MONSOON: A Coevolutionary Multiobjective Adaptation Framework for Dynamic Wireless Sensor Networks

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

Abstract

Wireless sensor applications (WSNs) are often required to simultaneously satisfy conflicting operational objectives (e.g., latency and power consumption). Based on an observation that various biological systems have developed the mechanisms to overcome this issue, this paper proposes a biologically-inspired adaptation mechanism, called MONSOON. MONSOON is designed to support data collection applications, event detection applications and hybrid applications. Each application is implemented as a decentralized group of software agents, analogous to a bee colony (application) consisting of bees (agents). Agents collect sensor data and/or detect an event (a significant change in sensor reading) on individual nodes, and carry sensor data to base stations. They perform these data collection and event detection functionalities by sensing their surrounding environment conditions and adaptively invoking biologically-inspired behaviors such as pheromone emission, reproduction and migration. Each agent has its own behavior policy, as a gene, which defines how to invoke its behaviors. MONSOON allows agents to evolve their behavior policies (genes) and adapt their operations to given objectives. Simulation results show that MONSOON allows agents (WSN applications) to simultaneously satisfy conflicting objectives by adapting to dynamics of physical operational environments and network environments (e.g., sensor readings and node/link failures) through evolution.

1. Introduction

Autonomous adaptability is a key challenge in wireless sensor networks (WSNs) [1–4]. With minimal intervention to/from human operators, WSN applications are required to adapt their operations to dynamic changes in physical operational environments (e.g., sensor readings) and network environments (e.g., network traffic and node/link failures). A critical issue in this challenge is that each WSN application tends to have conflicting operational objectives. For example, the success rate of data transmissions from individual nodes to base stations is an important objective because higher success rate ensures that base stations have more data for operators to better understand a physical operational environment and make better informed decisions. At the same time, the latency of data transmissions from individual nodes to base stations is another important objective. Lower latency ensures that base stations can collect sensor data for operators to understand a physical operational environment more quickly and make more timely decisions. Success rate and latency conflict with each other. For improving success rate, hop-by-hop recovery is often applied; however, this can degrade latency. For improving latency, nodes may transmit data to base stations with the shortest paths; however, success rate can degrade because of traffic congestion on the paths.

In order to address this adaptability issue, the authors of the paper envision autonomous WSN applications that understand their operational objectives and simultaneously satisfy them against the dynamics of network environments. Toward this vision, the authors observe that various biological systems have developed the mechanisms to overcome the above adaptability issue. For example, each bee colony autonomously satisfies conflicting objectives to maintain its well-being [5]. Those objectives include maximizing the amount of collected honey, maintaining temperature inside 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. Given this observation, the proposed application architecture, called BiSNET/e (Biologically-inspired architecture for Sensor NETworks, evolutionary edition), applies key biological mechanisms to implement adaptive WSN applications.

Figure 1 shows the BiSNET/e runtime architecture. The BiSNET/e runtime operates atop TinyOS on each node. It consists of two software components: *agents* and *middleware platforms*, which are modeled after bees and flowers, respectively. Each WSN application is designed as a decentralized group of agents. This is analogous to a bee colony (application) consisting of bees (agents). Agents collect sensor data and/or detect an event (a significant change in sensor reading) on platforms (flowers) atop individual nodes. Then, they carry sensor data to base stations, in turn, to a backend server (the MONSOON server in Figure 1), which is modeled after a nest of bees. Agents perform

1530-1605/08 $25.00 © 2008 IEEE

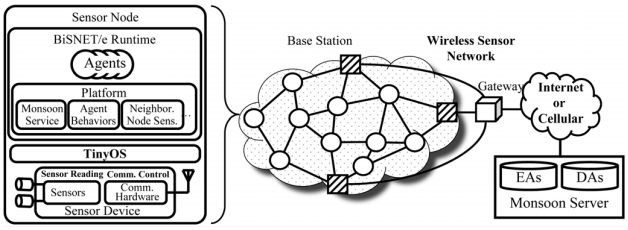

Figure 1. BiSNET/e Runtime Architecture

these data collection and event detection functionalities by autonomously sensing their surrounding environment conditions and adaptively performing biological behaviors such as pheromone emission, reproduction, migration, swarming and death. A middleware platform runs on each node, and hosts an arbitrary number of agents (Figure 1). It provides a series of runtime services that agents use to perform their functionalities and behaviors.

This paper describes a key mechanism in BiSNET/e, called MONSOON[1], which is an co-evolutionary adaptation framework for agents. Each agent possesses its own behavior policy, as a *gene*, which defines how to invoke its behaviors. MONSOON allows agents to evolve their behavior policies via genetic operations (mutation and crossover) across generations and simultaneously adapt the behavior policies to conflicting objectives in dynamic physical operational environments and network environments. Currently, MONSOON considers three objectives: success rate, latency and power consumption. The evolution process in MONSOON frees application designers from anticipating all possible environment conditions and tuning their agents's behavior policies to the conditions at design time. Instead, agents can autonomously evolve and tune their behavior policies. This significantly simplifies the implementation and maintenance of agents (i.e., WSN applications).

