Self-Configurable Publish/Subscribe Middleware for Wireless Sensor Networks

Pruet Boonma and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
{pruet, jxs}@cs.umb.edu

Abstract

In the publish/subscribe (pub/sub) communication scheme in wireless sensor networks (WSNs), there exist inherent tradeoffs among conflicting objectives in event publication. To address this issue, this paper investigates pub-/sub middleware for WSNs, called TinyDDS. With its self-configuring event routing protocol, TinyDDS adaptively performs event publication according to dynamic network conditions and autonomously balances its performance among conflicting objectives. TinyDDS leverages an evolutionary multiobjective optimization mechanism to seek the optimal tradeoffs among objectives and adjust parameters in its event routing protocol. Simulation results validate the ability of TinyDDS to tune its event publication against dynamic network conditions. TinyDDS is implemented lightweight and efficient enough to run on resource-limited nodes.

1. Introduction

This paper considers wireless sensor networks (WSNs) that are deeply embedded in the physical environment and intended to aid personalized data communication for users. For example, in disaster response for wildfire, fire fighters may carry devices that communicate with an emergency center and display personalized data such as their surrounding environments (e.g., temperature, CO, humidity and wind speed). WSNs can monitor an observation area (e.g. a mountain) to detect certain critical events and transmit them to a base station located at an emergency center. Officers in the center may forward those transmitted events to fire fighters in a particular area.

The publish/subscribe (pub/sub) communication scheme is suitable for this event notification with WSNs by decoupling space and time among data/event source nodes (publishers) and sink nodes (subscribers). A subscriber has the ability to express its interest in an event or a pattern of events in order to be notified subsequently. Each interest is subscribed to a publisher(s), and the publisher(s) notifies an event to a subscriber(s) when the event matches a subscribed interest. Publishers do not need to know the number and locations of subscribers, and vice versa. Moreover, publishers do not need to know the availability of subscribers, and vice versa. For example, subscribers may be active, sleeping or dead due to a lack of battery when a publisher publishes an event. Event subscription and publication are asynchronously transmitted among publishers and subscribers. Decoupling of space and time can increase the scalability of WSNs by removing explicit dependencies among publishers and subscribers.

A key issue in the pub/sub scheme in WSNs is that, in event publication, there exist inherent tradeoffs among conflicting operational objectives such as data yield, data fidelity and power efficiency. For example, hop-by-hop recovery is often used for packet transmission to improve data yield (the quantity of successfully-notified event data) from publishers to subscribers. However, this can degrade data fidelity (the quality of event data; e.g., event data freshness) and power efficiency. For improving data fidelity, publishers may publish event data to subscribers with the shortest paths; however, data yield can degrade because of traffic congestion and packet losses on the paths. For improving power efficiency, publishers may frequently sleep; however, this can degrades data yield and data fidelity.

In order to address this issue, this paper investigates TinyDDS, which is pub/sub middleware for event detection and dissemination applications in WSNs. With its self-configuring event routing protocol, TinyDDS adaptively performs event publication according to dynamic network conditions and autonomously balances its performance among conflicting operational objectives. TinyDDS leverages an evolutionary multiobjective optimization mechanism, called MONSOON, in order to seek the optimal tradeoffs among objectives and adjust parameters in its event routing protocol. Simulation results validate the ability of TinyDDS to self-configure its event publication against various dynamic changes in the network (e.g., node failures, base station failures, noises in communication and network separation). TinyDDS is implemented lightweight enough to run on resource-limited nodes.

2. OMG DDS Standard Specification

TinyDDS is a lightweight implementation of the Data Distribution Service (DSS), which Object Management Group (OMG) standardizes for topic-based pub/sub middleware.DDS provides standard interfaces for event subscription and publication in Interface Definition Language (IDL), and TinyDDS implements them with nesC. See [2] for the IDL-nesC mapping in TinyDDS.

Figure 1 shows the architecture of DDS middleware. An event sink expresses its interest to an event, or *topic*, and subscribes to its local `Subscriber` with associated `SubscriberListner` and `DataReader`.

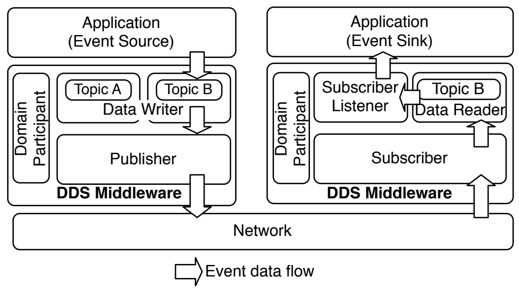


Figure 1. DDS Architecture

An event source creates an event/topic with its corresponding `DataWriter`, and the event is published by a `Publisher` to its subscribers in the network.

