the communication cost.

All these DPF efforts focused on completing the aggregation of particles in a distributed manner on different sensor nodes, and reducing the communication cost by compressing the raw data. These features will introduce the following two problems: 1) the aggregation of particle weights will experience a long delay, so will each iteration of PFs; and 2) though the total data amount is compressed, the number of communicated messages may remain or even increase. In addition, [23] strives to keep the computation result of each iteration consistent across the network, which is often unnecessary.

In [27], the authors presented a semi-distributed particle filter, which is the first to maintain particles on different sensor nodes. However, weight aggregation is still dependent on a global transceiver. Such kind of global transceiver, which is assumed to be able to communicate with all the nodes in the network directly, is usually hard to implement in real deployments. Moreover, [27] made many assumptions, such as binary proximity sensors, the optical communication and reflective devices, which are mostly unnecessary.

Except for these DPF efforts, Huang *et al.* studied target tracking using DPFs in [26], which was an efforts of applying DPFs in specific scenarios.

2.4.3 Sample Impoverishment Problem

For the practical problems like sample impoverishment, typical solutions for sample impoverishment include resample-move [24], regularization [22], and bridging densities [22]. The resample-move algorithm [24] moves the resampled particle to a new position by sampling a transition kernel function, and repeated particles may be moved to different positions due to the variation of kernel function sampling. In [22], particles are resampled from a continuous approximation based on the rescaled Kernel density instead of the previous particle set. Bridging densities [22] introduce intermediate distributions between the prior and the likelihood, so that the sampling takes both the prior distribution and the likelihood function into account.

2.5 Multipath Forwarding and Flooding

2.5.1 Real-time

Many sensor networks require real-time propagation for time sensitive data, in which the end-to-end delay needs to be satisfied [70].

Abdelzaher *et al.* discussed a WSN's real-time capacity from a macro perspective without making any detailed assumptions in [71]. On the contrary, He *et al.* presented specific delays for each chain during the end-to-end data propagation of the target tracking application in [6], for which all the calculations were based on the actual implementation of their testbed VigilNet [72].

Some research works utilized real-time performance as a constraint for studying other related problems, e.g., sleep scheduling [14], data aggregation [73], MAC protocol [74], and routing [75]. And some works aimed at achieving an optimized energy-latency tradeoff [51, 76].

2.5.2 Multipath Forwarding and Flooding

In unreliable network environments, past data delivery protocols use different approaches to guarantee timeliness. Single path forwarding protocols transmit one copy of data along a predetermined single path, and depend on retransmissions to guarantee reliability [77]. In contrast, multipath data delivery protocols transmit a number of copies through multiple routes simultaneously. This will increase the reliability and the probability of delivering real-time data in a timely manner, but often at the expense of additional energy consumption.

Multipath routing protocols can be broadly classified into two categories, static routing and dynamic routing. Static multipath routing protocols setup multiple routes, which are either disjoint [31] or braided [32], before sending out a data packet. The source nodes then either choose one of the routes or combine the resources of all the routes for a single flow. In contrast, dynamic multipath forwarding protocols decide the flooding recipients at each hop so that the data flow is split dynamically.

Dynamic routing protocols can be further classified into multicast, broadcast (or flooding), and gossip. Multicast has been extensively studied for WSNs [33], most of which aim at multihop one-to-many communications. Broadcasting and flooding are usually used interchangeably, but there are also works that distinguish them with minor differences [78]. The HHB scheme introduced in [34] is a hop-by-hop broadcast protocol that leverages the broadcasting capability of wireless medium to guarantee the reliable delivery. The HHB protocol broadcasts at each hop with a specific probability to avoid degenerating into a flooding storm. Zhang *et al.* present a constrained flooding protocol in [35], which exhibits good energy efficiency by constraining retransmissions. Gossip can be considered as a form of probabilistic flooding. In [36], Lu *et al.* present a gossip algorithm called NBgossip, which

forwards the linear combinations of the received messages.

Almost all of the multipath data delivery protocols introduce extra overheads even when unicasting is enough for satisfying QoS constraints such as timeliness. In contrast, CFlood uses flooding but constrains the energy consumption effectively by controlling the scale of flooding.

2.5.3 Data Aggregation

Existing work about data aggregation mainly focused on aggregation structure and performance, e.g., aggregating node selection [79] and aggregation delay [80]. Only a few discussed the aggregation function, which is the method to aggregate multiple packets into one.

From the perspective of the aggregation function, data aggregation efforts may be classified into semantics-based aggregation [81] and length-based aggregation [38]. Examples of semantics-based aggregation functions include the enumeration in a state vector used in [81], "sum" used in [80], and "max", "min", "average", and "count" listed in [37]. All of these semantics-based aggregation functions may compress the total length of multiple packets to that of a single original packet, or even less. On the contrary, length-based aggregation does not, or slightly, change the total length. For example, the aggregation in [38] simply connects multiple packets into one, with the same payload length.

Chapter 3

Preliminaries

Before presenting the detailed research work, we introduce system models and performance metrics in this chapter.

3.1 Network Model and Assumptions

We consider a homogeneous, static sensor network in a two-dimensional plane, with sensor nodes randomly and uniformly deployed at a density ρ. There is a single sink node, to which all the nodes report their detection. Such a many-to-one data propagation is usually called "convergecast" [82].

We assume that every node is aware of its own location (using GPS [83] or algorithmic strategies such as [84]), and is able to determine a target's position at detection (either by sensing or by calculating—e.g., [17,54,85]). Multiple targets are assumed to be distinguished from each other using a target classification algorithm such as [55].

Each node may detect targets within its sensing radius r, and communicate with other nodes within its adjustable communication radius R (e.g., with popularly used radio hardware such as CC2420 [86]). In addition, we assume that $2r \leq R$. This is a reasonable assumption, as the radio of a node is usually much more powerful than its sensing devices. For example, the radio range of MICA2 is up to 150 m [87], which is very difficult to reach for most sensing devices.

Finally, we assume that the underlying MAC protocol supports collision avoidance with the RTS/CTS (Request To Send and Clear To Send) exchange mechanism [88]. With the RTS/CTS exchange, a node will be able to know at an early stage if the transmission will be delayed by the channel contention.

3.2 Duty Cycling and Sleep Scheduling

The sleep pattern of a sensor node can be described by an application specific definition as in [19]. But a more commonly used description is duty cycling with specific toggling period and duty cycle [15]. In the duty cycling mode, a node operates in two states: active and sleep. When active, a node turns on its processor, sensing devices and RF transceiver for computing, sensing, and communicating. In the sleep state, a node puts most modules/devices into sleep except a wake-up timer with extremely low energy consumption [8].

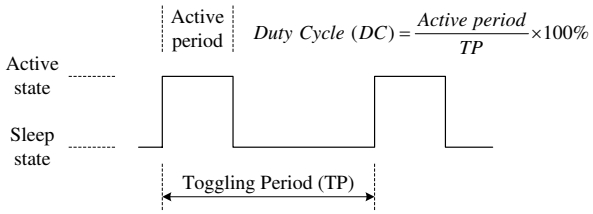


Figure 3.1: Toggling period and duty cycle

Figure 3.1 illustrates the concepts of *toggling period* (TP) and *duty cycle* (DC). Within each TP period, a node wakes up and remains active for $TP * DC$, and then sleeps for $TP * (1 - DC)$ [8].

Usually, nodes work in random duty cycling mode by default, i.e., TP and DC are the same for all the nodes, but their starting time point of each TP may be different. Though DC is often very low under this random mode [15], nodes are still able to communicate with each other using a MAC protocol such as B-MAC [89].

For a certain purpose, TP and DC may be changed dynamically, e.g., prolonging DC to improve the detection performance. This is sleep scheduling, which is used to improve energy efficiency of proactive wake-up.

3.3 Target Prediction

In this section, some of the vectors in a 2-dimensional plane are expressed with polar coordinates (polar coordinates and Cartesian coordinates will not be explicitly declared as long as there is no ambiguity). For example, $\overrightarrow{X} = (X, \theta)$, where $X = \|\overrightarrow{X}\|$ is its polar radius and $\theta \in (-\pi, \pi]$ is the polar angle. In a real deployment, we may simply assign the four directions, south, east, north and west respectively as $-\frac{\pi}{2}$, 0, $\frac{\pi}{2}$ and π.

A target's movement status is a continuous function of time. However, the estimation for a target's movement status is a discrete time process. The surveillance system can only estimate the target states at some time points, and predict the future motion based on the estimation results. Assume that TPSS estimates the target states at time points $\{t_n | n \in N\}$,

where $t_i < t_j \ for \ \forall \ i < j \in N$. We define the state vector to represent the target motion state:

Definition 1 (State Vector) *For each time point t_n, the state vector is defined as $State(n) = (t_n, x_n, y_n, \overrightarrow{v_n}, \overrightarrow{a_n})$, where (x_n, y_n) is the target position, $\overrightarrow{v_n} = (v_n, \theta_n)$ and $\overrightarrow{a_n}$ are respectively the average velocity vector and the average acceleration vector of the target during (t_{n-1}, t_n), v_n is the scalar speed and θ_n is the moving direction.*

At time point t_n, we may calculate the state vector $State(n)$ based on the discrete time state sequence $\{State(k)|k < n\}$, and then predict $\overrightarrow{S_{n+1}}$ (i.e., the displacement vector during (t_n, t_{n+1})) according to kinematics rules. These predicted values are denoted as variable names with a prime symbol, such as $\overrightarrow{S_{n+1}}'$ for $\overrightarrow{S_{n+1}}$.

In addition, we establish a probabilistic model for the displacement $\overrightarrow{S_{n+1}}'$ as $\overrightarrow{S_{n+1}}' = (S_{n+1}, \Delta_{n+1})$, where S_{n+1} and Δ_{n+1} signify the polar radius and the polar angle of $\overrightarrow{S_{n+1}}'$ respectively. Both S_{n+1} and Δ_{n+1} are discrete time random processes. Obviously $\mu_{S_{n+1}} = \|\overrightarrow{S_{n+1}}'\|$ and $\mu_{\Delta_{n+1}} = 0$, where μ is a random variable's expected value.

3.4 Tracking Problem and Particle Filters

The tracking problem is usually formulated as a dynamic system in the state space $\{\mathbf{x}_k, \ k \in \mathbb{N}\}$, where k is the index of discrete time. This dynamic system consists of a state transition model and a measurement (or observation[1]) model [21]:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{f}_k(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}) \\ \mathbf{z}_k &= \mathbf{h}_k(\mathbf{x}_k, \mathbf{n}_k) \end{aligned} \tag{3.1}$$

where $\mathbf{f}_k$ and $\mathbf{h}_k$ are possibly nonlinear functions, $\{\mathbf{v}_{k-1}, \ k \in \mathbb{N}\}$ and $\{\mathbf{n}_k, \ k \in \mathbb{N}\}$ are i.i.d. process noise and measurement noise sequences respectively, and $\{\mathbf{z}_k, \ k \in \mathbb{N}\}$ is a sequence of observations at time k.

Assuming that states $\{\mathbf{x}_k, \ k \in \mathbb{N}\}$ follow a first order Markov process and observations depend only on the states, the posterior probability density $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ (i.e., the degree-of-belief in the state $\mathbf{x}_k$) can be recursively estimated in two steps—prediction and updating:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \tag{3.2}$$

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{3.3}$$

[1]The terms "measurement" and "observation" will be used alternatively whenever there is no ambiguity.

The optimal Bayesian solutions for this tracking problem are tractable only when certain constraints hold. Kalman filter and grid-based filter are two examples of these optimal solutions. When the analytic solution is intractable, on the contrary, approximation will be the only choice. Particle filters are just one of the approximate approaches.

Particle filters represent the posterior pdf with a number of particles with associated weights, and make estimations based on these weighted particles. A generic particle filter, i.e. sequential importance sampling (SIS) algorithm, runs in an iterative manner. After the initialization step that draws N_s particles for the first time $t = 0$, each iteration consists of the following four steps [29]:

1) *Prediction.* Draw N_s particles for time k from an importance density $q(\mathbf{x}_k|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$;

2) *Updating.* Assign and normalize a weight w_k^i for each particle;

3) *Resampling (optional).* Eliminate particles with low importance weights and multiply those with high importance weights so as to reduce the degeneracy effect;

4) *Estimation.* Calculate the estimation as $\hat{\mathbf{x}_k} = \sum_{i=1}^{N_s} w_k^i \mathbf{x}_k^i$.

Here $N_s \in \mathbb{N}$ is the number of particles, $i = 1, \ldots, N_s$ is the index of particles, w_k^i is the weight of particle $\mathbf{x}_k^i$ at time k, and $\hat{\mathbf{x}_k}$ is the estimated $\mathbf{x}_k$.

Sampling importance resampling (SIR) filters are derived from generic particle filters by choosing the prior density $p(\mathbf{x}_k|\mathbf{x}_{k-1}^i)$ as the importance density $q(\mathbf{x}_k|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$, and resampling in every iteration [21].

The dynamic system shown in Equation (3.1) provides a very flexible model for the tracking problem. In fact, many kinematics-based motion models can be represented with this state transition model [90, 91]. For example in a two-dimensional plane, let $\mathbf{s}_k = (s_{kx}, s_{ky})$, $\mathbf{v}_k = (v_{kx}, v_{ky})$, and $\mathbf{a}_k = (a_{kx}, a_{ky})$ denote the position vector, the velocity vector and the acceleration vector at time k respectively. Then according to kinematics, we have:

$$\begin{bmatrix} \mathbf{s}_k^T \\ \mathbf{v}_k^T \end{bmatrix} = \begin{bmatrix} \mathbf{I}_2 & \Delta t \cdot \mathbf{I}_2 \\ \mathbf{0}_{2,2} & \mathbf{I}_2 \end{bmatrix} \begin{bmatrix} \mathbf{s}_{k-1}^T \\ \mathbf{v}_{k-1}^T \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \cdot \mathbf{I}_2 \\ \Delta t \cdot \mathbf{I}_2 \end{bmatrix} \mathbf{a}_{k-1}^T \tag{3.4}$$

where $\mathbf{I}_2$ is an identity matrix of size 2, $\mathbf{0}_{2,2}$ is a zero matrix, and t is the interval of state transition.

If we let the state $\mathbf{x}_k = (\mathbf{s}_k, \mathbf{v}_k)^T$, then Equation (3.4) can be considered as a linear state transition model with the acceleration $\mathbf{a}_{k-1}$ as the process noise. Here we assumed that $\mathbf{a}_{k-1}$ keeps unchanged during the interval $(k-1, k]$. If otherwise $\mathbf{a}_{k-1}$ changes at a certain jerk $\mathbf{j}_{k-1}$ (i.e., the rate of change of acceleration, but rarely used in physics), then Equation (3.4) can be expanded for the state $\mathbf{x}_k = (\mathbf{s}_k, \mathbf{v}_k, \mathbf{a}_k)^T$ with the jerk $\mathbf{j}_{k-1}$ as the process noise. This also results in a linear state transition model.

Since velocity is the derivative of position (i.e. $\mathbf{v}_k = \mathbf{s}_k'$) and acceleration is the derivative of velocity (i.e. $\mathbf{a}_k = \mathbf{v}_k' = \mathbf{s}_k''$), this state transition model can be further expanded using

Taylor's theorem. Then the state will be in the form of $\mathbf{x}_k = (\mathbf{s}_k^{(0)}, \mathbf{s}_k^{(1)}, \ldots, \mathbf{s}_k^{(n_x-1)})^T$, where $\mathbf{s}_k^{(i)}$ $(i \in [0, n_x - 1])$ denotes the ith derivative of $\mathbf{s}_k^{(0)} = \mathbf{s}_k$ and n_x is half of the dimension of $\mathbf{x}_k$. In this case, the difference introduced by various dimensions of the state vector $\mathbf{x}_k$ locates only in the granularity of noise and the computational complexity. Therefore, kinematics-based prediction and tracking can be completely represented by such a dynamic system thus solved with particle filters.

Many existing efforts follow this state transition model, e.g., [91] and [28] adopt a two-dimensional model, [90] and [92] discuss three-dimensional models. In fact, other vectors in a two-dimensional plane, or even scalar components can also be integrated in the state vector [93].

In this dissertation, we adopt a dynamic system model on the state $\mathbf{x}_k = (\mathbf{s}_k^{(0)}, \mathbf{s}_k^{(1)}, \ldots, \mathbf{s}_k^{(n_x-1)})^T$ for the algorithm development without any other specific assumptions.

3.5 Performance Metrics

In this section, we define the metrics to estimate energy efficiency and tracking performance for the three dissertation problems.

3.5.1 Energy efficiency

For Sleep Scheduling

Since the energy consumption of sleep scheduling is highly relative to the position where the target is detected, the network-wide energy consumption on different nodes may vary significantly. Given this inequity, the network lifetime—time until the first sensor node runs out of power—will be a less useful metric. We define *extra energy* (EE) as the criterion of energy efficiency for sleep scheduling:

Definition 2 (Extra Energy) *The extra energy is defined as the more energy consumed for sleep scheduling than the energy consumption without any target detected during the same tracking period.*

For Particle Filtering

VPF is a centralized implementation of particle filters, for which all the computations are completed in a central unit, e.g., the base station. Thus, we estimate the energy efficiency with the computation time per iteration.

Unlike VPF, CDPF is a distributed particle filter, which focuses on reducing the communication cost. Therefore, we use the communication cost, e.g., the total communication workload, to estimate the energy efficiency.

For Constrained Flooding

We define energy efficiency of CFlood with the real-time data propagation performance that CFlood achieves per unit of energy. Specifically, we define *real-time capacity per unit energy consumption* $\delta = \frac{DSR}{e}$ to describe the energy efficiency, which measures the percentage of time-sensitive packets that can be delivered meeting their deadlines with unit energy consumption. Therefore, the higher δ is, the more energy efficient the flooding protocol will be. DSR represents the deadline satisfaction ratio, which will be defined in the next section. e is the average energy consumed for transmitting a single time-sensitive packet, i.e., the ratio of the total energy consumption to the total number of time-sensitive packets generated. With a flooding protocol, a single packet may be copied multiple times. Thus, the total energy consumption consists of the energy for transmitting and receiving all the copies. In the simulation, we do not emphasize the unit of energy, since it depends on the specific hardware platform.

3.5.2 Tracking performance

For Sleep Scheduling

The tracking delay is one of the most important performance metrics for tracking. Since tracking can be considered as a process of continuously detecting a target, the tracking delay should be defined based on the detection delay to express the network-wide detection performance. Usually the detection delay is defined as the delay between the time when the target enters the surveillance field or gets lost and the time when it is detected. Based on this definition, we use *average detection delay* (AD) to measure the tracking delay, which is defined as follows.

When a target leaves the sensing region of the currently active nodes and no other nodes detect it in time, the target will be lost. A detection delay is defined as the time interval from when a target is lost till it is detected again. AD is defined as the average of the detection delays along the target's traversing route. Before the target is detected for the first time, TPSS protocol is not started and all the nodes work in the random sleep pattern as we assume. Thus the initial detection delay, like the initial activation delay defined in [6], is out of the scope of TPSS's performance. We define a term, *tracking period*, as the time interval from when a surveillance system detects the target for the first time to when the target leaves the surveillance field. Then, AD is estimated only within the tracking period.

Another important performance metric of tracking is the percentage of a target's escape

distance in the total trajectory. We call it *escape distance percentage* (EDP). The expected tracking result should be there always exist nodes that can detect the target, i.e., the escape distance percentage is zero. However, this is often impossible. A low escape distance percentage would mean a good coverage on the target's trajectory.

In the prototype implementation, the detection delays are difficult to measure without time synchronization in the real deployment. Therefore, we utilized the escape distance percentage to evaluate the tracking performance. We will detail the experiment result in Chapter 9.

For Particle Filtering

When target tracking is considered as a dynamic state estimation problem for trajectory estimation, we estimate the tracking performance with the estimation error. This estimation error is usually described with Root Mean Squared Error (RMSE) [21, 26]. We follow this method in the dissertation.

For Constrained Flooding

We adopt the *deadline satisfaction ratio* (DSR) [9] to characterize the real-time performance of a WSN, which is defined as the ratio of the number of real-time packets that arrive at sink nodes meeting their deadlines, to the total number of those transmitted.

Chapter 4

Sleep Scheduling for Single Target Tracking

4.1 Introduction

In this chapter, we present a target prediction and sleep scheduling protocol (TPSS), which sleep-schedules nodes for improving the tradeoff between energy efficiency and detection delay.

TPSS is designed based on proactive wake-up: when a node (i.e., *alarm node*) detects a target, it broadcasts an alarm message to proactively awaken its neighbor nodes (i.e., *awakened node*) to prepare for the approaching target. To enhance energy efficiency, we modify this basic proactive wake-up method to sleep-schedule nodes precisely. Specifically, TPSS selects some of the neighbor nodes (i.e., *candidate node*) that are likely to detect the target to awaken. On receiving an alarm message, each candidate may individually make the decision on whether or not to be an awakened node, and if yes, when and how long to wake up.

We utilize two approaches to reduce the energy consumption during this proactive wake-up process: 1) reducing the number of awakened nodes; and 2) scheduling their sleep pattern to shorten the active time. First, those nodes that the target may have already passed during the sleep delay do not need to be awakened. Secondly, nodes that lie on a direction that the target has a low probability of passing by could be chosen to be awakened with a low probability. Therefore, the number of awakened nodes can be reduced significantly. Moreover, these chosen awakened nodes could wake up and keep active only when the target is expected to traverse their sensing area. Thus, their wake-up time can be curtailed as much as possible, too.

Both of these energy reducing approaches are built upon target prediction results. The past efforts about target prediction utilized various predicting approaches: 1) in [20], the authors

only predict an exact position that the target is probably moving to; 2) in [13], the algorithm predicts a tracking contour, i.e., a small area that the target may pass with kinematics rules only; and 3) in [60], dynamics is used for prediction, and the system needs to classify what the target is and estimate external causes for the target motion. Unlike these research efforts, we develop a target prediction model based on both kinematics rules and probability theory. Kinematics-based prediction calculates the expected displacement of the target in a sleep delay, which shows the position and the moving direction that the target is most likely to be in and move along. Based on this expected displacement, probability-based prediction establishes probabilistic models for the scalar displacement and the deviation. Once a target's potential movement is predicted, we may make sleep scheduling decisions based on these probabilistic models: take a high probability to awaken nodes on a direction along which the target is highly probable to move, and take a low one to awaken nodes that are not likely to detect the target.

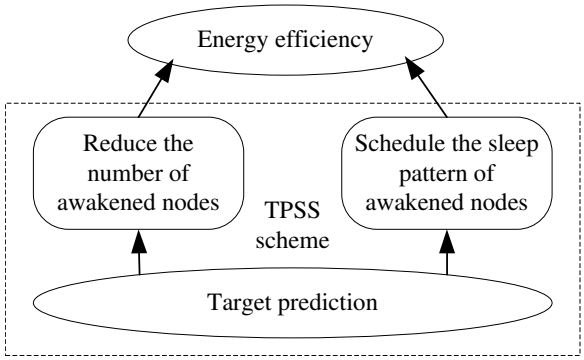


Figure 4.1: Foundation, approaches, and objective

Figure 4.1 shows the foundation, approaches, the objective of TPSS protocol and the relationship among them.

- The prediction scheme predicts (or describes) the prospective change of a target's velocity rather than its position. Since velocity is a physical vector, the prediction scheme establishes probabilistic models for both its scalar absolute magnitude (i.e. speed) and its direction. The normal distribution (or Gaussian distribution) is the major probabilistic model that we utilize for prediction.

- For reducing the number of awakened nodes, we introduce a concept of *awake region* and a mechanism for computing the scope of an awake region. An awake region is defined as the region that a target may traverse in a next short term, which should be covered probabilistically by active nodes. As some nodes in an awake region will be proactively awakened and keep active for some time, the number of these awakened nodes in the region should be as less as possible to enhance energy efficiency.

- For scheduling the sleep pattern of awakened nodes, we present a sleep scheduling protocol to further enhance energy efficiency. This mechanism schedules the sleep

patterns of awakened nodes individually according to their distance and direction away from the current motion state of the target. Therefore, the lasting time of awakened nodes in their active state could be reduced to as short as possible.

We analyze the detection probability and the detection delay of TPSS, and conduct simulation-based experimental studies to understand the tradeoff between energy efficiency and tracking performance. Our simulation results show that, compared with existing protocols such as Circle scheme [8] and MCTA [13] (a past algorithm for reducing the energy consumption of proactive wake-up), TPSS introduces an improvement of $25\% \sim 45\%$ in energy efficiency, at the expense of $5\% \sim 15\%$ increase in detection delay.

4.2 Target Motion Prediction

In the real world, a target's movement is subject to uncertainty, while at the same time it follows certain rules of physics. This apparent contradiction is because: 1) at each instant or during a short time period, there is no significant change on the rules of a target's motion, therefore the target will approximately follow kinematics rules; 2) however, a target's long term behavior is uncertain and hard to predict, e.g., a harsh brake or a sharp turn cannot be predicted completely with kinematics rules. In fact, even for a short term, it is also difficult to accurately predict a target's motion purely with a physics-based model. However, the prediction is absolutely helpful for optimizing the energy efficiency and tracking performance tradeoff. Thus, we consider a probabilistic model to handle as many possibilities of change of the actual target motion as possible.

At each time point t_n, the whole prediction process could be divided into three steps:

1) Calculate the current speed v_n, direction θ_n and acceleration $\overrightarrow{a_n}$, based on $State(n-1)$ and the current position (x_n, y_n) that is assumed to be obtained by sensing or by calculating;

2) Based on kinematics rules, predict the displacement $\overrightarrow{S_{n+1}}'$;

3) Establish the probabilistic model including the target's scalar displacement S_{n+1} and deviation Δ_{n+1}.

In the default random sleep pattern, the communication among nodes suffer a sleep delay with a MAC protocol like B-MAC, where a sending node broadcasts the preamble no less than the length of a toggling period to guarantee that each duty cycling receiver can hear it [89]. We suppose the sleep delay exactly as TP for simplification, i.e., $t_{n+1} = t_n + TP$.

Each time when time moves one step forward (e.g. from t_n to t_{n+1}), v'_{n+1} and θ'_{n+1} predicted in step 2 at time point t_n will be replaced with the actual values v_{n+1} and θ_{n+1} calculated in step 1 at time point t_{n+1}.

Notice that this prediction is based on a previously available observation. If this is the first

time that the target is detected and there is no $State(n-1)$, the prediction can be skipped and delayed to the next time point.

4.2.1 Calculate the Current State

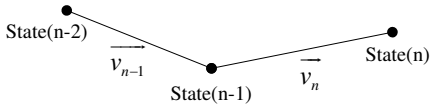


Figure 4.2: Target movement states

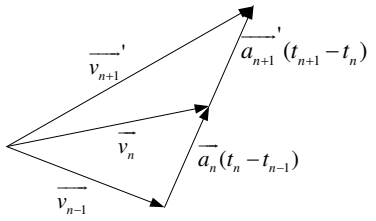


Figure 4.3: Prediction based on kinematics

First we compute the state vector $State(n)$ based on the previous state $State(n-1)$. Assume that the current time point is t_n, and the state at the previous time point is known as $State(n-1)$. Figure 4.2 shows the target motion at three continuous time points, and Figure 4.3 illustrates the change of the target velocity and its acceleration. As assumed, the target's current location (x_n, y_n) can be determined by sensing or calculating with existing algorithms. Then, $\overrightarrow{v_n}$ and $\overrightarrow{a_n}$ can be computed as,

$$\begin{cases} v_n &= \frac{\sqrt{(y_n-y_{n-1})^2+(x_n-x_{n-1})^2}}{t_n-t_{n-1}} \\ \theta_n &= \begin{cases} \arctan\frac{y_n-y_{n-1}}{x_n-x_{n-1}} & , \quad x_n \neq x_{n-1} \\ 0 & , \quad x_n = x_{n-1} \end{cases} \\ \overrightarrow{a_n} &= \frac{\overrightarrow{v_n}-\overrightarrow{v_{n-1}}}{t_n-t_{n-1}} \end{cases} \tag{4.1}$$

Notice that we didn't substitute $t_n - t_{n-1}$ with TP, because the actual time point for sampling t_n depends on whether or not the target is physically detected. TP is only used for the prediction. Here we assume that sensor nodes are time synchronized locally in a small range, so that the received t_{n-1} may be used in the calculation. Local time synchronization may be easily achieved with a protocol such as RBS [94], or simply with HELLO message exchange [31] detailed in Chapter 7.

4.2.2 Kinematics-based Prediction

Then we predict $\overrightarrow{S_{n+1}}'$ following kinematics rules. For simplifying the computation, we assume that the acceleration remains unchanged as $\overrightarrow{a_n}$ during (t_n, t_{n+1}) (because otherwise

we will have to consider the target's "jerk", i.e. the rate of change of acceleration, which is rarely used). Then

$$\overrightarrow{S_{n+1}}' = \overrightarrow{v_n} \cdot TP + \frac{1}{2}\overrightarrow{a_n} \cdot TP^2 \tag{4.2}$$

4.2.3 Probability-based Prediction

Since the alarm message broadcast experiences a sleep delay of TP, the target may have already passed by some candidate nodes before they receive the alarm. For TPSS we setup a probabilistic model for the length of the target's displacement during the sleep delay, which can also show the impact of the target's scalar speed on proactive wake-up and sleep scheduling.

Suppose that the random variable S_{n+1} is Gaussian, i.e., $S_{n+1} \sim N(\mu_{S_{n+1}}, \sigma^2_{S_{n+1}})$. The mean is calculated as $\mu_{S_{n+1}} = \|\overrightarrow{S_{n+1}}'\| = \|\overrightarrow{v_n} \cdot TP + \frac{1}{2}\overrightarrow{a_n} \cdot TP^2\|$ when the acceleration remains unchanged. During the sleep delay TP, the scalar speed is likely to change between $\|\overrightarrow{v_n}\|$ (i.e. the original speed) and $\|\overrightarrow{v_n} + \overrightarrow{a_n} \cdot TP\|$ (i.e. the final speed when the acceleration remains unchanged). Thus S_{n+1} is likely to change between $S_A = \|\overrightarrow{v_n} \cdot TP\|$ and $S_B = \|\overrightarrow{v_n} \cdot TP + \overrightarrow{a_n} \cdot TP^2\|$. Obviously $\mu_{S_{n+1}}$ is in the interval (S_A, S_B) or (S_B, S_A) depending on the included angle between $\overrightarrow{v_n}$ and $\overrightarrow{a_n}$. Let the standard deviation of S_{n+1} be $\sigma_{S_{n+1}} = |\mu_{S_{n+1}} - S_A|$. In general $|\mu_{S_{n+1}} - S_B| \neq |\mu_{S_{n+1}} - S_A|$, but the difference is very small when the target is not making a sharp turn. Thus the probability of $S_{n+1} \in (S_A, S_B)$ or $S_{n+1} \in (S_B, S_A)$ is approximately 68% according to the "68-95-99.7 rule" of Gaussian distribution. In summary, $S_{n+1} \sim N(\mu_{S_{n+1}}, \sigma^2_{S_{n+1}})$ where

$$\begin{cases} \mu_{S_{n+1}} &= \|\overrightarrow{v_n} \cdot TP + \frac{1}{2}\overrightarrow{a_n} \cdot TP^2\| \\ \sigma^2_{S_{n+1}} &= (\|\overrightarrow{v_n} \cdot TP + \frac{1}{2}\overrightarrow{a_n} \cdot TP^2\| - \|\overrightarrow{v_n} \cdot TP\|)^2 \end{cases} \tag{4.3}$$

Figure 4.4 shows an example of $S_{n+1} \sim N(20, 25)$.

Next, we establish a linear model for Δ_{n+1}, i.e., the angle by which the target may deviate from the central direction of $\overrightarrow{S_{n+1}}'$. Assume that at time point t_{n+1}, the probabilities that the target turns left or right are completely equal. We configure the probability density function (PDF) of Δ_{n+1} as Equation 4.4, where p, q are coefficients and $p \leq \pi$.

$$f_{\Delta_{n+1}}(\delta) = \begin{cases} -\frac{q}{p}\delta + q & , \quad (\delta \geq 0) \\ \frac{q}{p}\delta + q & , \quad (\delta < 0) \end{cases} \tag{4.4}$$

Figure 4.5 shows the probability density distribution of Δ_{n+1}. Since the total probability is equal to 1, we have $pq = 1$. Meanwhile the expected value of Δ_{n+1} is 0. Thus given a variance $\sigma^2_{\Delta_{n+1}}$,

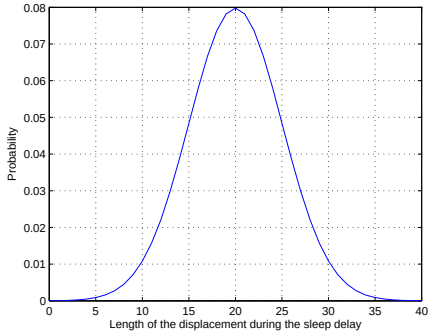


Figure 4.4: Example of Gaussian distribution of S_{n+1}

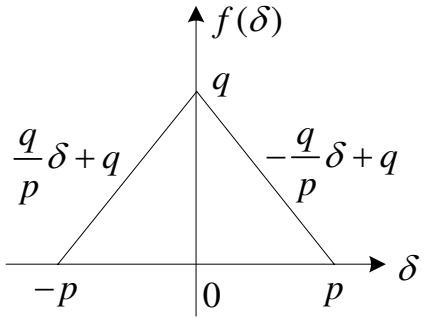


Figure 4.5: Probability density distribution of Δ_{n+1}

$$\sigma^2_{\Delta_{n+1}} = E[\Delta^2_{n+1}] - E[\Delta_{n+1}]^2 = E[\Delta^2_{n+1}] = \int_{-p}^{p} \delta^2 f(\delta) d\delta$$

Then the coefficients p and q are determined by $\sigma^2_{\Delta_{n+1}}$ as,

$$\begin{cases} p &= \sqrt{6}\sigma_{\Delta_{n+1}} \\ q &= \frac{\sqrt{6}}{6\sigma_{\Delta_{n+1}}} \end{cases} \tag{4.5}$$

The variance $\sigma^2_{\Delta_{n+1}}$ can be configured by the application or dynamically computed regarding to the acceleration. Next we present an example of computing $\sigma^2_{\Delta_{n+1}}$ from a maximum deviation angle δ_{max}.