MONSOON supports data collection applications, event detection applications and hybrid applications. Different types of applications are implemented with different types of agents. Data collection and event detection applications are implemented with *data collection agents* (DAs) and *event detection agents* (EAs), respectively. Both DAs and EAs are used to implement hybrid applications, which perform both data collection and event detection. In hybrid applications, DAs and EAs coevolve and adapt their behavior policies (genes) in a symbiotic manner. EAs helps DAs improve their behavior policies, and vice versa.

This paper is organized as follows. Section 2 overviews the BiSNET/e runtime, and Section 3 describes the design of MONSOON. Section 4 evaluates MONSOON with a series of simulation results. Simulation results show that MONSOON allows agents (WSN applications) to simultaneously satisfy conflicting objectives by adapting to dynamics of physical operational environments and network environments (e.g., sensor readings and node/link failures) through evolution. Sections 5 and 6 conclude with some discussion on related work.

2. The BiSNET/e Runtime

At the beginning of a WSN's operation, one DA and one EA are deployed on each node. They have randomly-generated behavior policies. A DA collects sensor data on each node periodically (i.e., at each data collection cycle) and carry the data to a base station on a hop-by-hop basis. An EA collects sensor data on each node periodically, and if it detects an event (i.e., a significant change in sensor data), carries the data to a base station on a hop-by-hop basis. If an event is not detected, the EA discards the data.

2.1. Agent Structure and Behaviors

Each agent consists of *attributes*, *body* and *behaviors*. *Attributes* carry descriptive information on an agent. They include agent type (i.e., EA or DA), behavior policy (gene), sensor data to be reported to a base station, the data's time stamp, and the ID of a node where the data is collected.

Body implements the functionalities of an agent: collecting and processing sensor data (e.g., discarding it or reporting it to a base station).

Behaviors implement actions inherent to all agents. Similar to biological entities (e.g., bees), agents sense their surrounding environment conditions and behave according to the sensed conditions without any intervention from/to other agents, platforms, base stations and human operators. This paper focuses on the following seven behaviors.

(1) Food gathering and consumption: Biological entities strive to seek food for living. For example, bees gather nectar to produce honey. Similarly, each agent periodically reads sensor data (as nectar) to gain *energy* (as honey)[2], and consumes a constant amount of energy for living.

(2) Pheromone emission: Agents may emit different types of pheromones: *migration and alert pheromones*. They 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 also emit alert pheromones when they fail migrations within a timeout period. Each alert pheromone references a possibly failed node that an agent could not migrate to. Each pheromone has its own concentration, which decays by half at every data collection cycle. A pheromone disapears when its concentration becomes zero.

(3) Replication: EAs may make a copy of themselves in response to the abundance of stored energy, while DAs always make a copy of themselves in each data collection

[1]Multiobjective Optimization for Network of Sensors using a cO-evOlutionary mechaNism

[2]The concept of energy in BiSNET/e does not represent the amount of physical battery in a node. It is logically affects agent behaviors.

cycle. A replicated (child) agent is placed on the node that its parent resides on, and it inherits the parent's agent type and behavior policy (gene). Replicated agents are intended to move toward base stations to report collected sensor data.

(4) Migration: Agents may move from one node to another. Migration is used to transmit agents (sensor data) to base stations. Each agent chooses a migration destination node by sensing three types of pheromones available on the local node: base station, migration and alert pheromones.

Each base station periodically propagates *base station pheromones* 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 the base stations by climbing pheromone's concentration gradient[3].

An agent may move to a base station by following a migration pheromone trace on which many other agents have traveled. The trace can be the shortest path to the base station. Conversely, an agent may goes off a migration pheromone trace and follows another path to a base station when the concentration of migration pheromones is too high on the trace (i.e., when too many agents have followed the trace). 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 consume more power and go down earlier than the others.

An agent may also avoid moving to a node referenced by an alert pheromone. This allows agents to reach base stations by bypassing link/node failures.

(5) Swarming: Agents may swarm (or merge) with others on their ways to base stations. Multiple agents become a single agent. (A DA can merge with both DAs and EAs, and an EA can merge with both EAs and DAs.) The resulting agent (swarm) aggregates sensor data contained in other agents, and uses the behavioral policy of the best agent in the swarm in terms of latency and power consumption. This data aggregation saves power consumption of nodes because in-node data processing requires much less power consumption than data transmission does.

(6) Reproduction: Once agents arrive at the MONSOON server (Figure 1), they are evaluated according to their objectives. Then, MONSOON selects best-performing (or elite) agents, and propagates them to individual nodes. An agent running on each node performs reproduction with one of the propagated agents. A reproduced agent inherits a behavior policy (gene) from its parents via crossover, and mutation may occur on the inherited behavior policy. Reproduced agents perform a generation change by taking over existing agents running on individual nodes.

Reproduction is intended to evolve agents so that the agents that fit better to the environment become more abundant. It retains the agents whose fitness to the current network conditions is high (i.e., the agents that have effective behavior policies, such as moving toward a base station in a short latency), and eliminates the agents whose fitness is low (i.e., 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.

(7) Death: Agents periodically consume energy for living, and expend energy to invoke their behaviors. (The energy costs to invoke behaviors are constant for all agents.) Agents die due to lack of energy when they cannot balance 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 the agent.

2.2. Behavior Sequences for DAs and EAs

Figures 2 and 3 show a sequence of behaviors that each DA and EA perform on a node in each data collection cycle.

A DA reads sensor data (as nectar) with the underlying sensor device and gains a constant amount of energy (as honey). Given the energy intake (E_F), each agent updates its energy level as follows.

