La Niña: An Evolutionary Noise-aware Optimization Framework in Self-Adaptive Publish/Subscribe Middleware for Wireless Sensor Networks

Pruet Boonma* and Junichi Suzuki

Pruet Boonma and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125, USA
{pruet, jxs}@cs.umb.edu
*Corresponding author

Abstract: This paper focuses on three key issues that the publish/subscribe (pub/sub) communication scheme faces for event routing in wireless sensor networks (WSNs): (1) balancing tradeoffs among conflicting performance objectives such as data yield, data fidelity and power efficiency; (2) satisfying quality of service (QoS) requirements such as latency; and (3) considering noise in evaluating event routing performance. To address these issues, this paper investigates self-adaptive event routing in TinyDDS, which is pub/sub middleware for WSNs. With its noise-aware and constraint-based evolutionary multiobjective optimization framework, La Niña, TinyDDS autonomously adapts its routing parameters to dynamic network conditions by reducing the impacts of noise on performance evaluation and seeking the optimal tradeoffs among performance objectives under given QoS requirements. Simulation results validate this ability of TinyDDS in large-scale, dynamic and noisy WSNs. TinyDDS is implemented lightweight enough to operate on resource-limited sensor nodes.

Keywords: Wireless sensor networks, Self-adaptive publish/subscribe middleware, Biologically-inspired networking, Evolutionary multiobjective optimization

1 Introduction

The publish/subscribe (pub/sub) communication scheme can improve scalability and failure resiliency of event notification in wireless sensor networks (WSNs) by decoupling space and time among event source nodes (publishers) and sink nodes (subscribers) (Wang et al., 2008). In the pub/sub scheme, 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. Thus, publishers can indirectly publish events to subscribers, and subscribers can indirectly subscribe their interests to publishers. 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 to them. Event subscription and publication are performed asynchronously.

This paper focuses on three key issues that the pub/sub scheme faces for event publication in WSNs. The first issue is that, in event publication, there exist inherent tradeoffs among conflicting performance 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 event data) from publishers to subscribers. However, this can degrade data fidelity (the quality of event data; e.g., event data freshness, or latency) and power efficiency. For improving data fidelity, publishers may transmit 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 often sleep for a long time; however, data yield and data fidelity can degrade because of frequent retransmissions of event data. Thus, in WSNs, the pub/sub scheme is required to balance the tradeoffs among conflicting performance objectives and find the optimal tradeoffs.

The second issue is that event publication often requires certain quality of service (QoS) in WSNs; e.g., power consumption and latency. The pub/sub scheme needs to satisfy given QoS requirements while balancing tradeoffs among conflicting performance objectives.

The third issue is that WSNs are inherently dynamic and noisy; noise always exists in measuring performance objective values (i.e., event publication performance). It is hard to predict and model the noise due to dynamics and uncertainty in, for example, routing structure, duty cycle pattern, packet loss rate and event occurrence pattern. Therefore, in WSNs, the pub/sub scheme is required to reduce the impacts of noise in measuring performance objective values in event routing.

In order to address the above three issues, this paper investigates event publication in TinyDDS, which is pub/sub middleware for WSNs. TinyDDS provides an event routing protocol that performs self-adaptive event publication. A component in TinyDDS, called La Niña, implements this protocol. It self-adapts the protocol's parameters to dynamic network conditions by reducing the impacts of noise on performance evaluation and seeking the optimal tradeoffs among performance objectives under given QoS requirements.

As an inspiration to design La Niña, the authors of the paper observe that various biological systems have developed the mechanisms to overcome the three issues described above. For example, a bee colony self-adapts to balance conflicting objectives simultaneously for maintaining its well-being. Those objectives include maximizing the amount of collected honey, maintaining the temperature in a nest and minimizing the number of dead drones. If bees focus only on foraging, they fail to ventilate their nest and remove dead drones. In noisy environments, a bee colony balances these tradeoffs under several constraints (e.g., the minimum amount of honey stored in a nest). Given this observation, La Niña applies key biological mechanisms to implement its event routing protocol.

Figure 1 overviews TinyDDS and La Niña. TinyDDS runs atop TinyOS on each sensor node. Currently, base stations act as subscribers, and individual nodes act as publishers. Each base station subscribes its interest in a particular event to nodes, and a node publishes it to a base station when it is detected in sensor reading. La Niña consists of agents, La Niña runtime and La Niña server. Agents and La Niña runtime are modeled after bees and flowers, respectively. When an event is detected, an agent (bee) obtains it as honey on the local La Niña runtime (flower) and carries it to a base station on a hop-by-hop (flower-to-flower) basis, in turn, to La Niña server, which is modeled after a nest of bees. Agents perform this event routing by autonomously sensing their local and surrounding network conditions (e.g., network traffic and node/link failures) and adaptively invoking their biological behaviors such as pheromone emission, reproduction, migration and death.

La Niña implements a noise-aware and constraint-based evolutionary multiobjective

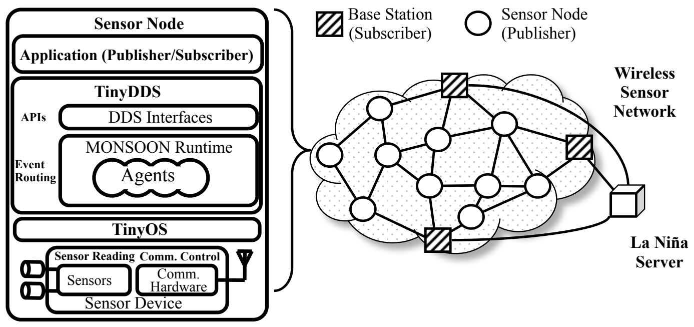


Figure 1 An Architectural Overview of TinyDDS and La Niña

optimization algorithm for agents to perform self-adaptive event routing. Each agent has its own behavior policy, as a set of *genes*, which defines when to and how to invoke its behaviors. La Niña allows agents to evolve their behavior policies via genetic operations (crossover and mutation) and adapt them to performance objectives in event routing. Currently, La Niña considers six objectives related to data yield, data fidelity and power consumption. To consider noise in measuring performance objective values, La Niña examines the level of confidence to evaluate each objective value and compare it with others.

La Niña frees WSN application developers from anticipating all possible network conditions and manually tuning protocol parameters to the conditions at design time. Instead, La Niña autonomously evolves and tunes its parameters (i.e., behavior policies of agents) at runtime. This can significantly simplify protocol implementation and maintenance.

La Niña also allows agents to adapt their behavior policies based on given constraints. Each constraint represents a QoS requirement; it is defined as an upper or lower bound of QoS. For example, a tolerable (upper) bound may be defined as a latency requirement. This feature allows application developers to flexibly specify their QoS requirements. Moreover, it often improves evolution speed by dedicating agents to satisfy given constraints.

2 Background: TinyDDS

TinyDDS is a lightweight implementation of the Data Distribution Service (DSS) specification, which Object Management Group (OMG) standardizes for pub/sub middleware (Object Management Group, 2007b)[a]. DDS provides standard middleware interfaces for event subscription and publication in the OMG Interface Definition Language (IDL) TinyDDS implements them with nesC, a dialect of the C language (Figure 1). See (Boonma and Suzuki, 2008b) for the IDL-to-nesC mapping in TinyDDS.