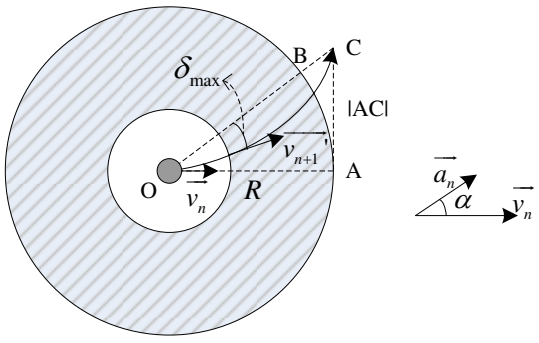


Figure 4.6: Probabilistic model of moving directions

Figure 4.6 shows the computation of the maximum deviation angle, where the small solid circle represents the target's current position, and the large circle is the transmission range of the alarm node. Here we simplify the computation by approximating the target's position with the alarm node's position, as the distance of the target from the alarm node is less than

a node's sensing range r, which is negligible compared with the transmission range R (e.g., the sensing range is configured as 10 m based on empirical data in [6], while the outdoor transmission range provided in the datasheet of Mica2 platform [87] is 500 $ft \approx 152.4$ m).

If the acceleration remains unchanged, the time that a target needs to escape the current alarm node's transmission range R can be computed approximately with the following equation.

$$R = v_n t_{escape} + \frac{1}{2} a_n \cos\alpha t_{escape}^2$$

where α is the included angle between $\overrightarrow{a_n}$ and $\overrightarrow{v_n}$, i.e.,

$$\cos\alpha = \frac{\overrightarrow{a_n} \cdot \overrightarrow{v_n}}{a_n \cdot v_n}$$

Such an approximate computation reduces the computational complexity significantly by substituting the solving process of a quadratic equation for that of a quartic equation used for precise solution.

Thus

$$t_{escape} = \frac{\sqrt{v_n^2 + 2Ra_n\cos\alpha} - v_n}{a_n \cos\alpha}$$

During this period of time, the target may move along the direction perpendicular to the current velocity for

$$|AC| = \frac{1}{2} a_n \sin\alpha t_{escape}^2 = \frac{\sqrt{1-\cos^2\alpha}}{a_n\cos^2\alpha}\left(v_n^2 + Ra_n\cos\alpha - v_n\sqrt{v_n^2 + 2Ra_n\cos\alpha}\right)$$

Then approximating the length of the arc $\widehat{AB}$ with $|AC|$, we have,

$$\delta_{max} = \frac{|AC|}{R} - \left|\theta'_{n+1} - \theta_n\right|$$

where $\frac{|AC|}{R}$ is the central angle corresponding to the target's deviation.

Let

$$\sigma_{\Delta_{n+1}} = \delta_{max} = \frac{\sqrt{1-\cos^2\alpha}}{Ra_n\cos^2\alpha}\left(v_n^2 + Ra_n\cos\alpha - v_n\sqrt{v_n^2 + 2Ra_n\cos\alpha}\right) - \left|\theta'_{n+1} - \theta_n\right| \quad (4.6)$$

then the probability of $\Delta_{n+1} \in (-\delta_{max}, \delta_{max})$ is approximately 68%.

4.3 Energy Conservation

In this section we introduce the approaches for reducing the energy consumption and describe the distributed algorithms.

4.3.1 Reducing the Number of Awakened Nodes

Usually, a sensor node's transmission range R is far longer than its sensing range r. Thus when the nodes are densely deployed to guarantee the sensing coverage, a broadcast alarm message will reach all the neighbors within the transmission range. However, some of these neighbors can only detect the target with a relatively low probability, and some others may even never detect the target. Then the energy consumed for being active on these nodes will be wasted. A more effective approach is to determine a subset among all the neighbor nodes to reduce the number of awakened nodes.

During the sleep delay, the target may move away from the alarm node for a distance. Then it is unnecessary for nodes within this distance to wake up, since the target has already passed by. Meanwhile, all the nodes in an awake region must in the one-hop transmission range of the alarm node. Therefore, an awake region should be in a ring shape, i.e., the part between two concentric circles.

Beyond the effort that limits the awakened nodes within an awake region, the number of awakened nodes can be further reduced by choosing only some nodes in the awake region as awakened nodes. Based on our prediction on the target's moving directions, the probabilities that the target moves along various directions are different. Obviously the number of awakened nodes along a direction with a lower probability could be less than the number along a direction with a higher probability. By choosing an awakened node based on a probability related to the moving directions, awakened nodes can be reduced significantly.

The term awake region, as we use it, is similar to the concept of a cluster used in the network architecture work (e.g., [17,95]) in that it encompasses some of a cluster's functions. However, unlike a cluster's head, neither an alarm node aggregates data from member nodes of the awake region, nor it imposes any control over members. An alarm node's responsibility here is just to broadcast an alarm message on detecting a target. An alarm message carries some target related information so that the neighbors (i.e. candidate nodes) could make decisions on whether or not to get prepared for the approaching target and how to prepare. In fact, an awake region is only a virtual concept. No functions are built upon this concept, except for the selection of awakened nodes. Only nodes within an awake region is likely to detect the target after the sleep delay, and only some of these nodes need to reschedule their sleep patterns preparing for the approaching target.

We first discuss the proactive wake-up mechanism based on awake regions, then introduce the two efforts for reducing the number of awakened nodes.

Proactive Wake-up with Awake Regions

First we present an awake region management mechanism as follows, which is different from cluster management. This mechanism is implemented in the three distributed procedures in

the algorithm description.

- *Creation.* On detecting a target, a sensor node will check its own status to determine if it is an awakened node in an existing awake region. If yes, it justifies if the target is leaving the current awake region. If an awake region exists and the target is not going to move out of the current awake region, the node does nothing. Otherwise if no previous awake region exists or if the target is leaving the current awake region, the node will run an alarm node election algorithm, e.g. [96], to decide whether or not to assume an alarm node's responsibility. If this node is elected as the alarm node, it broadcasts an alarm message to all the candidate nodes. On receiving this alarm message, each candidate node individually decides if it is in the scope of this awake region and whether or not to schedule the sleep pattern. Finally, a new awake region comes into being when every awakened node schedules their sleep patterns specifically for the approaching target.
- *Dismissal.* As time progresses, the sleep patterns of awakened nodes will automatically recover back to the default pattern, thus the awake region will be dismissed automatically. There is no explicit dismissal mechanism needed.

The approach for electing an alarm node of [96] is as follows. Upon detection, each node broadcasts a DETECTION message to nodes nearby containing a time stamp recording when the detection is declared. Then it checks all the DETECTION messages received from nodes nearby within an interval, and compares the time stamps of other nodes with its own. Nodes that detect a neighbor node's time stamp is earlier than its own simply keep silent. In fact, a simpler approach also works well. Without any alarm election algorithm used, multiple alarm messages may be broadcast from multiple nodes that detect the same target. Then on receiving the first alarm message, a neighbor node may simply ignore the following ones sent by nodes that are within a $2r$ distance from the first alarm node, as these alarms may be considered as for the same target.

For convenience of discussion, we refer to a sensor node's two working modes as *default mode* and *tracking mode*. When the sleep pattern of a sensor node is not scheduled for a specific target, it works in the default mode and follows the default sleep pattern. When it detects a target or decides to reschedule the sleep pattern on receiving an alarm message, it enters the tracking mode. This rescheduled sleep pattern has an end time point, after which the node will return to the default mode automatically. In other words, the proactive wake-up depends on the alarm broadcasting, and whether to actually wake up proactively is decided by a candidate node itself.

An alarm message contains the following information that is used for a candidate node to make decision and compute the scope of an awake region:

- ID and the position of the alarm node (id_r, x_r, y_r);

- The state vector $State(n)$; and
- The prediction results, including $\overrightarrow{S_{n+1}}'$, $\mu_{S_{n+1}}$, $\sigma_{S_{n+1}}$, and $\sigma_{\Delta_{n+1}}$.

Constraint on the Awake Region Scope

Denote d the distance of an awakened node from the alarm node. Next we determine an awake region's scope by deciding the value scope of d.

As previously discussed, the target may move by S_{n+1} during the sleep delay TP. Here we make the same approximate assumption that the target's position is exactly the alarm node's position to simplify the computation. If we set $d \geq \mu_{S_{n+1}} - \sigma_{S_{n+1}}$, the probability that awakened nodes cannot cover the target after a sleep delay will be less than 16%. Moreover, it is obvious that $d \leq R$ because nodes outside of the alarm node's transmission range cannot be awakened. Therefore we determine the scope of an awake region as $\max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\} \leq d \leq R$. Thus the number of nodes in an awake region is $\rho\pi\left[R^2 - \max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\}^2\right]$, where ρ is the node density. Figure 4.6 also shows the scope of an awake region, which is filled with diagonals.

Awakened Nodes Selection in an Awake Region

So far the computation of an awake region's scope depends on the target's scalar speed only. Moreover, the decrement percentage of the number of awakened nodes is only $\frac{\max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\}^2}{R^2}$ (e.g., 6.25% when $R = 60$, $\mu_{S_{n+1}} = 20$, and $\sigma_{S_{n+1}} = 5$), which is not significant enough for enhancing energy efficiency.

As discussed previously, only some of the member nodes in an awake region need to be awakened. By taking into account the prediction results on moving directions, we can further reduce the number of awakened nodes in an awake region so as to save more energy than solely constraining the scope of an awake region.

Since the probability that a target moves along the direction of $\overrightarrow{S_{n+1}}'$ (i.e. $E[\Delta_{n+1}]$), denoted as θ, is the highest, we force all the nodes on this direction to be awakened. As Δ_{n+1} decreases on other directions, the number of awakened nodes on those directions can also be decreased. Define the probability that a candidate node on the direction $(\theta + \delta)$ reschedules its sleep pattern (i.e., becomes an awakened node) as

$$P_{ss}(\delta) = \frac{f_{\Delta_{n+1}}(\delta)}{f_{\Delta_{n+1}}(0)} = \begin{cases} -\frac{1}{p}\delta + 1 & , \quad (\delta \geq 0) \\ \frac{1}{p}\delta + 1 & , \quad (\delta < 0) \end{cases} \tag{4.7}$$

where "ss" means sleep scheduling.

Then the total number of awakened nodes in an awake region would be

$$\begin{aligned} N &= \int_{-\pi}^{\pi} P_{ss}(\delta) \cdot \frac{\rho\pi(R^2 - \max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\}^2)}{2\pi} d\delta \\ &= \rho(R^2 - \max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\}^2) \cdot \int_0^{\pi} \big(-\frac{1}{p}\delta + 1\big) d\delta \\ &= \frac{\sqrt{6}}{2}\rho\sigma_{\Delta_{n+1}}(R^2 - \max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\}^2) \end{aligned}$$

As an example, the number of awakened nodes of TPSS is only approximately 19% of that of the Circle scheme when $R = 60$, $\mu_{S_{n+1}} = 20$, $\sigma_{S_{n+1}} = 5$, and $\sigma_{\Delta_{n+1}} = \frac{\pi}{6}$. In another word, the energy consumption of TPSS is only about 19% of that of the Circle scheme.

4.3.2 Sleep Scheduling for Awakened Nodes

After reducing the number of awakened nodes, energy efficiency can be enhanced further by scheduling the sleep patterns of awakened nodes, as not all the awakened nodes need to keep active all the time. We schedule the sleep patterns of awakened nodes by setting a start time and an end time of the active period. Out of this active period, awakened nodes do not have to keep active. Therefore the time that an awakened node has to keep active could be reduced compared with the Circle scheme.

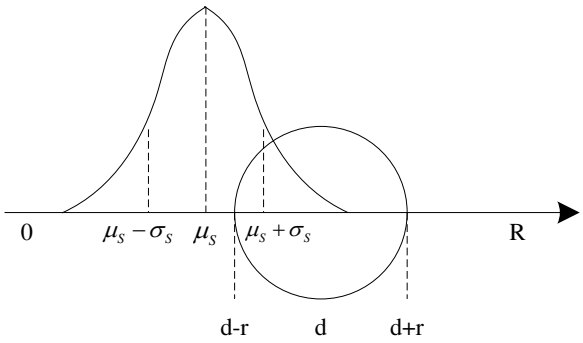


Figure 4.7: Relationship between S_{n+1} and d

As previously stated, the distance of an awakened node to the alarm node is $\max\{\mu_{S_{n+1}} - \sigma_{S_{n+1}}, 0\} \leq d \leq R$. At the moment that an awakened node receives the alarm message (i.e. after the sleep delay, we denote this time point as $t_{alarmed}$), the relationship between the awakened node's position and the distribution of the target's displacement length during a sleep delay is shown in Figure 4.7. In the figure, 0 means the position of the alarm node and r is the sensing range of nodes. According to the relative positions of the awakened node and the target's expected position after the sleep delay, we make the sleep scheduling decisions as follows.

When $\mu_{S_{n+1}} \geq d - r$, the awakened node is required to wake up immediately (i.e. at $t_{alarmed}$) since it is expected that the target has probably entered its sensing range. When $\mu_{S_{n+1}} < d - r$, the awakened node is required to wake up at $t_{alarmed} + \frac{d-r-\mu_{S_{n+1}}}{TS}$, where we suppose $TS = \frac{\mu_{S_{n+1}}}{TP}$ to be the average speed in the awake region. In both cases, the awakened node needs to keep active until $t_{alarmed} + \frac{d+r-(\mu_{S_{n+1}}-\sigma_{S_{n+1}})}{TS}$, i.e., when the probability that the target has moved out of the sensing range of the awakened node is greater than 84%. For the convenience of discussion, we denote $t_{start} = t_{alarmed} + \frac{max\{d-r-\mu_{S_{n+1}},0\}}{TS}$ and $t_{end} = t_{alarmed} + \frac{d+r-(\mu_{S_{n+1}}-\sigma_{S_{n+1}})}{TS}$. In summary, the rescheduled active period of an awakened node is

$$\left[t_{start} = t_{alarmed} + \frac{max\{d - r - \mu_{S_{n+1}}, 0\}}{TS}, t_{end} = t_{alarmed} + \frac{d + r - (\mu_{S_{n+1}} - \sigma_{S_{n+1}})}{TS}\right] \quad (4.8)$$

And the time of keeping active is,

$$T_{active} = \frac{min\{2r + \sigma_{S_{n+1}}, d + r - (\mu_{S_{n+1}} - \sigma_{S_{n+1}})\}}{TS}$$

Each awakened node is required to suspend its active/sleep toggling period for keeping active in the tracking mode, and then resume duty cycling after recovering to the default mode. Thus other than the timer for periodic active/sleep toggling (or *default timer*), a new wake-up timer (or *tracking timer*) is needed for the proactive wake-up. Compared with protocols that have no sleep scheduling, TPSS can save more energy by scheduling the sleep pattern instead of keeping nodes active all the time.

4.4 Algorithm Descriptions

Section 4.4 presented the rules for conserving the energy consumption, including reducing the number of awakened nodes, scheduling their sleep patterns and leveraging the redundant alarm messages of interfering targets. However for the actual implementation, all of these mechanisms have to be distributed on each sensor node. In this section, we present detailed algorithm descriptions for TPSS protocol in three procedures.

Procedure 1 is a handler for the event of detecting a target, which can be triggered by an interrupt that is raised on sensing something.

For the formation frequency of awake regions, the MCTA algorithm [13] uses a "refresh time" concept. Instead, we use the target's motion trend as the criterion: when the target moves close to the edge of the current awake region, a sensor node, who detects the target is leaving and is elected as the alarm node, broadcasts an alarm message to wake up neighbors and form a new awake region.

Algorithm 1 *OnDetectingTarget()* — Triggered when detecting a target

1: **if** (I am in tracking mode) **then**
2: **if** (The target is *NOT* leaving the current awake region) **then**
3: return;
4: **end if**
5: **end if**
6: (Optional:) Run an alarm election algorithm;
7: **if** (I am selected as the alarm node) **then**
8: Calculate $\overrightarrow{v_n}$ and $\overrightarrow{a_n}$ with Equation 4.1;
9: Predict $\overrightarrow{S_{n+1}}'$ with Equation 4.2;
10: Compute $\mu_{S_{n+1}}, \sigma_{S_{n+1}}, \sigma_{\Delta_{n+1}}$ with Equation 4.3 and 4.6;
11: Broadcast $id_r, x_r, y_r, State(n), \overrightarrow{S_{n+1}}', \mu_{S_{n+1}}, \sigma_{S_{n+1}}, \sigma_{\Delta_{n+1}}$;
12: **end if**
13: return;

Algorithm 2 *OnAlarmMsg()*—Triggered when receiving an alarm message

1: Compute the distance d to the alarm node;
2: **if** $(d < \mu_{S_{n+1}} - \sigma_{S_{n+1}})$ **then**
3: return; // constrain the scope of an awake region
4: **end if**
5: Compute δ with the alarm node position, my position and $\overrightarrow{S_{n+1}}'$;
6: Generate a random number $random = [0, 1]$;
7: **if** $(random > P_{ss}(\delta))$ **then**
8: return; // select awakened nodes
9: **end if**
10: Compute t_{start} and t_{end} with Equation 4.8;
11: SetTrackingTimer(t_{start}); // sleep until t_{start}
12: return;

Procedure 2 describes a sensor node's actions upon receiving an alarm message. This procedure can also be implemented as an interrupt handler.

In step 2 of Procedure 2, the node determines whether or not it is in the scope of the awake region. In step 6, the node decides whether to be an awakened node or not. Finally in step 10, the tracking timer is set so that the node can wake up at the scheduled time point.

Procedure 3 describes the tracking timer processing procedure, which controls the scheduled wake-up/sleep and the mode switch.

The computation workload of TPSS protocol is mainly located in step 6 and 7 of Procedure 1, i.e., the calculation for $\overrightarrow{v_n}$, $\overrightarrow{a_n}$, and the prediction for $\overrightarrow{S_{n+1}}', \mu_{S_{n+1}}, \sigma_{S_{n+1}}, \mu_{\Delta_{n+1}}, \sigma_{\Delta_{n+1}}$. Thus the computation workload is aggregated on the alarm node, which is more energy effective

Algorithm 3 *OnTrackingTimer()* — Triggered when the tracking timer is out

```
1: if (mode == "default") then
2:     mode = "tracking";
3:     SuspendDefaultTimer();
4:     SetTrackingTimer(t_end); // keep active until t_end
5: else
6:     mode = "default";
7:     ResumeDefaultTimer();
8: end if
9: return;
```

than distributed algorithms.

Most of the computations of the three procedures can be completed in the transport layer. The only support that TPSS needs from the link layer (usually a MAC protocol) is an interface for controlling the tracking timer, and suspending/resuming the default active/sleep toggling period.

4.5 Analysis

Based on the target prediction and the two approaches for saving energy for single target tracking, we analyze the detection delay and the tracking probability of TPSS. In [4] and [6], the authors discuss the tracking probability and the detection delay by leveraging two general results from theory of probability. Here we utilize a similar computation process, but add the impact of our target prediction and sleep scheduling protocol. The process of the computation is as follows.

First, we calculate the probability that a single node detects the target, denoted as $P_{single}(Y|\delta = \hat{\delta})$, where Y represents the event of successful detection, and $\hat{\delta}$ is a specific moving direction of the target. This probability is dependent on the relative position of the node to the target's potential route. Thus for all the possibilities where a node may detect the target, this probability needs to be normalized. We integrate it over the area where the node may locate, and compute a normalized probability for single node detection, denoted as $P^1(Y|\delta = \hat{\delta})$. Here the superscript 1 means single node detection.

Then, we calculate $P(Y|\delta = \hat{\delta})$ for the moving direction $\hat{\delta}$, i.e., the probability that at least one node around $\hat{\delta}$ detects the target before it moves away for a distance L. And we normalize it as $P(Y)$ over all the possible directions.

Finally, we compute $E[T_{detection}|\delta = \hat{\delta}]$ for the moving direction $\hat{\delta}$, i.e., the expected detection delay on $\hat{\delta}$. And we normalize it as $E[T_{detection}]$ (or $D_{detection}$) over all the possible directions.

These calculation operations are based on two general results from theory of probability [4]. The first result is, if the probability of an event A occurring in a single experiment is p, and if the number of experiments conforms to a Poisson Distribution with parameter λ, the probability of event A occurring at least once in the series of experiments is $P = 1 - e^{-p\lambda}$. The second result is, the expected value of a nonnegative random variable is $E[X] = \int_0^\infty [1 - P(X \leq x)]dx$.

In this section, we first introduce the definition of a detection area and its division, then present the computation step by step.

4.5.1 Detection Area

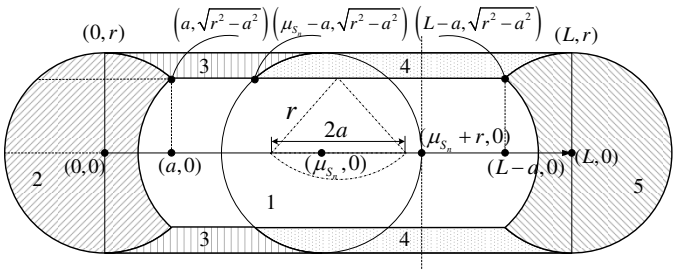


Figure 4.8: Detection probability and detection delay

Suppose that a target is at $(0, 0)$ at time point t_n, and moves for a distance L along the direction $\hat{\delta}$ until it arrives at $(L, 0)$. As shown in Figure 4.8, all the nodes that have a chance to detect the target locate in the rectangle or two semi-circles. Since tracking is a continuous detecting process, a new detection process may be started after a target is already inside the surveillance field. Therefore nodes in the semi-circle on the left side may also detect the target.

For the simplification of discussion, we let $Area_{total}$ denote this detection area where nodes that may detect the target locate. We divide $Area_{total}$ into five regions, labeled with 1 to 5 and filled with different patterns (except area 1 that is not filled) in the figure. Let $Area_i$ denote the i^{th} area. At the same time, $Area_{total}$ and $Area_i$ are also used to represent the size of these areas.

In average, the point $(\mu_{S_{n+1}}, 0)$ is the position of the target after a sleep delay TP. Therefore before the target arrives at $(\mu_{S_{n+1}}, 0)$, all the nodes work in the default mode. Only after that, awakened nodes can enter the tracking mode.

Suppose that the target's route overlaps with a node's sensing area by l. Then the probability that this node detects the target is $DC + \frac{l}{TP \cdot TS}$.

We define a parameter a as in

$$DC + \frac{2a}{TP \cdot TS} = 1$$

where $TS = \frac{\mu_{S_{n+1}}}{TP}$. Thus

$$a = \frac{1}{2}(1 - DC)\mu_{S_{n+1}}$$

Thus when $l \geq 2a$, the node must be able to detect the target, i.e., the detection probability will be 1. Bounded by a, it is obvious that all the nodes in $Area_1$ hold a detection probability of 1. The other four areas are divided based on two criteria: 1) if it is possible that a node enters its tracking mode when the target passes its sensing area; and 2) whether or not the target's route fully overlaps with the sensing area of a node. For all the nodes that may enter the tracking mode when the target passes by, the detection probability should be expressed in two parts, i.e., for the default mode and for the tracking mode. For those nodes that only partially overlap with the target's route, the probability is relatively lower than those nodes that fully overlap. Regarding to these two criteria, the other four areas are divided as in Table 4.1.

Table 4.1: Division of areas

Areas	2	3	4	5
Criterion 1	No	No	Yes	Yes
Criterion 2	No	Yes	Yes	No

4.5.2 Single Node Detection Probability

For $Area_2$ and $Area_3$, nodes always work in the default mode when they detect the target. Thus the detection probability is $DC + \frac{l}{TP \cdot TS}$. However for nodes in $Area_4$ or $Area_5$, they may work in either the default mode or the tracking mode, according to the scheduling probability $P_{ss}(\delta)$ defined in Equation 4.7. If they work in the default mode, the detection probability is also equal to $DC + \frac{l}{TP \cdot TS}$. If they work in the tracking mode, i.e., selected to be awakened nodes, the detection probability is equal to 1. This is decided by the sleep scheduling scheme discussed in Section 4.3.2, i.e., nodes will wake up when the target passes by. Actually the detection probability may not be strictly 1, as the target may change the movement status abruptly. But this is very rare for the real moving persons or vehicles. We suppose this probability exactly as 1 to simplify the computation. Thus for nodes in $Area_4$ or $Area_5$, the detection probability can be expressed as,

$$(1 - P_{ss}(\delta))\big(DC + \frac{l}{TP \cdot TS}\big) + P_{ss}(\delta) \cdot 1$$

Here another simplification is to utilize $P_{ss}(\delta)$ for all the nodes in $Area_4$ and $Area_5$. We make this simplification approach with two reasons. First, $R >> r$ and L is close to R, thus

$L >> r$. In another word, the area shown in Figure 4.8 is a sector-like shape in the alarm node's transmission range, with a very small central angle as shown in Figure 4.9. Therefore the difference between δ of different nodes is small. Second, all the areas are symmetric about the x-axis, and all the probabilities to be integrated are even functions. Thus the difference introduced by $P_{ss}(\delta)$ can be counteracted to the most extent.

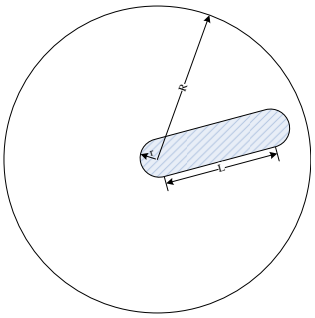


Figure 4.9: Detection area in the transmission radius

For a node (x, y) in $Area_3$ or $Area_4$, the target's route fully overlaps with their sensing area. Thus $l = 2\sqrt{r^2 - y^2}$. For a node (x, y) in $Area_2$ or $Area_5$, the target's route partially overlaps with their sensing area. Thus $l = x + \sqrt{r^2 - y^2}$ or $l = L - x + \sqrt{r^2 - y^2}$ respectively.

In summary, the single node detection probability and the area of all the five areas are listed in Table 4.2.

Table 4.2: Single node detection probability and size of areas

$Area_i$	$P_{single}(Y\|\delta = \hat{\delta})$	Size
1	1	$2r^2 \arccos\frac{a}{r} + 2(L - 3a)\sqrt{r^2 - a^2}$
2	$DC + \frac{x+\sqrt{r^2-y^2}}{TP \cdot TS}$	$\Phi(r, a)$
3	$DC + \frac{2\sqrt{r^2-y^2}}{TP \cdot TS}$	$2\mu_{S_{n+1}}(r - \sqrt{r^2 - a^2}) - \Phi(r, a) + 4a\sqrt{r^2 - a^2}$
4	$(1 - P_{ss}(\delta))\big(DC + \frac{2\sqrt{r^2-y^2}}{TP \cdot TS}\big) + P_{ss}(\delta)$	$2(L - \mu_{S_{n+1}})(r - \sqrt{r^2 - a^2})$
5	$(1 - P_{ss}(\delta))\big(DC + \frac{L-x+\sqrt{r^2-y^2}}{TP \cdot TS}\big) + P_{ss}(\delta)$	$\Phi(r, a)$

Here $\Phi(r, a) = \pi r^2 - 2r^2 \arccos\frac{a}{r} + 2a\sqrt{r^2 - a^2}$.

Next we integrate and normalize $P_{single}(Y|\delta = \hat{\delta})$ over all the five areas.

$$\begin{aligned} P^1(Y|\delta=\hat{\delta})\cdot Area_{total} &= \int_{Area_{total}} P_{single}(Y|\delta=\hat{\delta})ds \\ &= \sum_{i=1}^{5}\int_{Area_i} P_{single}(Y|\delta=\hat{\delta})ds \\ &= P^1_{random}(Y|\delta=\hat{\delta})\cdot Area_{total} + P^1_{plus}(Y|\delta=\hat{\delta})\cdot Area_{total} \end{aligned}$$

Here $P^1_{random}(Y|\delta=\hat{\delta})$ is the normalized probability when all the nodes work only in the default mode without sleep scheduling, and $P^1_{plus}(Y|\delta=\hat{\delta})$ represents the enhancement on the probability introduced by TPSS. In fact, the normalized probability when all the nodes work only in the default mode without sleep scheduling $P^1_{random}(Y|\delta=\hat{\delta})$ is independent of δ. Thus in the following discussion we abbreviate it as P^1_{random}. We have,

$$P^1_{random}\cdot Area_{total} = DC\cdot Area_{total} + \frac{L\cdot\Phi(r,a) - 2a\big[\Phi(r,a)-\pi r^2\big]}{TP\cdot TS}$$

and

$$P^1_{plus}(Y|\delta=\hat{\delta})\cdot Area_{total} = \frac{P_{ss}(\delta)}{TP\cdot TS}\{(L-\mu_{S_{n+1}})\big[4ar-\Phi(r,a)\big] + 2a\cdot\Phi(r,a) - \frac{8}{3}\Gamma(r,a)\}$$

where $\Gamma(r,a) = r^3 - (\sqrt{r^2-a^2})^3$. Let

$$A = (2-DC)\Phi(r,a) - \frac{8}{3TP\cdot TS}\Gamma(r,a) - 4ar$$

and

$$B = \frac{L}{TP\cdot TS}\big[4ar-\Phi(r,a)\big] + A$$

for concise expression, we have $P^1_{plus}(Y|\delta=\hat{\delta})\cdot Area_{total} = B\cdot P_{ss}(\delta)$.

4.5.3 Detection Probability

As the node density is D_s, we suppose that the number of nodes n in $Area_{total}$ conforms to Poisson distribution with the mean $\lambda = D_s\cdot Area_{total}$. According to the result from theory of probability, we have,

$$\begin{aligned} P(Y|\delta=\hat{\delta}) &= 1 - e^{-\lambda P^1(Y|\delta=\hat{\delta})} \\ &= 1 - e^{-D_s\cdot P^1_{random}\cdot Area_{total}}\cdot e^{-D_s\cdot P^1_{plus}(Y|\delta=\hat{\delta})\cdot Area_{total}} \end{aligned}$$

Integrating and normalizing it over $(-\pi, \pi]$, we can calculate $P(Y)$ as,

$$\begin{aligned} P(Y) &= \int_{-\pi}^{\pi} P(Y|\delta = \hat{\delta}) f_T(\hat{\delta}) d\hat{\delta} \\ &= \int_{-\pi}^{\pi} \left[1 - e^{-D_s \cdot P^1_{random} \cdot Area_{total}} \cdot e^{-D_s \cdot P^1_{plus}(Y|\delta=\hat{\delta}) \cdot Area_{total}}\right] f(\hat{\delta}) d\hat{\delta} \\ &= 1 - e^{-D_s \cdot P^1_{random} \cdot Area_{total}} \cdot \int_{-\pi}^{\pi} e^{-D_s \cdot B \cdot P_{ss}(\hat{\delta})} f(\hat{\delta}) d\hat{\delta} \end{aligned}$$

Here $f_T(\hat{\delta})$ is the probability density function of the target's actual turning potential. When we also use the prediction model, i.e., Equation 4.4, to describe it, $P(Y)$ can be calculated as,

$$\begin{aligned} P(Y) &= 1 - e^{-D_s \cdot P^1_{random} \cdot Area_{total}} \cdot 2 \int_0^{\pi} e^{-D_s \cdot B \cdot (-\frac{1}{p}\hat{\delta} + 1)} (-\frac{q}{p}\hat{\delta} + q) d\hat{\delta} \\ &= 1 - \frac{2}{D_s B} \cdot e^{-D_s \cdot (P^1_{random} \cdot Area_{total} + B)} \left[(\frac{1}{D_s B} + 1 - \pi q) e^{\frac{\pi D_s B}{p}} - (\frac{1}{D_s B} + 1) \right] \end{aligned} \tag{4.9}$$

Here p and q are both coefficients for our linear prediction model.