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

$E(t)$ is the current energy level of the DA, and $E(t-1)$ is the DA's energy level in the previous data collection cycle. t is incremented by one at each data collection cycle.

If a DA's ($E(t)$) becomes very low (below the death threshold: T_D), the DA dies due to starvation[4].

A DA replicates itself in each data collection cycle. A replicating (parent) agent splits its energy units to halves ($\frac{E(t)-E_R}{2}$), gives a half to its child agent, and keeps the other half. E_R is the energy cost for an agent to perform the replication behavior. A child agent contains the sensor data that its parent collected, and carries it to a base station.

Each replicated DA migrates toward a base station on a hop by hop basis. On each intermediate node, it examines Equation 2 to determine which next node it migrates to.

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

An DA 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},

[3]Base station pheromones are designed after the Nasonov gland pheromone, which guides bees to move toward their nest [6].

[4]If all agents are dying on a node at the same time, a randomly selected agent for each type (i.e., EA and DA) will survive. At least one agent of each type runs on each node.

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 concentration of P_t among neighboring nodes.

When a DA is migrating to a neighboring node, it emits a migration pheromone on the local node. If the DA's migration fails, it emits an alert pheromone. Each alert pheromone spreads to one-hop away neighboring nodes.

```
for each data collection cycle
do {
    Read sensor data and gain energy (E_F).
    Update energy level (E(t)).
    if E(t) < the death threshold (T_D)
      then Invoke the death behavior.
    Invoke the replication behavior to make a child agent.
    Give the half of the current energy level to a replicated (child) agent.
    for each migrating agent
    do {
        Determine the destination node of migration.
        Emit a migration pheromone on the local node.
        Migrate to a neighboring node.
        if Migration fails
          then {
            Emit an alert pheromone on the local node.
            Propagate it to neighboring nodes.
          }
    }
}
```

Figure 2. A Sequence of DA Behaviors in Each Data Collection Cycle

```
while true
do {
    Read sensor data and gain energy (E_F).
    Update energy level (E(t)).
    if E(t) < the death threshold (T_D)
      then Invoke the death behavior.
    while E(t) > the replication threshold (T_R(t))
    do {
        Invoke the replication behavior to make a child agent.
        Give the half of the current energy level to the child agent.
    }
    for each migrating agent
    do {
        Determine the destination node of migration.
        Emit a migration pheromone on the local node.
        Migrate to a neighboring node.
        if Migration fails
          then {
            Emit an alert pheromone on the local node.
            Propagate it to neighboring nodes.
          }
    }
}
```

Figure 3. A Sequence of EA Behaviors

When an EA reads sensor data (as nectar) with the underlying sensor device and gains energy (as honey), its current energy level ($E(t)$) is updated with Equation 3.

$$E(t) = E(t-1) + S \cdot M \tag{3}$$

S represents the absolute difference between the current and previous sensor data. M is metabolic rate, which is a constant value between 0 and 1.

Each EA replicates itself if its energy level exceeds the replication threshold: $T_R(t)$ (Figure 3). The replication threshold is continuously adjusted as EWMA (Exponentially Weighted Moving Average) of each EA's energy level:

$$T_R(t) = (1-\alpha)T_R(t-1) + \alpha E(t) \tag{4}$$

$T_R(t)$ is the current replication threshold, and $T_R(t-1)$ is the one in the previous data collection period. EWMA is used to smooth out short-term minor oscillations in the data series of E. It places more emphasis on the long-term transition trend of E; only significant changes in E have the effects to change T_R. The α value is a constant to control the responsiveness of EWMA against the changes of E.

Similar to DAs, a parent EA splits its energy units to halves, gives a half to its child agent, and keeps the other half. The EA keeps replicating itself until its energy level becomes less than its T_R. A child agent contains the sensor data that its parent collected, and carries it to a base station.

EAs perform the migration behavior with Equation 2 in the same way as DAs do.

2.3 Agent Behavior Policy

EAs and DAs have the same structure for behavior policies (genes). Each behavior policy contains a set of weight values in Equation 2 ($w_t, 1 \le t \le 3$). w_1 and w_3 are non negative, and w_2 can be negative. These weight values 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 the base stations by climbing the 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. A negative w_2 value allows an agent to go off a migration pheromone trace and follow another path toward a base station. If an agent has a positive w_3, it moves to a base station by bypassing link/node failures.

3. MONSOON

MONSOON is a coevolutionary multiobjective adaptation mechanism designed for agents in BiSNET/e. It allows agents to heuristically adapt to multiple objectives simultaneously. This adaptation process is performed through *elite selection* and *genetic operations*. The elite selection process evaluates each type of agents (DAs and EAs) that arrive at base stations, based on given objectives, and chooses the best (or elite) ones. Elite agents are propagated to the network in order to perform genetic operations and reproduce an offspring (next generation) agent on each node. Elite selection is performed in the MONSOON server (see Figure 1), and genetic operations are performed in each node.

3.1. Operational Objectives

Agents (DAs and EAs) consider three conflicting objectives: *latency*, *cost* and *success rate* of their migration (i.e., data transmission) from individual nodes to base stations.