A `Subscriber` on each node monitors every incoming event, and if the event matches an event sink's subscription, it will be notified to the event sink via `SubscriberListner` and `DataReader`.

DDS specifies a set of QoS parameters for distributed networks, for example, *LATENCY_BUDGET* and *RELIABILITY*. The former indicates the "urgency" of the event by providing a maximum bound of the event latency, i.e., the time period since the event is published until the event is received. The later indicates the level of reliability requested by `DataReader` or offered by `DataWriter`. In DDS specification, there are two reliability level, *BEST_EFFORT* which indicates that the middleware does not have to retransmit any failed transmission and *RELIABLE* which indicates that the middleware have to retransmit any failed transmission in order to improve the success rate the event transmission. DDS, however, does not standardize how to satisfy these QoS parameters and leaves to implementations.

3. TinyDDS

TinyDDS is a pub/sub middleware for WSNs application with self-configuration mechanism using evolutionary algorithm to satisfy QoS parameters. Applications implemented on top of TinyDDS can disseminate and collect data through a pub/sub interface provided by TinyDDS and the self-configuration mechanism inside TinyDDS will adjust the operational parameter, e.g., routing scheme, to satisfy QoS parameters autonomously according to current network conditions. In TinyDDS, each base station acts as a subscriber while each sensor node acts as publisher. This assumption is more strict than traditional pub/sub definition; however, this assumption fits well with general WSNs application where sensor nodes are intended to report data or event to base stations.

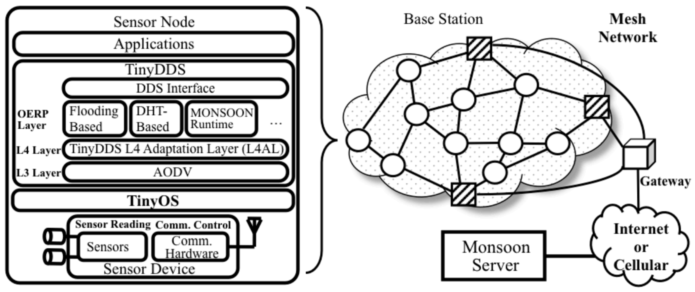


Figure 2. The Architecture of TinyDDS

The left-hand side of the figure 2 shows the architecture of the TinyDDS. With respect to the TCP/IP reference model, TinyDDS operates in transport layer and work on top of any network layer (L3) implementation. TinyDDS follows Layer design pattern [4] by separating different functionalities into different layers. At the top most layer, TinyDDS provides a subset of DDS interfaces to be used by applications. The DDS is an Object Management Group's standard specification for publish/subscribe middleware in distributed network systems. An application implemented on top of DDS can disseminate events, i.e. data or control messages, to the network with associated topic and the events are captured by any subscribers, i.e., base station, who have interest on the topic of that events. The implementation of the DDS interfaces operates on top of an overlay network for event routing. Different routing protocols can be used for routing on overlay network by implementing in this Overlay Event Routing Protocols (OERP) layer. This OERP layer allows application developer to choose appropriate routing protocol to suit their requirements and limitation. For example, in sensor network with very limited memory space sensor nodes, flooding-based routing protocol may be used because it needs minimal memory space to maintain routing table. On the other hand, sensor network which try to minimize the energy consumption of memory rich sensor nodes may use DHT-based routing protocol. By using this OERP layer, TinyDDS frees developers from the limitation of routing algorithm used in network layer which generally depends on sensor node platform such as MicaZ which based on Zigbee protocol stack. The routing protocol in OERP layer utilizes low-level network layer implementation through a transport layer interface called TinyDDS L4 Adaption Layer (L4AL). L4AL allows TinyOS to operates with any network and MAC layer protocol, such as AODV and Zigbee respectively.

To satisfy QoS parameters, TinyDDS utilizes an evolutionary mechanism called MONSOON which autonomously optimize the operational parameters of TinyDDS to meet with the QoS parameters. Evolutionary algorithm is a heuristic search algorithm for complex and dynamic problems which fits well with the dynamic and uncertain nature of WSNs. This optimization mechanism consists of two parts, the MONSOON runtime operated in the OERP layer and the MONSOON server operated at a central server. The MONSOON runtime encapsulate published data from upper layer into a software agent then the runtime sends out the software agent to the network. Each software agent has its own behavior policy which govern its behavior, e.g., how to move to the next hop toward a base station. This behavior policy, i.e., gene, is improved by MONSOON server at a central server. In particular, when an agent reaches a base station and in turn central server, MONSOON server evaluates the agent according to the operational objectives, including QoS parameters. Example of operational objectives are latency, which represents the *LATENCY_BUDGET* QoS parameters in DDS, and power consumption. Based on a collection of agents, MONSOON server selects some of the agents with well-balanced performance on the operational objectives as elite agents and send those agents to sensor nodes. At each sensor nodes, the incoming elite agents perform genetic operations with agents on the sensor node inside the MONSOON runtime in order to improve the behavior policy of next generation agents.