4.5.4 Detection Delay

According to the second result from theory of probability, we have

$$E[T_{detection}|\delta = \hat{\delta}] = \int_0^{\infty} [1 - P(T_{detection} \leq t|\delta = \hat{\delta})] dt$$

And letting $L = TS \cdot t$, $P(T_{detection} \leq t|\delta = \hat{\delta}) = P(T_{detection} \cdot TS \leq L|\delta = \hat{\delta})$ is the probability that a target is detected before it enters for a distance L, i.e., $P(Y|\delta = \hat{\delta})$. Thus

$$\begin{aligned} E[T_{detection}|\delta = \hat{\delta}] &= \int_0^{\infty} e^{-D_s \cdot P^1_{random} \cdot Area_{total}} \cdot e^{-D_s \cdot P^1_{plus}(Y|\delta=\hat{\delta}) \cdot Area_{total}} dt \\ &= exp\big(- D_s \big\{ \pi r^2 - (1 - DC)\Phi(r, a) + P_{ss}(\delta) \cdot A \big\} \big) \\ &\quad \cdot \int_0^{\infty} e^{-D_s \cdot TS \cdot \left\{ 2r \cdot DC + \frac{1}{TP \cdot TS} \left[\Phi(r,a) + P_{ss}(\delta)[4ar - \Phi(r,a)] \right] \right\} t} dt \end{aligned}$$

We simplify this equation by removing $P_{ss}(\delta)$ from the exponential inside the integral. Let us check the meaning of $\Phi(r, a) + P_{ss}(\delta)[4ar - \Phi(r, a)]$. In Figure 4.10, it is easy to prove that $\Phi(r, a)$ is exactly the size of the shape filled with dots. Then $[4ar - \Phi(r, a)]$ is the size

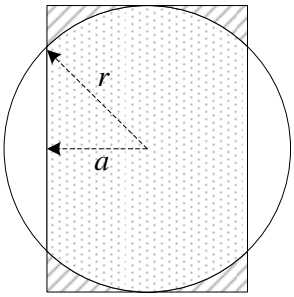


Figure 4.10: Simplify $E[T_{detection}|\delta = \hat{\delta}]$

of four small areas filled with diagonals. Compared with the dotted shape, the size of small areas filled with diagonals can be ignored, especially when multiplied with a real number $P_{ss}(\delta)$ that is less than 1. Therefore, $\Phi(r,a) + P_{ss}(\delta)[4ar - \Phi(r,a)] \approx \Phi(r,a)$, and

$$E[T_{detection}|\delta = \hat{\delta}] \approx \frac{exp\big(-D_s\big\{\pi r^2 - (1-DC)\Phi(r,a) + P_{ss}(\delta)\cdot A\big\}\big)}{D_s \cdot TS \cdot \big[2r \cdot DC + \frac{\Phi(r,a)}{TP \cdot TS}\big]}$$

Integrating and normalizing it over $(-\pi, \pi]$, we can calculate $E[T_{detection}]$ as,

$$\begin{aligned} D_{detection} = E[T_{detection}] &= \int_{-\pi}^{\pi} E[T_{detection}|\delta = \hat{\delta}] f_T(\hat{\delta}) d\hat{\delta} \\ &= \frac{e^{-D_s\left[\pi r^2 - (1-DC)\Phi(r,a)\right]}}{D_s \cdot TS \cdot \big[2r \cdot DC + \frac{\Phi(r,a)}{TP \cdot TS}\big]} \cdot \int_{-\pi}^{\pi} e^{-D_s \cdot A \cdot P_{ss}(\hat{\delta})} f_T(\hat{\delta}) d\hat{\delta} \end{aligned}$$

Similarly when we also use the prediction model, i.e., Equation 4.4, to describe it, $D_{detection}$ can be calculated as,

$$\begin{aligned} D_{detection} &= \frac{e^{-D_s\left[\pi r^2 - (1-DC)\Phi(r,a)\right]}}{D_s \cdot TS \cdot \big[2r \cdot DC + \frac{\Phi(r,a)}{TP \cdot TS}\big]} \cdot 2\int_0^{\pi} e^{-D_s \cdot A \cdot (-\frac{1}{p}\hat{\delta}+1)}(-\frac{q}{p}\hat{\delta} + q) d\hat{\delta} \\ &= \frac{2e^{-D_s\left[\pi r^2 - (1-DC)\Phi(r,a) + A\right]}}{D_s^2 A \cdot TS \cdot \big[2r \cdot DC + \frac{\Phi(r,a)}{TP \cdot TS}\big]} \cdot \big[(\frac{1}{D_s A} + 1 - \pi q) e^{\frac{\pi D_s A}{p}} - (\frac{1}{D_s A} + 1)\big] \end{aligned} \tag{4.10}$$

Here p and q are both coefficients for the linear prediction model, and

$$A = (2 - DC)\Phi(r,a) - \frac{8}{3TP \cdot TS}\Gamma(r,a) - 4ar$$

4.6 Performance Evaluation

We evaluated TPSS protocol in a simulation environment programmed in C++, and compared it to Circle scheme [8] and MCTA algorithm [13].

4.6.1 Simulation Environment

In the simulation, $500 - 2,000$ nodes are randomly deployed in a $200m \times 200m$ area, i.e., the node density is $0.5 - 4.0$ $nodes/100m^2$. Fifteen target speed values in an arithmetic progression ($\{2, 4, \ldots, 30\}$ m/s) are examined. We design a random curvilinear motion model to describe the actual target motion.

Table 4.3: Energy consumption rates

Status	Energy consumption rate (unit)
Active (P_{active})	9.6 (mJ/s)
Transmit (P_{send})	720 (nJ/bit), 5.76 ($mJ/Byte$)
Receive (P_{rcv})	110 (nJ/bit), 0.88 ($mJ/Byte$)
Sleep (P_{sleep})	0.33 (mJ/s)

We estimated the performance metrics against the node density and the target speed. In the simulation, we configured the node density as 1.25 $node/100m^2$ when retrieving data regarding to various target speeds, and the target speed as 20 m/s when retrieving data regarding to various node densities. For each value of the node density (or the target speed), we simulated 10 deployments (or the route samples), and repeated each deployment (or route sample) 10 times. We configured $r = 10$ m, $R = 60$ m, $TP = 1$ s, $DC = 10\%$, and the energy consumption rates as shown in Table 4.3.

The energy consumption data in Table 4.3 come from the actual Mica2 platform [97, 98]. We did not list the energy consumption rate for instruction execution, for it is difficult to measure in the simulation. As a compensation, we increased the energy consumption rate for TPSS's active state by 20%.

The random curvilinear motion model that we used in the experiments was completely different from the model that was established by our prediction in Section 4.2 (because otherwise everything would be perfect). Instead, we leverage some common sense based rules to imitate the actual target motion. Assume that the target moves in a random walk manner starting from a base speed value (e.g. 15 m/s), i.e., at each fixed time interval (e.g. 200 ms), the target speed may increase/decrease by 1 m/s or keep unchanged with a probability of $\frac{1}{3}$ respectively. For the moving direction, we assume that the target may turn with a random angle in $[0, \alpha_{max}]$ at each fixed time interval, where α_{max} decreases as the speed v increases. This matches the actual case because a target, especially a vehicle,

may probably turn over if it abruptly changes the direction when moving at a high speed. We describe this relation between v and α_{max} with a linear mapping $\alpha_{max} = av + b$, where a and b are linear mapping constants. Now, two (v, α_{max}) value pairs will fully determine the linear mapping. $\alpha_{max} = 25^o$ when $v = 9\ m/s$ and $\alpha_{max} = 5^o$ when $v = 30\ m/s$ were the example value pairs used in our simulation. Figure 4.11 shows an example moving route when a target traverses a tracking field of $200m \times 200m$ with a base speed $v = 20\ m/s$.

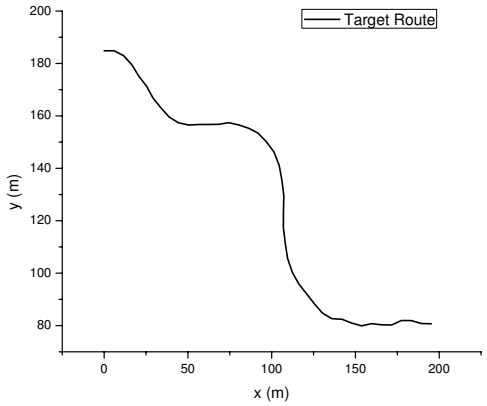


Figure 4.11: A typical target route

No sleep scheduling scheme and the Circle scheme are two extreme ones. When there is completely no sleep scheduling, all the sensor nodes sleep and wake up randomly. The energy consumption is the least, but so is the tracking performance. With the Circle scheme, all the neighbor nodes within one hop range of the alarm node are awakened, and all the awakened nodes keep active all the time until when the target is expected to leave the alarm node's transmission range. Then before awakened nodes recover back to the default sleep pattern or the target leaves the coverage area of awakened nodes, tracking performance is guaranteed, i.e., 100% covered with zero tracking delay. However, energy efficiency of the Circle scheme is the worst, which is same during its active period as the case that keeps all the nodes active all the time. Therefore, NOSS scheme and Circle scheme serve as the upper bound and the lower bound (may be reversed) on performance metrics for other protocols.

4.6.2 Experimental Results

Extra energy (EE)

We measure the extra energy consumption EE in the unit J to determine how much more energy is needed for tracking a single target than detecting nothing. Figure 4.12 and Figure 4.13 show the EEs of four protocols at various node densities and various target speeds respectively. In general, the energy consumption for tracking increases as the node density increases, or as the target speed decreases. The former is easy to understand, as denser de-

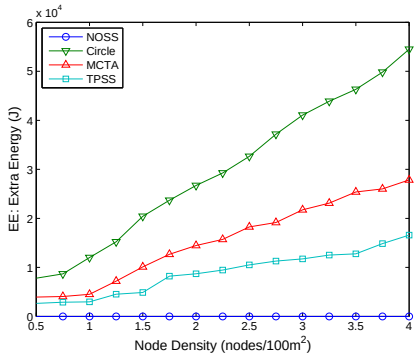


Figure 4.12: Extra energy vs. node density

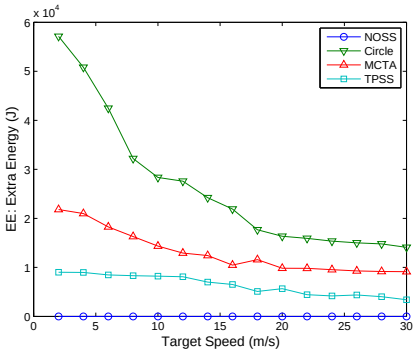


Figure 4.13: Extra energy vs. target speed

ployment means more nodes are awakened. The reason for the latter is that a slower target will stay longer in the alarm node's transmission range, thereby require awakened nodes to keep active for a longer time. In both cases, the extra energy for NOSS is the lower bound with the value 0, which is actually the basis of the definition of EE. As the scheme with the simplest sleep scheduling effort, Circle provides an upper bound for MCTA and TPSS in terms of the energy. We observe that TPSS consumes less energy than MCTA.

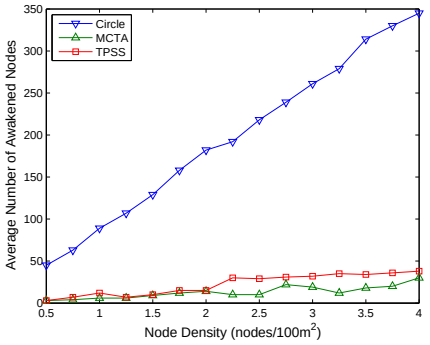


Figure 4.14: Number of awakened nodes vs. node density

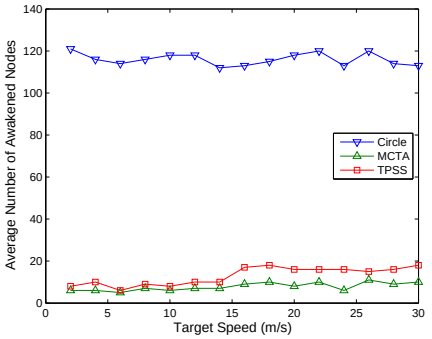


Figure 4.15: Number of awakened nodes vs. target speed

Something interesting is that, if we check the number of awakened nodes, the curves of MCTA and TPSS show reverse positions. Figure 4.14 and Figure 4.15 illustrate the number of awakened nodes (averaged for each proactive wake-up), in which the number of awakened nodes of TPSS is greater than that of MCTA.

This seeming contradiction can be explained by comparing energy for sensing and energy for communication separately. Note that energy is consumed primarily for sensing and communication. Sensing energy includes P_{active} and P_{sleep}, while communication energy includes P_{send} and P_{rcv}. Figure 4.16 shows the comparison among the three protocols (except Circle)

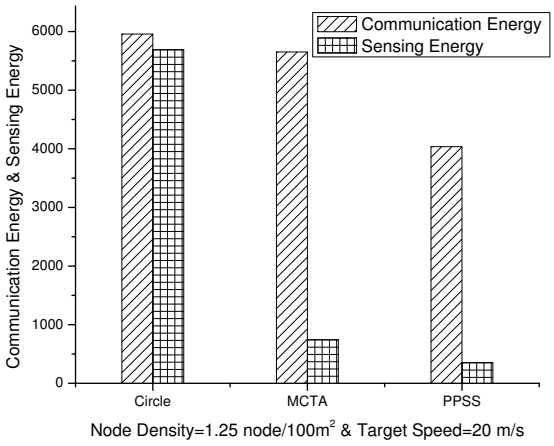


Figure 4.16: Sensing energy and communication energy when node density=1.25 $node/100m^2$ and target speed=20 m/s

for sensing energy and communication energy separately. We observe that sensing energy and communication energy of TPSS are both less than MCTA. On one hand, TPSS consumes less sensing energy because of the sleep scheduling effort that shortens the active time as much as possible. On the other hand, TPSS consumes less communication energy because the frequency of proactive wake-up of TPSS is lower than that of MCTA. TPSS awakens neighbor nodes up to one transmission range away from the alarm node, and executes the next wake-up only when the target is about to leave the alarm node's transmission range. However, the interval between two adjacent wake-up actions of MCTA depends on the refresh time which is not necessarily as long as the transmission range. Such a short time interval increases the frequency of wake-up actions, thereby increases the energy for sending wake-up messages (or alarm messages). This can be verified by checking the number of proactive wake-up actions: when the node density is 1.25 $node/100m^2$ and the target speed is 20 m/s, the average number of proactive wake-up actions of MCTA and TPSS are respectively 5.48 and 2.91. This accomplishment of TPSS is attributed to the design from its prediction scheme to the sleep scheduling mechanism.

Average detection delay (AD)

Figure 4.17 and Figure 4.18 show the average detection delay on various node densities and various target speeds respectively. The curve for the NOSS scheme is not shown in the figure, for otherwise the granularity will be too large to show the difference among Circle, MCTA and TPSS clearly. The NOSS scheme serves as an upper bound for AD, since the detection delay is the longest with no proactive wake-up completely. The values of NOSS's AD spreads around 10 s for the node density case and in $6.5 - 85$ s for the target speed case. We observe that TPSS has longer delay than Circle and MCTA as a cost for enhancing energy efficiency.

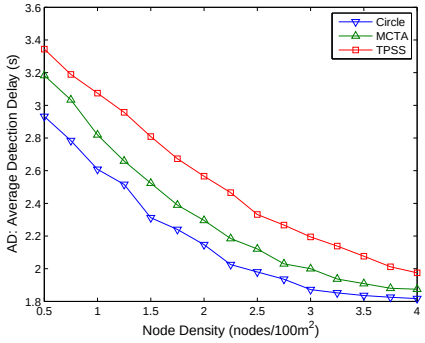


Figure 4.17: Average detection delay vs. node density

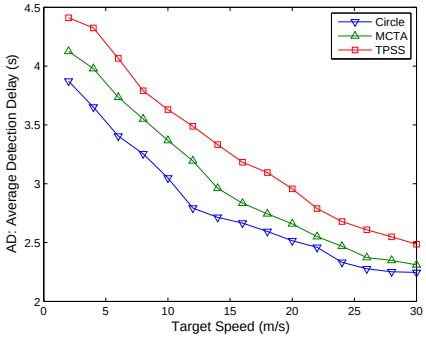


Figure 4.18: Average detection delay vs. target speed

Compared to MCTA on both EE and AD, we may observe that TPSS introduces an improvement of 25% $\sim$ 45% on energy efficiency, at the expense of only 5% $\sim$ 15% increase on detection delay.

4.7 Conclusion

In a duty-cycled sensor network, proactive wake-up and sleep scheduling can create a local active environment to provide guarantee for the tracking performance. By carefully and effectively limiting the scope of this local active environment, the energy efficiency can be improved with an acceptable loss on the tracking performance. For example, TPSS uses a prediction method based on both kinematics and probability, so that the energy consumed by low value-added sensor nodes (i.e., nodes that have a low probability of detecting the target) can be reduced. In addition, the design of TPSS protocol shows that it is possible to precisely sleep-schedule nodes without involving much physics.

Chapter 5

Sleep Scheduling for Multiple Target Tracking

5.1 Introduction

Recently, many interesting applications have emerged, which require concurrent tracking of multiple targets—e.g., search and rescue, disaster response, pursuit evasion games [99]. Like for single target tracking, energy efficiency is also a challenging problem for multiple target tracking systems.

In this chapter, we extend TPSS to multiple target tracking by presenting a Sleep Scheduling algorithm for Multiple Target Tracking (SSMTT). In TPSS protocol, an alarm node broadcasts an alarm message to activate its neighbor nodes (i.e., awakened nodes) toward preparing them to track the approaching target. When the routes of multiple targets interfere with each other, some neighbor nodes of an alarm node may have already been activated by another alarm node's alarm broadcast. Thus, our objective is to further improve energy efficiency of TPSS for multiple target tracking, by leveraging the overlapping broadcasts for multiple targets to reduce the energy consumed on proactive wake-up alarm transmission.

A problem of this multi-target-conscious algorithm is that some sensor nodes may be put into the sleep state by the alarm broadcast for a target, thereby miss the alarm broadcast for another. We present a solution to this problem by modifying the node sleep patterns.

The results from our experimental evaluations show that, compared to TPSS, SSMTT algorithm saves alarm transmission energy by $10\% \sim 15\%$ during the interfering phase.

5.2 Problem Formulation

Let an awake region be denoted as A, and an alarm node as γ. The alarm node γ will broadcast an alarm message to schedule the sleep pattern of its neighbor nodes upon detecting a target.

We assume that n targets $\{T_i | i \in [0, n-1]\}$ interfere with each other. Each target will trigger an alarm message and thus form an awake region. Let the awake regions triggered by target T_i be denoted as $\{A_{ip} | p \in N_0\}$, and their alarm nodes as $\{\gamma_{ip} | p \in N_0\}$, where N_0 is the non-negative integer set. Then, at the time that A_{ip} is formed, A_{ip} may overlap with $\{A_j q | j \in [0, i-1] \cup [i+1, n-1], q \in N_0\}$.

Next, we define the criteria for deciding whether or not SSMTT improves energy efficiency compared with tracking multiple targets separately with a single target tracking protocol, and how much it improves.

Since our basic idea is to enhance energy efficiency by leveraging the targets' interference, the energy consumption is a critical aspect in making this decision. When two targets are far away from each other such that there is no node that can detect both of them at the same time, they can be handled as two single targets with single target tracking and sleep scheduling protocols. Therefore, we only consider the difference in energy efficiency during the period when multiple targets interfere.

We define the interference period of two targets T_i and T_j as a time interval. This interval starts when the alarm nodes for T_i and T_j are close enough to hear each other for the first time ($|\gamma_{ip}\gamma_{jq}| \leq R, (p, q \in N_0)$), and ends when the alarm nodes of T_i and T_j are away from each other for the first time ($|\gamma_{ip}\gamma_{jq}| \geq R, (p, q \in N_0)$).

Let the tracking energy consumed during the interference period be denoted as E_s when tracking multiple targets through single target tracking protocols, and as E_m when tracking with SSMTT. We define the benefit that can be obtained by SSMTT as Energy Saving Ratio (ESR), where $ESR = \frac{E_s - E_m}{E_s}$.

Given these metrics, we formulate our problem as: *how to schedule the node sleep patterns and leverage the overlapping broadcasts for multiple targets, to achieve better ESR with acceptable AD and EDP loss.*

5.3 SSMTT Algorithm Design

The SSMTT algorithm is built upon TPSS protocol. Based on TPSS, we define Scheduled Period (SP) as the period that the node takes the scheduled sleep pattern instead of the default one, and Scheduled Active Period (SAP) as the period that the node does not sleep completely within a SP.

5.3.1 Energy Saving for Proactive Wake-up Alarm Transmission

Let the interfered target be denoted as T_i, and the interfering targets be denoted as $\{T_j | j \in [0, i-1] \cup [i+1, n-1]\}$. If just before a detection event, the alarm node γ_{ip} received alarm messages triggered by T_j, its awake region A_{ip} may overlap with awake regions $\{A_j q | j \in [0, i-1] \cup [i+1, n-1], q \in N_0\}$. If most of the awakened nodes in the overlapped area have already been awakened by $\{\gamma_{jq}\}$, then it will not be necessary for γ_{ip} to awaken them once again.

Now, we discuss the detailed definition and calculation of the criteria for γ_{ip} on saving this transmission energy. First, we start with the case of two targets.

Figure 5.1 shows the interference between two targets T_i and T_j. In the figure, the smallest circles are sensor nodes, and the dotted circles around them denote their sensing range. Assume that γ_{ip} just received an alarm message from γ_{jq} before it detects T_i, and it finds that there would be an overlapping area between A_{ip} and A_{jq} (the dash dotted line describes the alarm message from γ_{jq} to γ_{ip}). Then γ_{ip} divides A_{ip} into zones $\{Z_k | k \in N_0\}$ based on the distance from γ_{ip}, i.e., a point $P \in Z_k \Leftrightarrow |\overrightarrow{\gamma_{ip}P}| \in (kr, (k+1)r]$, where r is the sensing range of the sensor nodes. In the figure, the arcs $\{\widetilde{B_k} | k \in N\}$ are the boundaries between adjacent zones. The reason we divide the zones by the distance r is to facilitate the discussion on the criterion for saving the transmission.

Assume that in the overlapping area of A_{ip} and A_{jq}, there are m nodes, called Overlapping Nodes (ON) and denoted as $U_{ON} = \{ON_k | k \in [0, m-1]\}$. Now for each ON_k, we define the metric Node Overlapping Ratio (NOR), as $NOR_k = \frac{t_{overlap}}{SAP_i}$, to describe the overlapping degree of SAP_i and SAP_j. Here SAP_i and SAP_j are the SAPs scheduled respectively for the two targets, $t_{overlap}$ is the overlapping time of SAP_i and SAP_j. Figure 5.2 shows their relationship.

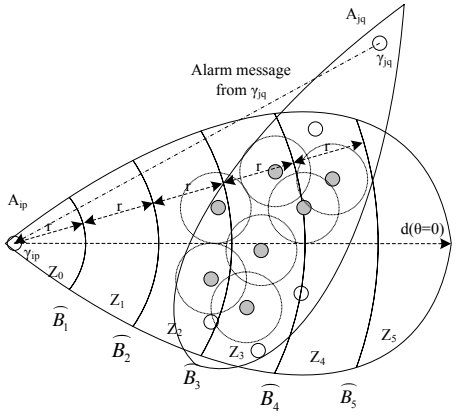


Figure 5.1: Interference of two targets

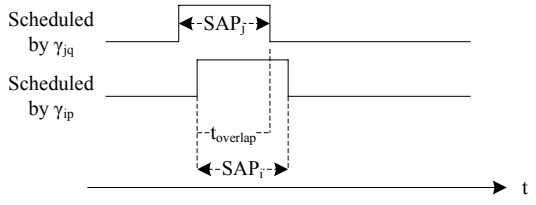


Figure 5.2: Node overlapping ratio

We call those ONs whose $NOR > THS_{NOR}$ as Reusable Nodes (RN) and denote them as

$U_{RN} = \{RN_k | k \in [0, m-1]\}$, where THS_{NOR} is a threshold specific for a given implementation. In Figure 5.1, the RNs are shown with solid gray circles. If the sensing coverage area of RNs in a zone Z_k is large enough so as to cover most of Z_k's area, the awakened nodes in Z_k may be omitted in γ_{ip}'s alarm broadcasting. For each zone Z_k, we define the metric Zone Overlapping Ratio (ZOR) to describe the overlapping degree of two targets in this zone. The overlapping ratio of Z_k, denoted ZOR_k, is calculated as:

$$ZOR_k = \frac{\bigcup S_{RN}}{S_{all}} \tag{5.1}$$

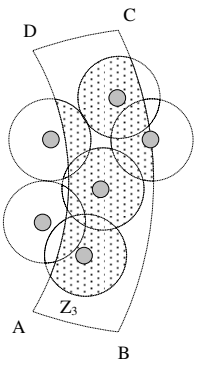


Figure 5.3: Definition of ZOR

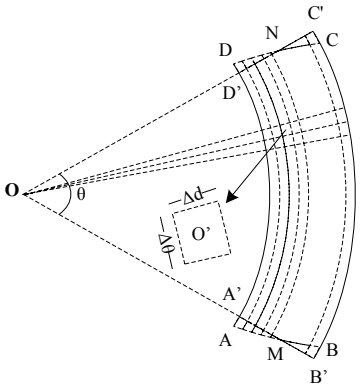


Figure 5.4: Calculation of covered area

Here S_{RN} is the covered area in Z_k of all the Z_k's RNs, and S_{all} is the total area of Z_k. Figure 5.3 shows the definition of ZOR_k, in which we use Z_3 in Figure 5.1 as the example. In Figure 5.3, $\bigcup S_{RN}$ is shown as the dotted area, and S_{all} equals to the area of $\overline{ABCD}$. We call those zones whose $ZOR_k > THS_{ZOR}$ as Reusable Zones (RZ) and denote them as $U_{RZ} = \{Z_k | k \in N_0\}$, where THS_{ZOR} is a threshold specific for a given implementation.

The core idea of the energy saving effort for proactive wake-up alarm transmission is (1) to cancel the alarm broadcast completely if a zone that is close to the alarm node is reusable, and (2) to reduce the transmission power of the alarm broadcast if a zone that is far from the alarm node is reusable and all the zones that are closer to the alarm node than it are not reusable.

Based on the case of two targets interference, we calculate ZOR_k using Equation 5.1 for the multiple target case, too. However, the difference is that the RNs for calculating $\bigcup S_{RN}$ includes the reusable nodes in all of the overlapping areas of A_{ip} and each interfering target.

Next, we present the calculation of $\bigcup S_{RN}$. To reduce the computational complexity, we adopt an approximate approach and again discuss with Z_3 in Figure 5.1 as the example. In Figure 5.4, $Z_3 = \overline{ABCD}$ is determined by the awake region A_{ip}'s edges, and the boundaries $\widetilde{B_3}$ and $\widetilde{B_4}$. O is the position of γ_{ip}. M and N are points on the intersection of A_{ip}'s edges and a circle with center O and radius $3.5r$ (i.e., $|OM| = |ON| = 3.5r$). θ is the central angle

corresponding to the arc $\widetilde{MN}$. Now, A', B', C', and D' are points on the intersection of line OM, line ON, and the boundaries $\widetilde{B_3}$ and $\widetilde{B_4}$.

We use the area $\overline{A'B'C'D'}$ as the approximation of $\overline{ABCD}$, and divide $\overline{A'B'C'D'}$ evenly into small "sectors" with concentric circles centered at O and lines through O. Here, sectors are not mathematical sectors but more like disk sectors in the context of computer disk storage. Each sector has a segment with the radius $\Delta d = \frac{r}{a}$ and corresponds to a central angle $\Delta\theta = \frac{\theta}{b}$, where a and b are application-specific constants.

Let a sector be denoted as $C(d, \theta)$. In polar coordinates with the radial coordinate ρ and the polar angle α, $C(d, \theta)$ is determined by the circle $\rho = d$, circle $\rho = d + \Delta d$, line $\alpha = \theta$, and line $\alpha = \theta + \Delta\theta$. Sector $C(d, \theta)$'s central point O' (i.e., $(d + \Delta d/2, \theta + \Delta\theta/2)$ in polar coordinates) is used as the representative of the sector. If O' is covered by a sensor node's sensing range, we consider that $C(d, \theta)$ is covered by this node.

Now, we approximate the calculation of ZOR_k in Equation 5.1 as $ZOR_k = \frac{SN_{RN}}{SN_{total}}$, where SN_{RN} is the number of sectors that are covered by reusable nodes, and SN_{total} is the total number of sectors in a zone.

5.3.2 Preventing Alarm Messages from being Missed

The consequence of the SSMTT mechanism is that, when a sensor node is scheduled to sleep by an alarm message, it may miss the alarm message broadcast for other approaching targets. Our solution to this problem is to force the awakened node, which has been scheduled to sleep until the expected target arrival time, to wake up with the default toggling cycle and an extremely short duty cycle. The only purpose of this is to check alarm messages from other approaching targets.

5.4 SSMTT Algorithm Description

To record each approaching target, a sensor node needs to manage an Alarm Message Database (AMD). Each entry in the AMD records all the information transmitted by an alarm message including target ID, awake region ID, γ's position, the target's instantaneous movement status, and TTL (i.e., time to live) et al. The AMD is updated whenever the node receives an alarm message, irrespective of whether it will become a member of the awake region.

When a node wakes up, it changes its sleep pattern according to the scheduled result and sets the wake-up timer for the subsequent wake-up. During this active period, it may detect a target or receive an alarm message, and corresponding interrupt handlers for them will be released for execution. The main function of the SSMTT algorithm is implemented in Algorithm 4.

Algorithm 4 Calculation of Reusable Zones

```
for all entries in AMD do
    Decide U_ON; // Calculate the set of overlapping nodes
end for
U_RN = φ; // Initialize the set of reusable nodes
for all ON_k in U_ON do
    Calculate NOR_k;
    if (NOR_k > THS_NOR) then
        U_RN = U_RN + ON_k; // Calculate the set of reusable nodes
    end if
end for
U_RZ = φ; // Initialize the set of reusable zones
for all zones Z_k in A_ip do
    SN_RN = 0;
    SN_total = 0;
    for all sectors C(λ, α) in Z_k do
        SN_total + +; // Summarize all the sectors in a zone
        if central point O' of C(λ, α) is covered by a RN_k (RN_k ∈ U_RN) then
            SN_RN + +; // Account for the sectors that are covered by reusable nodes
        end if
    end for
    ZOR_k = SN_RN/SN_total;
    if (ZOR_k > THS_ZOR) then
        U_RZ = U_RZ + Z_k; // Calculate the set of reusable zones
    end if
end for
return U_RZ;
```

5.5 Performance Evaluation

5.5.1 Simulation Environment

We evaluated SSMTT algorithm against three sleep scheduling protocols for single target tracking: Circle scheme [8], MCTA [13], and TPSS. As discussed in Section 5.1, the reason that we did not compare SSMTT with other sleep scheduling protocols with multiple target tracking support is because, we are not aware of any previous works in this area.

In the evaluation, 400 nodes were deployed in a 20×20 grid structure in a $100m \times 100m$ area. We tracked $2 \sim 10$ targets concurrently, which move with the speeds $\{3, 6, 9, 12, 15, 18, 21, 24, 27\}$ in Uniform Rectilinear Motion (URM) mode. For each combination of protocol, number of target, target speed, and interfering angle, we simulated 50 cases and reported the average

results. The energy consumption data comes from the actual Mica2 platform [87, 100]. The energy consumption with variable transmission power is determined with a model developed using curve fitting based on the empirical measurements in [100].

5.5.2 Simulation Results

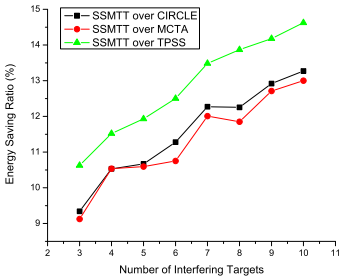


Figure 5.5: ESR vs. number of targets

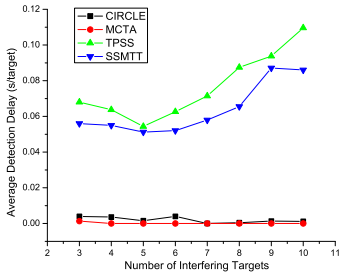


Figure 5.6: AD vs. number of targets

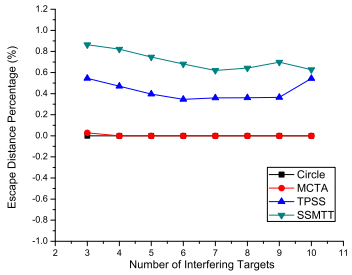


Figure 5.7: EDP vs. number of targets

Figure 5.5 shows ESR on alarm communication of SSMTT over the other three reference single target tracking protocols under different numbers of targets. We can observe that the energy saved increases as the number of interfering targets increases. This is because that the more targets interfere, the more overlapping broadcasts can be saved. Although TPSS protocol's energy consumption on alarm transmission is the most, its overall energy consumption is still better than Circle and MCTA [42]. Figure 5.6 shows AD of the four simulated protocols under different numbers of targets. Both TPSS and SSMTT introduce an increasing detection delay. However, this performance loss is acceptable, since in most of the cases the increased delay is within 0.1 second for each target per interference. Figure 5.7 shows the escape distance percentage (EDP) of the four simulated protocols under different numbers of targets. SSMTT introduces a little increase. But like AD, this performance loss is negligible compared with the ESR enhancement.

We also studied the correlation between ESR and the target speed. In the two targets case, the correlation is shown in Figure 5.8. Basically ESR decreases as the target speed increases. This is because that the randomness of targets will increase significantly as the speed increases, therefore the overlap among interfering targets' scheduling will decrease. Figure 5.9 shows the correlation between ESR and the interfering angle for two targets case. We can observe that little interfering angle for two targets moving on the same direction presents the best energy saving ratio, and the worst case occurs when the interfering angle is close to $\frac{2\pi}{3}$.

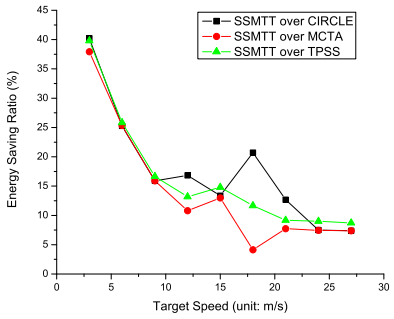


Figure 5.8: ESR vs. target speed

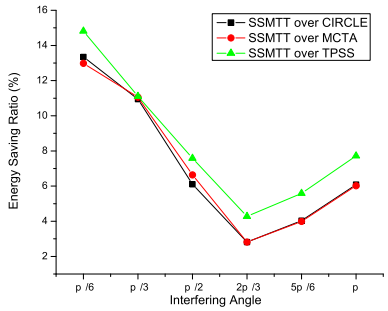


Figure 5.9: ESR vs. interfering angle

5.6 Conclusion

In this chapter, we extend TPSS to multiple target tracking by presenting SSMTT algorithm. SSMTT further improves energy efficiency of TPSS for multiple target tracking, by leveraging the overlapping broadcasts for multiple targets to reduce the energy consumed on proactive wake-up alarm transmission. Our experimental evaluations show that compared with TPSS, SSMTT algorithm reduces the energy consumption for alarm transmission by 10% $\sim$ 15%. Thus, it makes TPSS more energy-efficient on multiple target tracking.

Chapter 6

VPF: An Improved Particle Filter for Reduced Computation

6.1 Introduction

When particle filters are applied to WSNs, especially as a distributed implementation, the computation workload of PFs will be distributed onto individual sensor nodes. Since the computing capability of sensor nodes is very limited, the computational complexity of PFs is not trivial.

For the computational complexity of PFs, the number of particles N_s is a very important factor. It is shown in [21, 23] that the complexity of particle filters is linear to the number of samples. This is especially significant in duty-cycled sensor networks, where each communication will experience a sleep delay (usually the length of a duty cycling period) when using a MAC protocol like B-MAC [89]. Thus propagating a particle message for each hop will experience one sleep delay. As the fusion center (sink nodes or cluster headers) of PF has to receive particle data at one message once a time, the estimation cannot be made within less than N_s sleep delays in the best case. Therefore, it will be very helpful to reduce N_s so that the problem scale is controlled.