(1) Latency represents the time required for an agent (DA or EA) to travel to a base station from a node where the agent is born (replicated). As depicted below, latency is measured as a ratio of this agent travel time to the physical distance (PD) between a base station and a node where the

agent is born. The MONSOON server knows the location of each node with a certain localization mechanism.

$$Latency = \frac{Agent\ travel\ time\ (sec)}{PD\ (meter)} \quad (5)$$

(2) Cost represents the amount of power consumption required for an agent (DA or EA) to travel to a base station from a node where the agent is born. It is measured with the total number of data transmissions, each node's radio transmission range (radius), and PD.

$$Cost = \frac{Total\ \#\ of\ data\ transmissions}{Transmission\ range/PD} \quad (6)$$

The total number of data transmissions include successful and unsuccessful (failed) agent migrations as well as the transmissions of migration or alert pheromones.

(3) Success Rate is measured differently for DAs and EAs. For DAs, it is measured as follows.

$$Success\ rate_{DA} = \frac{\#\ of agents\ that\ arrive\ at\ base\ stations}{The\ total\ \#\ of\ nodes} \quad (7)$$

For EAs, success rate is measured as follows.

$$Success\ rate_{EA} = \frac{\#\ of successful\ agent\ migrations}{The\ total\ \#\ of\ attempts\ of\ agent\ migrations} \quad (8)$$

3.2. Elite Selection

Figure 4 shows how elite selection occurs at the MONSOON server in each data collection cycle. The MONSOON server performs the same selection process for EAs and DAs separately. The first step is to obtain three objective values (i.e., latency, cost and success rate) from each of the agents that reach the MONSOON server via base stations. Then, each agent is evaluated whether it is dominated by another agent. An agent is considered to be dominated if another agent outperforms it in all of three objectives.

```
Empty the archive
for each data collection cycle
do { Empty the population pool.
     Collect agents from the network.
     Add collected agents to the population pool.
     Move agents from the archive to the population pool.
     Empty the archive
     for each agent of the ones in the population pool
     do { if not dominated by all other agents in
          the population pool
          then Add the agent to the archive.
     Select elite agents from the archive.
     Propagate elite agents to the network.
```

Figure 4. Elite Selection in MONSOON

In the next step, a subset of non-dominated agents are selected as elite agents. This is performed with a hypercube space, which a three dimensional space whose axes represent three objectives (i.e., latency, cost and success rate). Each axis of the hypercube space is divided so that the space is divided into small cubes. Each non-nominated agent is plotted in this hypercube space based on their objective values. A single agent is randomly selected from each cube as an elite agent. This elite selection is designed to maintain the diversity of elite agents' genes. The diversification of agent genes contribute to improve agents' adaptation even to unanticipated network conditions.

Figure 5 shows an example hypercube space. Each axis is divided into two ranges; therefore, 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 in the hypercube space. Three agents (B, C, and D) are plotted in the lower left cube, while the other three agents (A, E, and F) are plotted in three different cubes. From the lower left cube, only one agent is randomly selected as an elite agent. A, E, and F are selected as elite agents because they are in different cubes.

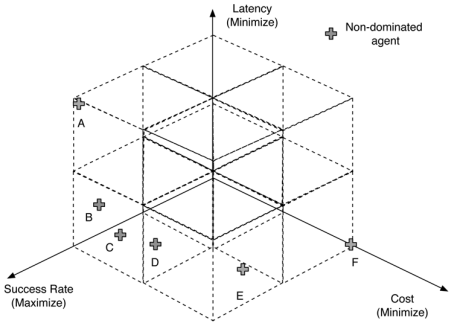

Figure 5. An Example Elite Selection

3.3. Genetic Operations

Once elite DAs and EAs are selected, the MONSOON server propagates them to each node in the network. They are propagated with base station pheromones.

Based on a certain reproduction probability, an agent performs the reproduction behavior on each node through genetic operations (crossover and mutation) when elite agents arrive at the node. As a mating partner, the agent selects one of the elite agents that has the most similar gene. Gene similarity is measured with the Euclidean distance between the values of two genes. DAs can mate with elite EAs, and EAs can mate with elite DAs. This cross-mating allows DAs and EAs to coevolve their behavior policies; DAs can improve EAs' genes, and vice versa.

During reproduction, an agent inherits the half of its gene from its parent agent and the other half from its parent's mating partner. Mutation occurs on the child agent's gene with a certain mutation probability by randomly changing gene values within a predefined value range.

4. Simulation Results

This section shows a set of simulation results to evaluate MONSOON. It is evaluated with a data collection applica-

tion (Section 4.1), event detection application (Section 4.2) and hybrid application (Section 4.3). A simulated WSN consists of 100 nodes uniformly deployed in an observation area of 300x300 square meters. Each node's communication range is 30 meters. A base station is deployed on the northwestern corner of the observation area. The base station links the MONSOON server via emulated serial port connection. All the software components in the BiSNET/e runtime are implemented in nesC, and the MONSOON server is implemented in Java. Simulation time is counted with ticks. Each tick represents five minutes. In genetic operations, the reproduction probability is 0.75, and the mutation probability is 0.025.

4.1 Data Collection Application

A data collection application is implemented with DAs that perform the sequence of behaviors shown in Figure 2. No EAs are used in this application. The data collection cycle corresponds to a simulation tick (five minutes).