Besides DDS interfaces, the DDS specification standardizes no specific algorithms or protocols for event subscription and publication; they are left to DDS implementations. TinyDDS implements La Niña for its event publication. The La Niña runtime assumes B-MAC in TinyOS and uses a subset of the OMG General Inter-ORB Protocol (GIOP) (Object Management Group, 2007a) to transmit data types in DDS interfaces (e.g., events) as well as data types in La Niña (e.g., agents and pheromones). The La Niña runtime is implemented in nesC, while the La Niña server is implemented in Java. See (Boonma and Suzuki, 2009) for implementation details of TinyDDS. The current codebase of TinyDDS contains 2,035 lines of nesC code and 4,060 lines of Java code.

[a]TinyDDS is freely available at http://dssg.cs.umb.edu.

3 Agents and La Niña Runtime

An agent is initially deployed with a randomly-generated behavior policy on the La Niña runtime at each node. Each agent collects sensor data on a node at each duty cycle, and if it detects an event in its sensor reading, it carries the event data to a base station.

3.1 Agent Behaviors

Each agent implements seven behaviors and performs them in the following order at each duty cycle.

Step 1: Energy gain. If an event is detected, each agent gains *energy*. In La Niña, the concept of energy does not represent the amount of physical battery power in a node. Instead, it is a logical concept that impacts agent behaviors. Upon an event detection, an agent updates its energy level with a constant energy intake (E_F):

$$E(t) = E(t-1) + E_F \tag{1}$$

$E(t)$ and $E(t-1)$ denote the energy levels in the current and previous duty cycles.

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 La Niña runtime removes the agent and releases all resources allocated to it[b]

Step 3: Replication. Each agent makes a copy of itself if an event is detected in Step 1. 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, while the parent stays at the local node to detect further events.

Step 4: Swarming. Each agent may swarm (or merge) with others at intermediate nodes on its way to a base station. On each intermediate node, it 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 migrating toward the 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 those aggregating agents in terms of performance objectives. (See Section 4 on how to find the best-performing 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 the next-hop node in its migration 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

[b]If all agents are dying on a node at the same time, a randomly selected agent will survive.

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 a next-hop node that an agent has migrated to. Agents may 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 signal noise Each alert pheromone references a 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 2 to determine which next-hop node it migrates to.

$$WS_j = \sum_{t=1}^{3} w_t \frac{P_{t,j} - P_{t_{min}}}{P_{t_{max}} - P_{t_{min}}} \qquad (2)$$

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, respectively. $P_{t_{max}}$ and $P_{t_{min}}$ denote the maximum and minimum concentrations of P_t among all neighboring nodes.

The weight values in Equation 2 ($w_t, 1 \leq t \leq 3$) 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 emits a migration pheromone on the local node at the probability of $1 - p_s$. If the agent's migration fails, it emits an alert pheromone at the probability of $1 - p_s$. Each pheromone spreads to one-hop away neighboring nodes. p_s indicates *selfishness* of an agent. If an agent is selfish with a high p_s value, it often emits no pheromones for saving its energy. (Agents expend energy to perform behaviors, as discussed above.)

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 a set of behavior probability values (p_m and p_s) and a set of weight values in Equation 2 ($w_t, 1 \leq t \leq 3$). Behavior probability values are non-negative between zero and one. p_m is used for each agent to decide whether it performs the migration behavior or swarming behavior. With a higher migration probability, an agent has a higher chance to perform the migration behavior. With a higher selfishness probability (p_s), an agent has a higher chance not to emit pheromones.

3.3 La Niña Runtime

Each La Niña runtime provides a set of middleware services for the agents running on the local host (Figure 1). For example, they implement agent behaviors as reusable services, maintain a set of neighboring nodes within the local node's communication range and manage the pheromones emitted on the local node. Also, each platform is responsible of controlling the local node's duty cycle by turning it on and off based on its sleep period.

4 Evolutionary Noise-aware Multiobjective Optimization with La Niña

The evolutionary optimization process in La Niña consists of *elite selection* and *genetic operations*, which are performed by the La Niña server and each node, respectively. The elite selection process evaluates the agents that arrive at base stations, based on given performance objectives, and chooses the best (or elite) ones. The La Niña server propagates elite agents 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 propagated elite agents. A child agent inherits behavior policies (genes) from its parents via crossover. In addition, mutation may occur on the child's behavior policy.

Reproduction is intended to evolve agents so that the agents that fit better to the network environment become more abundant. It retains the agents with high fitness to the current network conditions (i.e., agents that have effective behavior policies, such as moving toward a base station in a short latency). It also eliminates the agents with low fitness (i.e., 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 a population of agents while ineffective ones become dormant or extinct. This allows agents to adapt to dynamic network conditions.

4.1 Performance Objectives and Constraints

Each agent considers six conflicting performance objectives related to data yield, data fidelity and power consumption: latency, cost, success rate, the degree of data aggregation, sleep period and selfishness. Success rate and the degree of data aggregation impact data yield. Latency impacts data fidelity. Cost, sleep period and selfishness impact power consumption. La Niña strives to minimize latency, cost and sleep period and maximize success rate, the degree of data aggregation and selfishness. Each performance objective can have an associated constraint, which represents a QoS requirement. La Niña eliminates agents who violate given constraints in its optimization process.

(1) Latency (L) represents the time required for an agent to travel to a base station from a node where the agent is replicated. As depicted below, it is measured as a ratio of this agent travel time (t) since the agent is replicated until reaching a base station to the physical distance (d) between the base station and a node where the agent is replicated.

$$L = \frac{t}{d} \tag{3}$$

The La Niña server knows the approximated location of each node with a localization mechanism that performs a triangulation based on signal strength between nodes.

(2) Success rate (S) is measured as the ratio of the number of successful data transmissions (N_{succ}) to the total number of node-to-node data transmissions required for an agent to arrive at a base station (n_{tran}).

$$S = \frac{n_{succ}}{n_{tran}} \tag{4}$$

(3) Cost (C) represents power consumption required for an agent to travel to a base station from a node where it is replicated. C is measured with d, n_{tran} and each node's radio communication range (r). n_{tran} is considered because data transmission incurs the highest power consumption among other operations in a node (Shnayder et al., 2004).

$$C = \frac{n_{tran}}{d/r} \tag{5}$$

The total number of data transmissions counts successful and unsuccessful (failed) migrations of an agent as well as the transmissions of its migration and alert pheromones. The theoretical lower bound of cost is one; one agent migration (i.e., data transmission) per each node's communication range without emitting pheromones.

(4) Degree of data aggregation is measured as the number of sensor data in an agent. It is more than one in a swarming agent. The larger it is, the more often agents swarm. This results in less power consumption of nodes because of a less number of agents and agent migrations in the network. On the contrary, a large degree of data aggregation can degrade latency because agents wait for other agents to swarm on nodes.

(5) Sleep period is the period for which a node sleeps between two duty cycles. The longer a node sleeps, the less amount of power it consumes. However, it can degrade latency because a sleeping node receives and sends out no agents. It can also increase power consumption of the other nodes if they re-transmit agents to the sleeping node.

(6) Selfishness controls the level of cooperation among agents. Selfish agents can reduce power consumption of nodes because they emit pheromones less often. However, the other agents lose the opportunities to leverage pheromones in their migration decisions. For example, if a selfish agent does not emit an alert pheromone when it fails to migrate to a node, the other agents may try to migrate to the node and fail to do so. This can increase power consumption due to failed agent migrations and degrade success rate and latency.

4.2 Confidence-based Domination (α-domination) and Constraint Violation (α-violation)

In its elite selection, La Niña compares agents that arrive at base stations and select elite agents based on their objective values (i.e., event routing performance). For this comparison, the notion of domination ranking is commonly used in evolutionary multiobjective optimization algorithms. One of the most well-known domination operators (e.g., Srinivas and Deb (1995)) determines that agent A dominates agent B, i.e., $A \succ B$, iif:

- A's objective values are better than, or equal to, B's in all objectives, and
- A's objective values are better than B's in at least one objective.