However, simply reducing N_s may have negative influence on the performance of PFs. It may increase the estimation error. Besides, "sparse" particles caused by N_s reduction can make the sample impoverishment problem more severe than usual. Sample impoverishment is a problem introduced by resampling, when particles with high weights are selected many times and the diversity is lost. When particles are less, the selected times of those high-weight particles will increase further. Thus, we need to consider particle number reduction and sample impoverishment problem in an integrated manner.

Typical existing solutions for sample impoverishment include resample-move [24], regular-

ization [22], and bridging densities [22]. Although these algorithms provide an optimal or near-optimal resampling results, they introduce significant computation overhead. For example, RPF [22] requires to calculate the covariance matrix S_k and a matrix A_k where $A_k A_k^T = S_k$. With the constrained resources of WSNs, such kind of matrix operations are not trivial. With bridging densities, the resampling step is divided into multiple iterations—each for a small transition from the prior distribution to the posterior distribution. Then, the computation workload will be increased significantly, which is not feasible for limited resources of WSNs.

In this chapter, we present a Vector space-based Particle Filter, to improve the efficiency of sampling importance resampling filter (SIR) [21] based on vector space theory. Based on the distance between vectors, we first reduce the number of particles so as to alleviate the computational overhead, while satisfying the requirement of WSNs on the tracking accuracy. Then, we develop an efficient solution for the sample impoverishment problem by distributing the repeated particles with high weights to multiple individual ones with the same total weight. Finally, we apply the results of VPF to data fusion and sleep scheduling—a characteristic of duty-cycled WSNs and a prerequisite that guarantees enough active nodes to obtain the observations.

Although a few research efforts on PFs consider non-vector space [101], we focus on vector space-based dynamic systems, like most of PF works that were applied to WSNs. In vector space theory, the distance between two vectors in an Euclidean n-dimensional space is tractable. In addition, like most existing works on Bayesian estimation and particle filters, we also study the tracking problem in a possibly continuous dynamic system using the discrete-time approach [21].

Our experimental simulation studies show that compared to RPF, VPF introduces an improvement of 68% on the execution time per iteration, at the expense of about 34% increase on the tracking error.

6.2 Particle Number Reduction

As particle filters are numerical methods, we here develop a heuristic algorithm for reducing the number of particles based on the partition of the state space. Intuitively, if two particles are very close to each other in the state space, they may be combined to a single particle with a weight that is equal to the sum of the original two particles' weights. Moreover, such a combination is supposed to control the estimation accuracy within a given tolerable error. In this manner, the number of particles will be reduced without influencing the estimation performance much. Therefore, the basic idea of reducing the number of particles is to partition the state space into cells, sample one particle for each cell as the representative, and combine the weights of particles in the same cell into a single weight. For a specific cell size (determined by the partition density), the number of particles will then be decided by

the number of cells.

In fact, not all the cells in a state space need to be sampled, because the area that needs to sample at time k can be probabilistically constrained by the proposal distribution (chosen as the prior distribution in SIR) [102]. Based on vector space theory, this reduction method is supposed to be efficient.

Next we first discuss the cell size when given a specific tolerance for the estimation error, then introduce the partition method and calculate the reduced number of particles based on this partition method. Finally we describe a corresponding sampling algorithm.

6.2.1 Cell Size

The number of particles, i.e. N_s, directly influences the tracking accuracy, which is usually measured with Root Mean Squared Error (RMSE) [21, 26]. As particle filters are numerical methods instead of analytical methods, there is no definite quantitative connection between N_s and RMSE. In [103], instead, the author presented the tracking performance at various numbers of particles through the experimental simulation. It was shown that a larger N_s will result in a better approximation to the optimal solution than a smaller one.

Most of the existing work on PF aimed at improving the tracking accuracy by minimizing RMSE. However, this is merely the objective from the perspective of mathematics. For actual applications, a bounded tolerance for the tracking error is often enough. For example, the users may be interested in which sensor nodes they can query for detailed information [65] or collect data stored in local memories on sensor nodes using a mobile agent [104], rather than estimate the true trajectory of the target. For such applications, the accuracy on estimating a target's trajectory is less important, therefore the tolerance for the tracking error can be relaxed.

Lemma 1 *In a n_x-dimensional WSN with the average sensor density ρ and the sensing radius r, the achievable tolerance for the tracking error is in the order of $O(\frac{1}{\rho r^{n_x-1}})$.*

Proof: According to the results of [64], the achievable localization error in a n_x-dimensional WSN of binary proximity sensors with the average sensor density ρ and the sensing radius r is $\Omega(\frac{1}{\rho r^{n_x-1}})$. Since the information that binary sensors can provide is the least, the tracking error in regular WSNs must outperform that in binary WSNs. Therefore, the achievable tolerance for the tracking error is in the order of $O(\frac{1}{\rho r^{n_x-1}})$. **Done.**

Next, we map Lemma 1 to the vector space of state $\mathbf{x}_k$ in order to determine the cell size.

Theorem 1 *When given a partition scheme with a cell size d in a state space of n_x dimensions ($V : \mathcal{R}^{n_x}$), the tolerance for the estimation error is in the order of $O(d)$.*

Proof: We setup a mapping from sensor nodes to particles in an Euclidean n_x-dimensional space ($\mathcal{R}^{n_x}$). Here for example, $n_x = 4$ for the state transition model in Chapter 3. Then the mapping will be:

$$\begin{aligned}\mathbf{g} \ &: \mathcal{R}^{n_x} \rightarrow \mathcal{R}^{n_x} \\ &\text{sensor nodes} \rightarrow \text{particles} \\ &\rho \rightarrow \rho_c \propto \tfrac{1}{d^{n_x}} \\ &r \rightarrow r_c \propto d \\ &\text{tolerance for the tracking error} \rightarrow \\ &\qquad \text{tolerance for the estimation error}\end{aligned}$$

where ρ_c is the partition density, and r_c is half of the diameter (i.e., the longest projection) of a cell.

Obviously this mapping is a perfect mapping, as for any particle in the particle space, there must be one and only one sensor node in the sensor node space. Then the result of Lemma 1 could be directly applied to the vector space of particles. Therefore we may draw the conclusion that, given a partition scheme with a cell size d in a state space of n_x dimensions, the tolerance for the estimation error is in the order of

$$O(\frac{1}{\rho_c r_c^{n_x-1}}) \propto O(\frac{1}{\frac{1}{d^{n_x}} \cdot d^{n_x-1}}) = O(d)$$

Done.

Based on Theorem 1, the cell size can be determined given a specific tolerance for the estimation error. For example, if a bound for the estimation error is given as b_e, then configuring the cell size as $d = b_e$ will constrain the estimation error within the bound.

6.2.2 Partition Method

We leverage the scheme of selecting sampling area in [102]: the sample area is centered at $m_{k|k-1} = E(\mathbf{x}_k|\mathbf{x}_{k-1})$ and with a "radius" of $3\sigma_{k-1}$, where $m_{k|k-1}$ is the mean of the prior distribution and σ_{k-1} is the standard deviation of the posterior pdf at time $k-1$. Then each point $m_{k|k-1} + d \cdot (c_1, c_2, \ldots, c_{n_x})$ in the sample area will be a vertex of 2^{n_x} cells, where $c_i \in \mathbb{Z}$ ($i \in [1, n_x]$). In addition, the number of these points is exactly the reduced number of particles N_s.

In fact, we do not have to partition the state space into cells explicitly. Next when we discuss the sampling algorithm, we may observe that a sample may be located in a specific cell easily.

Algorithm 5 Sampling Algorithm

```
Calculate m_{k|k-1} and 3σ_{k-1} for the sample area;
Initialize n = 0, n_try = 0;
while (n < N_s and n_try < N_trymax) do
    Draw a sample x_k^i from p(x_k|x_{k-1}^i);
    Decide the cell C that x_k^i locates in;
    if (x_k^i is outside of the sample area) then
        continue to the next loop;
    end if
    n_try = n_try + 1;
    Assign a weight {x_k^i, w_k^i};
    if (C is marked) then
        Combine the existing particle in cell C and {x_k^i, w_k^i} using Equation 6.1;
    else
        Add {x_k^i, w_k^i} to cell C and mark it;
        n = n + 1;
    end if
end while
```

6.2.3 Sampling Algorithm

Based on the partition method, a sampled particle $\mathbf{x}_k^i = (x_{k1}^i, x_{k2}^i, \ldots, x_{kn_x}^i)$ may be located in the cell with the least vertex $\mathbf{C} = (c_1, c_2, \ldots, c_{n_x})$, where $c_j = x_{kj}^i - x_{kj}^i \bmod d$ $(j \in [1, n_x])$ and mod is the modulo operation. Then two particles in the same cell, $\{\mathbf{x}_k^i, w_k^i\}$ and $\{\mathbf{x}_k^j, w_k^j\}$, can be combined as a single new particle $\{\mathbf{x}_k^l, w_k^l\}$:

$$\begin{cases} \mathbf{x}_k^l = \frac{w_k^i}{w_k^i + w_k^j}\mathbf{x}_k^i + \frac{w_k^j}{w_k^i + w_k^j}\mathbf{x}_k^j \\ w_k^l = w_k^i + w_k^j \end{cases} \tag{6.1}$$

In this way, two particles are combined with their weighted probability density also combined, i.e., $w_k^i \mathbf{x}_k^i + w_k^j \mathbf{x}_k^j = w_k^l \mathbf{x}_k^l$.

Finally the sampling algorithm is described by Algorithm 5. Note that we added a maximum number N_{trymax} for the sampling efforts so that it can be ended in finite steps. In the real deployment, N_{trymax} can be configured as a multiple of N_s, and its coefficient may be determined through experiments.

6.3 High-weight Particle Expansion

The approach for reducing the number of particles is to combine particles in the same cell into one, so that the number of particles could be reduced according to the estimation

accuracy requirement. On the contrary, we introduce a particle expansion scheme in this section, which decomposes a single particle into multiple new ones, to solve the sample impoverishment problem.

After resampling using a typical algorithm used by SIR, e.g. systematic resampling [103], particles with high weights may be selected many times, and those with low weights may be removed from the particle set. Then some cells may be unmarked (or empty), and the others may have their particle weights increased. When the process noise ($\mathbf{v}_k$) is small, this problem will become very severe. Even distributing particles into cells cannot solve this problem completely, because a particle with a high weight may be resampled multiple times and the new particles are likely to be transited to a very small area in the state space during the prediction step, thus be combined into one single particle again. Therefore, we present a particle expansion algorithm as a modification step after the resampling step.

First we modify the widely used systematic resampling algorithm [103] by folding all the new particles from the same parent to a single one with the total weight of these new particles. As the modification is not complicated, we do not describe it in detail.

Then we define the neighbor cells, to which a particle in a cell C may be expanded to, as the number of neighbor cells of C which share a $n_x - 1$ plane with C. These neighbors will be the closest to cell C. It can be easily shown that a cell's number of neighbor cells in a n_x-dimensional state space is $2n_x$.

Based on the repeated times and the number of neighbor cells, we expand a high-weight particle in an iterative way, until the estimation of the effective sample size $\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s}(w_k^i)^2}$ is lower than a specific threshold. In fact, a simpler measure of degeneracy is to use $\frac{(w_k^i)_{max}}{(w_k^i)_{min}}$. A higher ratio means a severer degeneracy. Then in each iteration, we distribute the particle with the maximum w_k^i in the following steps:

1) Keep $\frac{1}{2n_x+1}$ of the particle weight in the original cell;

2) Distribute the other to $2n_x$ neighbor cells, each with a particle in the center of the neighbor cell and a weight of $\frac{1}{2n_x+1}$ of the original; and

3) Combine the distributed part with the local particle in a neighbor cell if it is marked, using Equation 6.1.

Figure 6.1 shows the expansion process with a two-dimensional state space as the example. The particle with a high weight (the slashed circle in the figure) is expanded to neighbor cells. If a neighbor cell has already been occupied by another particle, the weighted combination operation discussed in Section 6.2 will be executed.

Finally we show the particle expansion algorithm is described by Algorithm 6.

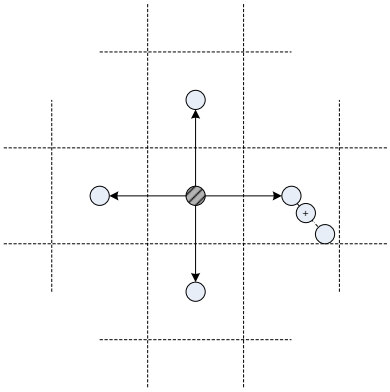


Figure 6.1: High-weight particle expansion

Algorithm 6 Particle Expansion Algorithm

while (1) **do**
  Calculate $\widehat{N_{eff}}$;
  if ($\widehat{N_{eff}} \geq N_T$) **then**
    break;
  end if
  Choose the particle $\{\mathbf{x}_k^i, w_k^i\}$ where w_k^i is the highest;
  $\{\mathbf{x}_k^i, w_k^i\} = \{\mathbf{x}_k^i, \frac{w_k^i}{2n_x+1}\}$;
  for all (neighbor cells C_j) **do**
    Distribute a new particle $\{\overline{\mathbf{x}_k^j}, \frac{w_k^i}{2n_x+1}\}$ in the center of C_j;
    if (C_j is marked) **then**
      Combine the existing particle in cell C_j and $\overline{\mathbf{x}_k^j}$ using Equation 6.1;
    else
      Add $\overline{\mathbf{x}_k^j}$ to cell C_j and mark it;
    end if
  end for
end while

6.4 VPF Algorithm

We first describe the VPF algorithm in Algorithm 7, then discuss its complexity and constraint.

In Algorithm 5, an extra vector subtraction (for judging if it is in the sample area) and a possible vector addition (for combining particles) are appended to the sampling step in SIR. In addition, the sampling operation may be executed more than N_s times until N_s particles are sampled or N_{trymax} is reached, whichever comes first. However, the complexity is still $O(N_s)$. For Algorithm 6, the complexity depends on the configuration of N_T, i.e., the thresh-

Algorithm 7 VPF Algorithm

1: Sample N_s particles using Algorithm 5;
2: Summarize the weights: $t = \sum_{i=1}^{N_s} w_k^i$;
3: Normalize the weights: $w_k^i = t^{-1} w_k^i$;
4: **if** $(\widehat{N_{eff}} < N_T)$ **then**
5: Resample using the modified systematic resampling algorithm;
6: **end if**
7: Modify the new samples using Algorithm 6;

old for the estimation of the effective sample size. But it can be estimated approximately as $O(\log \frac{(w_k^i)_{max}}{(w_k^i)_{min}})$. In summary, the complexity of VPF is potentially lower than that of RPF. We will discuss it in detail in Section 6.6.

Since VPF is based on vector space theory, its complexity will increase as the dimension of states increases. Therefore, it is more suitable for low-dimensional dynamic systems than high-dimensional ones.

6.5 Potential Applications

The primary purpose of particle filters is to estimate the trajectory of a target across the surveillance field. However, an improved particle filter should support more functions specific to WSNs, e.g., data fusion and sleep scheduling, rather than estimating the trajectory only. Next we illustrate how VPF facilitates data fusion and sleep scheduling.

6.5.1 Data Fusion

As Algorithm 5 returns particles as well as the actual number of particles, data fusion is possible along the propagation path of particles. Then the only part of PFs that is necessary for centralized computing—the summation and weighted average [27]—can be completed on the way.

Suppose that particle data is handled by sink nodes, and for all the sensor nodes that observe the target there exists a lowest common ancestor (LCA) in the routing tree rooted at a sink node. Starting from an empty set ϕ, the LCA node may maintain a particle set $\{\mathbf{x}_k^i\}$ and a temporary average weight $\{\overline{w_k}, n_s\}$, where n_s is the number of received particles to date. Then each time the LCA node receives a new particle data with a weight w_k^i, it can accumulate the particle onto the existing data set and update the average weight to $\{\frac{\overline{w_k} n_s + w_k^i}{n_s+1}, n_s + 1\}$.

In this manner, the computation could be offloaded from sink nodes. Moreover, the commu-

nication cost and the delay may be reduced significantly.

6.5.2 Sleep Scheduling

It is well known that target prediction, which sleep scheduling is usually dependent on [42], is not suitable to be completed using particle filters. This is because the estimation at time k, i.e. the basis for predicting the state at time $k+1$, cannot be obtained until the resampling step is completed [28]. Strictly speaking, PF was indirectly used for prediction in [28]—the particle filter was only used to estimate, and the prediction was made based on this estimation. As previously discussed, collecting all the particle data means a high communication cost thereby a long delay, while this delay will make the prediction result less valuable.

Unlike other PFs, VPF is still helpful for sleep scheduling. Based on the partition scheme, different areas in the state space may be given different weights. Therefore, sensor nodes that locate in the area corresponding to a specific cell may be proactively awakened with a probability proportional to the weight of the cell. That means sensor nodes that are likely to observe the target hold a high probability of waking up. This will increase the observation quality as well as save the energy as much as possible. In addition, this prediction does not require to complete the whole iteration of VPF. Instead, the prediction step for time $k+1$ can be inserted after the sampling step at time k. In this manner, the prediction result could be propagated early and neighbor nodes may get ready for observing the approaching target.

6.6 Evaluation

We evaluated VPF in Matlab and compared it with regularized particle filter. This section reports our evaluation results measured in terms of RMSE and the algorithm efficiency.

6.6.1 Simulation Environment

We study the bearings-only tracking problem [24] in a 600 $m \times 600$ m plane for the simulation. The dynamic system model is:

$$\begin{aligned} \mathbf{x}_k &= \Phi \mathbf{x}_{k-1} + \Gamma \mathbf{v}_{k-1} \\ z_k &= \arctan \tfrac{y_k}{x_k} + n_k \end{aligned} \tag{6.2}$$

Table 6.1: Evaluation result summary

	RPF	VPF
RMSE	93.8562	126.037
Average N_s	25	20.02
Average execution time per iteration	0.01156 s	0.00374 s

where $\mathbf{x}_k = (\mathbf{s}_k^{(0)}, \mathbf{s}_k^{(1)})^T = (x_k, y_k, x'_k, y'_k)^T$, z_k is the observed bearing,

$$\Phi = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Gamma = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

In addition, $\mathbf{v}_{k-1} = (v_x, v_y)_{k-1}^T$ and n_k are zero mean Gaussian white noises, the variances of which are respectively $\sigma_v^2 = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$ and σ_n^2.

The detailed parameter configurations are as follows. The initial state vector is $\mathbf{x}_1 = [20, 30, 10, 8]^T$. The initial particle set is drawn from a Gaussian distribution with a known mean $\overline{\mathbf{x}_1} = \mathbf{x}_1 + \Delta\mathbf{x}$ and a covariance matrix $P_0 = diag([4, 6, 0.1, 0.1])$, where $\Delta\mathbf{x} = [10, 50, 0, 0]^T$. In Φ and Γ, Δt is set as 1 s. The standard deviations of noises are $\sigma_x = \sigma_y = \sigma_n = 0.05$. The simulation includes 50 steps, and the resampling threshold is $N_T = N_s/5$.

For VPF, we configured the cell size as $d = 15\ m$, and started the algorithm with 25 particles that were distributed in 5×5 cells. To make a fair comparison, we configured the number of particles of RPF also as $N_s = 25$. In addition, we set the volume of the unit hypersphere $c_{n_x} = 2$ and the Epanechnikov kernel for RPF.

6.6.2 Experimental Results

First, we summarize the evaluation results in Table 6.1. We observe that although VPF increases RMSE by about 34% more than RPF, it significantly saves 68% of the execution time per iteration compared with RPF. Except for the efficiency improvement introduced by the reduced problem size, another reason for VPF's significant enhancement on the computation time is that based on vectors, VPF involves with very less computational workload. As discussed in Sections 6.2 and 8.2, VPF requires simple arithmetic operations. On the contrary, RPF requires complicated matrix operations [22]. Therefore, VPF is much more efficient than RPF on the computation.

Next, we report the detailed evaluation results. Figure 6.2 and Figure 6.3 show the estimation results of the position $\mathbf{s}_k^{(0)}$ and the velocity $\mathbf{s}_k^{(1)}$ respectively. The estimation of RPF is

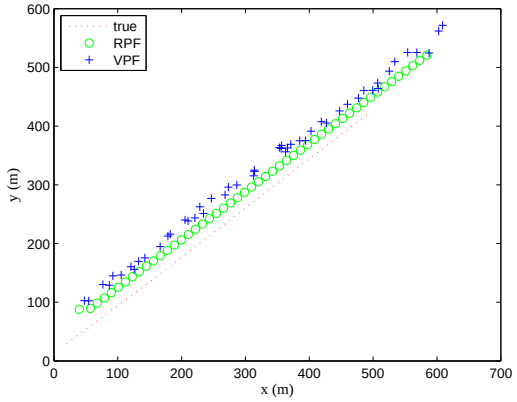


Figure 6.2: Estimation on the position

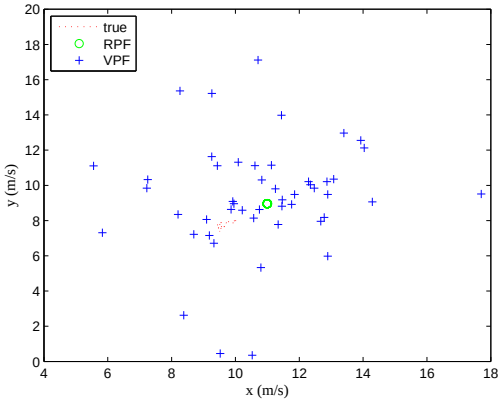


Figure 6.3: Estimation on the velocity

relatively stable, but VPF shows a more variable estimation. This is caused mainly by VPF's flexible sample size, i.e., the number of particles of VPF may dynamically change across the iterations. Such an adaptive change on N_s results in the efficiency improvement.

Figure 6.4 just shows this change of N_s along the time. Each time a resampling step is executed with a severe sample impoverishment, N_s may increase significantly to compensate for this degeneracy. This is the result of high-weight particle expansion as discussed in Section 8.2. When there is no severe sample impoverishment, on the contrary, particles may be propagated outside of the sampling area during the prediction step. Therefore, N_s may decrease gradually along the time.

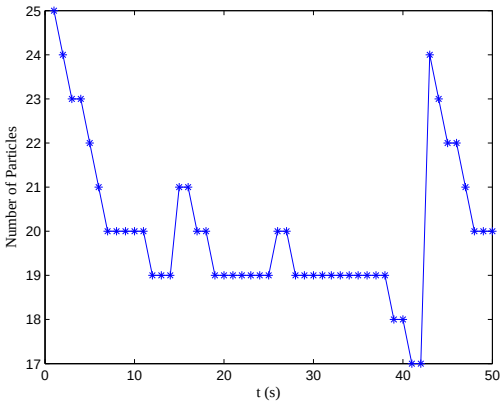


Figure 6.4: Adaptive number of particles

6.7 Conclusion

In this chapter, we designed VPF based on vector space theory and described the evaluation results. As particle filtering is a numerical method based on statistics, the quantitative relation between the number of particles and the estimation error is still not clear. Instead, the vector space partition method helps to bound the estimation error. However, the current bound, which is obtained from binary sensor networks, is still a loose one and needs to be further tightened.

Chapter 7

CDPF: A Distributed Particle Filter for Reduced Communication

7.1 Introduction

Distributed particle filters (DPFs) [23] have been studied to offload the computation from as well as reduce the convergecast communication to the computational center, which is used by centralized particle filters (CPFs) [105]. However, few existing efforts on DPFs were implemented in a completely distributed manner [27]. This is because that the aggregation of weights, which is critical for weight normalization and resampling, is inevitable.

For efficiently transmitting the particle data to the computational center, there exist several DPF efforts that focus on reducing the communication cost by compressing messages, such as Gaussian mixture approximation [25], non-parametric particle compression based on support vector machine [29], and adaptive encoding [23, 69]. But like CPFs [21], these DPF efforts all share a common problem: the number of messages remain unchanged. Usually, reducing the number of messages is more efficient for energy conservation than compressing the data amount contained in each message, especially in duty-cycled WSNs where nodes need to wake up from the sleep state for transmitting data [15], irrespective of how much data they need to transmit. Therefore, only when PFs are implemented in a completely distributed manner, the communication cost could be minimized and the energy efficiency of WSNs can be enhanced significantly (this motivation will be detailed in Section 7.2 through analysis).

For this purpose, we need to develop a method to aggregate the particle weights without any extra communication other than necessary particle propagation and/or those needed for sharing of measurements. This may be achieved by maintaining particles on different nodes and propagating it along the target trajectory. Based on the overhearing effect [106], nodes may receive all the propagated particles, thereby obtaining the aggregation as a side product of particle propagation.

Another important feature of WSNs that we may use for this purpose is that the local status in a WSN is relatively stable in the short term. The local status may include, but is not limited to, node positions, the topology, and the detection capability of neighbor nodes. Based on this stable local status, it is possible for a node to estimate the working status of its neighbor nodes thus the contributions they may make toward target estimation. We fully leverage this feature to approximate the contributions of neighbor nodes and further reduce the communication cost.

In this chapter, we design a completely distributed particle filter for target tracking in WSNs, called CDPF, so as to minimize the communication cost. First, we develop a mechanism for maintaining particles on sensor nodes and propagating them along the target trajectory. Then, we design CDPF by adjusting the order of PFs' four steps and leveraging the data aggregation during particle propagation. Finally, we introduce a neighborhood estimation method so that each node may replace the likelihood functions with approximate, estimated contributions of its neighbors, and eliminate the communication cost of measurement broadcasting. Like most existing work on Bayesian estimation and particle filters, we study the tracking problem in a possibly continuous dynamic system using the discrete-time approach [21].

We compare CDPF to CPF [21] and SDPF [27], the latter of which is a state-of-the-art effort that considers the distributed implementation of PFs from the perspective of network architecture and protocol of WSNs. Our experimental simulation studies show that, compared to SDPF, CDPF reduces the communication cost by 90%, with a similar estimation error.

7.2 Motivations

In this section, we discuss the potential improvements on reducing the communication cost if we implement a completely distributed particle filter.

It is shown in [23] that the communication workload of centralized particle filters at each iteration is $\sum_{i=1}^{N_m} D_m H_i$, where N_m is the number of sensor nodes with measurements, D_m is the data amount of a measurement message, and H_i is the number of hops that D_m data needs to be propagated to the computational center. In a sensor network with N nodes, the average distance (hop counts) to the sink node is approximately $\sqrt{N}$ [107]. Then, we have the communication complexity of CPFs [21] as $O(D_m \sum_{i=1}^{N_m} H_i) = O(D_m N_m \sqrt{N})$.

Coates elaborated the achievable compression on the disseminated raw data of particles, i.e., compressing D_m either by training parametric models or with adaptive encoding in [23]. Like the communication cost of CPFs, the achievable communication cost of DPF is $O(P N_m \sqrt{N})$, where P is the data amount of each compressed measurement message. This result only provides a possibility of reducing the total data amount of communication if $P \ll D_m$. In fact, the number of communication messages is equal to or even higher than that of CPFs (due to the backward parameter exchange). Thus the efficiency of DPF completely depends

Table 7.1: Analyzed communication costs of various PFs

Particle filter methods	Communication costs
CPF	$D_m N_m \sqrt{N}$
DPF	$P N_m \sqrt{N}$
SDPF	$N_s(D_p + D_m + 2D_w)$
CDPF	$N_s(D_p + D_m + D_w)$

on the compression efficiency on the raw data.

In [27], Coates and Ing developed a semi-distributed particle filter, named SDPF, in which particles are maintained on different sensor nodes and weight aggregation is completed on a global transceiver. The communication of SDPF consists of three parts: particle propagation, measurement sharing and weight aggregation/dissemination. First, the particles maintained on different sensor nodes are propagated in the predicted direction of the target at each iteration. As each sensor node that maintains particles needs to broadcast a message containing both particles and their weights to its neighbors within one hop, the communication cost is $\sum_{i=1}^{N_n} N_i(D_p + D_w) = (D_p + D_w)\sum_{i=1}^{N_n} N_i = N_s(D_p + D_w)$. Here N_n is the number of sensor nodes that are maintaining a subset of particles, N_i ($N_i \geq 1$) is the number of particles maintained on sensor node i, thus $\sum_{i=1}^{N_n} N_i = N_s$. Moreover, we denote the data amount for each particle D_p (the subscript p represents "particle"), and the data amount of a particle's weight D_w (the subscript w represents "weight"). Secondly, the measurements of neighbor nodes are shared locally among N_n nodes. Then the communication cost will be $\sum_{i=1}^{N_n} D_m = N_n D_m \leq N_s D_m$. Thirdly, at each iteration, each sensor node that maintains particles needs to transmit the weights of particles on it to a global transceiver, which is assumed to be one hop away from every node in the network. After a three-way query-response handshaking, the transceiver sends the calculated total weight back to active nodes. The communication cost during the whole aggregation process is $\sum_{i=1}^{N_n} N_i D_w + 2 = N_s D_w + 2$, where 2 comes from the two broadcast messages from the transceiver. Therefore, the total communication cost of SDPF is $N_s(D_p + D_w + D_m + D_w) + 2 \approx N_s(D_p + D_m + 2D_w)$.

Unlike DPFs and SDPF, a completely distributed particle filter does not have to collect the particle weights for the aggregation. Based on the calculation for SDPF, the communication cost of such a PF like CDPF that we present in this dissertation will approximately be $N_s(D_p + D_m + D_w)$.

We compare the analyzed communication costs of four PFs in Table 7.1. Obviously, SDPF and CDPF may significantly reduce the communication cost of CPFs and DPFs by constraining the communication within one hop, especially when the network scale is large. Except for this, CDPF eliminates the communication for weight aggregation completely, thereby achieves the minimal communication cost.

7.3 Particle Maintenance and Propagation

For the sake of clarify, we first explicitly interpret the term "distributed" in DPFs. In the existing work, it was defined in several different ways. For example in [23], "distributed" means that the aggregation of particles and their weights is completed in a distributed manner on different sensor nodes. Other operations, such as the calculation of factorized likelihood functions, the training of parametric models and measurement quantization, all serve for this purpose. Unlike [23], the term "distributed" in [27] was interpreted as meaning that disjoint subsets of particles are maintained on different sensor nodes.

We define "distributed" in CDPF following the interpretation of [27], i.e., particles on nodes. Its advantages include: 1) the computational workload may be distributed onto different sensor nodes; 2) particles are easy to manage and propagate; and 3) particles can be combined or divided based on node positions.

Next, we introduce the maintenance and propagation mechanism of particles.

7.3.1 Particle Maintenance

Like [27], we constrain particles to locate on sensor nodes only, thus each particle will automatically have the position of its "host" node that maintains it. This may increase the estimation error of PFs. But if nodes are deployed densely enough or an error bounded by the sensing radius is tolerable, the error increment will be less important.

We do not distinguish the different particles on the same node. Hence multiple particles on a single node may be combined to one particle, with the total weights of original particles as its weight. On the contrary, a single particle may also be divided into multiple ones during the propagation, which will be detailed in Section 7.3.2. Though the number of particles N_s may vary when being combined and divided, this variable N_s is controllable. This is because that N_s corresponds to the number of sensor nodes that maintain particles and participate into filtering, while these nodes are always around the target trajectory thus will be bounded when given a certain deployment density.

7.3.2 Particle Propagation

At the initialization step, each node that first detects an intruding target is given a particle with a certain weight. This particle weight may be configured as a constant, or adaptively determined according to the received signal strength. In the following tracking process, these particles will be propagated along with the moving target.

Figure 7.1 shows the propagation of particles at iteration k. In the figure, small circles represent sensor nodes, squares mean the target positions, and dotted lines and circles signify

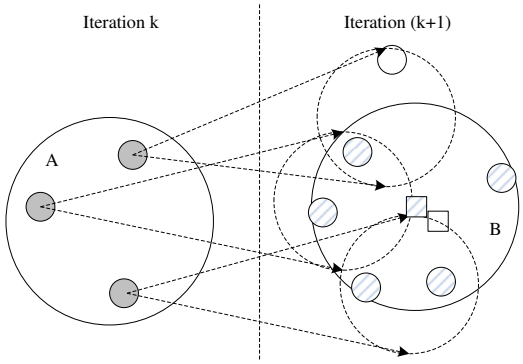


Figure 7.1: Particle propagation

the prediction and the direction of particle propagation. For the discussion convenience, we call the dotted circles "predicted areas". The other symbols in the figure will be introduced in Sections 7.4 and 7.5, where they are discussed.

Nodes always propagate particles towards the predicted target position, so that the particles in the current iteration can be reused in the next iteration, with their weights updated. A dynamic clustering mechanism as in [42] may be used for this purpose: a node broadcasts the particles on it to all of its neighbors, but only those that are highly likely to detect the target record the particles (i.e., nodes in predicted areas). We leverage the linear probability model in [42] to decide which neighbors should record the particles. If there are more than one node in the predicted area, a single particle will be divided into multiple ones, so will its weight. The weight is divided based on the following rule: 1) the total weight of divided particles is equal to the original particle's weight; and 2) the ratio of any pair of divided particles' weights is equal to the ratio of their host nodes' probabilities in the linear probability model.

Particle propagation from multiple source nodes may also overlap on neighbor nodes, e.g., nodes in the intersection of two predicted areas in Figure 7.1. In this case, particles from different source nodes will be combined into one on the receiving node.

It is possible that some nodes receive and record particles, but they are not able to detect the target at the next iteration, e.g., the blank node in Figure 7.1. Then, the weight update of particles on it will depend on the likelihood function. If the likelihood function shows zero or almost zero density, this node may drop the particle on it and stop broadcasting.

It is also possible that a node that does not receive any propagated particles detects the target, e.g., the node outside of any predicted areas. Then a new particle will be created as in the initialization step.

7.3.3 Node Scheduling

Sensor nodes need to be scheduled for maintaining and propagating particles. In a duty-cycled WSN [15], nodes around the predicted target position may be in the sleep state when a target is approaching. Then, it needs to be proactively awakened so as to receive the propagated particles. We leverage TPSS sleep scheduling algorithm presented in [42]: nodes around the predicted target position are awakened to prepare for the approaching target, and the energy consumption could be reduced simultaneously.

7.4 CDPF Design

Based on the mechanism of particle maintenance and propagation, we design the CDPF algorithm in this section. First, we partition the updating step and adjust the order of filtering steps in CPFs. Then the CDPF algorithm will be specified.