3.1. Agent Behaviors

Each agent implements seven behaviors and performs them in the following sequence.

Step 1: Energy gain. When an event is given to an agent, the agent also gain some *energy*. In MONSOON, the concept of energy does not represent the amount of physical battery in a node. It is a logical concept that impacts agent behaviors. Each agent updates its energy level with a constant energy intake (E_F):

Step 2: Energy expenditure and death. Each agent consumes a constant amount of energy to use computing/networking resources available on a node (e.g., CPU and radio transmitter). It also expends energy to invoke its behaviors. The energy costs to invoke behaviors are constant for all agents. An agent dies due to energy starvation when it cannot balance its energy gain and expenditure. The death behavior is intended to eliminate the agents that have ineffective behavior policies. For example, an agent would die before arriving at a base station if it follows a too long migration path. When an agent dies, the local platform removes the agent and releases all resources allocated to it[1].

Step 3: Replication. Each agent makes a copy of itself when it gains energy, i.e., when it receive a new event from upper layer. A replicated (child) agent is placed on the node that its parent resides on, and it inherits the parent's behavior policy (gene). A replicating (parent) agent splits its energy units to halves, gives a half to its child agent, and keeps the other half. A child agent contains the event that its parent received, and carries it to a base station on a hop by hop basis.

Step 4: Swarming. Agents may swarm (or merge) with others at intermediate nodes on their ways to base stations. On each intermediate node, each agent decides whether it migrates to a next-hop node or waits for other agents to arrive at the current node and swarm with them. This decision is made based on the migration probability (p_m). If an agent meets other agents destine at a same base station during a waiting period, it merges with them and contains the event they carry. It also uses the behavioral policy of the best one in the aggregating agents in terms of operational objectives. (See Section 4. on how to find the "best" agent.) The swarming behavior is intended to save power consumption by reducing the number of data transmissions. If the size of data an agent carries exceeds the maximum size of a packet, the agent does not consider the swarming behavior.

Step 5: Pheromone sensing and migration. On each intermediate node toward a base station, each agent chooses a migration destination node (next-hop node) by sensing three types of *pheromones* available on the local node: base station, migration and alert pheromones.

Each base station periodically propagates a base station pheromone to individual nodes in the network. Their concentration decays on a hop-by-hop basis. Using base station pheromones, agents can sense where base stations exist approximately, and move toward them by climbing a concentration gradient of base station pheromones.

Agents may emit migration pheromones on their local nodes when they migrate to neighboring nodes. Each migration pheromone references the destination node an agent has migrated to. Agents may also emit alert pheromones when they fail migrations within a timeout period. Migration failures can occur because of node failures due to depleted battery and physical damages as well as link failures due to interference and congestion. Each alert pheromone references the node that an agent could not migrate to. Each of migration and alert pheromones has its own concentration. The concentration decays by half at each duty cycle. A pheromone disappears when its concentration becomes zero.

Each agent examines Equation 1 to determine which next-hop node it migrates to.

$$WS_j = \sum_{t=1}^{3} w_t \frac{P_{t,j} - P_{t_{min}}}{P_{t_{max}} - P_{t_{min}}} \tag{1}$$

An agent calculates this weighted sum (WS_j) for each neighboring node j, and moves to a node that generates the highest weighted sum. t denotes pheromone type; P_{1j}, P_{2j} and P_{3j} represent the concentrations of base station, migration and alert pheromones on the node j. $P_{t_{max}}$ and $P_{t_{min}}$ denote the maximum and minimum concentrations of P_t

[1] If all agents are dying on a node at the same time, a randomly selected agent will survive. At least one agent runs on each node.

among all neighboring nodes.

The weight values in Equation 1 ($w_t, 1 \leq t \leq 3$) govern how agents perform the migration behavior. For example, if an agent has zero for w_2 and w_3, the agent ignores migration and alert pheromones, and moves toward a base station by climbing a concentration gradient of base station pheromones. If an agent has a positive value for w_2, it follows a migration pheromone trace on which many other agents have traveled. The trace can be the shortest path to a base station. Conversely, a negative w_2 value allows an agent to go off a migration pheromone trace and follow another path to a base station. This avoids separating the network into islands. The network can be separated with the migration paths that too many agents follow, because the nodes on the paths run out of their battery earlier than the others. If an agent has a negative value for w_3, it avoids moving to a node referenced by an alert pheromone, thereby bypassing failed nodes and links.