This domination operator ranks agents based on domination relationships among agents. Non-dominated agents have the highest rank, and in general, they are elite agents.

This classical domination operator does not work well in noisy problems. As discussed in Section 1, noise considerably exists in evaluating objective values and constraint (QoS) violation in WSNs. This means that, even if agents have the exactly same behavior policy, they can yield different objective values and different levels of constraint violation depending on network conditions. Therefore, La Niña employs new statistical operators to reduce the impacts of noise on evaluating objective values and constraint violation.

The proposed operator, called *α-domination operator*, determines the domination relationship between two agents by statistically processing multiple sample sets of objective values. With this operator, agent A is said to *α-dominate* agent B, i.e., $A \succ_\alpha B$, iif:

- A's and B's samples are classifiable with the statistical confidence level of α, and
- $C(A,B) = 1 \wedge C(B,A) >= 0$.

In order to examine the first condition, the α-domination operator classifies A's and B's samples with Support Vector Machine (SVM) and measures classification error (Step 1 in Figure 2) The error (e) is calculated as the ratio of the number of miss-classified samples to the total number of samples. For considering confidence level (α) in classification error, the α-domination operator uses the confidence interval of classification error (e_{int}):

$$e_{int} = e \pm t_{\alpha,n-1}\hat{\sigma} \tag{6}$$

$t_{\alpha,n-1}$ represents a $t-$distribution with α confidence level and $n-1$ degrees of freedom. σ is the standard deviation of classification errors. It is approximated as follows. n denotes the total number of samples.

$$\sigma \cong \sqrt{\frac{e}{n}} \tag{7}$$

If e_{int} is significant (i.e., if e_{int} does not span zero), the α-domination operator cannot classify A's and B's samples with the confidence level of α. If e_{int} spans zero, the operator can classify A's and B's samples with the confidence level of α. In between the two cases, the operator can classify A's and B's samples with a small enough classification error. See also Figure 2 for these three cases.

In order to examine the second condition in α-domination, the α-domination operator measures C-matric (Zitzler and Thiele, 1999) with a classical domination operator described above. $C(A,B)$ denotes the fraction of agent B's samples that at least one sample of agent A dominates:

$$C(A,B) = \frac{|\{b \in B \mid \exists a \in A : a \succ B\}|}{|B|} \tag{8}$$

If $C(A,B) = 1$, all of B's samples are dominated by at least one sample of A.

The α-domination operator determines $A \succ_\alpha B$ if $C(A,B) = 1$ and $C(B,A) \geq 0$ as far as A's and B's samples are classifiable. On the other hand, if $C(A,B) \geq 0$ and $C(B,A) \geq 0$, the α-domination operator determines neither $A \succ_\alpha B$ nor $B \succ_\alpha A$. See also Figure 2.

Figure 2 shows an example α-domination with two objectives, f_1 and f_2, to be minimized. Agent A and B have seven samples each (black circles and gray squares). The first step is to classify these 14 samples with SVM and calculate e_{int}. SVM provides a classification vector in the objective space as a boundary to classify samples. In Figure 2, two samples of agent B are miss-classified; e is $\frac{2}{14}$ (0.143). Thus, $\sigma \cong \sqrt{\frac{0.143}{14}} = 0.1$. Assuming $\alpha = 0.95$, $e_{int} = 0.143 \pm 1.771 * 0.1 = 0.143 \pm 0.1771$. Since e_{int} spans zero, the α-domination operator can classify A's and B's samples with the confidence level of 95%. The second

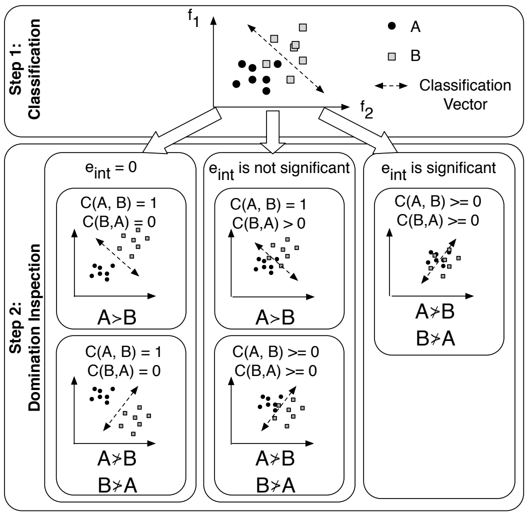


Figure 2 An Example α-domination

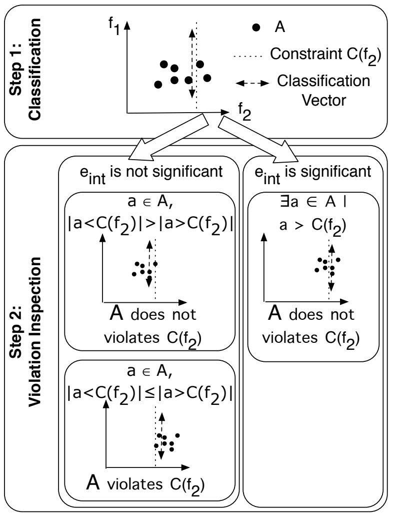


Figure 3 An Example α-violation

step is to calculate $C(A,B)$ and $C(B,A)$. In Figure 2, $C(A,B) = 1$ and $C(B,A) = 2/14$ (0.143). Therefore, the α-domination operator determines $A \succ_\alpha B$.

In addition to the α-domination operator, La Niña provides the *α-violation operator* to statistically determine constraint violation of agents. The operator chooses an agent whose at least one sample violates a constraint. Then, it classifies the agent's samples and the closest point(s) on a violated constraint front(s) (Step 1 in Figure 3). It also calculates the confidence interval of classification error (e_{int}) with Equation 6. In the same way as the α-domination operator does, the α-violation operator examines if it succeeds its classification with the confidence level of α.

Figure 3 shows an example α-volation with two objectives, f_1 and f_2, to be minimized. A constraint (upper bound) is given on f_2: $C(f_2)$. Agent A has seven samples (black circles), and two of them violate the constraint. The first step is to classify these 7 samples with SVM and calculate e_{int}. SVM provides a classification vector in the objective space as a boundary to classify samples and the constraint front. In Figure 3, two samples of agent A are miss-classified ; e is $\frac{2}{7}$ (0.286). Thus, $\hat{\sigma} \cong \sqrt{\frac{0.286}{7}} = 0.2$. Assuming $\alpha = 0.95$, $e_{int} = 0.286 \pm 1.943 * 0.2 = 0.286 \pm 0.3886$. Since e_{int} spans zero, the α-violation operator classifies A's samples and the constraint front with the confidence level of 95%. The second step is to count the number of samples in A that do not violate ($|a < C(f_2)|$) and violate ($|a \geq C(f_2)|$) the constraint. Because the number of samples that do not violate the constraint is greater than the number of samples that violate the constraint; therefore, the α-violation operator determines that A does not violate the constraint on f_2.

4.3 Elite Selection

Figure 4 shows how the La Niña server periodically performs elite selection. The first step is to measure six objective values of each agent that arrives at base stations. If an agent α-violates at least one of constraints, it is eliminated. Remaining agents examine *α-domination* relationships among them.