7.4.1 Algorithm Design

Among the four steps of generic PFs, the prediction step may depend only on individual particles (e.g., by choosing the prior distribution as the importance density). Thus, every sensor node that maintains a subset of particles may complete the prediction step independently. However, all the other three steps (i.e., updating, resampling and estimation) require collection of all the measurements and particle weights. To design CDPF, we need to develop a method to achieve the same objective without any extra communications.

Particle propagation designed in Section 7.3.2 provides us a feasible approach. Based on the overhearing effect, it is possible that every node in any of the predicted areas hears the particles from all the broadcasting nodes. Since we assume that the sensing radius of nodes is no greater than half of the communication radius, this goal is achievable as long as the propagation does not reach too far (i.e., the time interval of the dynamic system is not very long). Then every node that receives particles will receive all the particles, so that every node may obtain the total weight.

However, one problem of this approach is that the obtained total weight is for the previous iteration instead of the current one. Therefore, we have to adjust the order of four steps to produce a working scheme. Figure 7.2 shows the steps of both CPF and CDPF. We partition the "updating" step into three sub-steps: 1) the "likelihood" step shares the measurements locally and calculates the likelihood functions; 2) the "assign weight" step assigns weights to particles; and 3) the "normalization" step calculates the total weight and normalizes the assigned weight. The dotted curves in Figure 7.2(a) signifies the reorder direction: we move normalization, resampling, and estimation steps forwards and insert them after the prediction step. The reordered steps for CDPF are shown in Figure 7.2(b).

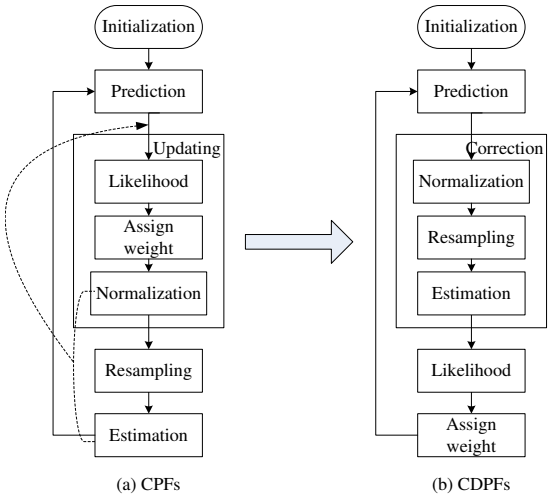


Figure 7.2: Steps of CPF and CDPF

After reordering the steps, we form normalization, resampling and estimation into a new step, named "correction". The correction step normalizes the updated weights, resamples and calculates the estimated target position for the previous iteration. Obviously, the correction step depends on the total weight that is aggregated by overhearing during particle propagation. This is the reason that we insert it after the prediction step.

Then the working process of CDPF will be:

1) *Prediction*. Predict the target motion and propagate particles towards that direction.

2) *Correction*. Normalize the propagated weights based on the total weight, resample, estimate the target position for the previous iteration, and possibly report it to sink nodes.

3) *Likelihood*. Share the measurements at the current iteration locally and calculate the likelihood functions.

4) *Assign weight*. Assign or update weights to particles based on the likelihood functions.

To update the weights, the measurement of each node should be shared locally with other nodes. Thus in the likelihood step, each node will receive the broadcast measurements from all of the neighbor nodes that are maintaining particles. In fact, this is also an approach to calculate the total weight without extra communication cost. However, we do not take this approach, because the broadcasting communication in this step can be eliminated, so that the communication cost can be reduced further. We will discuss the details in Section 7.5.

In Figure 7.1, we drew two squares, meaning two possible positions of the target. Based on the correction step, we now explain their difference. We use the blank square to represent the real position of the target, which is why the blank node cannot detect it. After the correction step, the estimated target position for the previous iteration will be obtained. Then we use

the slashed square to represent the "predicted" target position based on this estimation. In fact, this cannot be called "prediction" any longer, because the current iteration has started. We use this term simply to show our calculation method. Then this predicted position will be an approximation to the real position, and we will use it to estimate the neighbor nodes' contributions and finally eliminate the likelihood function calculation in Section 7.5.

7.4.2 Algorithm Details

Based on our previous design, we now detail an iteration of the CDPF algorithm in Algorithm 8.

Algorithm 8 CDPF algorithm at iteration $k+1$

1: Draw N_s samples for iteration $k+1$ from an importance density $q(\mathbf{x}_{k+1}|\mathbf{x}_k^i, \mathbf{z}_{k+1})$, i.e., propagate particles from iteration k to $k+1$;
2: Calculate the total weight by overhearing;
3: Normalize the received weights;
4: Resampling;
5: Make the estimation for iteration k;
6: Broadcast/receive measurements;
7: Calculate the likelihood function;
8: Assign/update a weight to the particle;

7.5 Improving CDPF: Neighborhood Estimation

As discussed in Section 8.1, the local status in a WSN (including node positions, the topology and the detection capability of neighbor nodes) is relatively stable in the short term. Based on this feature, a sensor node may estimate the working status of its neighbor nodes thereby the contributions they may make to the target estimation. In this section, we develop an approximate estimation method to improve CDPF by further reducing the communication cost. We first discuss a prerequisite for the neighborhood estimation, which answers how a node may obtain information about its neighbors. Then, we introduce the estimation method and present the improved CDPF algorithm. Finally, we discuss the potential overhead that this estimation may introduce, and potential factors that may impact the estimation result.

7.5.1 Prerequisite

A prerequisite of neighborhood estimation is that local knowledge can be easily shared among the neighbor nodes. Basically whatever a node knows, its one-hop neighbors may easily

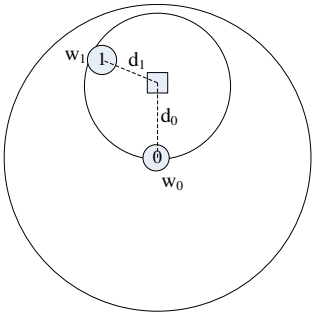


Figure 7.3: Neighborhood estimation

know it via local direct communication. In many existing efforts, short message exchange was commonly used for sharing local knowledge among neighbors, e.g., for updating routing information [31] or for maintaining synchronization [108]. Therefore we may reasonably assume that every sensor node knows all the detailed information about its one-hop neighbors, especially their positions.

7.5.2 Estimation Method

Figure 7.3 shows a local topology including two sensor nodes (shown as small circles) and the predicted position of a target (the square). The large circle with a communication radius represents the one-hop neighbor area of node 0, and the middle circle with a sensing radius signifies the area in which nodes may detect the target. We define this middle circle as an estimation area:

Definition 3 (Estimation Area) *In a two-dimensional plane, we define the estimation area as the circular area that is centered at a target's predicted position and has the sensing radius as its radius.*

In Figure 7.1, the two large solid circles (labeled as A and B) are right the estimation areas of two iterations. Since we assume that the sensing radius of nodes is no greater than half of the communication radius, an estimation area will never exceed the scope of the large circle, i.e., the communication range of any sensor node within it. This means that the information of all the nodes that may detect the target thereby participate into particle filtering can be shared with each other.

In Figure 7.3, we let d_0 and d_1 denote the distances of two nodes from the predicted target position, and c_0 and c_1 denote the contributions of nodes 0 and 1 respectively. Based on the prerequisite, the positions of nodes 0 and 1 are known to each other, so is the predicted target position. Thus, both nodes may easily calculate d_0 and d_1.

We set the contribution of a node for a specific target inverse proportional to its distance from the target. Then, the weighted distance of any nodes from the target in the estimation area will be constant. We argue that this model is reasonable, as the closer a node is to the target, the more contribution it will make for estimating the target feature, i.e., the more information users may obtain from it. In fact, this is also intuitive in terms of the idea of PFs: when particles are maintained on sensor nodes, the closer a node is to the target, the more weight the particle maintained on it should have. Therefore, we setup the following proportion equation:

$$c_0 d_0 = c_1 d_1 = \epsilon \tag{7.1}$$

where ϵ is a constant.

By assuming that $c_0 = 1$, node 0 will obtain the relative contribution of its neighbor node 1 as $c_1 = \frac{d_0}{d_1}$. Similarly, node 1 will also obtain the relative contribution of its neighbor node 0 as $c_0 = \frac{d_1}{d_0}$ when assuming $c_1 = 1$. From now on, we only discuss the estimation process of node 0. According to the symmetry, each node in the local area may complete the same estimation process. The only difference would be the values of these relative contributions.

Since node 0 may obtain such an estimation for each of its one-hop neighbors, we assume that all the estimated contributions form a set $\{c_0, c_1, \ldots, c_m\}$, where m is the number of its one-hop neighbors. Then the normalized contributions will be $\{c_0/C, c_1/C, \ldots, c_m/C\}$, where $C = 1 + \sum_{i=1}^{m} c_i$. We define the estimated neighbor contributions as the following:

Definition 4 (Estimated Neighbor Contributions) *Within an estimation area, the contributions of neighbor nodes that are estimated by node* 0 *are defined as:*

$$\{c_0, c_1, \ldots, c_m\} = \{\frac{1}{d_0 \cdot D}, \frac{1}{d_1 \cdot D}, \ldots, \frac{1}{d_m \cdot D}\}$$

where c_0 *represents the contribution of node* 0, c_i $(1 \leq i \leq m)$ *are the contributions of* m *other neighbor nodes in the estimation area,* d_i $(0 \leq i \leq m)$ *are the distances of each node from the predicted target position, and* $D = \sum_{i=0}^{m} \frac{1}{d_i}$.

Based on this definition, we may easily prove the following two propositions are true:

1) The estimated neighbor contributions are normalized; and

2) When the shared node positions and the predicted target position are consistent on all the nodes, so will the contributions estimated by all the nodes.

Theorem 2 *The estimated neighbor contributions are normalized.*

Proof: First, the total contribution from Definition 4 is equal to 1:

$$\sum_{i=0}^{m} c_i = \sum_{i=0}^{m} \frac{1}{d_i \cdot D} = \frac{1}{D} \sum_{i=0}^{m} \frac{1}{d_i} = \frac{1}{D} \cdot D = 1$$

Secondly, the ratio of any two contributions follows the model in Equation 7.1.

Hence, all the defined contributions are normalized. **Done.**

Theorem 3 *When the shared node positions and the predicted target position are consistent on all the nodes, so will the contributions estimated by all the nodes.*

Proof: To prove this proposition, we only need to prove that a node's contribution estimated by itself is equal to that estimated by any other node in the estimation area. Without loss of generality, we evaluate the contribution of node 0 estimated by itself and node 1.

According to Definition 4, node 0's contribution estimated by itself is $\frac{1}{d_0 \cdot D}$. At the same time, its contribution is estimated by node 1 as $\frac{1}{d_0 \cdot D}$. Since the shared node positions and the predicted target position are consistent on all the nodes, either d_0 or D will be consistent in both results. Therefore, the two results are identical. **Done.**

7.5.3 Improved CDPF

The result of this neighborhood estimation can replace the measurement sharing and likelihood function calculation, i.e., the likelihood step in Figure 7.2(b) or steps 6 and 7 in Algorithm 8. The detailed method is:

1) Each node in the estimation area estimates the contributions of itself as well as its neighbors.

2) Based on Definition 4, each node updates the particle weight as $w_{k+1} = w_k \cdot c_0$.

We name this improved version CDPF-NE, where the suffix "NE" represents neighborhood estimation. In this way, c_0 replaces the likelihood function (in case that the proposal function is chosen as the prior). Therefore broadcasting for measurement sharing could be completely eliminated. The analyzed communication cost of CDPF in Table 7.1 will then become $N_s(D_p + D_w)$, i.e., the only communication cost left is for particle propagation. Based on the architecture of "particles on nodes", this communication cost is already the minimum.

7.5.4 Discussion

First, we discuss the frequency of this estimation and its potential overhead. From the definitions above, we may observe that the local status used for neighborhood estimation

mainly involves with node positions, the predicted target position, and the working status of neighbor nodes. First, the node positions never change in a static WSN. Even in a mobile WSN, nodes rarely move fast, either. Secondly, the predicted target position of CDPF is calculated by each individual node based on consistent data. So it is also consistent within the estimation area. Finally, the working status of neighbor nodes is subject to change. However, as long as the change can be anticipated, the estimation still can work correctly. For example, duty cycling is widely used [15] to reduce the energy consumption during idle listening, which is a major source of energy waste [14], thereby improve the network lifetime. With duty cycling, nodes are put into sleep states for most of the time, and only awakened periodically. In certain cases, the sleep pattern of nodes may also be explicitly scheduled, via proactive wake-up [18, 42] for instance. No matter what sleep pattern is taken, the working status can still be anticipated as long as the pattern is certain.

Based on these conditions, we may hence exchange the local status of neighbor nodes and execute the neighborhood estimation at a low frequency, e.g., once per day, once per week or even longer. This will introduce little communication overhead, but gain much improvement on the communication efficiency for target tracking, especially in a WSN where target intrusion events are not rare.

Then we discuss the potential factors that may impact the estimation. According to the previous analysis, the most significant impacts are those uncertain factors, e.g., a random sleep pattern, unexpected node failure, mobile sensor nodes at a high speed, or overloaded nodes due to network congestion. These uncertain factors will propose more requirements on time synchronization. The level of synchronization is dependent on the impact level of these factors. For a real deployment with any of these uncertain features, CDPF-NE needs to be applied carefully. In addition, the estimation depends on the predicted target position. Thus, a wide prior distribution may result in a large error on the estimation result.

7.6 Evaluation

We evaluated CDPF and CDPF-NE in Matlab and compared them with CPF [21] and SDPF [27]. This section reports our evaluation results using the communication cost as the overhead criterion, and root mean squared error (RMSE) as the estimation correctness criterion.

7.6.1 Simulation Environment

The sensor network includes $2,000 - 16,000$ nodes in a two-dimensional plane, which are randomly deployed in a $200m \times 200m$ area. Thus, the node density is $5 - 40$ $nodes/100m^2$. The sensing radius of nodes is set as 10 m, and the communication radius is set as 30 m. A target crosses the surveillance field from the start point $(0, 100)$ with a constant speed

3 m/s. At each time step of 1 s, the target turns a random angle bounded by $[-15^o, +15^o]$. We study the bearings-only tracking problem [24] in the simulation:

$$\begin{aligned} \mathbf{x}_k &= \Phi \mathbf{x}_{k-1} + \Gamma \mathbf{v}_{k-1} \\ z_k &= \arctan \frac{y_k}{x_k} + n_k \end{aligned} \tag{7.2}$$

where $\mathbf{x}_k = (\mathbf{s}_k, \mathbf{v}_k)^T = (x_k, y_k, x'_k, y'_k)^T$, z_k is the observed bearing, and

$$\Phi = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Gamma = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

In addition, $\mathbf{v}_{k-1} = (v_x, v_y)^T_{k-1}$ and n_k are zero mean Gaussian white noises, and the variances of which are respectively $\sigma_v^2 = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$ and σ_n^2.

The detailed parameter configurations for the dynamic system above are as follows. The time step of CDPF is 5 s. The standard deviations of noises are $\sigma_x = \sigma_y = \sigma_n = 0.05$. The simulation includes 50 steps. For CPF, we adopt the number of particles $N_s = 1000$.

For all the four algorithms simulated in the experiments, i.e., CPF, SDPF, CDPF and CDPF-NE, we adopt SIR filters [21] as the basis: we use the prior distribution as the importance density, and execute the resampling step at every iteration.

7.6.2 Experimental Results

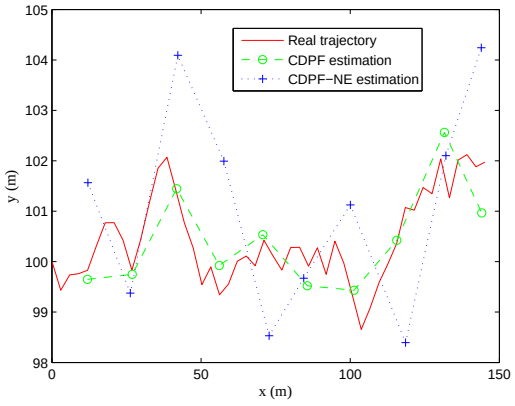
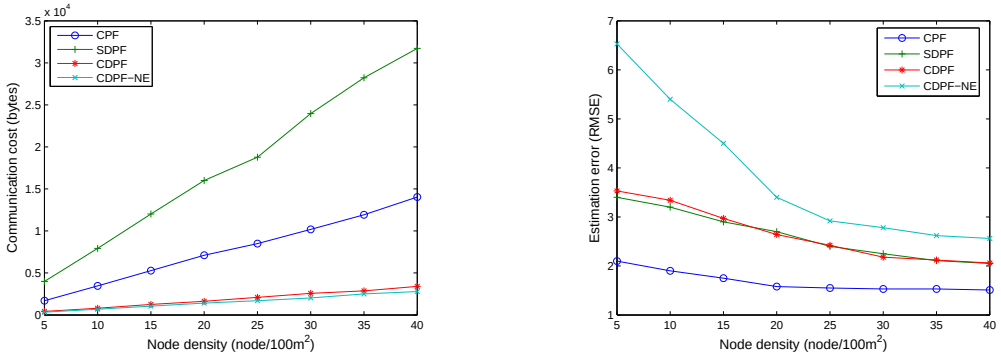


Figure 7.4: Estimation example

First in Figure 7.4, we show an estimation example including CDPF and CDPF-NE, when the node density is 20 $nodes/100m^2$. The real trajectory of the target is shown in a solid curve, which was simulated based on the target model. We may observe that the estimation error of CDPF-NE is a little greater than CDPF, as CDPF-NE replaces the measurement sharing with neighborhood estimation. However, the error of up to 3 m is still tolerable given the node density of 5 $m^2/node$.

Then we examine the communication costs of four algorithms in various node densities. Based on the dynamic model of the bearings-only tracking problem, we assume that a particle includes four integers, and either a measurement or a weight includes one integer only. On a 32-bit platform, we have $D_p = 16$, $D_m = 4$ and $D_w = 4$, all in bytes.



(a) Communication cost (b) Estimation error

Figure 7.5: Transmission delay and multipath forwarding

In Figure 7.5a, the communication costs of all the four algorithms increase as the node density increases. This is because that the number of sensor nodes that detect the target and report the measurement increases. We observe that CDPF and CDPF-NE reduce the communication cost significantly, in which CDPF-NE achieves the minimal communication overhead. Compared with SDPF, their reduction on the communication cost reaches up to 90%. If compared with CPF, they can also reduce the communication by about 70%. Except for their communication reduction efforts, another reason for this is that multiple particles on a single node can be combined into one, thus the data amount for propagation decreases significantly.

A counterintuitive observation is that the communication cost of SDPF is higher than that of CPF. This is caused by the network scale: in the configured network environment, any node can propagate the particle data to the sink node in the center of the network within four hops at the most. Thus, the hop count factor in the communication cost of CPF is not dominant. On the contrary, the eight particles on each node that detects the target increase SDPF's communication workload significantly. Therefore the two curves show a reverse relation. If the surveillance field is large enough so that the hop count factor dominates the communication cost, two curves are supposed to reverse their positions.

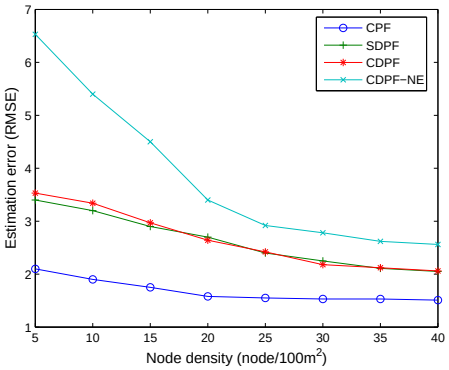


Figure 7.6: Estimation error

Finally, we present the result of the estimation error. In Figure 7.6, CDPF shows a similar RMSE to SDPF, as their operations on measurement sharing and particle propagation are similar. CDPF-NE shows the greatest estimation error, which is about 100% to 30% more than SDPF, as it simulates the likelihood function by estimating the contribution of neighbor nodes. However, the estimation error of CDPF-NE decreases faster than others, because the error difference will become less remarkable as the node deployment reaches a certain level of density. As long as a reasonably big estimation error can be tolerated, CDPF-NE would be the most efficient choice.

7.7 Conclusion

Based on the experimental evaluation in Section 7.6, we observe that compared to SDPF, CDPF introduces a significant reduction of 90% on the communication cost, with a similar estimation error. Though CDPF-NE introduces a much larger increase on the estimation error than its reduction on the communication cost, it provides an option to achieve the minimum communication. The application of CDPF-NE is subject to several conditions, e.g., static nodes and stable working status of nodes. In many deployments, these conditions are easy to satisfy. Therefore, CDPF-NE is still helpful for bounding the network performance.

Chapter 8

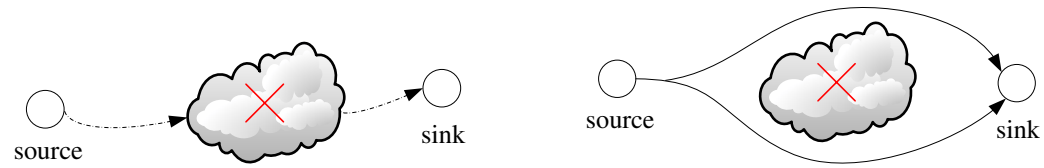
Constrained Flooding

8.1 Motivations

Once nodes detect a new position of the target during the tracking process, they need to transmit the captured data, i.e., target information, to sink nodes. Like local tracking operations, this propagation process also involves lots of communications, sometimes more than necessary. On one hand, there may exist multiple source nodes that transmit data, which are highly likely identical or similar. We name this *identical report effort*. On the other hand, when certain performance metrics are required to be satisfied, source nodes may need to transmit redundant data copies. We name this *redundant copy effort*. Both of these efforts will result in an increment on the communication overhead thereby on the energy consumption. In this chapter, we present an energy efficient solution to the extra communication overhead problem introduced by the redundant copy effort. At the same time, this solution also works for the problem introduced by the identical report effort to some extent.

Real-time performance is one of the most important QoS metrics for time-sensitive applications of WSNs. For example, a target tracking system [6] may require sensors to collect and report target information to sink nodes before the target leaves the surveillance field. For improving real-time performance, we need to ensure that as many time-sensitive packets as possible, arrive at sink nodes within their deadlines. The delay that a packet may experience during propagation may be caused by many reasons, including those due to network congestion and node/link failures.

Multipath forwarding is a commonly used redundant copy approach for enhancing real-time performance of WSNs [30]. Figure 8.1a shows the transmission delay caused by network congestion or node/link failure, either of which will cause a "hole" without regular connection in the network. In Figure 8.1b, multiple paths are shown to bypass such a network hole, with multiple data copies forwarded along different paths. However, it is possible that



(a) Network congestion or node/link failure (b) Multipath forwarding

Figure 8.1: Transmission delay and multipath forwarding

network congestion or the connection status is not significant enough throughout the entire path from source to sink to warrant multipath forwarding for each hop. Therefore sometimes, it is unnecessary to transmit redundant data copies generated by multipath forwarding protocols, which often consumes additional energy and bandwidth. Since sensor nodes are battery-powered and therefore energy must be efficiently consumed, energy-efficient forwarding protocols are critical toward enhancing the capability of delivering real-time packets within given end-to-end time constraints, while reducing the energy consumption.

Previous research efforts that study the efficiency of multipath forwarding protocols have focused on reducing the number of flooding recipients at each hop [31–36]. However, even though the number of recipients is reduced, the redundancy introduced by this approach remains due to its probability-based recipient selection mechanism. In fact, for those hops in which the connection status is good enough so that unicasting does work, redundant multipath flooding can be completely eliminated, and therefore the efficiency of flooding therefore can be further improved.

Data aggregation has been widely used to reduce the redundancy of data propagation, no matter where the redundancy is introduced, either by the identical report effort or the redundant copy effort [37]. Whenever possible, combining multiple data packets into one will reduce the communication overhead, even if the aggregated data length is no less than the total length of data packets before the aggregation [38].

Unlike the aggregation research work introduced previously, we aim at leveraging data aggregation as an auxiliary effort for enhancing energy efficiency. We do not consider the aggregation structure, or the aggregation function itself. Instead, we try to understand how much improvement on energy efficiency a data aggregation algorithm, with either semantics-based aggregation function or length-based aggregation function, may introduce to our constrained flooding protocol.

In this chapter, we present a constrained flooding protocol, named CFlood, to improve the energy efficiency on transmitting real-time data. The primary objective of CFlood is to enhance the deadline satisfaction ratio (DSR) per unit energy consumption. CFlood uses flooding to enhance DSR, and effectively improves the energy efficiency by controlling the flooding scale and aggregating redundant data. In addition, we improve CFlood with a data aggregation effort. During the time that a node waits for the possible abort message,

incoming packets may be aggregated to control the flooding scale further. CFlood is designed as a hop-by-hop routing protocol with no global network information needed. Therefore, it is scalable for large-scale sensor networks.

We conducted extensive simulation-based experimental studies to evaluate CFlood's performance on transmitting real-time packets and improving the energy efficiency. Especially, we compare the performance of CFlood with or without data aggregation to understand the contribution of data aggregation on controlling the communication overhead. Our results reveal that CFlood achieves a higher deadline satisfaction ratio per unit energy consumption than previous multipath data delivery protocols, such as a multipath routing protocol MCMP [30] and a directional flooding protocol DFP [46], especially in sparse or unreliable network environments.

8.2 Design Overview

CFlood is a decentralized flooding-based routing protocol. Routes are determined, i.e., recipients are chosen at each hop dynamically during data propagation. CFlood uses flooding to increase the deadline satisfaction ratio. But flooding may not be necessary at each hop. It is desirable to constrain the scale of flooding as much as possible to enhance energy efficiency.

Our approaches for controlling the flooding scale thereby enhancing energy efficiency δ include:

1) Reduce the flooding actions as much as possible, and using unicasting instead;

2) Reduce the number of recipients when flooding is necessary; and

3) Aggregate data whenever possible without extra waiting time.

Our intuition in controlling the flooding scale of CFlood with the first approach is as follows. After a recipient at a given hop finds that the transmission of its next hop can meet the sub-deadline, it is unnecessary for other recipients to continue flooding for the next hop. Thus, this recipient can abort the subsequent flooding of other ones. For this hop, it seems as unicasting is used instead of flooding. This way, the end-to-end time constraint can be satisfied and the energy efficiency can be enhanced.

Even if flooding is necessary, we should reduce the number of recipients. We use several criteria for recipient selection, in which the end-to-end time constraint is the primary one. First, we introduce a deadline partition method to distribute the end-to-end deadline to multiple hops. Then, CFlood estimates the per-hop delays between nodes. If the estimated per-hop delay is longer than the distributed sub-deadline, it is highly unlikely that the route through the node can meet the end-to-end time constraint. Otherwise, the node can be chosen as a recipient of the flooding. With this approach, the number of recipients can potentially be reduced with respect to meeting the end-to-end time constraint.

During the time when SRs are waiting for ABORT messages from PR or timer expiration, it is likely that new data packets arrive at these SRs, especially when the distributed sub-deadline is not tight and the timer is configured relatively long. Without any extra waiting effort, incoming packets may be aggregated with the ones that are being held on the nodes so that the communication overhead may be reduced further. As each flooding recipient may be chosen as either a PR or a SR for each individual packet, a SR for one packet may be chosen as either the PR or one of SRs for another packet. In this case, the aggregation operation needs to be designed carefully.

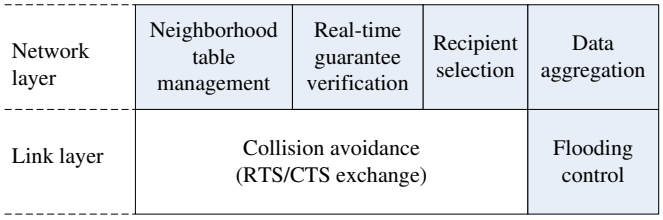


Figure 8.2: Functional components of CFlood

We design CFlood in five functional components, which are shown in a network protocol stack in Figure 8.2. The five components work in the following manner:

- *Neighborhood table management.* CFlood maintains a neighborhood table on each node to save routing information, neighboring relations, and estimated per-hop delays. Periodic HELLO message exchange is used to share these information fields among neighbor nodes, update the table entries, estimate per-hop delays between each pair of nodes, and synchronize the clocks of nodes locally.

- *Real-time guarantee verification.* Among all the one-hop neighbors, we want to flood only to those nodes that can satisfy the time constraint. This component verifies whether a neighbor can meet the end-to-end time constraint, and therefore could be considered as a flooding recipient. This decision is made by comparing the estimated per-hop delay, denoted as L_h, and the distributed per-hop sub-deadline, denoted as D_h. The per-hop delay L_h is estimated using HELLO message exchange, while the distributed per-hop sub-deadline D_h comes from a deadline partition method that we develop to partition the end-to-end deadline to sub-deadlines of multiple hops. If $L_h < D_h$, i.e., the transmission at this hop can be completed within the sub-deadline distributed to this hop, then this neighbor could be selected as a prospective recipient.

- *Recipient selection.* Among the neighbors that can meet the deadline, the recipient selection component selects a primary recipient and several secondary recipients as potential flooding nodes at the next hop. First, each node periodically computes a next-hop neighbor (or parent as in [45]), which has the highest probability of meeting the time constraint. This parent node is used as the primary recipient (PR) of flooding. Then, each node also selects several secondary recipients (SRs) based on the time

constraint as well as the criteria on flooding-controllability, congestion avoidance and computation simplicity. When unicasting is determined to be sufficient for meeting a packet's time constraint, the PR aborts the SRs' next-hop flooding (this is done through the flooding control mechanism discussed next). Otherwise, the PR and all the SRs continue to flood the packet further.

- *Flooding control.* After flooding to the recipients, the flooding control component takes the responsibility of aborting further flooding from secondary recipients if unicasting to the primary recipient can meet the distributed sub-deadline. The working sequence of a MAC protocol, which supports collision avoidance with the RTS/CTS exchange mechanism, can be summarized as RTS-CTS-DATA or RTS-BACKOFF, depending on whether the channel reservation using RTS/CTS exchange succeeds. For controlling the flooding scale, CFlood inserts an ABORT step after CTS for the PR's flooding, and a WAIT step before RTS for SRs' flooding. Thus, the working sequence of a PR will be modified as RTS-CTS-ABORT-DATA or RTS-BACKOFF, and that of SRs will be modified as WAIT-ABORT, WAIT-RTS-CTS-DATA or WAIT-RTS-BACKOFF. If a PR finds that the channel is clear after receiving a CTS, it broadcasts an ABORT message so that SRs can abort their subsequent flooding actions.
- *Data aggregation.* During the time interval when a PR is waiting for CTS reply or SRs are waiting for the abort messages, they may receive new data packets. We name this time interval a *waiting time*. Then, the data aggregation component may optionally aggregate these incoming packets with the ones held on the nodes. Therefore, the communication overhead for the next hop will be further reduced.

Before discussing the functional components in detail, we finally summarize all the notations in Table 8.1 for convenience in discussion.

8.3 CFlood Design Details

8.3.1 Neighborhood Table Management

This component manages the neighborhood table, each entry of which corresponds to a neighbor node. An entry includes the following fields:

(NeighborID, ParentID, HopCount, SendDelay, NeighborhoodTable, ClockDifference, TTL)

ParentID is the ID of this neighbor node's parent. HopCount is the number of hops by which the neighbor node is away from the sink node. SendDelay is the estimated delay for sending a packet to this neighbor node. NeighborhoodTable is a structure composed of this neighbor node's neighborhood table. ClockDifference is its time difference from this neighbor node.

Table 8.1: Notations

R	Transmission radius	$NB(N_i)$	Neighbors of node N_i
r	Sensing radius	$Parent(N_i)$	The parent of node N_i
L_h	Estimated per-hop delay	$Source(N_i)$	The source node from which N_i receives a packet
D_h	Distributed per-hop deadline	$Forward(N_i)$	Flooding recipients of node N_i
PR	Primary recipient	$Hop(N_i)$	The number of hops that N_i is away from the sink
SR	Secondary recipient	ρ	Node density
T_h	The average throughput of a node at hop h	C_h	The number of nodes that are h hops away from the sink node
SL_i	Slack time ratio	DSR	Deadline satisfaction ratio
δ	Real-time capacity per unit energy consumption	e	The average energy for transmitting a single time-sensitive packet

TTL is short for Time To Live. The table management operations include adding, updating, expiring, and removing.

The entries in the neighborhood table are updated via HELLO message exchanges, which is a commonly used approach for sharing local knowledge among neighbors [31]. The mechanism has the advantage that it can adapt the network to possible topology changes (e.g., those caused by link failure, node failure). Each HELLO message includes the fields (SenderID, ParentID, HopCount, SendDelay, NeighborhoodTable, CurrentClock), so that all the receivers may: 1) update this neighbor node's routing information in their individual neighborhood tables (i.e., neighbor management); 2) update its own routing information if necessary (i.e., routing); and 3) update their clock difference from this neighbor node (i.e., time synchronization).

At the beginning when a network is deployed, each node in the network holds an empty neighborhood table, except for the sink node. From a HELLO message received from a sink node, all the one hop neighbors of the sink node will know that their HopCount is 1 and their parent is the sink node. By iteration, the neighborhood tables will be filled hop by hop throughout the network. Specifically, the selection of a node's parent and the estimation of SendDelay will be detailed later in this section.

8.3.2 Real-time Guarantee Verification

As previously discussed, CFlood compares the estimated per-hop delay L_h with the distributed per-hop sub-deadline D_h to determine whether or not a potential recipient can satisfy the time constraint. This component is responsible for estimating L_h, computing D_h, and conducting the comparison.

Per-hop delay estimation

The delay experienced at a hop usually consists of the transmission delay, the propagation delay, and the receiving delay. The transmission delay is the time that a packet experiences at the sender's MAC and PHY layers. The propagation delay is the duration when a data signal together with its carrier travels in the air. The receiving delay is the time that a packet experiences at the receiver's PHY and MAC layers. Since the delay includes parts at both the sender and the receiver, the precise measurement will require time synchronization, which is generally energy inefficient [109]. We introduce a feasible mechanism without assuming time synchronization, although our estimation result may not be perfectly precise due to the asymmetry of wireless channels.