Figure 6 (a) shows the average objective values produced by DAs at each simulation tick. Each objective value gradually improves and converges at the 17th tick. This simulation result shows that MONSOON allows DAs to simultaneously satisfy conflicting objectives by evolving their behavior policies.

Figures 8, and 9 and 10 show the objective values that elite DAs produced at the 20th tick. Since each objective value's change is less than 1% from the 17th to 20th tick, it is fair enough to say that the elite DAs are on the Pareto front at the 20th tick. Figures 8, and 9 and 10 plot the elite DAs in three different perspectives: latency-cost, cost-success rate, and latency-success rate perspectives. Each gray dot represents an elite DA, and a black dot represents overlapping elite DAs. These figures demonstrate that elite agents are well diversified as intended by an elite selection process described in Section 3.2.

Figure 6 (b) shows how the performance of DAs changes against a dynamic node addition. 25 nodes are added at random locations at the 20th tick. Upon this change in the network environment, objective values degrade dramatically because DAs have randomly-generated behavior policies on the new nodes. Those DAs cannot migrate efficiently toward the base station. Also, enough pheromones are not available on new nodes; DAs cannot make proper migration decisions when they move to the new nodes. However, DAs gradually improve their performance again, and objective values converge again at the 43th tick. MONSOON allows DAs to autonomously recover application performance despite dynamic node addition by evolving their behavior policies.

Figure 6 (c) shows how the performance of DAs changes against dynamic node failures. 25 nodes randomly fail at the 20th tick. Objective values degrade because some DAs try to migrate to failed nodes referenced by migration