Then, a subset of non-dominated agents is selected as elite agents. This is performed with the objective space; a six dimensional hypercube space whose axes represent six objectives. Each axis is divided between the maximum and minimum objective values of non-dominated agents so that the space contains small cubes. Each non-dominated agent is plotted in the objective space based on their objective values. If multiple agents are plotted in the same cube, a single agent is randomly selected as an elite agent. If no agents are plotted in a cube, no elite agent is selected from the cube. This hypercube-based elite selection is designed to maintain the diversity of elite agents. Diversity can improve agent adaptability even to unanticipated network conditions.

```
Empty the archive
while true
do
    Empty the population pool.
    Collect the agents that have arrived at base stations.
    Add collected agents to the population pool.
    Move agents from the archive to the population pool.
    Empty the archive
    for each agent of in the population pool
    do
        Obtain the agent's objective values.
        if one or more objective values $\alpha$-violate constraints
            then Remove the agent from the population pool.
    for each agent in the population pool
    do
        if not $\alpha$-dominated by any other agents
         in the population pool,
            then Add the agent to the archive.
    Select elite agents from the archive.
    Propagate elite agents (their behavior policies),
     mutation rate and sleep period to individual nodes.
    Sleep for the sleep period.
```

Figure 4 Elite Selection in the La Niña Server

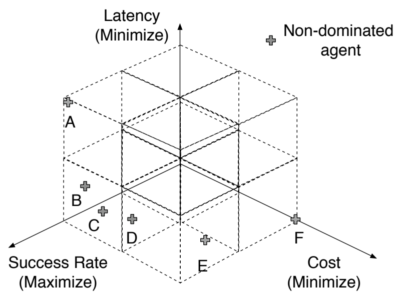


Figure 5 An Example Elite Selection

Figure 5 shows an example hypercube that shows three objectives (success rate, cost and latency). It is divided to two ranges over each objective; eight cubes exist in total. Thus, the maximum number of elite agents is eight. In this example, six (A to F) non-dominated agents are plotted. From the lower left cube, one agent is randomly selected as an elite agent. A, E, and F are selected as elite agents because they are in different cubes.

As Figure 4 shows, La Niña performs elitism based on a $(\mu + \lambda)$ evolution strategy. Non-dominated agents compete with (or are evaluated together with) the agents that arrive at base stations in the future. Elitism is intended to improve evolution speed of agents.

In addition to select elite agents, the La Niña server adjusts the mutation rate of agents and the sleep period of nodes. Mutation rate is adjusted based on the disorderliness of the current population of agents in the objective space. If agents are disordered (or sparse) in the objective space, it indicates that agents have not adapted to the current network conditions yet. In other words, they are still distant to the Pareto optimal front or unable to form the Pareto front yet. Thus, in this case, La Niña uses a high mutation rate to have agents evolve further. In contrast, if agents are ordered (or dense) in the objective space, it indicates that they are close to the Pareto optimal front. Thus, La Niña uses a low mutation rate to suppress agent evolution. This also helps agents reduce the fluctuation in their objective values. La Niña calculates the disorderliness of agents by measuring entropy of agents (H) in the objective space:

$$H = -\sum_{i \in C} P(i) \log_2 \left(\frac{n_i}{\sum_{i \in C} n_i}\right) \quad (9)$$

C is a set of cubes in the objective space. (In Figure 5, C is a set of eight cubes.) $P(i)$ denotes the probability that an agent is in cube i. n_i is the number of agents in cube i. Then, H is normalized with the maximum entropy value:

$$H_o = \frac{H}{H_{\max}} = \frac{H}{\log_2 n} \tag{10}$$

WIth normalized entropy (H_o), mutation rate m is calculated as follows.

$$m = m_{\max} \times \sqrt{1-(1-H_o)^2} \tag{11}$$

$m_{\max}$ denotes the maximum mutation rate. Mutation rate is adjusted in a non-linear manner based on the current normalized entropy of agents.

Sleep period is adjusted in a stepwise manner in between the predefined minimum and maximum values. When agents break at least one of latency, cost and success rate constraints, the sleep period is decreased by one minute; otherwise, increased by one minute.

The La Niña server propagates adjusted mutation rate and sleep period as well as elite agents (their behavior policies) to individual nodes in the network. This propagation is performed with a base station pheromone. When a node receives an adjusted sleep period, it's local La Niña runtime changes its current sleep period accordingly.

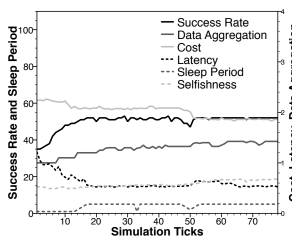
4.4 Genetic Operations

Upon receiving a base station pheromone, an agent performs the reproduction behavior on each node with a certain reproduction rate. It selects one of propagated elite agents, as a mating partner, which has the most similar genes (behavior policy). Gene similarity is measured with the Euclidean distance between two sets of gene values. If two or more elite agents have the same similarity, one of them is randomly selected. In reproduction, a child agent performs one-point crossover; it randomly inherits the half of its genes from its parent agent and the other half from the parent's mating partner. If reproduction does not occurs, the most similar elite agent replaces a parent agent on each node.

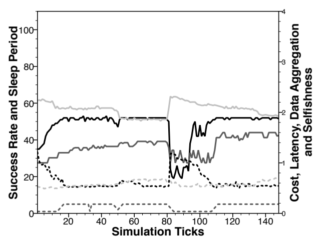
Mutation occurs on a child agent's genes, with a certain mutation rate, by randomly changing gene values within a predefined value range. As described in Section 4.3, mutation rate is periodically adjusted by the La Niña server and propagated to individual nodes. After reproduction, a child agent takes over its parent as the next generation agent.

5 Simulation Evaluation

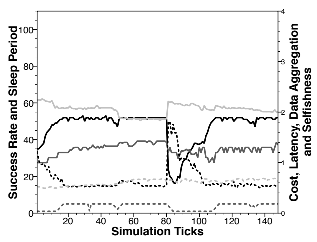
This section shows a series of simulation results to evaluate La Niña. All simulations were carried out with the TOSSIM simulator. A simulated WSN consists of 100 nodes deployed uniformly in a 300 meters x 300 meters observation area. Each node's communication range is 30 meters. At the beginning of each simulation, a simulated forest fire starts at the middle of the observation area and spreads throughout the area. A base station is deployed at the northwestern corner of the observation area. The base station connects to the La Niña server via emulated serial port connection. The initial sleep period is one minute on each node, and its minimum and maximum are one and five minutes, respectively. For genetic operations, the reproduction rate is 0.5, and the maximum mutation rate ($m_{\max}$ in Equation 11) is 0.2. The constraint (lower bound) of success rate in data transmission is 0.5. The confidence level of 95% is used in α-domination and α-violation operators.