Step 6: Pheromone emission. When an agent is migrating to a neighboring node, it may emits a migration pheromone on the local node. If the agent's migration fails, it may emits an alert pheromone. The decision to emit pheromone is made based on the migration pheromones emission probability (p_{mpe}) and alert pheromones emission probability (p_{ape}), respectively. If the emission probability is high, it is a higher chance that agent will emit the pheromone; thus, the other agents can gain knowledge about how this agent moves or the node failure. As a consequence, the success rate or the latency of the other agents may be improved. However, this pheromone emission will increase the power consumption of agent. In the other words, these emission probabilities control how agent is cooperative with the others or be selfish. Each pheromone spreads to one-hop away neighboring nodes.

Step 7: Reproduction. Two parent agents may produce a child agent. A child agent is placed on the node that their parents reside on, and it inherits the parents' behavior policies (genes). This behavior is intended to evolve agents. (See Section 4. for more details.)

3.2. Agent Behavior Policy

Each behavior policy consists of two distinctive information: a set of behavior probability (p_m, p_{mpe}, and p_{ape}) and a set of weight values in Equation 1 ($w_t, 1 \leq t \leq 3$). The behavior probability is a non-negative value between zero and one. They are used for each agent to decide whether it performs the migration behavior, migration pheromone emission, and alert pheromone emission respectively.

4. Optimization Process in MONSOON

The optimization process in MONSOON is performed in the MONSOON Server (see Figure 2). The MONSOON optimization process is based on evolution process, i.e., genetic algorithm, which consists of, *elite selection* and *genetic operation*. In the evolution process in MONSOON, elite selection and genetic operations are performed in the MONSOON server and each node, respectively.

The elite selection process evaluates the agents that arrive at base stations, based on given operational objectives, and chooses the best (or elite) ones. Elite agents are propagated to individual nodes in the network. Through genetic operations (crossover and mutation), an agent running on each node performs the reproduction behavior with one of elite agents. A reproduced agent inherits a behavior policy (gene) from its parents via crossover, and mutation may occur on the inherited behavior policy.

Reproduction is intended to evolve agents so that the agents that fit better to the environment become more abundant in the network. It retains the agents that have effective behavior policies, such as moving toward a base station in a short latency, and eliminates the agents that have ineffective behavior policies, such as consuming too much power to reach a base station. Through successive generations, effective behavior policies become abundant in agent population while ineffective ones become dormant or extinct. This allows agents to adapt to dynamic network conditions.

4.1. Operational Objectives

Each agent considers four conflicting objectives related to data yield, data fidelity and power consumption: latency, cost, success rate and the degree of data aggregation. Success rate and the degree of data aggregation are related to data yield. Latency is related to date fidelity. Cost is related to power consumption. MONSOON strives to minimize latency and cost and maximize success rate and the degree of data aggregation.

(1) Latency represents the time required for an agent to travel to a base station from a node where the agent is replicated. It (L) is measured as a ratio of this agent travel time (t) to the physical distance (d) between a base station and a node where the agent is replicated. Latency represents the *LATENCY_BUDGET* QoS parameter in DDS.

$$L = \frac{t}{d} \tag{2}$$

(2) Success rate (S) is measured as the ratio of the number of successful data transmissions (N_{succ}) to the total number of data transmissions (n_{tran}). Success rate represents *RELIABILITY* QoS parameters with *RELIABLE* setting in DDS.

$$S = \frac{n_{succ}}{n_{tran}} \tag{3}$$

(3) Cost represents the amount of power consumption required for an agent to travel to a base station. It (C) is measured with d, each node's radio communication range (r), and the total number of node-to-node data transmissions required for an agent to arrive at a base station (n_{tran}).

$$C = \frac{n_{tran}}{r/d} \quad (4)$$

The total number of data transmissions counts successful and unsuccessful migrations of an agent as well as the transmissions of its migration and alert pheromones.

(4) Degree of data aggregation is measured as the number of sensor data in an agent. It is more than two in a swarming agent.