The problem of estimating the round-trip delay has been well studied in the past [110]. We simply apply the existing method into the HELLO message exchange mechanism. Suppose the neighbors of a node N are $\{N_i | N_i \in NB(N)\}$. Node N may append a round-trip delay estimation request for a specific neighbor node N_i in a randomly chosen HELLO message (e.g., one out of every twenty continuous HELLO messages). Neighbors other than N_i deal with this HELLO message as usual, while N_i is supposed to reply with a HELLO message immediately. Then a round-trip delay is obtained by node N, the half of which can be used as the estimated per-hop delay and is saved in the SendDelay field of N_i's entry. This does not require time synchronization since the starting and ending time points of the round trip are sampled at the same node.

Per-hop deadline computation

Most of the past works on the end-to-end deadline partition [71, 111] have adopted either uniform or exponential models. Uniform distribution allocates the total end-to-end deadline evenly to all the hops from the source to the sink, implicitly assuming that a packet suffers the same delay at each hop. The exponential model computes the per-hop sub-deadline as $D_h = \frac{D}{2^h}$, where h is the number of hops from the sink node and D represents the end-to-end deadline. These schemes are based on analytical models and do not consider the actual throughputs of the network. We now introduce a throughput-based model by establishing a relationship between the per-hop sub-deadline and the number of nodes at each hop in an intuitive manner.

Let ρ denote the node density of the network. Now, the average number of nodes that are h hops away from the sink node, denoted as C_h, can be computed as $\rho(\pi(hR)^2 - \pi[(h-1)R]^2) = \rho\pi R^2(2h-1)$. Intuitively, at a specific hop in a convergecast network, lesser the number of nodes, greater will be the traffic that each node has to transport toward the sink node. Thus, longer will be the delay that a packet will suffer at the hop. Consequently, a longer sub-deadline will be needed for the hop. This relationship can be approximately modeled as $D_h \sim T_h \sim \frac{1}{C_h}$, where T_h is the average throughput of a node at hop h. When the node density ρ is fixed for a given implementation, we have $D_h \sim \frac{1}{2h-1}$. Thus, the end-to-end deadline D over an h-hop transmission can be distributed according to its weight at each hop k as $D_k = \frac{\frac{1}{2k-1}}{\sum_{k=1}^{h}\frac{1}{2k-1}} \cdot D$. Especially the sub-deadline of the first hop from the source is:

$$D_h = \frac{\frac{1}{2h-1}}{\sum_{k=1}^{h}\frac{1}{2k-1}} \cdot D \tag{8.1}$$

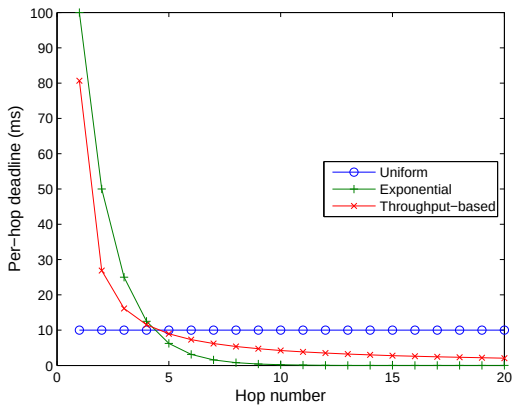


Figure 8.3: Deadline distribution

Figure 8.3 shows the comparison among the uniform model, the exponential model, and the throughput-based model, for an example with a 200 ms end-to-end deadline over 20 hops. We can observe that compared with the uniform model, the throughput-based model is more adaptive for the many-to-one convergecast architecture of WSNs. In addition, unlike the exponential model for which the distributed per-hop deadline decreases quickly to zero, the throughput-based model supports larger-scale networks.

For a single node N, its hop count from the sink node may be different when considering different routes via different neighbor nodes. Therefore the result of (8.1) needs to be computed for each potential recipient. Suppose HopCount of a neighbor node N_i is h_i. When N_i is used for relaying, the distributed per-hop sub-deadline at this hop is:

$$D_{h_i+1} = \frac{\frac{1}{2(h_i+1)-1}}{\sum_{k=1}^{h_i+1}\frac{1}{2k-1}} \cdot D \tag{8.2}$$

Real-time guarantee verification

For a specific neighbor N_i, we establish a function $CanMeetDeadline(N_i)$, in which the per-hop deadline D_{h_i+1} is computed and compared with the per-hop delay L_{h_i+1} (i.e., SendDelay in the neighborhood table). The function $CanMeetDeadline(N_i)$ returns true when $L_{h_i+1} < D_{h_i+1}$ and false otherwise. Only those neighbors that has a true return value will be considered as potential recipients.

8.3.3 Recipient Selection

The recipient selection component is responsible for selecting the flooding recipients (both a PR and multiple SRs) from the one-hop neighbors. With CFlood, each node computes a parent node as the PR periodically even when no data packets are passing through. The criteria used for SR selection include real-time guarantee, flooding-controllability, congestion avoidance, and simplicity of computation.

Primary recipient

As one of CFlood's techniques to constrain the flooding scale is to substitute unicasting for flooding as much as possible, a parent needs to be prepared for each node as the next-hop neighbor of unicasting.

For a neighbor N_i, (8.2) shows the distributed per-hop deadline D_{h_i+1}. We can also estimate the per-hop delay L_{h_i+1} with the round-trip HELLO message exchange. Thus, we define a ratio $SL_i = 1 - \frac{L_{h_i+1}}{D_{h_i+1}}$ to describe the proportion of the slack time, and call it, the *slack time ratio*. The slack time ratio describes how likely N_i can meet a packet's sub-deadline for this hop. Based on our throughput-based deadline partition model, the slack time ratio also describes how likely a route via N_i can meet the end-to-end time constraint. Therefore, we select a neighbor N_i with the maximum SL_i as the parent node, i.e., a neighbor with

$$\max_i(SL_i) \sim \min_i \left\{ \frac{L_{h_i+1}}{D_{h_i+1}} \right\} = \frac{1}{D} \cdot \min_i \left\{ L_{h_i+1} \cdot \frac{\sum_{k=1}^{h_i+1} \frac{1}{2k-1}}{\frac{1}{2(h_i+1)-1}} \right\}$$

Secondary recipients

For selecting SRs, we progressively remove those neighbor nodes that cannot satisfy the following criteria:

1) *Real-time guarantee.* An SR should meet the time constraint (i.e., $CanMeetDeadline(N_i)$ returns true).

2) *Flooding-controllability.* The subsequent flooding of an SR should be able to be aborted by the PR.

3) *Congestion avoidance.* An SR should not introduce new congestion, since CFlood's major objective is to quickly bypass network congestion or connection failure. Thus we remove redundant SRs that share the parent with other SRs and have lower probabilities for meeting the time constraint. By strictly prohibiting two recipients from sharing a common parent, the network congestion could be avoided at least for the next hop.

4) *Simplicity of computation.* CFlood is designed to be as simple as possible due to the constrained computing capability of sensor nodes. We use a set of "common sense-based" operations to quickly reduce the problem size before applying the first three ones. These quick reduction operations include: a) remove all the neighbors whose parent is also a neighbor of the flooding node, and thus also has a chance to receive the packet at this hop, i.e., $Parent(N_i) \in NB(N)$; b) remove all the neighbors that send packets to the flooding node at the last hop, i.e., $N_i = Source(N)$; and c) remove all the neighbors that have received packets at the last hop, i.e., $N_i \in Forward(Source(N))$.

Next, we describe the SR selection algorithm at a high-level of abstraction in Algorithm 9. The algorithm complexity is $O(n^2)$ for searching N_i in $Forward(Source(N))$, where n is the average number of one-hop neighbors of a node. The magnitude of n is small. For example, our simulation shows that in a network with node density 0.005 $node/m^2$ (i.e., each node covers an area of 200 m^2), n is only up to 5.

8.3.4 Flooding Control

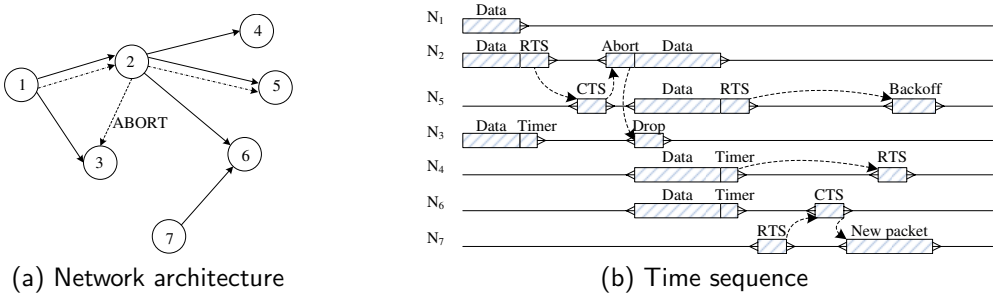
One of the most important contributions of the research work in this chapter is the flooding control mechanism, i.e., to abort the subsequent flooding after the current flooding occurs. In detail, the PR and the SRs forward packets in different ways. As the flooding node of the next hop, the PR initiates an RTS/CTS exchange with its PR immediately after receiving a packet. If a CTS is received successfully, the PR broadcasts an ABORT message and then unicasts the data packet. Otherwise, the PR backs off for some period of time and again initiates the RTS/CTS exchange later. Unlike the PR, the SRs set an ABORT timer for each received data packet. If an ABORT message from a PR is received for a buffered data packet, the SRs drop that packet. Otherwise, if the timer runs out first, the SRs know that the PR's flooding for the next hop is delayed (i.e., backed off), and therefore they start to flood the packet to the next hop. In this way, when the network condition is good (e.g., the RTS/CTS exchange initiated by the PR succeeds without backoff), the flooding is reduced to unicast (because the SRs drop the packet on receiving the ABORT message).

Overhearing is also an approach for controlling flooding [106], e.g., the SRs abort the subsequent flooding upon overhearing the transmission of the PR. However, overhearing is not a good choice under the time constraint. Not until the SRs overhear the complete packet

Algorithm 9 Secondary recipient selection

Initialize the SR candidate set as $SR = \{N_i | N_i \in NB(N), Hop(N_i) < Hop(N)\}$;
for all $(N_i \in SR)$ **do**
    Examine N_i with the conditions of three quick reduction operations, i.e., remove N_i if $(Parent(N_i) \in NB(N))$ or $(N_i = Source(N))$ or $(N_i \in Forward(Source(N)))$;
    Remove N_i if it violates the time constraint, i.e., if $(CanMeetDeadline(N_i)$=**false**$)$;
    Remove N_i if it violates the flooding-controllability criterion, i.e., if it cannot hear from the PR $(N_i \notin NB(PR))$;
    Remove N_i if it violates the congestion avoidance criterion by sharing a parent with the PR, i.e., if $(Parent(N_i) = Parent(PR))$;
end for
if $(SR == \phi)$ **then**
    return ϕ
end if
Sort the remaining SRs in a descending order of SL_i;
for all $(N_i \in SR)$ **do**
    Remove N_i if it violates the congestion avoidance criterion by sharing a parent with another SR with a higher SL_i, i.e., if $(\exists j < i, Parent(N_i) = Parent(N_j))$;
end for
return SR

payload and send it up to the network layer, can they drop the corresponding packet saved in the buffer. Such a transmission through the network stack may introduce extra delays, especially when the data packet is long.



(a) Network architecture (b) Time sequence

Figure 8.4: An example of CFlood's flooding control mechanism

Figure 8.4 shows an example of CFlood's flooding control mechanism. In the figure, the circle with the number i represents node N_i. In Figure 8.4a, the dash-dot arrows show the parent relation (e.g., $Parent(N_2) = N_5$), the solid lines show the actual data propagation (e.g., $Forward(N_1) = \{N_2, N_3\}$), and the dotted line represents the ABORT messages. Figure 8.4b shows the time sequence of the nodes. The dotted curves show the working mechanism of flooding control. When N_2 (as the PR of N_1) finds that the channel is clear

on receiving the CTS reply, it broadcasts an ABORT message and then sends out the data packet. Node N_3 (as an SR of N_1) receives the ABORT message from N_2 before the timer runs out. Thus, it aborts the subsequent flooding and drops the packet. On the contrary, N_5 does not receive the CTS reply from its parent node. Thus, it has to backoff for sometime without broadcasting an ABORT message. As N_4 and N_6 do not receive the expected ABORT message before the timer runs out, they have to continue the flooding by initiating the RTS/CTS exchange. The expiration time for the ABORT timers on SRs can be either determined by specific application configurations, or computed as the minimum allowed slack time $\min_i\{D_{h_i+1} - L_{h_i+1}\}$ of that SR node at the next hop.

8.3.5 Data Aggregation

As discussed previously, data may be aggregated when a PR or SRs are waiting for CTS or ABORT messages in the waiting time. But actually, the aggregation can only happen in the failure case, where the PR does not receive CTS thus SRs cannot receive ABORT messages. From Figure 8.4b, we may observe that in the successful aborting case where node N_3 receives the ABORT message from N_2, the local wireless channel of node N_3 will be fully occupied by RTS and the ABORT message that N_2 sends. Even if there is a short gap between RTS and ABORT, N_3 cannot reply to any other source nodes with CTS. This is because that N_3 has to keep silent in this short term to reserve the channel for N_2 to receive CTS from its PR successfully. On the contrary, in the failure case when node N_5 does not receive any CTS and accordingly node N_6 does not receive any ABORT message from N_5 in the expected time, they will consider the channel becomes available. Then, for any incoming RTS requests from other source nodes, e.g. node N_7, node N_6 may reply with CTS and receive new data packets, so that it is possible to aggregate them with the packet held on the node. Therefore, we mainly discuss the aggregation in the failure case, i.e., when new packets arrive between RTS and BACKOFF (for a PR) or between WAIT and RTS (for a SR).

During the waiting time, there must be already a packet on an aggregating node, either a PR or a SR. We call it the *resident packet*. In the failure case, the resident packet needs to be flooded out before the distributed sub-deadline, no matter it is a PR or a SR, no matter there are new packets incoming or not. However, the remaining slack time to the sub-deadline may or may not be enough to receive a new packet. If it is not enough (i.e., the remaining slack time for the resident packet is shorter than the expected transmission time of the new packet in RTS), receiving the new packet will exceed the sub-deadline of the resident packet, thereby failing to delivering it timely. Even if it is enough otherwise, we should still consider whether to receive and aggregate depending on whether the aggregating node is a PR or a SR for the new packet. If the aggregating node is a SR of the new packet and the sub-deadline of the new packet is longer than the resident packet, aggregating them and flooding them out before the resident packet's sub-deadline may lose the opportunity of constraining the flooding for the new packet. Given such a complicated situation, we make

the decision of CFlood's data aggregation based on the following several conditions:

1) *Enough time condition.* True if the remaining slack time for the resident packet is enough for receiving a new packet and aggregating it with the resident packet.

2) *Primary recipient condition.* True if the aggregating node is the primary recipient of the new packet.

3) *Semantics-based aggregation condition.* True if the aggregation operation is semantics-based.

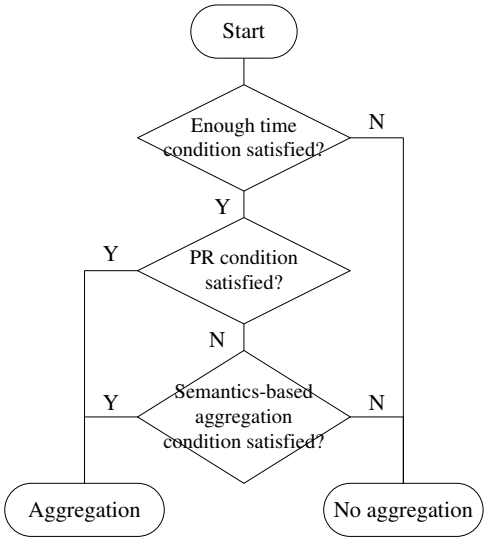


Figure 8.5: Decision process for data aggregation

Then based on the three conditions, we make the decision for data aggregation in the process shown in Figure 8.5. We explain the decision making process as follows. When the remaining slack time for the resident packet is not enough for receiving a new packet and aggregating it with the resident packet, aggregating will delay the transmission of the resident packet, thus we do not execute the aggregation. If time is enough, we further distinguish the cases of whether the aggregating node is a PR or a SR of the new packet. If it is a PR, data aggregation will introduce no extra overhead. Therefore, CFlood will aggregate the packets and flood the aggregated version to the next hop. If otherwise it is a SR of the new packet, CFlood checks if the aggregation operation is semantics-based or length-based. As previously discussed, semantics-based data aggregation may reduce the data amount during propagation. In this case, the flooding of the new packet on this SR does not introduce more communication workload, and CFlood will take the aggregation choice. Otherwise if the aggregation operation is length-based, we flood two packets separately without data aggregation to simplify the computation.

8.4 Experimental Evaluation

We compared CFlood (without data aggregation) and CFlood-DA (with data aggregation) with three past competitor protocols. Mint routing (MR) is a single path delivery protocol [45], which serves as a lower bound on both real-time performance and energy consumption. MCMP [30], a multipath routing protocol, is one of the latest efforts on optimizing data delivery under both real-time and reliability constraints. DFP [46] (short for Directional Flooding Protocol) is a forwarding protocol that optimizes the delivery probability. Next, we first introduce the simulation environment configuration, then report the evaluation results.

8.4.1 Simulation Environment

We evaluated CFlood and CFlood-DA using the simulation tools Qualnet 4.0 [112] and sQualnet [113], which is an extension to Qualnet for sensor networks. The simulation is based on the CSMA/CA MAC protocol implemented in sQualnet.

In the simulation, we deploy 20 to 150 sensor nodes uniformly in a square area of $200m \times 200m$, and assume Mica motes [87] as the hardware platform. We set $R = 60m$, $r = 30m$, and the data rate as 38.4 kbps. We leverage the statistics provided by sQualnet to estimate the energy consumption.

Each sensor node samples and reports an event (e.g., detection of a target) once per second. We configure the lengths of a data packet, a HELLO message, and an ABORT message as 150 bytes, 50 bytes, and 10 bytes, respectively. Usually 10 bytes (e.g. including the ID of the source node and the ID of this packet) are long enough for an ABORT message to identify a specific data packet.

We measure the performance metrics of interest of CFlood, CFlood-DA and the competing protocols under varying degrees of node density, link reliability, and end-to-end time constraint. The default values of these varying conditions are set as 0.0015 $node/m^2$, 75%, and $100ms$, respectively.

8.4.2 Simulation Results

Figure 8.6 shows the deadline satisfaction ratio of the four protocols under different node densities. Due to the competition of the two factors, the number of recipients and the congestion level, the curves present crests. We observe that CFlood yields the best DSR.

Figure 8.7 shows the deadline satisfaction ratio under various per-hop wireless link reliability. Among the four protocols, CFlood yields the best DSR, especially when the link reliability is low. As the link reliability increases, the DSRs of CFlood, MCMP, and DFP tend to be comparable, and MR remains as the lower bound. We observe that MCMP performs well

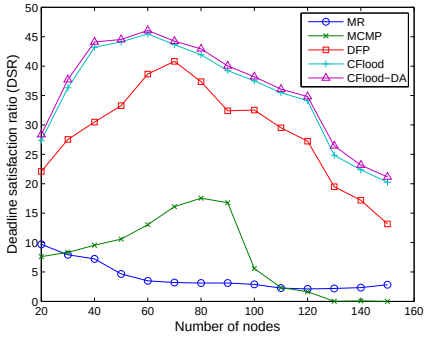


Figure 8.6: Deadline satisfaction ratio DSR vs. node density

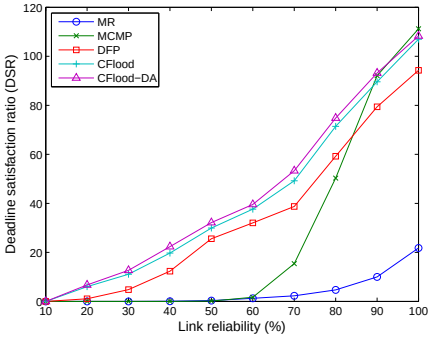


Figure 8.7: Deadline satisfaction ratio DSR vs. link reliability

especially when the network condition is good, while CFlood is more adaptive to unreliable network environments.

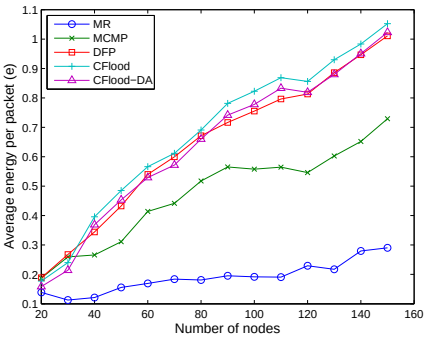


Figure 8.8: Average energy consumption e vs. node density

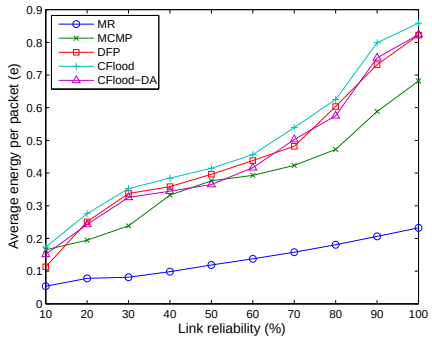


Figure 8.9: Average energy consumption e vs. link reliability

Figure 8.8 shows the average energy consumption per real-time packet under different node densities. We observe that MR, as the lower bound, consumes the least energy, and CFlood consumes the most. But the energy consumption of CFlood is very close to that of DFP.

Figure 8.9 shows that CFlood consumes the most energy. However, this performance loss will be compensated by its improvement on DSR.

Figure 8.10 shows the real-time capacity per unit energy consumption δ under different node densities. We observe that CFlood performs the best, especially when the node density is low. Detailed simulation results show that on average, CFlood is 197%, 346%, and 20% better than MR, MCMP and DFP, respectively.

As shown in Figure 8.11, when we consider the real-time capacity per unit energy consumption δ, CFlood outperforms the other three protocols when the link reliability is lower than

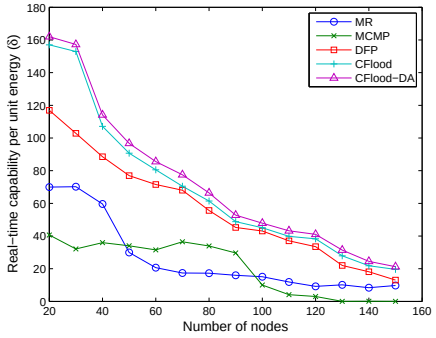


Figure 8.10: Real-time capacity δ vs. node density

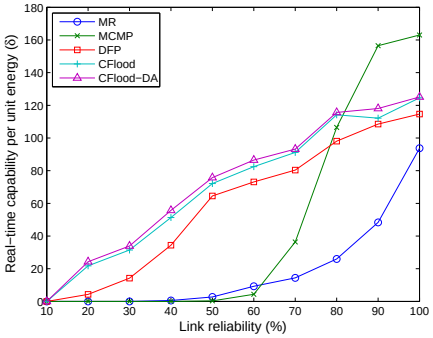


Figure 8.11: Real-time capacity δ vs. link reliability

80%.

In the previous plots, the performance of CFlood-DA is very similar to CFlood. Though it introduces a little improvement based on CFlood, the improvement is not significant. This is all because that the end-to-end deadline that we configured was a little tight, so that there is no enough time to aggregate data during the waiting time.

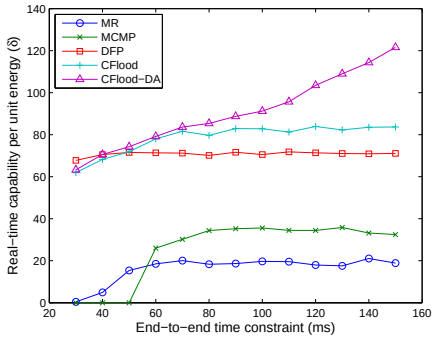


Figure 8.12: Real-time capacity δ vs. end-to-end time constraint

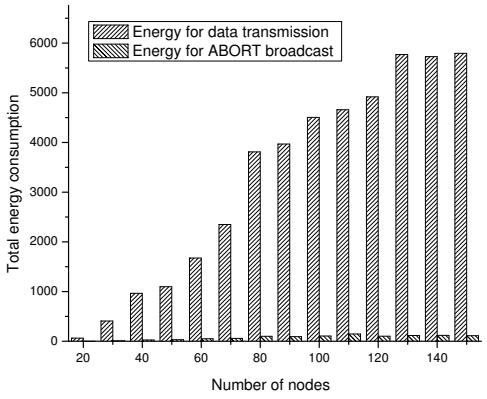


Figure 8.13: Energy for data transmission and ABORT broadcast

Figure 8.12 shows that the real-time capacity per unit energy consumption δ of CFlood is higher than that of the other three protocols, as long as the end-to-end time constraint is not very tight. Especially, CFlood-DA performs much better than CFlood when the time constraint is loose, because the aggregation opportunity during the waiting time for ABORT messages increases. Therefore, CFlood-DA provides a better choice on enhancing energy efficiency for real-time data propagation.

CFlood's flooding control mechanism based on ABORT messages does introduce some overheads. However, the extra energy consumption is negligible. Figure 8.13 shows the contrast

between the energy consumption for data transmission and that for the ABORT message exchange. We observe that the energy consumption for ABORT messages is only about 1% of the energy consumption for data transmission. This implies that CFlood's flooding control mechanism introduces little overhead.

Thus, our simulation results reveal that CFlood achieves better real-time capacity per unit energy consumption than past protocols, especially for sparse node deployment, unreliable wireless links, and loose end-to-end time constraints.

8.5 Conclusion

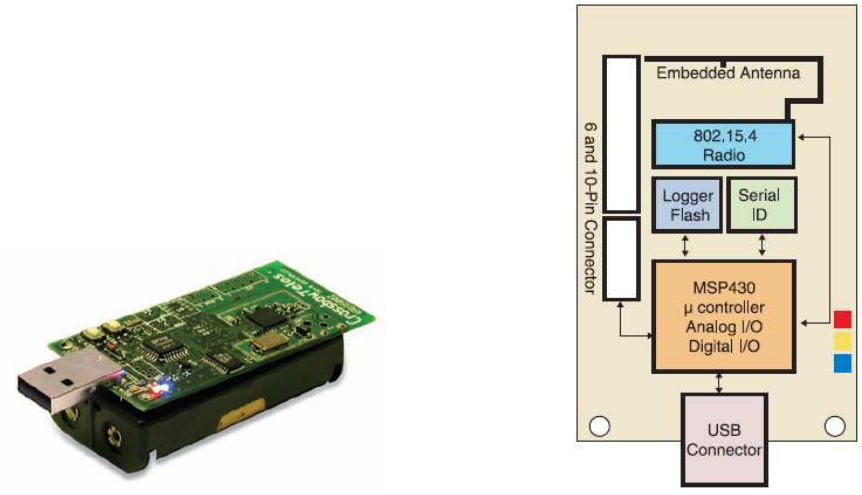
This chapter presents CFlood protocol to enhance the deadline satisfaction ratio per unit energy consumption in WSNs. Based on the design of CFlood protocol, we notice that modifying the underlying MAC protocol enables nodes to estimate the result of the following transmission at an early stage. In addition, local short message exchange, e.g., HELLO messages used by CFlood, not only enables routing information propagation, but also helps on local knowledge share. Besides, it can be used for round-trip delay estimation and local time synchronization. Another observation from the constrained flooding problem is, for real-time data propagation, the effect of data aggregation depends on the end-to-end deadline. A loose deadline will increase the successful probability of data aggregation, thereby improve the flooding efficiency.

Chapter 9

Prototype Implementation

We implemented a prototype system based on TelosB motes [39] and TinyOS version 2.1.1 [41] to evaluate our algorithms and protocols. In this chapter, we first introduce hardware and software platforms as well as overview the implementation and experiment, then present the experiment results in details.

9.1 Hardware and Software Platform



(a) TelosB mote (b) TelosB block diagram

Figure 9.1: TelosB

The hardware platform used in the implementation is Crossbow TPR2400 mote [39], i.e., TelosB. TelosB is an open-source experimental platform developed and published to the

research community by University of California, Berkeley. This platform integrates an 8 MHz TI MSP430 microcontroller, 10kB RAM, 48kB internal and 1MB external flash, an IEEE 802.15.4 radio with integrated antenna, an optional sensor suite (including integrated light, temperature and humidity sensors, i.e. TPR2420), and a USB interface for programming and data collection. Moreover, TelosB is powered with two AA batteries. Figure 9.1a is a TelosB mote, and Figure 9.1b shows its block diagram. This TelosB mote is fully compatible with the open source TinyOS distribution [41] introduced below.

The software platform is TinyOS version 2.1.1 [41], an open source operating system designed for wireless sensor networks. TinyOS features a component-based architecture, which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. TinyOSs component library includes network protocols, distributed services, sensor drivers, and data acquisition tools. All of these components can be used as-is or be further refined for a custom application. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage. TinyOS applications are written in nesC [40], a dialect of the C language optimized for the memory limits of sensor networks.

9.2 Implementation and Experiment Overview

We implemented TPSS protocol and CFlood protocol in an application, and particle filters (including VPF and CDPF) in another. TPSS propagates the target information along its trajectory, while CFlood protocol propagates the report messages toward the sink node. Therefore, we may collect the results for them respectively in one application, without influencing each other. Like TPSS, particle filters also propagate the target information along the trajectory. Though they may be implemented in one application, it is difficult to distinguish the experiment result of TPSS from that of particle filters. Thus, we designed a separate program to implement VPF and CDPF.

Except for these two applications, we also developed two assistant programs: target node and data collection. In this section, we will overview performance metrics, experiment steps, two assistant programs, and mote deployment. Then, the design details of the two application programs for TPSS, PFs, and CFlood, will be specified in Sections 9.3 through 9.5.

9.2.1 Performance Measurement

Before discussing the design overview, we first introduce the types of data that we collected from the experiments, i.e., the performance metrics that we utilized to evaluate our algorithms and protocols.

Energy Consumption

From the perspective of measurement on hardware, there are mainly three approaches to measure the energy consumption on a mote:

1) *Battery emulation/simulation.* A DC power source, such as the AA batteries used in TelosB motes, can be emulated or simulated using specific devices [114–116]. These special devices may provide DC power supplies as well as simulating and recording the characteristic curve of a battery. Based on the run-time data, the energy consumption of a device under test may be precisely collected.

2) *On-board power meter.* An on-board power meter is a micro circuit that can be attached to a sensor mote, so as to measure the run-time current. An example of the on-board power meter is the scalable power observation tool (SPOT) developed by University of California, Berkeley [117]. It may transmit the captured energy-related data to external devices via I2C bus [118].

3) *Online multimeter.* Connecting a multimeter to the battery on the mote, we may easily measure the run-time currency and voltage. Then, the corresponding energy consumption can be calculated with physics rules $E = UIt = U^2t/R$, where U, I, and R respectively represent voltage, current, and resistance. Based on its feasibility, the online multimeter approach is the most commonly used one for energy measurement [119–121].

However, all of these hardware-based, real measurement methods have their drawbacks. First, battery emulation/simulation devices are usually very expensive and cumbersome the usage in field experiments. Secondly, the on-board power meter itself is still an academic research effort, thus unavailable in the commercial market. Thirdly, the energy data captured by an online multimeter is hard to be saved in permanent storages, thus it can hardly measure the ever-changing energy consumption. Finally and most importantly, for all the three approaches, it is difficult to attach a measurement device to each of the working mote in the surveillance field.

Given this situation, we take the emulation approach for energy measurement. In our implementation program, we record the number of operations, and the communication data amount or the operating time of each operation. Then, after the experiments, we collect these raw data from motes and saved them on the PC. For a specific operation (e.g., send or receive a packet, or keep active for some time), we calculate the energy consumed based on the configured energy consumption rate of an operation, and its actual operation time or data amount. Usually the energy consumption rate of a mote for a specific operation at a specific status is consistent along the time. As long as we measure the operation time correctly, the calculated result will approach the actual value closely.

Table 9.1 shows the energy consumption rate of TelosB at different status, which is obtained from TelosB's data sheet [39]. In the experiment results, we ignore the energy consumed in the sleep state, as approximately it is only $1\% - 2\%$ of the energy consumed in the active

Table 9.1: TelosB energy consumption rates

Status	Energy consumption rate
Active	5.5 (mJ/s)
Sleep	18.3 ($\mu J/s$)
Transmit	96 (mJ/s)
Receive	74.4 (mJ/s)

state.

Table 9.2: Energy consumption of various messages

Message type	Alarm	Report	HELLO	ABORT
Length (bytes)	100	28	28	10
Transmission time (ms)	3.2	0.896	0.896	0.32
Energy for sending a packet (μJ)	307.2	86.016	86.016	30.72
Energy for receiving a packet (μJ)	238.08	66.6624	66.6624	23.808

In the experiments, the lengths of an alarm message, a report data message, a HELLO message, and an ABORT message are respectively configured as 100, 28, 28, and 10 bytes. Based on 250 kbps data rate of radio [39], the transmission times for various types of packets, as well as the energy consumptions for sending and receiving a packet, are shown in Table 9.2. Then, we only need to record the number of various packets that a mote receives, creates, floods, or aborts. By multiplying them with their respective energy consumption rates on sending or receiving, we will be able to obtain the total energy consumption.

Escape Distance Percentage

We used the escape distance percentage to estimate the tracking performance. The escape distance percentage is defined as the ratio of the length of a target's moving route in which no mote detects it, to the whole route length. A sensor network with the node sleep pattern well scheduled should minimize this percentage to keep the target always in sight.

Number of Packets

We record the number of packets mainly for CFlood. For each mote, we save the number of packets that it receives, creates, floods, and aborts. Based on these numbers, we may calculate the total number of redundant copies of a packet, and the deadline satisfaction ratio. Compared with the energy and the escape distance percentage, the number of packets is the easiest metric to collect.

9.2.2 Experiment Steps

We adopted Cygwin [122] on Windows [123] as our programming environment. The development consists of two steps: TOSSIM [124] simulation and real experiments on TelosB motes.