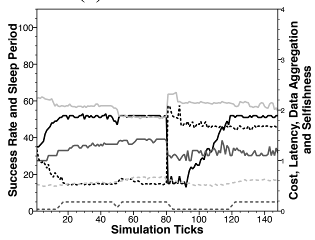
(a) Static Network



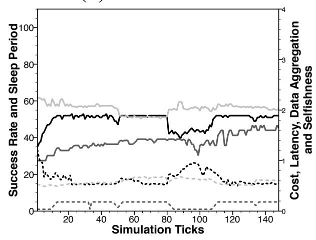
(b) Node Addition



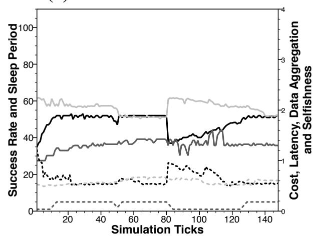
(c) Random Node Failure



(d) Noise Addition



(e) Base Station Failure



(f) Network Split

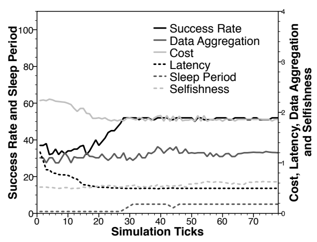
Figure 6 Event Routing Performance with a Success Rate Constraint

5.1 Event Routing Performance with a Single Constraint

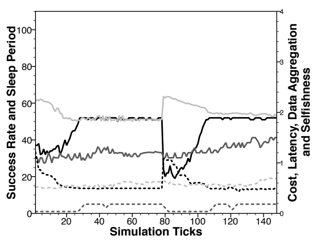
Figure 6 shows the average objective values that agents yield with a constraint for success rate. Each simulation tick represents a duty cycle. Figure 6 (a) shows a result in a static network where no dynamic changes occur in the network. All objective values improve and converge by the 65th tick. This demonstrates that La Niña allows agents evolve their behavior policies and autonomously improve their performance under conflicting objectives. Note that agents always satisfy a given success rate constraint since the 20th tick.

Figures 6 (b) to (f) show how agents perform when network conditions change at the 80th tick. In Figure 6 (b), 25 nodes are added at random locations. As a result, objective values degrade because agents initially have random behavior policies on new nodes. Those agents cannot migrate efficiently to the base station. Also, pheromones are not available on new nodes; agents cannot make proper migration decisions on those nodes. In Figure 6 (c), when 25 nodes are randomly removed, objective values degrade because agents try to migrate to the missing nodes. Those agents cannot migrate efficiently to the base station. In Figure 6 (d), node-to-node packet loss rate increases from 0.05 to 0.2. Thus, agents fail migration more often. In Figure 6 (e), two base stations are initially deployed at the northwestern and southeastern corners of the observation area. When the southeastern base station fails, objective values drop because some agents migrate to the failed base station. In Figure 6 (f), two base stations are deployed as in Figure 6 (e). Then, the network is split at the middle and separated into two sub-networks. As a result, objective values drop because some agents try to migrate to the base station on the other sub-network.

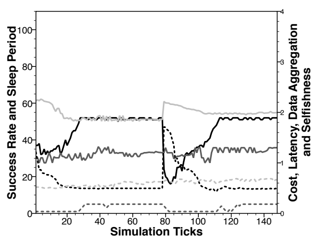
As Figures 6 (b) to (f) show, La Niña allows agents to autonomously adapt to dynamic changes in the network and recover their performance with a given success rate constraint satisified. Objective values are mostly same before and after each dynamic change. In Figure 6 (b), upon node addition, agents yield a higher degree of data aggregation because more agents migrates in the network and they can aggregate more often.



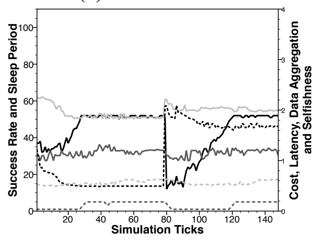
(a) Static Network



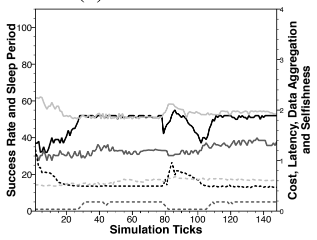
(b) Node Addition



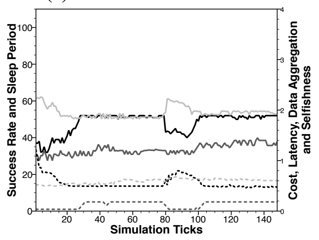
(c) Random Node Failure



(d) Noise Addition



(e) Base Station Failure



(f) Network Split

Figure 7 Event Routing Performance with Success Rate, Latency and Cost Constraints

5.2 Event Routing Performance with Multiple Constraints

Figure 7 shows the average objective values that agents yield with constraints for success rate, latency and cost. Latency and cost constraints are 0.5 second per 30 meters and 2.0 transmissions, respectively. All other simulation configurations are same as the ones in Section 5.1. In Figure 7 (a), success rate, latency and cost improve faster than the other objectives because agents strive to satisfy given constraints. In fact, latency and cost improve faster and remain more stable than they do in Figure 6 (a). They are always around or lower than their constraints since the 20th tick, and success rate is always higher than its constraint since the 30th tick. La Niña allows agents to evolve and autonomously improve their performance under conflicting objectives and satisfy multiple constraints.

In Figures 7 (b) to (f), agents perform similarly to Figures 6 (b) to (f) in that they exhibit self-healing and self-adaptation properties against dynamics of the network. Object values converge again after each dynamic change. Compared with Figures 6 (b) to (f), agents recover their latency and cost performance faster and retain them more stable by following given constraints. These results show that La Niña allows agents to autonomously adapt to dynamic network conditions and recover their performance under multiple constraints.

5.3 Impacts of Adaptive Mutation on Event Routing Performance

In order to evaluate the impacts of adaptive mutation on event routing performance, Figure 8 shows the average objective values that agents yield with adaptive mutation disabled. Mutation rate is fixed at 0.2. All other simulation configurations are same as the ones in Section 5.2. Compared with Figure 7, all objective values improve slower and fluctuate more. For example, in Figure 8 (b), it takes 40 simulation ticks for success rate to reach 50% while it takes only 30 ticks in Figure 7 (b). As discussed in Section 4.3, La Niña increases mutation rate to stimulate agent evolution when agents do not adapt well to the current network conditions. Thus, objective values converge faster. These results show that adaptive mutation in La Niña allows agents to evolve more efficiently and stably.