4.2. Elite Selection

Listing 1 shows how the MONSOON server periodically performs elite selection. The first step is to measure four objective values of each agent that arrives at base stations. Each of agent is evaluated whether it is *dominated* by another one. It is considered to be dominated if another outperforms it in all four objectives. However, because the objectives of agents are measured from their performance on WSN, which contains lots of "noise", i.e., uncertainty, in objective function. For example, an agent with a good gene might has poor performance because it is unlucky, i.e., blocked by many ineffective agents. Thus, the agent might has very good objective values in all objective, except the very high latency. Hence, MONSOON relaxes the domination condition by allows agents which is weakly dominated by the other agents to be considered as non-dominated agent. In the other words, an agent is dominated only when it is strongly dominated by the other agents. In particular, an agent is strongly dominated by the other agent when all of its objective values are worse than that of the other agents. Formally, for a minimization problem, a solution $\vec{x} \in \mathcal{S}$, where $\mathcal{S}$ is the decision variable space, dominates a solution $\vec{y} \in \mathcal{S}$, i.e. $\vec{x} \succ \vec{y}$, if and only if $f_i(\vec{x}) < f_i(\vec{y}) \; \forall \; i = 1, \cdots, m$, where f_i is an objective function. As a consequence, MONSOON can eliminate the effect of noise in the objective function by allowing agents who might get some noise in some objectives to survive to the next generation. Finally, a subset of non-dominated solutions is selected as elite agent and propagated to WSN to perform reproduction.

Listing 1. Elite Selection in MONSOON

```
A = ∅
while true {
  P_net = CollectAgentFromNetwork()
  P = A ∪ P_net
  for each a ∈ P {
    for each a′ ∈ P {
      if a′ ≠ a and a′ ≻ a {
        P = P \ a
        break
      }
    }
  }
  A = P
  P_elite = EliteSelection(P)
  PropagateToNetwork(P_elite)
}
```

Table 1. Variables and Functions in Elite Selection in MONSOON

A	Archive population
P_{net}	Agent population collected from sensor network
P	Current population
a, a'	An agent
P_{elite}	Elite agent selected from non-dominated population
CollectAgentFromNetwork()	A function returning agents collected from sensor network
EliteSelection(P)	A function selecting a subset of agents from non-dominated population
PropagateToNetwork(P_{elite})	A function propagating elite agents to sensor network

The MONSOON server propagates elite agents to individual nodes in the network. This propagation is performed with a base station pheromone.

4.3. Genetic Operations

Upon receiving a base station pheromone, an agent running on each node performs the reproduction behavior with a certain reproduction rate. It selects one of propagated elite agents, as a mating partner, which has the most similar gene (behavior policy). Gene similarity is measured with the Euclidean distance between the values of two genes. If two or more elite agents have the same similarity to the local agent, one of them is randomly selected. During reproduction, a child agent performs one-point half-and-half crossover; it randomly inherits the half of its gene from its parent agent and the other half from the parent's mating partner.

Mutation occurs on a child agent's gene, with a certain mutation rate, by randomly changing gene values within a predefined value range. As described in the previous section, mutation rate is periodically adjusted by the MONSOON server and propagated to individual nodes. After reproduction, a child agent takes over the local parent as the next generation agent.

5. Simulation Results

This section shows simulation results to evaluate TinyDDS in static and dynamic network conditions in terms of performance and degree of self-organization. All simulations were run with the TOSSIM simulator [7].

The network consists of 100 nodes deployed uniformly in a 200x200 meters observation area to detect a forest fire.

A simulated forest fire starts at the middle of the observation area and spreads in the observation area. Each node's communication range is 30 meters. A base station is deployed on the northwestern corner of the observation area. The base station connects to the MONSOON server via emulated serial port connection. Packet loss rate is 0.05. On each node, an agent keeps monitoring the change of temperature in the current and previous duty cycle, and if it exceeds a threshold, the agent publishes an event by replicating itself. The duty cycle of each node is fixed at five minutes.

The degree of self-organization is measured with the notion of *entropy*. In this paper, entropy indicates how similar performance different agents yield. The lower entropy, the more similar performance they yield. To measure the entropy, the objective space is divided into a set of small non-overlapped cubes with the same number of division in each axis. Then, entropy (E) is given as follows.

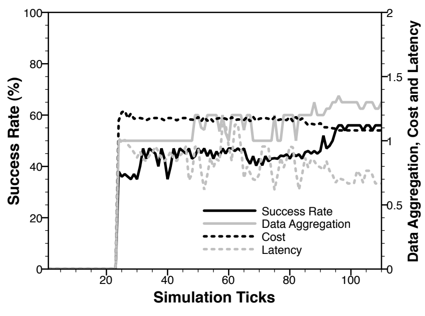
$$E = \sum_{i \in S} p_i \log(p_i) \tag{5}$$

i identifies individual cubes and S denotes the set of all cubes. p_i denotes the probability in which agents are plotted in the cube i, i.e., the number of agents in the cube i over the total number of agents.