As mentioned in Chapter 4, we initially simulated TPSS in a self-developed simulation tool programmed in C++. Then for the implementation, we first ported the simulation codes to TOSSIM—a discrete event simulator for TinyOS sensor networks. The TOSSIM framework running on a PC allows users to debug, test, and analyze algorithms in a controlled and repeatable environment.

After making sure the program runs well in TOSSIM, we removed all the TOSSIM related components, compiled it for TelosB motes, and installed it onto real hardware platforms. Downloading a compiled program onto a TelosB mote through a PC USB port includes the following steps:

1) Plug in a mote and identify its COM port—plug a TelosB mote into a PC USB port, start a Cygwin terminal and run the command "motelist.exe". Then the COM number where the TelosB mote is plugged in will be shown in the terminal;

2) Compile and download the application—enter the application directory where Makefile locates in the Cygwin terminal, and run the command "make telosb install,<*node ID*> bs1, <*N*>", where the optional <node ID> sets the node address and <N> is an integer one less than the COM port number assigned to the TelosB;

3) Run the program—unplug the mote and install the batteries. Then the mote is ready for the experiments.

Finally, we deployed the motes in an outdoor parking lot and collected the real experiment data in the field.

9.2.3 Target Emulation and Detection

As there are only three sensing devices on TelosB platform, i.e., light, temperature and humidity, it is very difficult for TelosB motes to detect a physical target, either a mobile vehicle or a person. Given this strictly constrained sensing capability, we adopted an alternative approach to emulate a mobile target: we designed a special TelosB mote as a target, and used its broadcasting messages as its physical behavior that other motes may "sense". For convenience in discussion, we call it the *target node*. The target node broadcasts short messages periodically at every 200 ms. Each time when a mote receives this short message, we consider that it has "detected" a target. For the convenience of the experimental result collection, and without the loss of generality, we assume that the sink mote does not report the target. Instead, it only receives and summarizes the report packets from other motes.

Obviously, the messages broadcast by the target node will influence the application's regular communications, as after all these messages need to consume some bandwidth in the wireless environment. However, this is inevitable without the actual sensing devices. What we can do is to minimize this influence by shortening the message length to the least: the broadcast message only includes the target's position, as we let the target move in a uniform rectilinear route.

During the experiments, we attached the target node to a remotely controlled toy car, so that it can move in the surveillance field along the trajectory that we designed.

The target node program was specifically designed to be run on the target node. Its primary purpose is to broadcast its positions periodically. The position information was *a priori* configured in a header file. Thus, we need to configure a start point and an end point for the broadcast, so that the broadcast messages can match its actual positions in the field experiments. For this purpose, we designed a testing state machine, which flashes LED0 and LED1 sequentially before the broadcast starts. As long as we control the toy car to enter the surveillance field after the flash of LED0 and LED1 is over, the target node's movement will approximately match the content it broadcasts.

Among the performance metrics that we collected from the experiments, the escape distance percentage was the only one that we collected on the target node. Since we did not implement the network wide time synchronization, it is difficult to summarize the overlap of motes' detection intervals. Thus, it is also difficult to observe when the target escapes from the motes. On the contrary, this could be completed easily on the side of the target node, given the following two facts: 1) motes that receive the broadcast messages from the target node always report to the sink node immediately; and 2) the sensing radius of motes (i.e., the communication radius of the target node) r is less than their communication radius R, as introduced in Chapter 3. Based on these facts, the target node will always overhear [106] the reported message by those motes that detect it. If no message is overheard in a short time, the target node may consider that it has escaped from the detection of sensor motes for the current time step. By recording these time steps, the escape distance percentage can be calculated after the target node leaves the surveillance field. To avoid false data by overhearing messages for previous detection events, we added a sequence number for each broadcast message of the target node. Each mote that receives this message thereby presumes a detection is required to add this sequence number in its report messages. In this way, false overhearing can be avoided.

The structure of the target node program is shown in Figure 9.2, in which Timer2 is the control timer of the state machine, Timer3 is the timer for overhearing, and DataCollectorC is the integrated component that is responsible of replying the collection request from the collection node introduced below. This figure was created by running the command "make telosb docs" provided by TinyOS in the application directory.

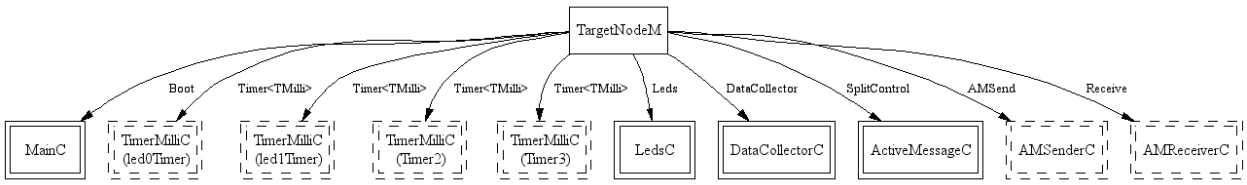


Figure 9.2: Target node program

9.2.4 Data Collection

Whenever a mote detects the "target", it will transmit a report to the sink node, which is how sensor networks work. However, we do not want them to transmit the experiment data to the sink node together with the regular report messages. This is because the experiment data is usually collected for the purpose of testing, thereby is an extra overhead for the application's regular operations. Therefore, we prefer to save the experiment data on each mote temporarily, and collect them later after the experiment.

TelosB mote has three storage spaces: an internal RAM, an internal flash, and an external flash. Among them, the external flash is a permanent (non-volatile) data storage in TinyOS, which allows a mote to persist data even if power is disconnected or the mote is reprogrammed with a new image. However, accessing the external flash is not efficient: the access to ST M25P40 flash used on TelosB is both slow (100 kB/s) and energy consuming (1 $\mu J/byte$) [125]. An experiment result saved on a mote needs to be updated frequently, e.g., at each communication and/or computation operation. Thus, such a frequent access action will spend much time thereby impact the application's regular operation. In addition, it is not necessary to save the experiment result of each operation. Instead, we pay more attention to the final result, e.g., the total energy consumption or the average detection delay. So the data amount will not occupy much storage space. Therefore, we used the internal RAM, i.e. variables in the program, to save the experiment results of each mote. As long as the power is not disconnected, these results will not be lost.

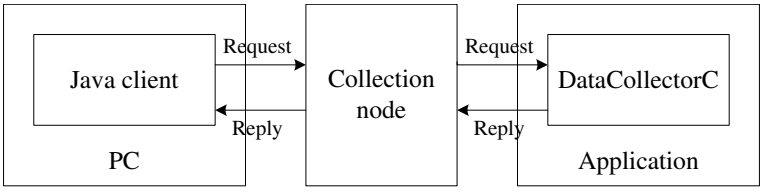


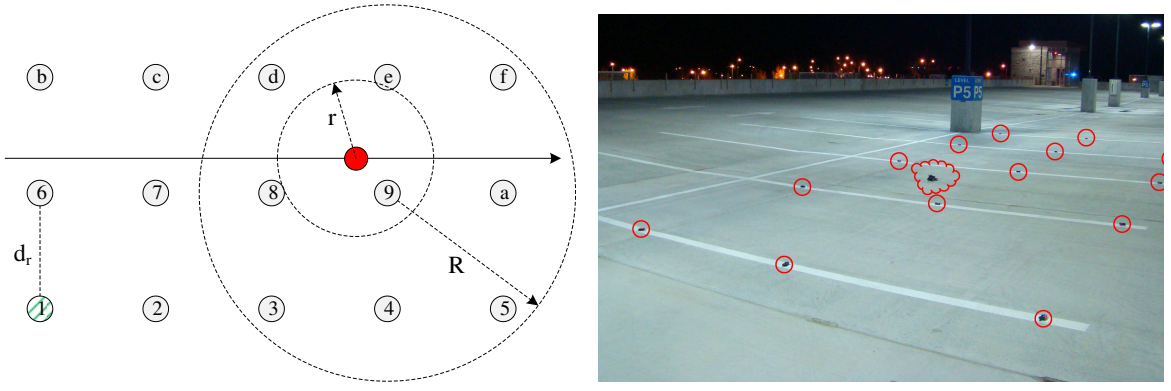
Figure 9.3: Data collection structure

We developed a data collection program, and used it to collect the data from all the motes as well as the target node after the experiment was completed. The program is composed of three parts: a client on PC, a collection node program, and a collection response component. Figure 9.3 shows the structure of this collection system. The client program developed in

Java [126] runs on a PC and communicates with the attached mote via the PC's serial port. It is responsible of receiving user commands, sending requests on demand, displaying and saving the replied results. The collection node program runs on a mote attached to the PC, which we call the *collection node*. It forwards the requests from the serial port to the motes to collect, and the replies from the motes back to the serial port. The collection response component is integrated in each of the motes that may have experiment results to collect. Upon receiving a request, it replies with all the results to the collection node. In this way, we are able to collect the experiment results with less manual effort as well as without impacting the application's regular operations.

9.2.5 Mote Deployment and Parameter Configuration

In an outdoor parking lot, we deployed 15 TelosB motes in a 3×5 grid. As shown in Figure 9.4a, small circles represent the motes, in which the red one is the target node and the slashed one in green serves as the sink node. Numbers in the small circles represent their individual node ID, which is used as their address for data propagation. To make the figure clear, we numbered the motes in the hex base, e.g., a means 10 and f means 15. The medium circle shows the sensing radius of the motes, in which a target may be detected, i.e., the broadcast messages of the target node may be received by other motes. The large circle means the communication radius of motes, i.e., the maximum distance that each pair of motes (excluding the target node) may communicate with each other. d_r is the distance between two adjacent motes in the grid. Finally, the solid line signifies the trajectory of the target, along which we remotely control the toy car to move.



(a) Network topology (b) Snapshot of real deployment

Figure 9.4: Mote deployment

Since we used a mote to emulate the target, the sensing radius of motes is actually the communication radius of the target node. Thus, the configuration of the sensing radius can be completed by configuring the communication radius of the target node. The configuration of TelosB's communication radius is completed by appending a compilation option "CFLAGS

+= -DCC2420_DEF_RFPOWER=<*PL*>" in the application's Makefile, where <*PL*> is an integer signifying the power level. The program running on the target node was specifically designed for it, thus the value of <*PL*> in its Makefile can be configured differently from other motes. Since <*PL*> can only be integers, the configured power levels can only be a series of discrete values.

Based on the node model introduced in Chapter 3 and the discussion above, we approximately configure the sensing radius $r = 1.5\ m$, the communication radius $R = 4\ m$, and $d_r = 2.5\ m$ as shown in Figure 9.4a, i.e., $2r < R$ and $\sqrt{2}d_r < R < 2d_r$. Thus, any mote may communicate with its neighbors on the opposite end of a grid's diagonal, but only very close (four at the most) motes may detect the target. This is consistent with our settings in the simulation, and the real configuration of commonly used mote platforms [98]. In addition, $r = 1.5\ m$ and $R = 4\ m$ correspond to $PL = 2$ and $PL = 3$ respectively.

We configured the target speed as $v = 1\ m/s$. Thus, the target node program will broadcast $4 * d_r * 1000/(v * 200) = 50$ messages showing its positions.

Figure 9.4b is a snapshot of the real deployment. The configured d_r value made it easy to align the motes along the borders of parking spaces. Since $d_r = 2.5\ m$, the 3×5 grid occupied $(3-1) * d_r \times (5-1) * d_r = 5 \times 10\ m^2$.

9.2.6 Assistant Components

For the convenience during the experiments, we utilized the three LEDs on TelosB mote to show the working status of a mote: 1) the red LED (LED0) is a sign of working program, which flashes at a frequency of 2 Hz as long as the downloaded program is running on the mote; 2) the green LED (LED1) is a sign of detection, which flashes once when a target is detected; and 3) the blue LED (LED2) is a sign of communication, which flashes whenever sending or receiving a message. With these three LEDs, we may easily observe the overall status of motes.

Timers are usually used to control the on/off of LEDs and the state machines in the program.

9.3 Sleep Scheduling

We implemented TPSS, the sleep scheduling protocol for single target tracking. The reason that we did not implement SSMTT—the sleep scheduling algorithm for multiple target tracking—is explained as follows. In our experiment environment, motes were deployed sparsely given the limited number of motes. Thus, the number of awakened motes at each period of proactive wake-up is small. When multiple targets intersect in the surveillance field, it is highly likely that the awakened motes for one target will not overlap with those for the other. Therefore, it is very difficult to leverage the alarm message of each other.

In this section, we first introduce the program design, then compare TPSS with competing protocols based on the experiment results.

9.3.1 Program Design

As in the simulation, we implemented TPSS as well as all of its competing protocols including NOSS, Circle [8], and MCTA [13]. The primary difference among the four protocols is how they choose the proactively awakened motes: 1) NOSS does not awaken neighbor nodes proactively, and always follows the default sleep pattern; 2) Circle awakens all the neighbor nodes in one hop of the alarm node; 3) MCTA awakens all the neighbor nodes in the expected tracking contour; and 4) TPSS chooses the awakened nodes based on kinematics and probabilistic prediction. In the following discussion, we use TPSS as the example to illustrate our implementation whenever there is no ambiguity.

TPSS protocol was implemented in the structure shown in Figure 9.5. The target sensor component receives the broadcast message from the target node, and signals an event of detection. The communication module is responsible for communicating with neighbor motes. Once a target is "detected", its potential movement is first predicted by the target predictor module. Then, the alarm node elector module elects a mote to broadcast the alarm message. When such an alarm message is received, the awakened node selector module decides if this mote should be awakened proactively. Besides these modules, we designed a few supporting modules, including local clock, timer, LED control, and random number generator. Each of these modules serves for a specific function, while TPSS protocol module controls the operation of TPSS and its cooperation with peripheral function modules.

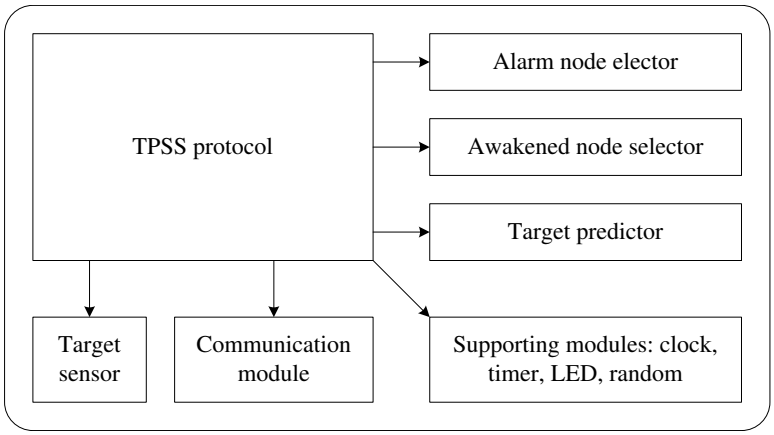


Figure 9.5: TPSS protocol structure

Among these modules, four competing protocols are only distinguished with different efforts in the awakened node selector module and TPSS protocol module.

In TPSS protocol control module, we designed an internal finite state machine as shown in Figure 9.6.

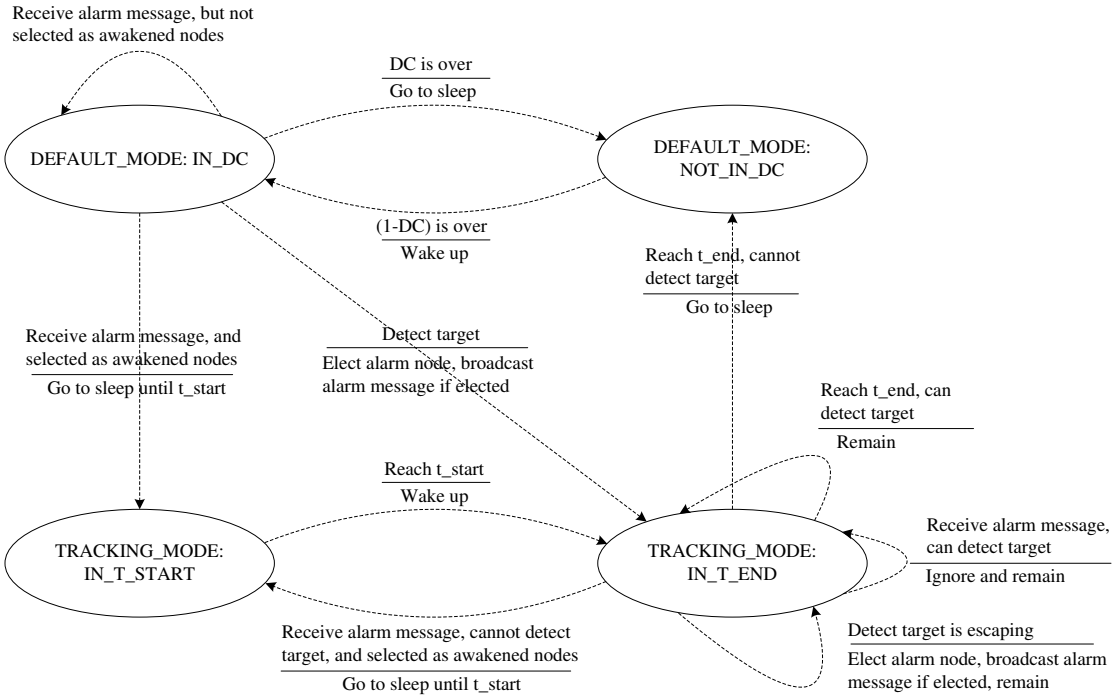


Figure 9.6: TPSS protocol finite state machine

The state machine consists of four states:

1) Default mode/IN_DC: the sleep pattern is not specifically scheduled, and the mote is active;

2) Default mode/NOT_IN_DC: the sleep pattern is not specifically scheduled, and the mote is sleeping;

3) Tracking mode/IN_T_START: the sleep pattern is scheduled for an approaching target, and the mote is scheduled to sleep until t_{start}; and

4) Tracking mode/IN_T_END: the sleep pattern is scheduled for an approaching target, and the mote is scheduled to wake up from t_{start} until t_{end}.

In Figure 9.6, the transitions between pairs of states are shown as fractions: with the transition condition as numerators, and the action taken along the transition as denominators. Each time a service of a functional module is needed, TPSS protocol module will call a command provided by that module, and possibly handle the events triggered by the command call.

We configured the default toggling period as $TP = 1\ s$ and the default duty cycle as $DC = 0.2$. In the default mode, a mote transits its state between Default mode/IN_DC and Default

mode/NOT_IN_DC, which is controlled by a timer based on the configured TP and DC.

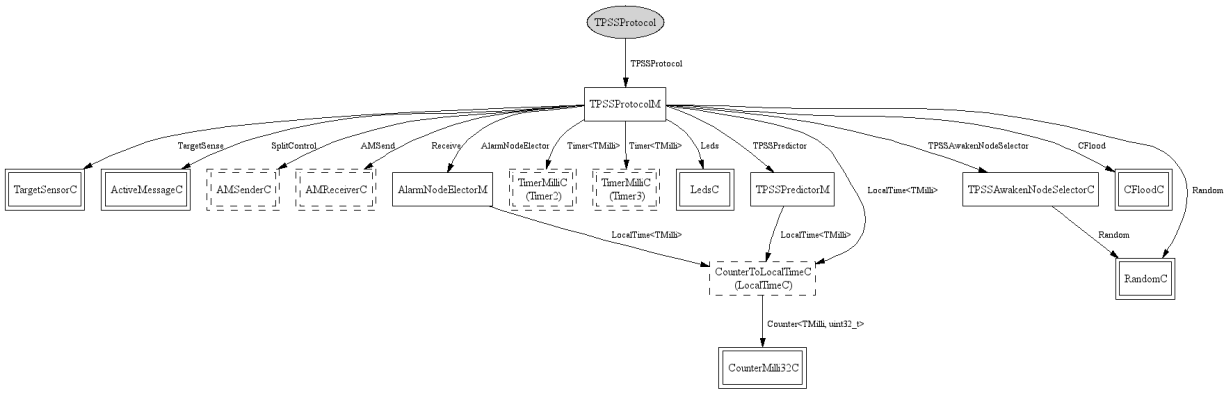


Figure 9.7: TPSS protocol implementation

Figure 9.7 shows the detailed program structure of TPSS. Like the target node program, this figure was also created with the command "make telosb docs" provided by TinyOS.

9.3.2 Experiment Results

Based on the energy consuming rates shown in Table 9.1, we calculate the extra energy consumption (EE) as the performance metric for energy efficiency in the real experiments. As discussed previously in Chapter 3, the detection delays are difficult to measure without time synchronization in the real deployment. Instead, we utilize the escape distance percentage, i.e. EDP, to evaluate the tracking performance. EDP is obtained using the overhearing method described in Section 9.2.3.

For each protocol under testing, the experiment was repeated five times, and we calculated the sample means and confidence intervals as the final result.

Table 9.3: TPSS experiment results

Protocol	NOSS	Circle	MCTA	TPSS
Energy for alarm messages	0	8.01	7.92	8.13
Energy for scheduled wake-up	0	74.1	38.6	33
Total EE (mJ)	0	82.11 (78.63–85.59)	49.52 (45.57–53.47)	41.13 (38.59–43.67)
EDP (%)	84 (79.14–88.86)	14 (12.65–15.35)	19.6 (18.00–21.20)	20.4 (18.80–21.99)

Since we deployed the network with a single configuration of mote density/topology/position and target movement, we present the final result in Table 9.3 instead of plots. Assuming that the samples are normally distributed, the values in the parenthesis in the table are 90% confidence intervals obtained from the samples.

From the table, we may observe:

1) Compared with Circle scheme, TPSS improves energy efficiency by 49.9%, with the cost of a 45.7% increase in the escape distance percentage.

2) Compared with MCTA algorithm, TPSS improves energy efficiency by 16.9%, with the cost of a 4.1% increase in the escape distance percentage.

3) In the extra energy consumed for sleep scheduling, the energy for alarm message communication takes a small part, and most of the energy is consumed for keeping motes active to wait for the approaching target. Just because TPSS reduces the number of awakened motes and their active time, energy efficiency is improved significantly.

4) Though TPSS introduces a large relative value on tracking performance loss, the absolute loss is not significant.

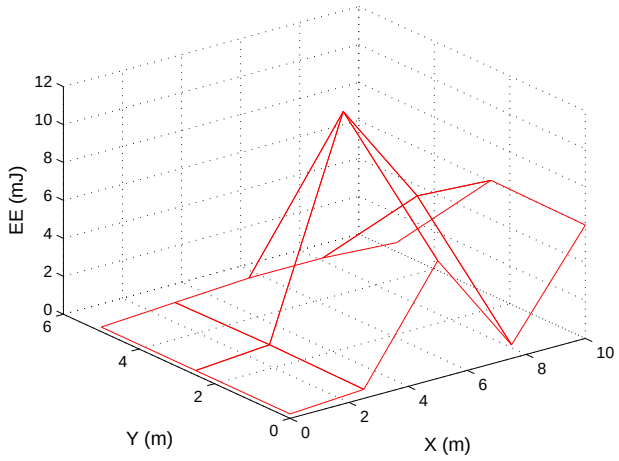


Figure 9.8: Extra energy of motes with TPSS

In Figure 9.8, we plot the extra energy consumption of individual motes to show the energy consumption distribution. The energy data was obtained from one sample of our experiments. Notice that the extra energy includes the energy consumed for sleep scheduling only. Since the sink node (i.e., mote 1) neither sends alarm messages nor was sleep-scheduled, its extra energy was found zero.

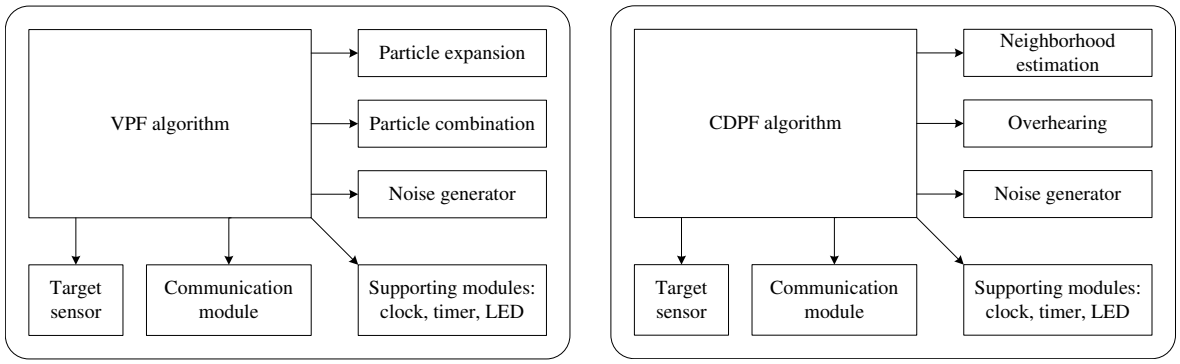
9.4 Particle Filters

We implemented VPF, CDPF and their respective competing algorithms, i.e., RPF [22] and SDPF [27].

9.4.1 Program Design

VPF algorithm was implemented in the structure shown in Figure 9.9a. Since particle filtering methods depend on the actual observation to estimate the target status, and the sensing device is not available on our TelosB motes, we use a noise generator module to emulate the observation: for each detected target, we append a noise to the received target position, so that the target positions detected by different motes are different.

VPF is a centralized particle filtering method. Thus, all the observations are propagated to the sink node. On the base station, particles are combined or expanded by particle combination module and particle expansion module respectively. Then, the estimation will be made with these particles on the base station.



(a) VPF algorithm implementation structure (b) CDPF algorithm implementation structure

Figure 9.9: VPF and CDPF algorithm implementation

CDPF algorithm was implemented in the structure shown in Figure 9.9b. The primary difference of CDPF from VPF is embodied in neighborhood estimation and overhearing modules. We implemented the neighborhood estimation module based on CFlood protocol's HELLO exchange. The overhearing module is responsible for collecting the particle data broadcast by motes. Unlike VPF, CDPF is a distributed implementation of particle filters. Therefore, both neighborhood estimation and overhearing modules are implemented on every mote.

Like TPSS, both VPF and CDPF integrate target sensor module, communication module, and supporting modules. Furthermore, both VPF and CDPF execute the algorithm in an iterative manner. In each iteration, the operations are executed in a sequential order.

Therefore, there is no complicated state change for the motes. We do not have to design the finite state machine like for TPSS.

Since VPF propagates particle information to the sink node and CDPF propagates it along the target trajectory, just like the difference between TPSS and CFlood, we combine them in a single application.

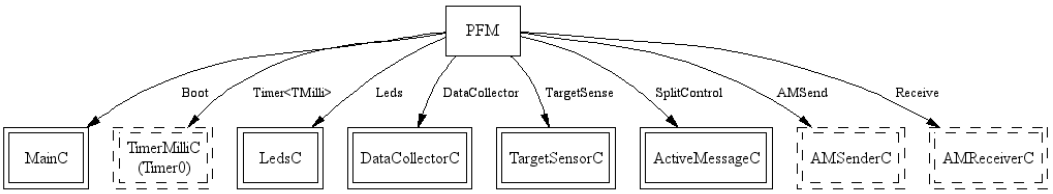


Figure 9.10: VPF and CDPF implementation

Figure 9.10 shows the detailed program structure of VPF and CDPF. Among all the modules of VPF and CDPF, target sensor module, communication module, and the supporting module were implemented as separate ones. As introduced previously, particle combination module and particle expansion module were implemented on the base station using Java language. In addition, neighborhood estimation and overhearing modules of CDPF, and noise generator were integrated in the core algorithm.

We configured the broadcasting frequency of the target node as 500 ms. Thus, each particle filter includes 20 iterations. Other parameters specific to the algorithm will be detailed in the following sections.

9.4.2 VPF Experiment Results

Unlike the bearings-only tracking problem and the four-dimensional state vector used in the simulation, we study two-dimensional state vector to simplify the implementation. Therefore, the cell partition can be simplified to a grid partition of the two-dimensional plane.

In the experiment, the cell size was configured as $d = 0.5\ m$. Thus, the maximum number of particles for VPF will be $5 \times 10/0.5^2 = 200$.

On the PC, we collect the number of particles, computation time, and RMSE from the Java program. For each protocol under testing, the experiment was repeated five times, and we calculated the sample means and confidence intervals as the final result as shown in Table 9.4. Assuming that the samples are normally distributed, the values in the parenthesis in the table are 90% confidence intervals obtained from the samples.

From the table, we may observe:

1) Compared with RPF, VPF introduces an increase of 50.6% on RMSE, but reduces the

Table 9.4: VPF experiment result

	RPF	VPF
RMSE	0.154 (0.150–0.158)	0.232 (0.226–0.238)
Average N_s	1000	25.12 (23.26–26.97)
Average execution time per iteration	7.33 ms (7.29–7.37)	2.04 ms (1.98–2.10)

computation time by 72.2%.

2) The number of particles remains in the same order as the number of cells in the sensing area of a mote.

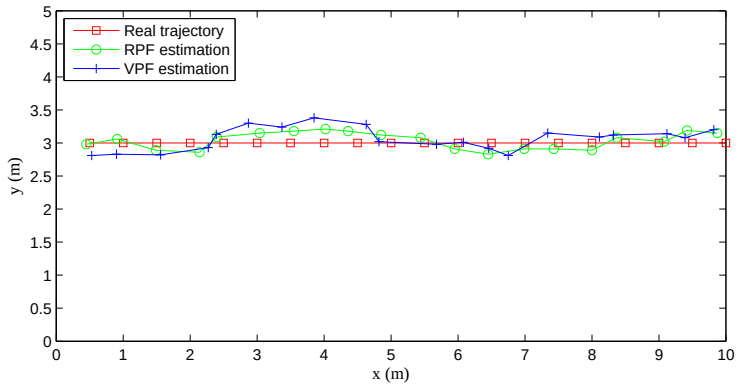


Figure 9.11: RMSEs of RPF and VPF

We show the estimation at an experiment sample in Figure 9.11, and the change of the number of particles in Figure 9.12. At the beginning of the experiment, we configured the number of particles of VPF as the maximum number, i.e., 200. Once the target is detected and observations are available, the number of particles decreased immediately to the level of the number of cells in a mote's sensing area. This particle reduction effort alleviated the computation significantly.

9.4.3 CDPF Experiment Results

CDPF is a completely distributed implementation of particle filtering method. Thus, we executed the program on the motes only. Moreover, we only recorded and collected the number of packets for propagating the particles along the trajectory, which has no conflict with the communication of VPF.

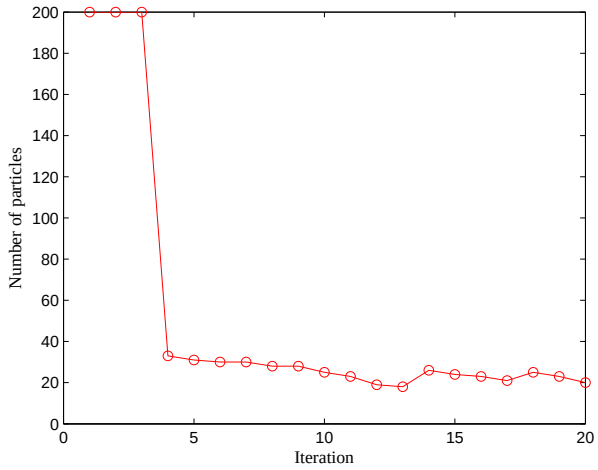


Figure 9.12: Change of VPF's number of particles

Table 9.5: CDPF experiment result

	CPF	SDPF	CDPF	CDPF-NE
RMSE	0.158 (0.155–0.161)	0.364 (0.357–0.371)	0.413 (0.402–0.424)	0.675 (0.662–0.688)
Average N_s	1000	12.9 (12.3–13.5)	1.67 (1.63–1.71)	1.67 (1.63–1.71)
Average communicated cost (bytes)	3.49 (3.40–3.58)	46.7 (45.1–48.3)	5.13 (5.02–5.24)	4.94 (4.82–5.06)

In the experiment, we configured the number of particles of CPF as 1000. For the competing algorithm SDPF, we configured the central unit for weight aggregation as mote 6 in the first half phase and mote 8 in the second half phase. In this way, weight aggregation can be completed at the closest mote for every mote that holds particles, which is the best situation for SDPF.

After the experiment, we collect the total length of communicated packets and the estimation error (i.e., RMSE) from each mote in the network. As every mote completes the filtering process independently, there may exist many RMSEs for each iteration. We choose the maximum one as the estimation error for that iteration, and calculate the average value of these maximum RMSEs. The experiment result is listed in Table 9.5. Assuming that the samples are normally distributed, the values in the parenthesis in the table are 90% confidence intervals obtained from the samples.

From the table, we may observe:

1) Compared with SDPF, CDPF and CDPF-NE introduce an increase of 13.5% and 85.4% on RMSE, but reduces the communication cost by 89.0% and 89.4% respectively.

2) Like in the simulation, CPF shows the least communication cost. This is because in a small network like our experiment environment, any mote can propagate the weights to the sink node within three hops at the most. Thus, the hop count factor in the communication cost of CPF is not dominant. At the same time, the particle data for CPF is stored on the base station, thus does not require to propagate. However, this situation will change as the network scale increases.

3) CDPF-NE has the lease communication cost in the distributed implementations, although it introduces the greatest estimation error.

4) CDPF and CDPF-NE introduce the least number of particles, so that the problem scale can be minimized.

From the experiment result, it seems that the decrease of CDPF-NE's communication cost is not worthy of its increase on the estimation error. However, we are not discussing the tradeoff or aiming at optimizing the tradeoff. By introducing CDPF-NE, we provide an option to minimize the communication cost of distributed particle filters. It can be used when the communication cost is critical, while the estimation error does not matter much.

9.5 Constrained Flooding

We implemented CFlood protocol together with TPSS protocol for the following two reasons:

1) The purpose of TPSS protocol is to schedule motes for local tracking, and CFlood protocol is designed for improving energy efficiency of data propagation process. They can work together in real deployments.

2) In the implementation, their results to be collected have no overlap.

Thus, CFlood module is integrated in Figure 9.7, where TPSS protocol module leverages the flooding services provided by CFlood module. Each time a mote detects a target, it floods the report out to the primary recipient and a few secondary recipients determined by CFlood.

9.5.1 Program Design

In the experiments, we compared CFlood and CFlood-DA with Mint routing [45] and DFP [46]. Compared with the simulation in Chapter 7, we did not implement the MCMP protocol [30], because its performance and behavior is very different from CFlood.