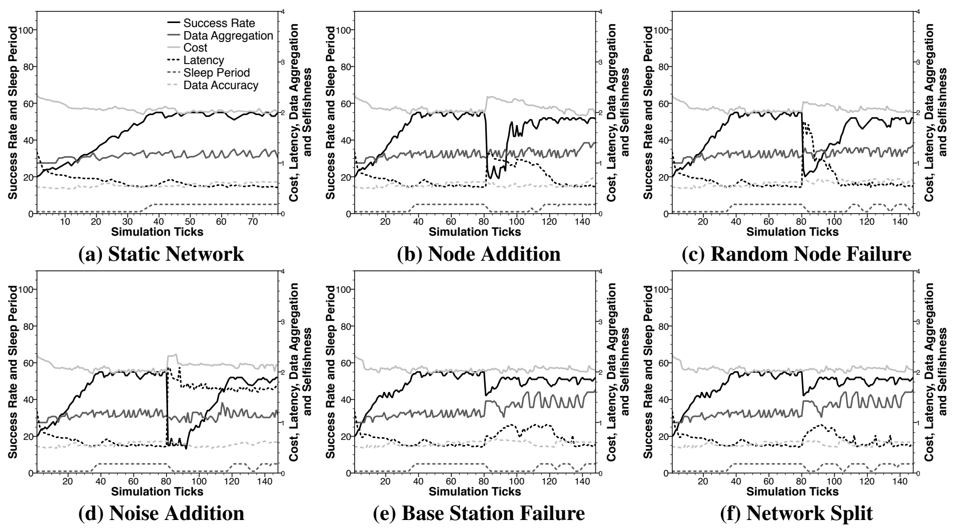


Figure 8 Event Routing Performance with Adaptive Mutation Disabled

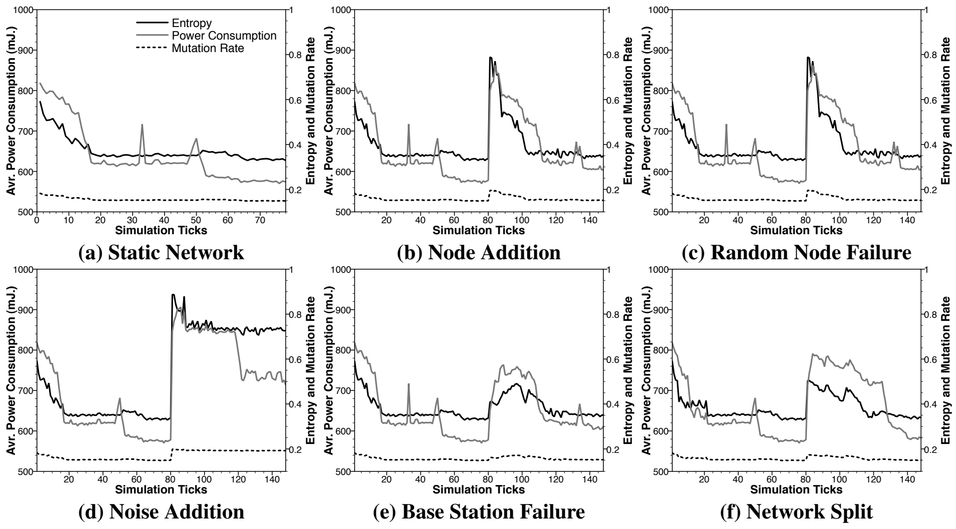


Figure 9 Mutation Rate, Power Consumption and Entropy

5.4 Mutation Rate, Power Consumption and Self-organization

Figures 9 shows the average amount of power that nodes consume as well as the average mutation rate that agents use. All simulation configurations are same as the ones used in Section 5.1. La Niña adjusts mutation rate lower as agents adapt to network conditions. This allows them to stabilize the fluctuation in their performance, as discussed in the previous section. Power consumption decreases as agents adapt to network conditions and sleep period increases. When a network condition(s) changes, the degree of agent adaptation drops. Sleep period also decreases because more agents violate constraints. As a result, power consumption increases upon a dynamic change in the network. However, power

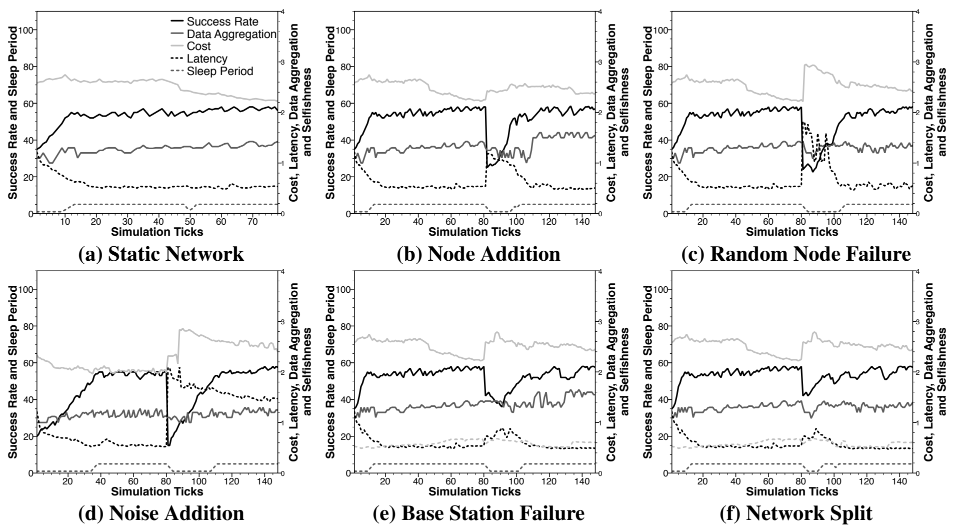


Figure 10 Impacts of Selfishness on Event Routing Performance

consumption goes down again. La Niña strives to minimize power consumption.

Figures 9 also shows the degree of self-organization in agent population. It is measured with normalized entropy, which is calculated with Equations 9 and 10. In this paper, normalized entropy indicates how similar performance different agents yield. The lower it is, the more similar performance agents yield. As Figures 9 shows, La Niña allows agents to adapt to network conditions and perform similar with each other. Entropy spikes upon a dynamic change in the network; however, it goes lower again through agent evolution.

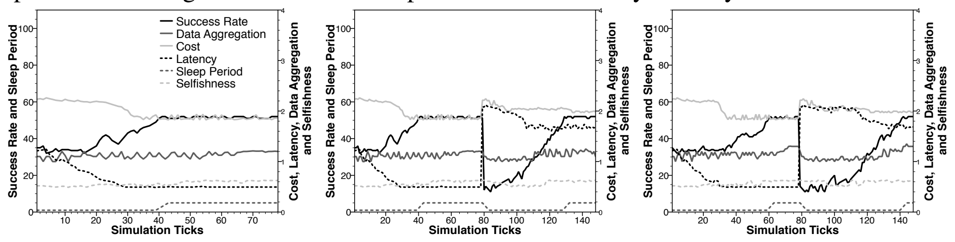
5.5 Impacts of Agent Selfishness on Event Routing Performance

As Section 3.1 describes, when its p_s is not zero, an agent becomes selfish and skips emitting migration and alert pheromones for reducing power consumption (i.e., the number of data transmissions). For example, in Figure 7 (a), agents increase their selfishness around the 50th tick. Correspondingly, power consumption decreases (Figure 9 (a)).

In order to evaluate the impacts of selfishness further, Figure 10 shows the average objective values that agents yield when they are not selfish at all (i.e., $p_s = 0$). Compared with Figure 7, agents incur higher costs because they always emit pheromones. For example, in Figure 10 (b), the average cost is approximately 2.25 transmissions per hop while it is less than 2.0 transmissions per hop in Figure 7 (b). This means that La Niña allows agents to effectively save power consumption by reducing the number of pheromone emissions.

5.6 Impacts of α-domination and α-violation on Event Routing Performance

For evaluating the impacts of α-dominanation and α-violation on event routing performance, Figure 11 shows the average objective values that agents yield with a traditional domination ranking (Srinivas and Deb (1995)). All other simulation configuration are same as the ones used in Section 5.2. Compared with Figure 7, all objective values improve slower in Figure 11. For example, in Figure 11 (b), it takes approximately 10 ticks longer for success rate to reach 50% than in Figure 7 (d). The α-domination and α-violation operators allow agents to evolve and operate more efficiently in noisy WSNs.

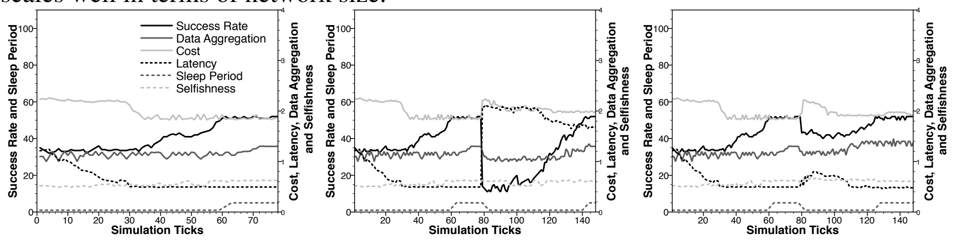


(a) Static Network **(b) Noise Addition** **(c) Network Split**

Figure 11 Event Routing Performance with α-domination and α-violation Disabled

5.7 *Impacts of Network Size on Event Routing Performance*

In order to evaluate the impacts of network size on event routing performance, Figure 12 shows the average objective values that agents yield when the network contains 400 nodes. All other simulation configuration are same as the ones used in Section 5.2. All objective values converge around the 65th tick and satisfy given constraints, although they improve slower than in Figure 7. Compared with Figure 7, convergence speed doubles approximately as network size becomes four times larger. These results show that La Niña scales well in terms of network size.