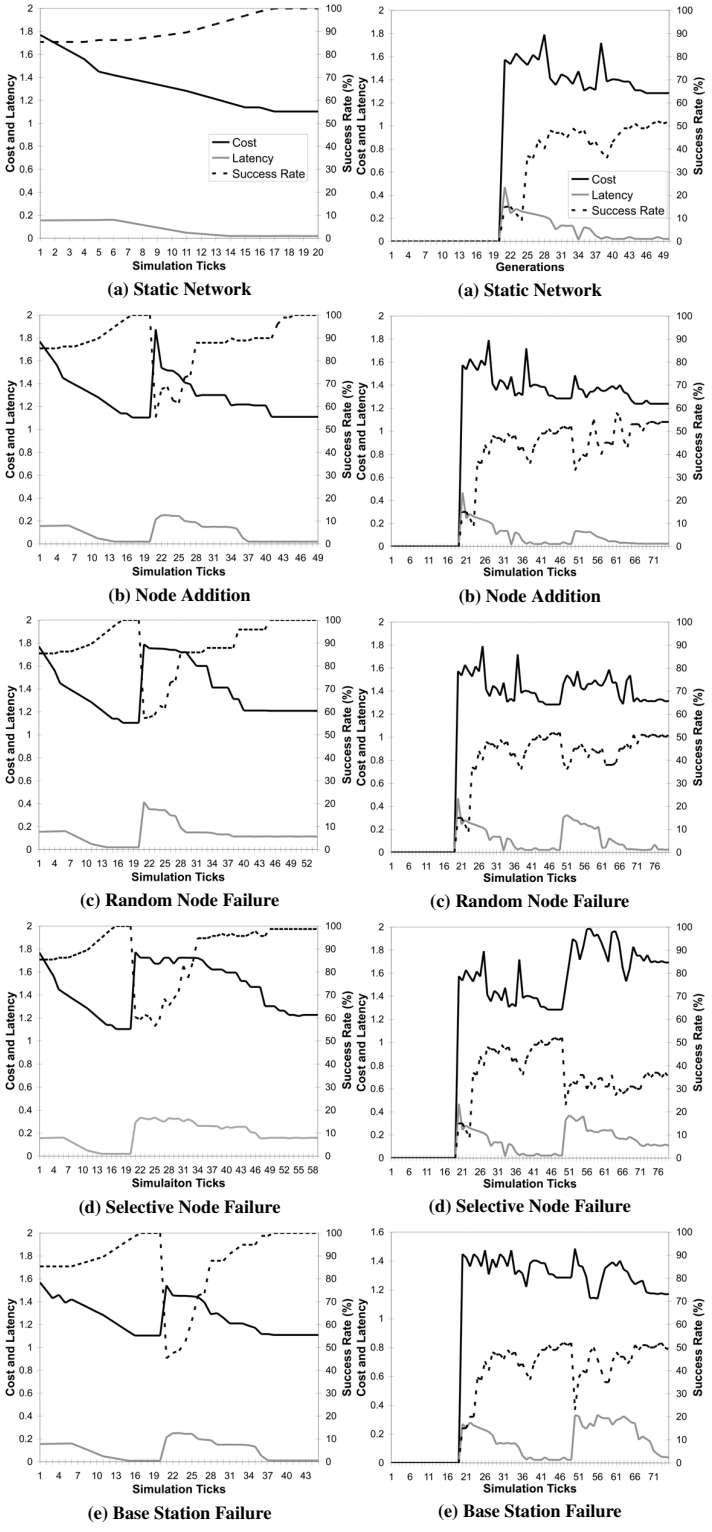

Figure 6. Objective Values of DAs without EAs

Figure 7. Objective Values of EAs without DAs

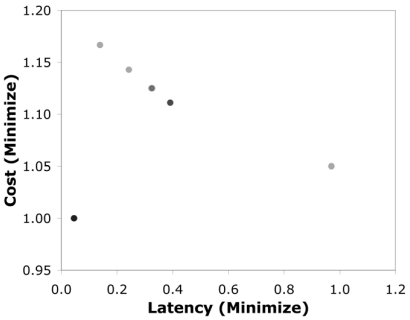

Figure 8. Latency-Cost Objective Values on the Pareto Front

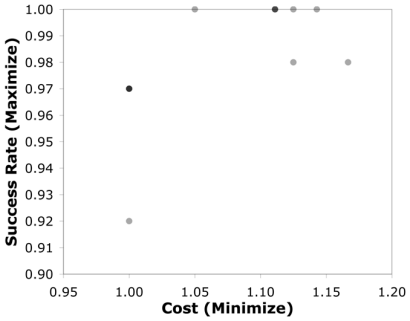

Figure 9. Cost-Success Rate Objective Values on the Pareto Front

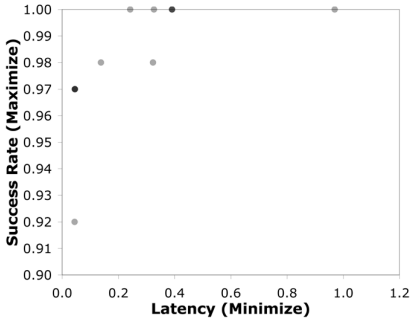

Figure 10. Latency-Success Rate Objective Values on the Pareto Front

pheromones. This increases the number of unsuccessful agent migrations. However, DAs gradually improve their performance again, and objective values converge again at the 45th tick. MONSOON allows DAs to autonomously recover application performance despite dynamic node failures by evolving their behavior policies.

Figure 6 (d) shows how the performance of DAs changes when nodes selectively fail in a specific area. At the 20th tick, 20 nodes fail in the middle of WSN observation area. Hence, a WSN has a hole in its middle area. Compared with Figure 6 (c), it takes longer time for DAs to recover their performance. Objective values converge at 50th tick again. The converged cost and latency are worse than the ones at the 20th tick because DAs have to detour a hole (i.e., a set of failed nodes) and take longer migration paths to the base station. This simulation results shows that MONSOON allows DAs to survive selective node failures through evolution.

Figure 6 (e) shows how the performance of DAs changes against base station failures. In this simulation scenario, two base stations are deployed at the northwestern and southeastern corners of WSN observation area. At the 20th tick, a base station at the southeastern corner fails. Objective values degrade because some DAs try to migrate toward the failed base station referenced by base station pheromones. This increases the number of unsuccessful agent migrations. However, DAs gradually improve their performance again, and objective values converge again at the 37th tick. MONSOON allows DAs to autonomously evolve and recover application performance despite dynamic base station failures.

4.2 Event Detection Application

An event detection application is implemented with EAs that perform the sequence of behaviors shown in Figure 3 in every simulation tick. No DAs are used in this application. This simulation study simulates an event, which occurs in the middle of WSN observation area at the 50th tick and radially spreads over time.

Figure 7 (a) shows the average objective values at each simulation tick. Upon an event detection, objective values are low because EAs use random behavior policies at first. However, each objective value gradually improves and converges at the 45th tick. This simulation result shows that MONSOON allows EAs to simultaneously satisfy conflicting objectives by evolving their behavior policies.

Figure 7 (b) shows how the performance of EAs changes against a dynamic node addition. 25 nodes are added at random locations at the 50th tick. Upon this environmental change, objective values degrade slightly because EAs have randomly-generated behavior policies on the new nodes. Those EAs cannot migrate efficiently toward the base station. However, EAs gradually improve their performance immediately, and objective values converge again at the 70th tick. MONSOON allows EAs to autonomously recover application performance despite dynamic node addition by evolving their behavior policies.

Figure 7 (c) shows how the performance of EAs changes against dynamic node failures. 25 nodes randomly fail at the 50th tick. Objective values degrade slightly because some EAs try to migrate to failed nodes referenced by migration pheromones. This increases the number of unsuccessful agent migrations. However, EAs gradually improve their performance again, and objective values converge again at the 72th tick. MONSOON allows EAs to autonomously recover application performance despite dynamic node failures by evolving their behavior policies.

Figure 7 (d) shows the result of a simulation when 20 sensor nodes are selected in selective fashion, i.e. create a hole in the middle of network, to be deactivated at the 50th tick. Hence, the sensor network contains a hole in the middle of the network. Compared with the result in figure 7 (c), MONSOON takes longer time to improve the performance of the WSN. The success rate converges at about the 75th tick to approximately 38%. The cost and latency also show the similar trend. Particularly, after the 52nd tick, the av-

erage value of cost and latency are higher than the values just before the 20th tick because agents have to detour in a longer path to avoid the hole in the middle of the network. The simulation results shows that MONSOON allows WSN to survives a selective sensor nodes failure by adjusting the operational parameters of WSN to be suitable to the changes in network condition.

Figure 7 (e) shows the result of a simulation which initially has two base stations deployed at the northwestern and southeastern corner of the observation area. Then, at the 50th tick, the base station at the southeastern corner is deactivated. From the figure, at the 51st tick, the success rate drops sharply to about 20% from around 50% in the 50th tick because more than a half of the agents still try to move to the base station at the southeastern corner. However, the success rate is improved successively and reach the same level as before the base station is deactivated at the 66th tick. Cost and latency show the same trend. MOSOON allows WSN to survives a base station failure by autonomously directing all agents to the remaining base station.