CFlood protocol was implemented in the structure shown in Figure 9.13. The HELLO

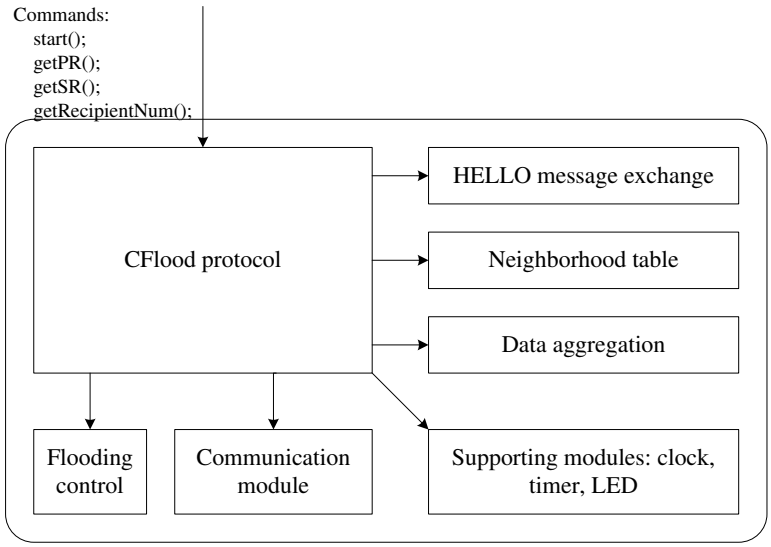


Figure 9.13: CFlood protocol structure

message exchange module is responsible for managing the neighborhood table and estimating the transmission delay between each pair of motes. The flooding control module was implemented by modifying the TKN15.4 MAC protocol, which is an IEEE 802.15.4 MAC Implementation for TinyOS. Data aggregation module aggregates packets when waiting for the ABORT message, whenever possible. Finally, CFlood protocol module provides an interface for the upper layer applications to search routing/flooding information.

In CFlood protocol module, we designed an internal finite state machine as shown in Figure 9.14.

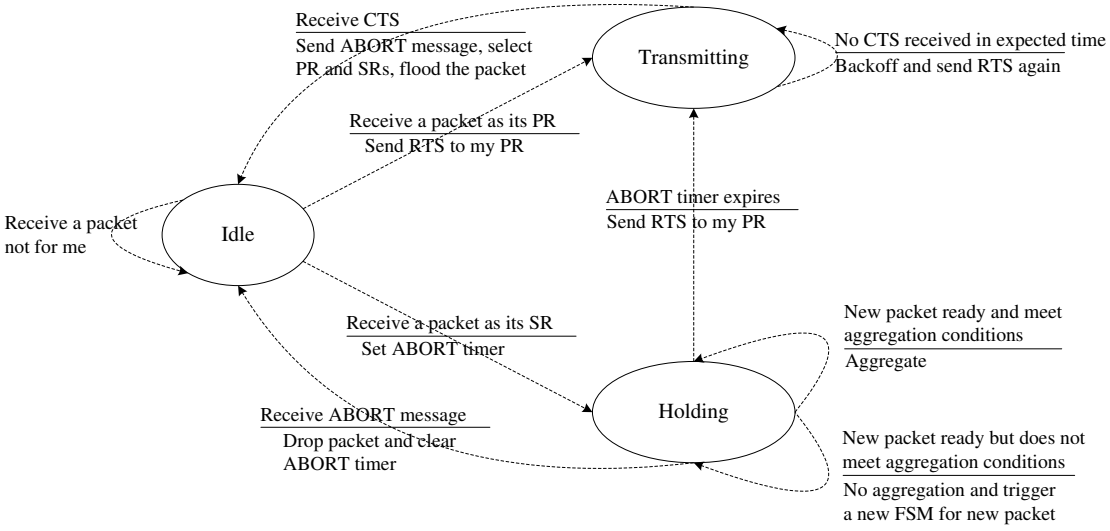


Figure 9.14: CFlood protocol finite state machine

The state machine consists of three states:

1) Idle: the mote is waiting for incoming packets;

2) Transmitting: the mote has sent RTS request, and is waiting for the CTS reply;

3) Holding: the mote is holding a packet for potential flooding, but it may be aborted on receiving an ABORT message.

Like TPSS protocol, the transitions between pairs of states are shown as fractions in Figure 9.14: with the transition condition as numerators, and the action taken along the transition as denominators.

9.5.2 Experiment Results

Based on the energy consuming rates shown in Table 9.1 and numbers of all types of packets collected from motes, we calculate the deadline satisfaction ratio (DSR), average energy consumption (e), and real-time capacity (δ).

For each protocol under testing, the experiment was repeated five times. The final result in Table 9.6. Assuming that the samples are normally distributed, the values in the parenthesis in the table are 90% confidence intervals obtained from the samples.

Table 9.6: CFlood experiment results

Protocol	MR	DFP	CFlood	CFlood-DA
DSR (%)	27.8 (27.3–28.3)	78.1 (77.3–78.9)	87.2 (86.3–88.1)	88.0 (87.1–88.9)
e (mJ)	90.3 (89.1–91.5)	184.1 (182.5–185.7)	190.6 (188.9–192.3)	172.0 (170.4–173.6)
δ (%)	30.79	42.42	45.75	51.16

From the table, we may observe:

1) CFlood and CFlood-DA improve the deadline satisfaction ratio significantly, and introduce little increase on the energy consumption. Therefore, they perform the best, especially CFlood-DA, in terms of real-time capacity per unit energy consumption, i.e., the criterion for energy efficiency.

2) Except for conserving more energy, the effort of data aggregation also increases DSR a little, as it alleviates the congestion around the sink node.

Chapter 10

Conclusion and Future Work

This dissertation was motivated by the energy efficiency challenge of sensor networks due to their constrained resources. We studied energy efficiency of target tracking in multiple forms and shapes. Specifically, we solved three dissertation problems: prediction-based sleep scheduling, efficient particle filtering, and constrained flooding.

In a duty-cycled sensor network, proactive wake-up and sleep scheduling can create a local active environment to provide guarantee for the tracking performance. By carefully and effectively limiting the scope of this local active environment, we can improve the energy efficiency with an acceptable loss on the tracking performance. The design of TPSS protocol and SSMTT algorithm shows that this purpose is achievable based on target prediction, and it is possible to precisely sleep-schedule nodes without involving much physics.

Our prediction method does not make any assumptions on the movement pattern of targets. For example, the prediction method does not care if the target is a vehicle or a person, fast or slow, moves stably or full of uncertainty such as abrupt turnarounds. This helps to improve the energy efficiency by constraining the awake region to the least, as well as generalize the application of TPSS and SSMTT. At the same time, however, this prediction method cannot cover those special cases, e.g., the sudden turning around of the target. In such a case, the target is highly likely to move out of the local active environment scheduled by proactive wake-up and sleep scheduling. Then, it may be detected as a new target, after a relatively long delay, as if it first enters the surveillance field. Though considering the movement pattern of targets in the design is helpful for reducing the loss on the tracking performance, it depends on specific applications with specific purposes. In addition, the energy efficiency may also be negatively influenced.

For particle filtering problems, we learned from the dissertation that the quantitative relation between the number of particles and the estimation error is still not clear. This is because that particle filtering is a numerical method based on statistics, the result of which is difficult to quantify. In the design of VPF, the vector space partition method provides a bound to

the estimation error. Moreover, the similar result can be applied to CDPF, i.e., bounding the estimation error with the maximum distance among nodes. However, these bounds are still loose, because they are obtained from binary sensor network [64] which is the simplest type of WSNs. By studying the connection between the estimation error and the sensing capabilities of nodes, we can further tighten these bounds.

The design of CDPF also shows that certain conditions specific to wireless networks (e.g., the overhearing effect [106]) can facilitate the application of PFs in WSNs. However, due to the complexity of PFs (including the significant computational/communication complexity and the aggregation delay), the application of PFs in WSNs still needs intensive research.

Based on the design of CFlood protocol, we notice that modifying the underlying MAC protocol enables nodes to estimate the result of the following transmission at an early stage. Though this cross-layer design is sometimes limited by certain requirements of the applications, it can provide the researchers space for improving MAC protocols. In addition, local short message exchange, e.g., HELLO messages used by CFlood, not only enables routing information propagation, but also helps local knowledge share. Besides, it can be used for round-trip delay estimation and local time synchronization.

Another observation from constrained flooding problem is, for real-time data propagation, the effect of data aggregation depends on the end-to-end deadline. A loose deadline will increase the successful probability of data aggregation, thereby improve the flooding efficiency.

We implemented most of the solutions and conducted field experiments. Though the emulation is sometimes unavoidable, the implementation can still provide more real and convincing results than the simulation. For example, motes work in a real deployment, thus they will experience real environmental noises and unstable links. In addition, the implementation itself can verify the rationality of the solutions, and the feasibility of applying them into the constrained resources of actual mote hardware platforms.

In the implementation, we have to carefully consider more real-life problems than those in the simulation, e.g., experimental data collection. The operations specifically for testing and experiments should not influence the regular operations of sensor networks, because otherwise the experimental results will be less reliable. Therefore, the influence of experimental operations on regular operations should be minimized. For this purpose, we designed the data collection node to collect experimental results after the experiments. But it is difficult to avoid this influence completely. For example, the broadcast of the target node will occupy the wireless channel, so that the transmission of report messages toward the sink node may be delayed.

We also learned from the implementation that compared to the simulation, changing network configuration (e.g. node density) is more difficult in the implementation. Besides the significant workload, another reason is that the transmission power level of motes' RF radio can only be configured as a series of discrete integer values, i.e., the communication radius cannot be configured to any number. Therefore, simply re-deploying nodes in a different

density, even in the same topology, may introduce a different connection status of nodes. Comparing the experimental results obtained from these different connection status will be less helpful.

Next, we review the contributions, discuss the dissertation's limitations, and finally present the potential future work.

10.1 Summary of Contributions

For sleep scheduling, we presented a prediction-based sleep scheduling protocol (i.e., TPSS) and its expansion for multiple target tracking (i.e., SSMTT). The target prediction scheme, which we developed based on both kinematics rules and theory of probability, not only predicts a target's next location, but also describes the probabilities with which it moves along different directions. Based on the prediction results, we improve the energy efficiency of proactive wake-up by reducing the number of awakened nodes and scheduling their sleep patterns to shorten the active time in an integrated manner. In addition, SSMTT leverages the redundant alarm messages of interfering targets to further improve the energy efficiency.

For trajectory estimation, we designed two particle filters, i.e. VPF and CDPF, to improve the energy efficiency and facilitate the application of particle filters in sensor networks. VPF reduces the problem size of particle filtering by reducing the number of particles, and CDPF provides a completely distributed implementation of PFs. Both of them enhance energy efficiency significantly, either by simplifying the computation or by reducing the communication cost. To the best of our knowledge, our CDPF is the first ever implementation of a completely distributed particle filter.

For constrained flooding, we developed a constrained flooding protocol (i.e., CFlood) to improve the flooding efficiency by aborting the flooding that has already occurred and utilizing unicasting whenever possible. Based on a simple modification to RTS/CTS handshaking used in MAC protocols, we enable nodes to know at an early stage if unicasting can satisfy the real-time constraint or not. Data aggregation was also used as an auxiliary effort to enhance the flooding efficiency further.

Another very important contribution of this dissertation is that we implemented a prototype for most of the dissertation solutions and conducted field experiments. By successfully implementing the designs and running them on actual hardware platforms of motes, we verified the rationality of the dissertation's solutions, and the feasibility of applying our designs onto the constrained resources of actual motes. Most importantly, our implementation-based experiment results, which are more convincing than the simulation-based experiments, showed similar results as those in the simulation. This significantly strengthened the dissertation's contributions.

10.2 Limitations of Dissertation

Except for the strengths previously discussed, the research in this dissertation has a few limitations as well, which are the foundations of the potential future work.

First, the sleep scheduling solution in this dissertation does not impose performance constraints when reducing the energy consumption. Instead, it is an experiment-based effort. Without performance constraints, it is difficult to quantify the quality of the solutions: on one hand, if the requirements on tracking performance are loose enough, energy efficiency may be improved further and the current effort could be more efficient; on the other hand, it is also possible that the enhancement on energy efficiency of the current effort has already exceeded the tolerance of a system on the performance loss, which will make the current solution less helpful.

Secondly, both VPF and CDPF introduce a higher estimation error than their respective competitors, which dilutes their achievements on enhancing energy efficiency. Though the estimation error cannot be anticipated based on the numerical method of PFs, we provided bounds for that of VPF as the cell size d. The similar bound can be obtained for CDPF as the maximum distance among nodes, which is related to the deployment density.

Thirdly, the verification method for deadline satisfaction used by CFlood protocol is based on an end-to-end deadline partition method that we designed in an intuitive manner. In fact, the end-to-end deadline partition method has never been fully studied for sensor networks. All the existing methods [71,111] were built on simple mathematical models without considering the actual network requirements. Compared with these existing methods, our partition method considers the actual network throughput, therefore is more realistic. However, the method is still at its early stage.

Finally, the current dissertation solutions do not use optimization methods. The optimization of these problems is difficult, especially for: 1) prediction-based sleep scheduling, for which the optimization will involve many physics problems, and these are out of this dissertation's scope; and 2) particle filtering, the performance of which is difficult to optimize because it is a numerical method based on statistics.

10.3 Future Work

Based on the current research results and limitations, we propose the following problems for future research:

- *Performance-based sleep scheduling.* Since it is difficult to quantify the quality without performance constraints, adding performance constraints to the sleep scheduling protocol will make it more helpful for network configuration and performance optimization.

- *Delay of particle filtering methods.* Besides reducing the estimation error, another promising research problem for particle filters is around the significant delays that both centralized PFs and distributed PFs experience: 1) centralized PFs concentrate particle weights in a central server for normalization and resampling, while such a convergecast traffic will cause severe congestion around the central server thereby produce delays; 2) distributed PFs propagate particles as well as their weights along the trajectory or throughout the network, but broadcasting in a local neighborhood has to be completed in a sequential order thereby results in heavy delays. It will be very helpful if these delays could be reduced.

- *End-to-end deadline partition method.* Based on a specific network topology and given constraints such as link quality, it is possible to estimate a model of network data streams through experiments. Then, we may design an end-to-end deadline partition method, which is most suitable for the real network environments.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 38, no. 4, pp. 393–422, 2002.

[2] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless microsensor network models," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 2, pp. 28–36, 2002.

[3] "Wireless sensor network." [Online]. Available: http://en.wikipedia.org/wiki/Wireless_sensor_network

[4] Q. Cao, T. Yan, J. Stankovic, and T. Abdelzaher, "Analysis of target detection performance for wireless sensor networks," in *Intl Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2005, pp. 276–292.

[5] G. Wittenburg, N. Dziengel, C. Wartenburger, and J. Schiller, "A system for distributed event detection in wireless sensor networks," in *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. New York, NY, USA: ACM, 2010, pp. 94–104.

[6] T. He, P. Vicaire, T. Yan, L. Luo, L. Gu, G. Zhou, R. Stoleru, Q. Cao, J. A. Stankovic, and T. Abdelzaher, "Achieving real-time target tracking using wireless sensor networks," in *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 37–48.

[7] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare event detection," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, p. 4.

[8] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking*, 2004, pp. 129–143.

[9] P. Li, B. Ravindran, J. Wang, and G. Konowicz, "Choir: A real-time middleware architecture supporting benefit-based proactive resource allocation," *Object-Oriented*

Real-Time Distributed Computing, IEEE International Symposium on, vol. 0, p. 292, 2003.

[10] S. Pattem, S. Poduri, and B. Krishnamachari, "Energy-quality tradeoffs for target tracking in wireless sensor networks," in *In Proceedings of IPSN03*, 2003, pp. 32–46.

[11] M. Ding and X. Cheng, "Fault tolerant target tracking in sensor networks," in *MobiHoc '09: Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing.* New York, NY, USA: ACM, 2009, pp. 125–134.

[12] D. Tian and N. D. Georganas, "A coverage-preserving node scheduling scheme for large wireless sensor networks," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 2002, pp. 32–41.

[13] J. Jeong, T. Hwang, T. He, and D. Du, "Mcta: Target tracking algorithm based on minimal contour in wireless sensor networks," in *INFOCOM*, 2007, pp. 2371–2375.

[14] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay efficient sleep scheduling in wireless sensor networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4, March 2005, pp. 2470–2481.

[15] Y. Gu and T. He, "Data forwarding in extremely low duty-cycle sensor networks with unreliable communication links," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, 2007, pp. 321–334.

[16] Y. Wu, S. Fahmy, and N. Shroff, "Energy efficient sleep/wake scheduling for multi-hop sensor networks: Non-convexity and approximation algorithm," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007, pp. 1568–1576.

[17] X. Wang, J.-J. Ma, S. Wang, and D.-W. Bi, "Cluster-based dynamic energy management for collaborative target tracking in wireless sensor networks," *Sensors*, vol. 7, pp. 1193–1215, 2007.

[18] J. Fuemmeler and V. Veeravalli, "Smart sleeping policies for energy efficient tracking in sensor networks," *Signal Processing, IEEE Transactions on*, vol. 56, no. 5, pp. 2091–2101, May 2008.

[19] X. Wang, J.-J. Ma, S. Wang, and D.-W. Bi, "Prediction-based dynamic energy management in wireless sensor networks," *Sensors*, vol. 7, no. 3, pp. 251–266, 2007.

[20] Y. Xu, J. Winter, and W.-C. Lee, "Prediction-based strategies for energy saving in object tracking sensor networks," in *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*, 2004, pp. 346–357.

[21] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, 2001.

[22] A. Doucet, N. De Freitas, and N. Gordon, Eds., *Sequential Monte Carlo methods in practice*. New York, USA: Springer, 2001.

[23] M. Coates, "Distributed particle filters for sensor networks," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*. New York, NY, USA: ACM, 2004, pp. 99–107.

[24] W. R. Gilks and C. Berzuini, "Following a moving target-monte carlo inference for dynamic bayesian models," *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, vol. 63, no. 1, pp. 127–146, 2001.

[25] X. Sheng, Y.-H. Hu, and P. Ramanathan, "Distributed particle filter with gmm approximation for multiple targets localization and tracking in wireless sensor network," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 24.

[26] Y. Huang, W. Liang, H.-b. Yu, and Y. Xiao, "Target tracking based on a distributed particle filter in underwater sensor networks," *Wirel. Commun. Mob. Comput.*, vol. 8, no. 8, pp. 1023–1033, 2008.

[27] M. Coates and G. Ing, "Sensor network particle filters: motes as particles," in *IEEE Workshop on Statistical Signal Processing*, 2005, pp. 1152–1157.

[28] X. Wang, J. Ma, S. Wang, and D. Bi, "Distributed energy optimization for target tracking in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 73–86, 2010.

[29] H.-Q. Liu, H.-C. So, F. K. W. Chan, and K. W. K. Lui, "Distributed particle filter for target tracking in sensor networks," *Progress In Electromagnetics Research*, vol. 11, pp. 171–182, 2009.

[30] X. Huang and Y. Fang, "Multiconstrained qos multipath routing in wireless sensor networks," *Wirel. Netw.*, vol. 14, no. 4, pp. 465–478, 2008.

[31] Y. M. Lu and V. W. S. Wong, "An energy-efficient multipath routing protocol for wireless sensor networks: Research articles," *Int. J. Commun. Syst.*, vol. 20, no. 7, pp. 747–766, 2007.

[32] S. De, C. Qiao, and H. Wu, "Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks," *Computer Networks*, vol. 43, no. 4, pp. 481–497, 2003.

[33] J. S. Silva, T. Camilo, P. Pinto, R. R. A. Rodrigues, F. Gaudncio, and F. Boavida, "Multicast and ip multicast support in wireless sensor networks," in *Networks*, 2008, pp. 19–26.

[34] B. Deb, S. Bhatnagar, and B. Nath, "Information assurance in sensor networks," in *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 2003, pp. 160–168.

[35] Y. Zhang and M. Fromherz, "Constrained flooding: A robust and efficient routing framework for wireless sensor networks," in *AINA: Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, 2006, pp. 387–392.

[36] F. Lu, L.-T. Chia, K.-L. Tay, and W.-H. Chong, "Nbgossip: An energy-efficient gossip algorithm for wireless sensor networks," *Journal of Computer Science and Technology*, vol. 23, no. 3, pp. 426–437, 2008.

[37] R. Rajagopalan and P. K. Varshney, "Data aggregation techniques in sensor networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 8, pp. 48–63, 2006.

[38] J. Benson, T. O'Donovan, U. Roedig, and C. J. Sreenan, "Opportunistic aggregation over duty cycled communications in wireless sensor networks," in *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, 2008, pp. 307–318.

[39] CrossBow, "Telosb data sheet," http://www.willow.co.uk/TelosB_Datasheet.pdf.

[40] "nesc programming language." [Online]. Available: http://nescc.sourceforge.net/

[41] "Tinyos." [Online]. Available: http://www.tinyos.net/

[42] B. Jiang, K. Han, B. Ravindran, and H. Cho, "Energy efficient sleep scheduling based on moving directions in target tracking sensor network," in *IPDPS*, 2008, pp. 1–10.

[43] B. Jiang, B. Ravindran, and H. Cho, "Energy efficient sleep scheduling in sensor networks for multiple target tracking," in *Distributed Computing in Sensor Systems*. Springer Berlin / Heidelberg, 2008, vol. 5067, pp. 498–509.

[44] ——, "Cflood: A constrained flooding protocol for real-time data delivery in wireless sensor networks," in *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2009, pp. 413–427.

[45] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 14–27.

[46] Y.-B. Ko, J.-M. Choi, and J.-H. Kim, "A new directional flooding protocol for wireless sensor networks," in *Springer-Lecture Notes in Computer Science*, vol. 3090, 2004, pp. 93–102.

[47] Y. Zhuang, J. Pan, and L. Cai, "Minimizing energy consumption with probabilistic distance models in wireless sensor networks," March 2010, pp. 1–9.

[48] D. Jung, T. Teixeira, and A. Savvides, "Sensor node lifetime analysis: Models and tools," *ACM Trans. Sen. Netw.*, vol. 5, pp. 3:1–3:33, February 2009.

[49] T. He, P. Vicaire, T. Yan, Q. Cao, and G. Z. et al., "Achieving long-term surveillance in vigilnet," *INFOCOM*, 2006.

[50] C. Sengul, M. J. Miller, and I. Gupta, "Adaptive probability-based broadcast forwarding in energy-saving sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, pp. 6:1–6:32, April 2008.

[51] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Energy-latency tradeoffs for data gathering in wireless sensor networks," in *INFOCOM*, 2004.

[52] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, "Tracking a moving object with a binary sensor network," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03, 2003, pp. 150–161.

[53] D. Eickstedt and M. Benjamin, "Cooperative target tracking in a distributed autonomous sensor network," in *OCEANS 2006*, September 2006, pp. 1–6.

[54] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, and H. Z. et al., "A line in the sand: A wireless sensor network for target detection, classification,and tracking," *Computer Networks (Elsevier)*, vol. 46, no. 5, pp. 605–634, 2004.

[55] J. Liu, M. Chu, and J. Reich, "Multitarget tracking in distributed sensor networks," *Signal Processing Magazine, IEEE*, vol. 24, no. 3, pp. 36–46, May 2007.

[56] S. Oh, L. Schenato, and S. Sastry, "A hierarchical multiple-target tracking algorithm for sensor networks," *International Conference on Robotics and Automation*, 2005.

[57] L. Fan, H. Wang, and H. Wang, "A solution of multi-target tracking based on fcm algorithm in wsn," in *IEEE International Conference on Pervasive Computing and Communications Workshops*, 2006, p. 290.

[58] L. Chen, M. Cetin, and A. Willsky, "Distributed data association for multi-target tracking in sensor networks," in *International Conference on Information Fusion*, 2005.

[59] S. Liu, K.-W. Fan, and P. Sinha, "Dynamic sleep scheduling using online experimentation for wireless sensor networks," in *SenMetrics*, 2005.

[60] R. M. Taqi, M. Z. Hameed, A. A. Hammad, Y. S. Wha, and K. K. Hyung, "Adaptive yaw rate aware sensor wakeup schemes protocol (a-yap) for target prediction and tracking in sensor networks," *IEICE - Transactions on Communications*, vol. E91-B, no. 11, pp. 3524–3533, 2008.

[61] Y. Zou and K. Chakrabarty, "Distributed mobility management for target tracking in mobile sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 6, no. 8, pp. 872–887, Aug. 2007.

[62] "Kinematics." [Online]. Available: http://en.wikipedia.org/wiki/Kinematics

[63] R. Olfati-Saber, "Distributed kalman filtering for sensor networks," in *Decision and Control, 2007 46th IEEE Conference on*, Dec. 2007, pp. 5492–5498.

[64] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley, "Tracking multiple targets using binary proximity sensors," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, pp. 529–538.

[65] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, "Collaborative signal and information processing: An information-directed approach," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1199–1209, 2003.

[66] D. Fox, "Adapting the sample size in particle filters through kld-sampling," *International Journal of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.

[67] N. Kwak, I.-K. Kim, H.-C. Lee, and B.-H. Lee, "Adaptive prior boosting technique for the efficient sample size in fastslam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 630–635.

[68] M. Rosencrantz, G. Gordon, and S. Thrun, "Decentralized sensor fusion with distributed particle filters," in *Proceedings of Uncertainty in Artificial Intelligence Acapulco*, 2003.

[69] G. Ing and M. J. Coates, "Parallel particle filters for tracking in wireless sensor networks," in *Proceedings of IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, 2005, pp. 935 – 939.

[70] V. C. Gungor, O. B. Akan, and I. F. Akyildiz, "A real-time and reliable transport (rt)2 protocol for wireless sensor and actor networks," in *IEEE/ACM Transactions on Networking*, 2008.

[71] T. F. Abdelzaher, S. Prabh, and R. Kiran, "On real-time capacity limits of multihop wireless sensor networks," in *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*, 2004, pp. 359–370.

[72] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 1–38, 2006.

[73] "Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 923–933, 2006, zhu,, Jin and Papavassiliou,, Symeon and Yang,, Jie.

[74] M. Caccamo, L. Zhang, L. Sha, and G. Buttazzo, "An implicit prioritized access protocol for wireless sensor networks," 2002, pp. 39–48.

[75] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic, and T. Abdelzaher, "Real-time power-aware routing in sensor networks," *Proceedings of the 14th IEEE International Workshop on Quality of Service*, pp. 83–92, 2006.

[76] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing sensor networks in the energy-latency-density design space," *IEEE Transactions on Mobile Computing*, vol. 1, no. 1, pp. 70–80, 2002.

[77] A. Willig and H. Karl, "Data transport reliability in wireless sensor networks - a survey of issues and solutions," in *Praxis der Informationsverarbeitung und Kommunikation*, 2005, pp. 86–92.

[78] S. Pleisch, M. Balakrishnan, K. Birman, and R. van Renesse, "Mistral: Efficient flooding in mobile ad-hoc networks," in *In Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*. ACM Press, 2006, pp. 1–12.

[79] S. Chatterjea, T. Nieberg, N. Meratnia, and P. Havinga, "A distributed and self-organizing scheduling algorithm for energy-efficient data aggregation in wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 4, pp. 1–41, 2008.

[80] C. Joo, J.-G. Choi, and N. B. Shroff, "Delay performance of scheduling with data aggregation in wireless sensor networks," in *Proceedings of the 29th conference on Information communications*, Piscataway, NJ, USA, 2010, pp. 893–901.

[81] T. He, L. Gu, L. Luo, T. Yan, J. Stankovic, and S. Son, "An overview of data aggregation architecture for real-time tracking with sensor networks," apr. 2006, p. 8 pp.

[82] H. Zhang, A. Arora, Y.-r. Choi, and M. G. Gouda, "Reliable bursty convergecast in wireless sensor networks," *Comput. Commun.*, vol. 30, no. 13, pp. 2560–2576, 2007.

[83] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–66, August 2001.

[84] R. Stoleru, J. A. Stankovic, and S. H. Son, "Robust node localization for wireless sensor networks," in *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, 2007, pp. 48–52.

[85] L. Yang, C. Feng, J. W. Rozenblit, and H. Qiao, "Adaptive tracking in distributed wireless sensor networks," in *Engineering of Computer Based Systems, 13th Annual IEEE International Symposium and Workshop*, 2006, p. 9.

[86] Chipcon, "Cc2420 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver," http://www.chipcon.com.

[87] CrossBow, "Mica2 data sheet," http://www.xbow.com.

[88] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, approved in 1999, reaffirmed in 2003, revised in 2007.

[89] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, New York, NY, USA, 2004, pp. 95–107.

[90] Y. Yu and Q. Cheng, "Particle filters for maneuvering target tracking problem," *Signal Process.*, vol. 86, no. 1, pp. 195–203, 2006.

[91] J. Lin, W. Xiao, F. L. Lewis, and L. Xie, "Energy-efficient distributed adaptive multi-sensor scheduling for target tracking in wireless sensor networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 6, pp. 1886–1896, 2008.

[92] F. Gunnarsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation and tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 425–437, 2002.

[93] Y. He and E. K. P. Chong, "Sensor scheduling for target tracking in sensor networks," in *43rd IEEE Conference on Decision and Control*, 2004, pp. 743–748.

[94] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, pp. 147–163, 2002.

[95] J. Denga, Y. S. Hanb, W. B. Heinzelmanc, and P. K. Varshney, "Balanced-energy sleep scheduling scheme for high density cluster-based sensor networks," in *Computer Communications: special issue on ASWN04*, vol. 28, 2005, pp. 1631–1642.

[96] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao, "Distributed group management in sensor networks: Algorithms and applications to localization and tracking," in *Telecommunication Systems*, vol. 26, no. 2-4, 2004, pp. 235–251.

[97] M. Athanassoulis, I. Alagiannis, and S. Hadjiefthymiades, "Energy efficiency in wireless sensor networks: A utility-based architecture," in *European Wireless*, 2007.

[98] J. Hill and D. Culler, "Mica: a wireless platform for deeply embedded networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, Nov/Dec 2002.

[99] S. Oh, L. Schenato, P. Chen, and S. Sastry, "A scalable real-time multiple-target tracking algorithm for sensor networks," *Memorandum*, 2005.

[100] G. Xing, C. Lu, Y. Zhang, Q. Huang, and R. Pless, "Minimum power configuration in wireless sensor networks," in *MobiHoc*, 2005, pp. 390–401.

[101] G. Wang, Y. Liu, and H. Shi, "Covariance tracking via geometric particle filtering," in *ICICTA '09: Proceedings of the 2009 Second International Conference on Intelligent Computation Technology and Automation*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 250–254.

[102] L. Li, Y. Liu, L. Sun, and J. Ma, "Lightweight particle filters based localization algorithm for mobile sensor networks," *Sensor Technologies and Applications, International Conference on*, pp. 135–140, 2008.

[103] G. Kitagawa, "Monte carlo filter and smoother for non-gaussian nonlinear state space models," *Journal of Computational and Graphical Statistics*, vol. 5, no. 1, pp. 1–25, 1996.

[104] Y. Xu and H. Qi, "Mobile agent migration modeling and design for target tracking in wireless sensor networks," *Ad Hoc Netw.*, vol. 6, no. 1, pp. 1–16, 2008.

[105] N.-L. Lai, C.-T. King, and C.-H. Lin, "On maximizing the throughput of convergecast in wireless sensor networks," in *GPC'08: Proceedings of the 3rd international conference on Advances in grid and pervasive computing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 396–408.

[106] P. Basu and J. Redi, "Effect of overhearing transmissions on energy efficiency in dense sensor networks," *Third International Symposium on Information Processing in Sensor Networks (IPSN), 2004.*, pp. 196–204, 2004.

[107] M. N. Halgamuge, "Performance evaluation and enhancement of mobile and sensor networks," Ph.D. dissertation, The University of Melbourne, Victoria 3010, Australia, November 2006.

[108] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *IEEE Infocom*, vol. 3, 2002, pp. 1567–1576.

[109] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *Network, IEEE*, vol. 18, no. 4, pp. 45–50, July-Aug. 2004.

[110] IETF, "Rfc 2681 - a round-trip delay metric for ippm," http://www.ietf.org/rfc/rfc2681.txt?number=2681.

[111] K. Liu, N. Abu-Ghazaleh, and K.-D. Kang, "Jits: just-in-time scheduling for real-time sensor data dissemination," *Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pp. 5–46, 2006.

[112] Scalable-Networks, "Qualnet network simulator," http://www.scalable-networks.com/.

[113] B. Vasu, M. Varshney, R. Rengaswamy, M. Marina, A. Dixit, P. Aghera, M. Srivastava, and R. Bagrodia, "Squalnet: a scalable simulation framework for sensor networks," in *SenSys: Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM, 2005, pp. 322–322.

[114] A. T. Inc., "Dc source with battery emulation," http://www.home.agilent.com/agilent/product.jspx?pn=66319D.

[115] D. F. Circuits, "Battery simulator (bs)," http://www.digatron.com/battery_simulation.html.

[116] K. I. Inc., "Battery/charger simulators," http://www.keithley.com/.

[117] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, pp. 186–195.

[118] P. Electronics, "I2c bus," http://www.i2cbus.com/.

[119] C. Margi, V. Petkov, K. Obraczka, and R. Manduchi, "Characterizing energy consumption in a visual sensor network testbed," in *In 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2006)*, 2006.

[120] S. Gurun, "Energy consumption and conservation in mobile peer-topeer systems," in *In Proceedings of the 1 st ACM International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare)*, 2006.

[121] C. B. Margi, R. M, and K. Obraczka, "Energy consumption tradeoffs in visual sensor networks," in *In 24th Brazilian Symposium on Computer Networks (SBRC 2006)*, 2006.

[122] "Cygwin." [Online]. Available: http://www.cygwin.com/

[123] "Windows." [Online]. Available: http://www.microsoft.com/windows/

[124] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 126–137.

[125] D. Gay and J. Hui, "Tep 103: Permanent data storage," http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html.

[126] "Java programming language." [Online]. Available: http://en.wikipedia.org/wiki/Java_(programming_language)