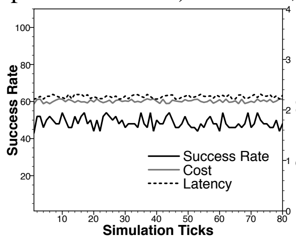
(a) Static Network **(b) Noise Addition** **(c) Network Split**

Figure 12 Event Routing Performance in a Network of 400 nodes

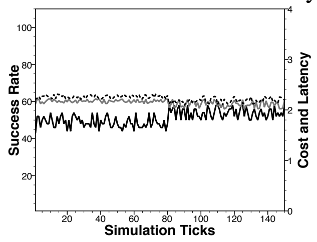
5.8 *Comparison with a Traditional Fault-tolerant Routing Algorithm*

In order to compare La Niña with a traditional routing algorithm, Figure 13 shows the average objective values with a fault-tolerant routing algorithm proposed by Gregoire and Koren (2007). The routing algorithm is designed to be tolerant against node and link failures by having each node examine whether its next-hop node forwards a packet to a next-next-hop node. If the next-hop node fails to do so, the packet is retransmitted to another next-hop node. Since this routing algorithm does not consider dynamic sleep period adjustment, data aggregation and selfishness of packets, Figure 13 shows the other three objective values only. The algorithm does not consider constraints either. All the other simulation configurations are same as the ones used in Section 5.1.

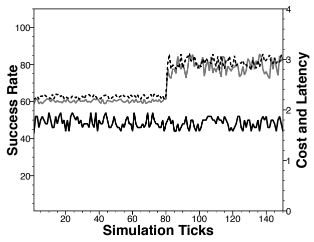
Compared with Figure 6, all objective values are less sensitive to dynamic changes in the network. However, they do not improve over time because no performance improvement mechanisms are considered. Latency and cost tend to be higher because hop-by-hop routing recovery (i.e., routing inspection and packet retransmission) is expensive. In contrast, La Niña can improve performance through evolution and reduce latency and cost lower than an algorithm proposed by Gregoire and Koren (2007). (For example, latency is up to 75% lower.) However, La Niña is more sensitive to dynamics of the network.



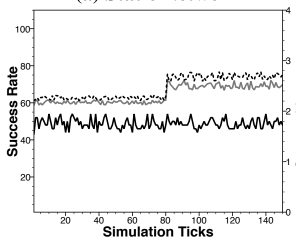
(a) Static Network



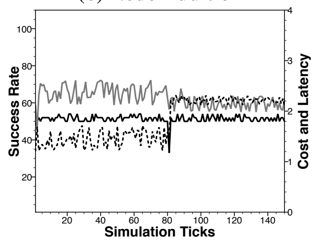
(b) Node Addition



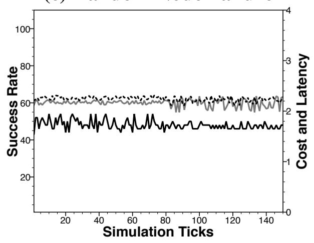
(c) Random Node Failure



(d) Noise Addition



(e) Base Station Failure



(f) Network Split

Figure 13 Event Routing Performance with a Traditional Fault-torelant Routing Protocol

5.9 Memory Footprint

Table 1 shows the memory footprint of TinyDDS on a MICA node. The total memory footprint is approximately 32 KB in flash memory and 3.5 KB in RAM. A fault-tolerant routing algorithm used in Section 5.8 consumes 24 KB in flash memory and 2.2 KB in RAM. The difference in memory footprint is quite small between this algorithm and TinyDDS. TinyDDS is lightweight enough to run on resource-limited nodes such as MICA2.

Software Component		Flash Memory (Bytes)	RAM (Bytes)
Pub/sub application		1,340	36
TinyDDS	DDS Interface	1,784	2,240
	La Niña Runtime	10,184	805
TinyOS		18,520	418
Total		31,828	3,499

Table 1 Memory Footprint of TinyDDS

6 Related Work

This paper describes a set of extensions to the authors' prior work (Boonma and Suzuki, 2008a,b). TinyDDS is originally proposed in (Boonma and Suzuki, 2008b); however, it does not consider self-adaptive event routing. An earlier version of La Niña is proposed in (Boonma and Suzuki, 2008a); however, it does not consider noisy WSNs. This paper investigates self-adaptive and noise-aware event routing in WSNs. In addition, this paper studies a new adaptive mutation method that is not studied in (Boonma and Suzuki, 2008a).

There exist several pub/sub middleware for WSNs (Marrón et al., 2005; Souto et al., 2005; Costa et al., 2007; Choon-Sung Nam, 2008; Hoffert et al., 2008). They allow application developers to reconfigure a series of configuration parameters. However, they do not consider self-adaptation of those parameters; developers need to manually conduct a time-consuming and error-prone process to optimize the parameters. In fact, these existing work do not consider dynamic WSNs, but assume static and noise-free WSNs. In contrast, TinyDDS is designed to inherently consider self-adaptation in dynamic and noisy WSNs.

Costa et al. (2005) and Carzaniga et al. (2004) propose pub/sub event routing protocols for WSNs. They consider dynamic network changes such as node/link failures. However, both protocols do not self-adapt to those changes; human administrators need to manually adjust protocol parameters in order to retain/improve event routing performance. Unlike them, TinyDDS can self-adapt to dynamic network conditions and dynamically improve event routing performance Moreover, TinyDDS assumes noise in monitoring network conditions, while Costa et al. (2005) and Carzaniga et al. (2004) do not.

Evolutionary multiobjective optimization algorithms (EMOAs) are used for routing (Rajagopalan et al., 2005; Xuea et al., 2006; Sin et al., 2008), node placement (Jia et al., 2008; Molina et al., 2008) and duty cycle management (Yang et al., 2007). Unlike La Niña, all of these work do not assume dynamic WSNs, but static WSNs. Rajagopalan et al. (2005) and Xuea et al. (2006) investigate EMOAs that optimize migration routes for mobile agents to travel from a base station to cluster head nodes and collect sensor data from clusters. In TinyDDS, La Niña allows agents to make their migration and other behavior decisions by themselves. La Niña optimizes their behavior policies, not agents' migration routes.

Mahjoub and El-Rewini (2007) propose a constraint-based EMOA for routing in WSNs. In routing a packet to a base station, each intermediate node decides the next-hop node by applying certain policies supplied by the central server. Human administrators manually define those policies by anticipating possible network conditions; thus, routing decision does not dynamically adapt to unanticipated network conditions. In contrast, La Niña allows agents to dynamically evolve their behavior policies and adapt to unanticipated network conditions without any intervention to/from human administrators.

Several EMOAs consider noise in objective functions (Teich, 2001; Lakshmikantha and Babbar, 2003; Eskandari et al., 2007). They assume that noise follows particular probability distribution such as normal distribution and uniform distribution. Given a probability distribution, each of existing noise-aware EMOAs statistically estimate each gene's objective value by collecting its samples. In contrast, La Niña assumes no probability distribution because, in general, it is hard to predict and model it in dynamic and noisy WSNs due to uncertainties such as packet loss rate and packet transmission pattern. Therefore, instead of estimating each gene/agent's objective values, La Niña measures the effect of noise and determines whether it is confident enough to compare genes/agents.