4.3 Hybrid Application

This section represents simulation results from a sensor network with two application deployed simultaneously. Figure 11 shows the average objective values from collected DAs, i.e. for data collection application, in each simulation ticks. On the other hand, figure 12 shows the average objective values from collected EAs, i.e. for event collection application, in each simulation ticks.

In the figure 12 (a), at 50th simulation ticks, oil spill happens and EAs start detecting and moving to the base station. The impact of EAs on DAs can be observed from the figure with the drop in success rate and the increase of cost and latency. However, within ten simulation ticks, MONOON allows DAs to adapt to the EAs and retain their performance. The simulation results shows that MONSOON allows a WSN application to adapt to the other application such that they can co-exist tranquilly in a same sensor network.

Figure 12 (b), (c), (d) and (e) show the similar scenario as in figure 12 (b), (c), (d) and (e), respectively. The simulation result in the former set of the figures also show the similar trend as in the later set of the figures; therefore, MONSOON allows a WSN application to adapt to network changes, i.e. partial node failure or the base station failure, even when it has to work simultaneously with another application on the same network.

Figure 12 (a) portraits the same scenario as in figure 7 (a). In the figure, 12 (a), sensor network hosts two applications, data collection and event detection. However, the objective values of event detection application, i.e. EAs, in figure 12 (a) are improved faster than in figure 7 (a). For example, the latency is reduced to lower than 0.05 at around the 28th tick in figure 12 (a) but it takes about the 38th tick

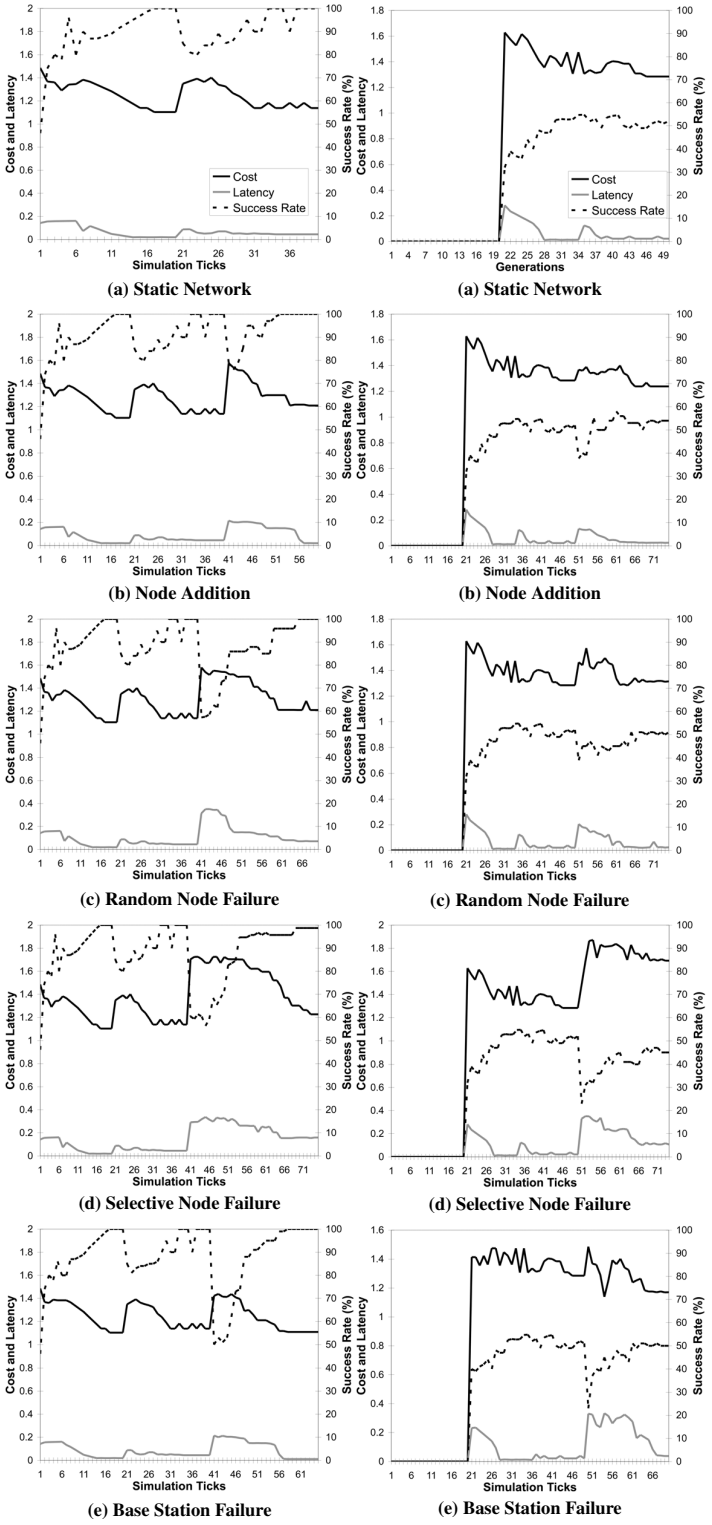

Figure 11. Objective Values of DAs with EAs

Figure 12. Objective Values of EAs with DAs

in figure 7 (a) to reduce to the same level. Thanks to crossmating (see Section 3.3) , MONSOON allows event detection application, i.e., EAs, to improve its objective values by using information from the other application. Figure 12 (b), (c), (d) and (e) also show the similar results.