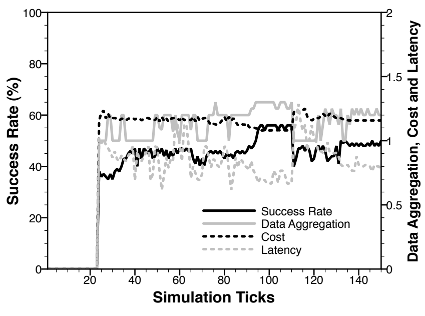
Figure 3 shows the average objective values that agents yield over a simulation. Each simulation tick represents a duty cycle. Figure 3 (a) shows a result in the static network, in which node/base station failures never occur. Each objective value improves and converges around the 90th tick. This result shows that , through evolution, agents self-configure their behavior policies and self-optimize their performance against conflicting objectives. Thus, TinyDDS can improves its performance in order to satisfy the QoS parameters, i.e., latency (in second per 30 meters) and reliability. Figures 3 (b), (c), (d) and (e) show how agents perform against dynamic changes in the network. Upon each changes that occurs at the 110th tick, objective values drops. In Figures 3 (b), when 25 nodes are added at random locations, objective values degrade because agents initially have random behavior policies on new nodes. Those agents cannot perform its behavior efficiently, e.g., cannot migrate to the base station in a shortest path. Also, enough pheromones are not available a new node when they are deployed; agents cannot make proper migration decision on those nodes. In figure 3 (b), randomly-selected 25 nodes fail, i.e., because of the fire. As a result, objective values drop because some agents try to migrate to failed nodes. In Figure 3 (d), the packet loss rate in the network is increased from 0.05 to 0.3; thus, agents have higher chance to failed migration. However, after 35 ticks, each objective value improves and converges again. In Figure 3 (e) and (f), two base stations are initially deployed at the northwestern and southeastern corners of the observation area. In Figure 3 (d), at 110th tick, a base station fails at the southeastern corner. Consequently, objective values drop because some agents try to migrate to the failed base station. In figure 3 (e), the forest fire spreads to northeastern and southwestern corner and destroy all sensor nodes along the way; thus, the network is split into two islands.

Once objective values drop due to a dynamic change in the network, agents gradually improve and converge their objective values again. Objective values are mostly same before and after each dynamic change. In figure 3 (b), agents yield a slightly higher degree of data aggregation after a dynamic node addition because there are more agents migrating in the network and there are higher chances for them to aggregate. Figures 3 (b), (c), (d), (e) and (f) show that MONSOON allows agents to posses a self-configuration against dynamic network condition; agents autonomously detect and configure itself according to the changes in the network. Despite those changes, agents self-configure their behavior polices and self-optimize their performance by evolving their behavior policies. As a consequence, TinyDDS can improves its operational parameters in order to satisfy with the required QoS parameters.

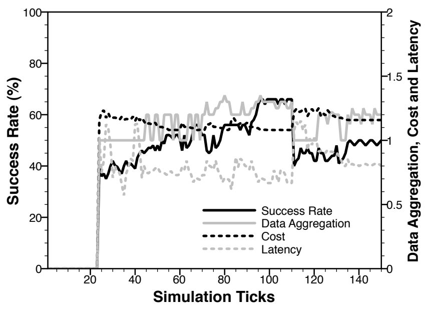
Figure 4 (a) and (b) show the average objective values that agents yield over a simulation when the weak domination is used instead of strong domination (see Section 4.2.). In the weak domination, an agent is weakly dominated by the other agents when all of its objective values do not better than the other agents and at least one of its objective value is worse than that of the others. Formally, for a minimization problem, a solution $\vec{x} \in \mathcal{S}$, where $\mathcal{S}$ is the decision variable space, dominates a solution $\vec{y} \in \mathcal{S}$, i.e. $\vec{x} \succeq \vec{y}$, if and only if $f_i(\vec{x}) \leq f_i(\vec{y}) \;\forall\; i = 1, \cdots, m$ and $f_i(\vec{x}) < f_i(\vec{y}) \;\exists\; i = 1, \cdots, m$. where f_i is an objective function. Compared with the figure 3 (a) and (b) , the results in the figure 4 (a) and (b) show slower convergence rate. For example, in the figure 3 (a), success rate converges at around 90th simulation ticks. However, in the figure 4 (a), it takes about 105 simulation ticks before the success rate is converged. The same phenomena can be observed from the other objectives. This can be attributed from the fact that strong domination can preserve more moderate individuals in each generation; thus, diversity is better maintained across generation. The simulation results show that strong domination allows MONSOON to converge faster. In the figure in the figure 3 (b) and in the figure 4 (b), the results are very similar the both figures. However, in the figure 4 (c) which shows the entropy of the system suggests that MONSOON with strong domination has better degree of self-organization after the noise addition occurs at 110th simulation ticks. Thus, by replacing weak domination with strong domination, MONSOON can be more self-organized and tolerant to the noise in the system.