7 Conclusion

This paper proposes and evaluates a self-adaptive event routing in TinyDDS, a pub/sub middleware for WSNs. Its noise-aware and constraint-based evolutionary multiobjective optimization framework, La Niña, allows TinyDDS to autonomously adapt its routing parameters to dynamic network conditions by reducing the impacts of noise and seeking the optimal tradeoffs among conflicting performance objectives under given QoS requirements. Simulation results validate this ability of TinyDDS in various dynamic and noisy WSNs. TinyDDS is lightweight enough to run on resource-limited sensor nodes.

References

BANAVAR, G., CHANDRA, T. D., STROM, R. E., AND STURMAN, D. C. 1999. A case for message oriented middleware. In *Proc. of Int'l Symposium on Distributed Computing*.

BOONMA, P. AND SUZUKI, J. 2008a. Exploring self-star properties in cognitive sensor networking. In *Proc. of IEEE/SCS Int'l Symposium on Performance Evaluation of Computer and Telecommunication System*.

BOONMA, P. AND SUZUKI, J. 2008b. 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*.

BOONMA, P. AND SUZUKI, J. 2009. Toward interoperable publish/subscribe communication between wireless sensor networks and access networks. In *Proc. of IEEE Int'l Workshop on Information Retrieval in Sensor Networks*.

CARZANIGA, A., RUTHERFORD, M. J., AND WOLF, A. L. 2004. A routing scheme for content-based networking. In *Proc. of IEEE Conf. on Computer Communications*.

CHOON-SUNG NAM, HEE-JIN JEONG, D.-R. S. 2008. Design and implementation of the publish/subscribe middleware for wireless sensor network. In *Proc. of IEEE Int'l Conf. Networked Computing and Advanced Information Management*.

COSTA, P., COULSON, G., MASCOLO, C., MOTTOLA, L., PICCO, G. P., AND ZACHARIADIS, S. 2007. A reconfigurable component-based middleware for networked embedded systems. *Springer J. of Wireless Information Networks 14*.

COSTA, P., PICCO, G. P., AND ROSSETTO, S. 2005. Publish-subscribe on sensor networks: A semi-probabilistic approach. In *Proc. of Int'l Conf. Mobile Ad-hoc and Sensor Sys.*

ESKANDARI, H., GEIGER, C., AND BIRD, R. 2007. Handling uncertainty in evolutionary multiobjective optimization: SPGA. In *Proc. of IEEE Congress on Evolutionary Comp.*

EUGSTER, P. T., FELBER, P. A., GUERRAOUI, R., AND KERMARREC, A.-M. 2005. The many faces of publish/subscribe. *ACM Computing Surveys 35,* 2.

GREGOIRE, M. AND KOREN, I. 2007. An adaptive algorithm for fault tolerant re-routing in wireless sensor networks. In *Proc. of IEEE Int'l Conf. on Pervasive Comp. and Comm.*

HADIM, S. AND MOHAMED, N. 2006. Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online 7,* 3.

HENRICKSEN, K. AND ROBINSON, R. 2006. A survey of middleware for sensor networks: state-of-the-art and future directions. In *Proc. Int'l Wksp Middleware for Sensor Nets*.

HOFFERT, J., BALAKRISHNAN, M., SCHMIDT, D., AND BIRMAN, K. 2008. Supporting large-scale continuous stream datacenters via pub/sub middleware and adaptive transport protocols. In *Proc. of ACM Workshop on Large-Scale Distributed System and Middleware*.

JIA, J., CHEN, J., CHANG, G., AND TAN, Z. 2008. Energy efficient coverage control in wireless sensor networks based on multi-objective genetic algorithm. *Elsevier Computers & Mathematics with Applications 10*.

Jourdan, D. B. and de Weck, O. L. 2004. Multi-objective genetic algorithm for the automated planning of a wireless sensor network to monitor a critical facility. In *Proc. of SPIE Defense and Security Symposium*.

Lakshmikantha, A. and Babbar, M. 2003. A modified NSGA-II to solve noisy multi-objective problems. In *Proc. of ACM Genetic and Evolutionary Computation Conf.*

Mahjoub, D. and El-Rewini, H. 2007. Adaptive constraint-based multi-objective routing for wireless sensor networks. In *Proc. of IEEE Int'l Conf. on Pervasive Services*.

Marrón, P. J., Lachenmann, A., Minder, D., Gauger, M., Saukh, O., and Rothermel, K. 2005. Management and configuration issues for sensor networks. *Wiley Int'l J. of Network Management 15,* 4.

Molina, G., Alba, E., and Talbi, E.-G. 2008. Optimal sensor network layout using multi-objective metaheuristics. *J. of Universal Computer Science 14,* 15, 2549–2565.

Object Management Group. 2007a. Common object request broker architecture (CORBA) specification, version 3.1; part 2: CORBA interoperability.

Object Management Group. 2007b. Data Distribution Service for real-time systems, v1.2.

Raich, A. M. and Liszkai, T. R. 2003. Multi-objective genetic algorithm methodology for optimizing sensor layouts to enhance structural damage identification. In *Proc. of Int'l Workshop on Structural Health Monitoring*.

Rajagopalan, R., Mohan, C. K., Varshney, P. K., and Mehrotra, K. G. 2005. Multi-objective mobile agent routing in wireless sensor networks. In *Proc. of IEEE Congress on Evolutionary Computation*.

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

Seeley, T. 2005. *The Wisdom of the Hive*. Harvard University Press.

Shnayder, V., Hempstead, M., rong Chen, B., Werner-Allen, G., and Welsh, M. 2004. Simulating the power consumption of large-scale sensor network applications. In *Proc. of ACM Conf. on Embedded Networked Sensor Systems*.

Sin, H., Lee, J., Lee, S., Yoo, S., Lee, S., Lee, J., Lee, Y., , and Kim, S. 2008. Agent-based framework for energy efficiency in wireless sensor networks. *World Academy of Science, Engineering and Technology 35*, 305–309.

Souto, E., Guimarães, G., Vasconcelos, G., Vieira, M., Rosa, N., Ferraz, C., and Kelner, J. 2005. Mires: a publish/subscribe middleware for sensor networks. *Springer J. of Personal Ubiquitous Computing 10,* 1.

Srinivas, N. and Deb, K. 1995. Multiobjective function optimization using nondominated sorting genetic algorithms. *MIT Evolutionary Computation J. 2,* 3.

Teich, J. 2001. Pareto-front exploration with uncertain objectives. In *Proc. of Int'l Conf. on Evolutionary Multi-Criterion Optimization*.

Wang, M.-M., Cao, J.-N., Li, J., and Dasi, S. K. 2008. Middleware for wireless sensor networks: A survey. *Springer J. of Computer Science 23,* 3.

Xuea, F., Sanderson, A., and Graves, R. 2006. Multi-objective routing in wireless sensor networks with a differential evolution algorithm. In *Proc. of IEEE Int'l Conf. on Networking, Sensing and Control*.

Yang, E., Erdogan, A. T., Arslan, T., and Barton, N. 2007. Multi-objective evolutionary optimizations of a space-based reconfigurable sensor network under hard constraints. In *Proc. of ECSIS Symp. on Bio-inspired, Learning, and Intelligent Sys. for Security*.

Yu, Y., Krishnamachari, B., and Prasanna, V. 2004. Issues in designing middleware for wireless sensor networks. *IEEE Network 18,* 1.

Zitzler, E. and Thiele, L. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evol. Comp. 3,* 4, 257–271.