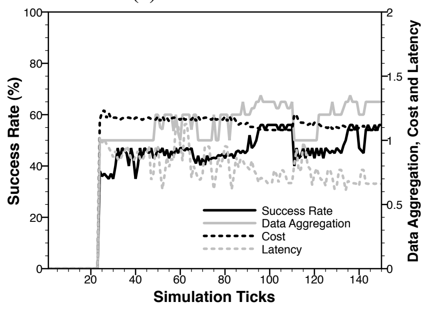
(a) Static Network



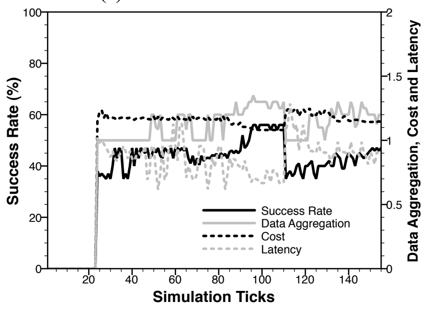
(c) Random Node Failure



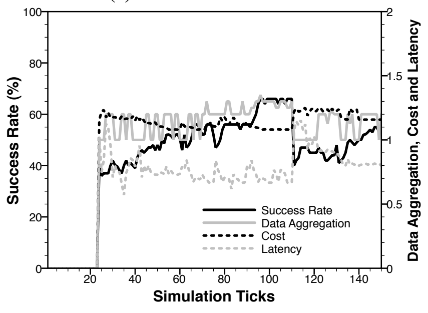
(e) Base Station Failure



(b) Node Addition



(d) Noise Addition



(f) Network Splitting

Figure 3. Performance with Success Rate, Data Aggregation, Cost and Latency

6. Processing and Power Efficiency

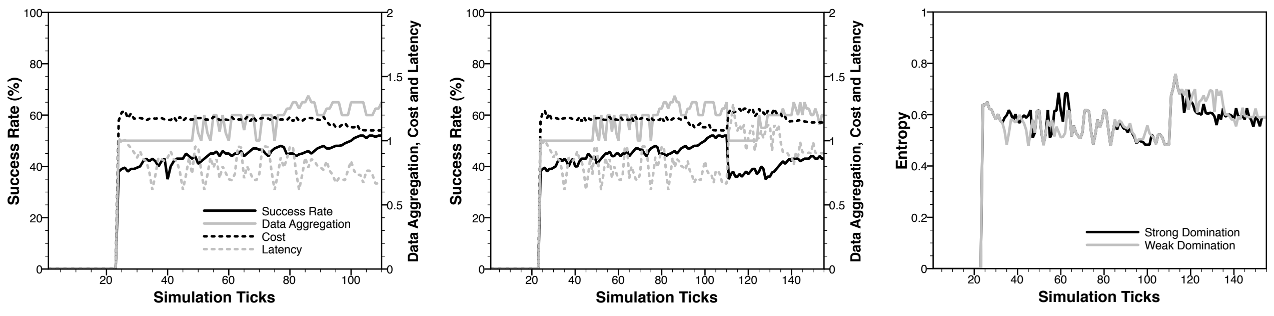
When TinyDDS is deployed on TinyOS, the average total latency is 0.79 second to route an event between source and destination nodes in a single hop. Of this total latency, TinyDDS spends 0.08 second; 0.03 second on a source node and 0.05 second on a destination node. This means TinyDDS occupies approximately 10% of the total latency. The total power consumption is 168mW per hop in the same setting. TinyDDS consumes only 3% of the total consumption (6 mW). These measurements results demonstrate that TinyDDS is implemented lightweight and efficient enough to operate on a resource-limited nodes.

7. Related Work

This paper describes a set of extensions to the authors' prior work [1, 2]. TinyDDS was originally proposed in [2]. [2] focuses on the (manual) configurability of TinyDDS and describes how it allows application developers to configure various policies and parameters in, for example, concurrency, data re-retransmission, data aggregation and event filtering. This paper extends [2] to study (non-manual) self-configuration of event publication against dynamic network conditions. MONSOON is proposed and evaluated in [1]. This paper extends [1] to explore how MONSOON's optimization mechanism complements the standard DDS specification and augments the pub/sub communication scheme in WSNs. In this paper, the design of MONSOON is extended to replace a traditional strong dominance operator with a weak dominance operator for guiding agent genes (i.e., behavior policies) more diverse and robust against dynamic network conditions. Moreover, this paper evaluates MONSOON in new simulation configurations (e.g., packet losses and network separation) that [1] does not consider.

There exist several pub/sub middleware for WSNs [5, 8, 10]. These research efforts propose reconfigurable middleware that allows application developers to flexibly customize a series of parameters and policies. However, they do not consider self-adaptation of middleware; application developers need to manually conduct the tedious, time-consuming and error-prone process to optimize middleware parameters and policies. In fact, the aforementioned research efforts do not consider the dynamics of sensor networks, but assume static and noise-free networks. In contrast, TinyDDS is inherently designed to consider self-adaptation to dynamic and noisy networks.

Genetic algorithms are applied to several aspects in WSNs, such as cluster-based routing [6], localization [11] and node placement [12]. Every work uses a fitness function that aggregates multiple objective values as a weighted sum, and uses the function to rank agents/genes in elite selection. Application designers need to manually configure every weight value in a fitness function through trial and errors. In MONSOON, no manually-configured parameters exist for elite selection because of domination ranking. MONSOON imposes much less configuration cost. Also, [3, 6, 11, 12]



(a) Static Network **(b) Noise Addition** **(c) Entropy**

Figure 4. Performance with Weak Domination and Entropy

do not assume dynamic networks, but static networks. Finally, all of the mentioned works do not address the noise in the objective function, which is very common on WSNs because of the dynamic nature of the network.

Evolutionary multiobjective optimization algorithms are used for node placement [9] in WSNs. In [9], an optimization process is performed only in a central server. This can lead to a scalability issue as the network size increases. In contrast, MONSOON is carefully designed to perform its adaptation process in both the MONSOON server and individual nodes.

8. Conclusion

This paper proposes and evaluates a self-configurable pub/sub middleware for WSNs. TinyDDS is designed to adaptively perform event publication by balancing conflicting objectives according to dynamic network conditions. Simulation results validate the ability of TinyDDS to tune its event publication against dynamic network conditions. TinyDDS is implemented lightweight and efficient enough to run on resource-limited nodes.

References

[1] P. Boonma and J. Suzuki. Exploring self-star properties in cognitive sensor networking. In *Proc. of IEEE/SCS Int'l Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2008.

[2] P. Boonma and J. Suzuki. Middleware support for pluggable non-functional properties in wireless sensor networks. In *Proc. of IEEE Workshop on Methodologies for Non-functional Properties in Services Computing*, 2008.

[3] A. L. Buczaka and H. Wangb. Optimization of fitness functions with non-ordered parameters by genetic algorithms. In *Proc. of IEEE Congress on Evolutionary Comp.*, 2001.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns.* Wiley and Sons, 1996.

[5] P. Costa, G. Coulson, C. Mascolo, L. Mottola, G. P. Picco, and S. Zachariadis. A Reconfigurable Component-based Middleware for networked Embedded Systems. *Springer J. of Wireless Information Networks*, 14, 2007.

[6] K. P. Ferentinos and T. A. Tsiligiridis. Adaptive design optimization of wireless sensor networks using genetic algorithms. *Elsevier J. of Computer Networks*, 51, 2007.

[7] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proc. of Int'l Conf. on Embedded Network Sensor System*, 2003.

[8] P. J. Marrón, A. Lachenmann, D. Minder, M. Gauger, O. Saukh, and K. Rothermel. Management and configuration issues for sensor networks. *Wiley Int'l J. of Network Management*, 15, 2005.

[9] R. Rajagopalan, P. K. Varshney, C. K. Mohan, and K. G. Mehrotra. Sensor placement for energy efficient target detection in wireless sensor networks: A multi-objective optimization approach. In *Proc. of Annual Conf. on Information Sciences and Systems*, 2005.

[10] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Springer J. of Personal Ubiquitous Computing*, 10, 2005.

[11] V. Tam, K. Y. Cheng, and K. S. Lui. Using micro-genetic algorithms to improve localization in wireless sensor networks. *J. of Comm., Academy Publisher*, 1(4), 2006.

[12] J. Zhao, Y. Wen, R. Shang, and G. Wang. Optimizing sensor node distribution with genetic algorithm in wireless sensor network. In *Proc. of Int'l Symp. on Neural Nets.*, 2004.