4.4 Power Consumption

Figure 13 shows the impact of MONSOON and BiSNET/e on power consumption, and compare it with the power consumption by RUGGED [7, 8]. RUGGED is a gradient-based routing protocol. Figure 13 compares the average power consumption of nodes running BiSNET/e and RUGGED in the simulation scenario of Figure 6 (a). BiSNET/e consumes more power than RUGGED first because agents use random behavior policies. However, MONSOON allows agents to evolve their behavior policies and, in turn, reduce power consumption. After the 17th tick, power consumption is mostly same in BiSNET/e and RUGGED. Power consumption is nearly constant in RUGGED because it does not have dynamic adaptation mechanisms.

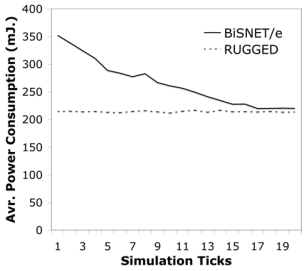

Figure 13. Average Power Consumption

4.5 Memory Footprint

Table 1 shows the memory footprint of the BiSNET/e runtime in a MICA2 mote, and compares it with the footprint of Blink (an example program in TinyOS), which periodically turns on and off an LED, RUGGED and Agilla, which is a mobile agent platform for WSNs [9]. The BiSNET/e runtime is lightweight in its footprint thanks to the simplicity of the biologically-inspired mechanisms in BiSNET/e. BiSNET/e can even run on a smaller-scale nodes, for example, TelosB, which has 48KB ROM.

Table 1. Memory Footprint in a MICA2 Mote

	RAM (KB)	ROM (KB)
BiSNET	2.5	30.0
Blink	0.04	1.6
RUGGED	0.84	20
Agilla	3.59	41.6

5. Related Work

This work is an extension to the authors' prior work, BiSNET [10]. BiSNET allows agents to autonomously adapt to dynamic network conditions. However, it does not investigate evolutionary adaptation (i.e., MONSOON); agent behavior policies are manually configured through trial-and-errors and fixed at runtime. Unlike BiSNET, BiSNET/e allows agents to dynamically adapt their behavior policies even to unanticipated network conditions.

MONSOON is designed as an extension to an existing mutiobjective optimization algorithm, called PESA-II [11], which in turn extends the NSGA-II algorithm [12]. MONSOON executes elite selection and genetic operations at physically different locations (i.e., at the MONSOON server and individual nodes, respectively), while both PESA-II and NSGA-II execute the two processes at the same location. In MONSOON, an agent chooses a mate that has the closest gene to its own gene, in order to consider the agent's performance stability. A mate is randomly chosen from the elite archive in PESA-II. In NSGA-II, a mate is selected with a binary tournament. Moreover, unlike PESA-II and NSGA-II, MONSOON considers coevolution between different types of agents (DAs and EAs).

kOS is an operating system that applies biological mechanisms to implement adaptive WSN applications [13]. However, kOS has not implemented specific biologically-inspired mechanisms yet. Also, [13] does not provide any evaluation results as well as the implementation details of kOS. In contrast, BiSNET/e implements specific biologically-inspired mechanisms such as pheromone emission, reproduction, genetic operations and migration. Moreover, this paper evaluates the impacts of those mechanisms on WSN applications' (i.e., agents') adaptability.

Agilla proposes a programming language to implement mobile agents for WSNs, and provides a runtime system (interpreter) to operate agents on TinyOS [9]. On the other hand, BiSNET/e does not focus on investigating a new programming language for WSNs. BiSNET/e and Agilla provide a similar set of behaviors such as migration and replication. However, Agilla does not address the research issues that BiSNET/e focuses on: evolutionary adaptation to conflicting objectives. In addition, BiSNET/e focuses on its design simplicity and runtime lightweight. As shown in table 1, BiSNET/e is much more lightweight than Agilla.

Several research efforts have applied genetic algorithms to WSNs, for example, to cluster-based routing [14–17], data processing [18], localization [19] and node placement [20, 21]. Every work uses a fitness function that combines multiple objective values as a weighted sum, and uses the function to rank agents/genes in elite selection. Application designers need to manually configure the weight values in a fitness function through trial-and-errors. In BiSNET/e, no manually-confired parameters exist for elite selection because of a domination ranking mechanism. As a result, BiSNET/e requires much less configuration cost for application designers. Also, [14, 15, 17, 19–21] do not consider dynamics in the network, but assumes the network is static.

Evolutionary multiobjective optimization algorithms

have been used for node placement [22–24] and routing [25, 26]. In each of these work, an optimization process is performed in a central server. This can lead to 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.

6. Conclusion

This paper describes an evolutionary multiobjective adaptation framework, MONSOON, in a biologically-inspired application architecture, called BiSNET/e. MONSOON allows WSN applications to simultaneously satisfy conflicting operational objectives by adapting to dynamics of physical operational environments and network environments (e.g., sensor readings and node/link failures) through evolution. Thanks to a set of simple biologically-inspired mechanisms, the BiSNET/e runtime is implemented lightweight.

References

[1] K. Akkaya and M. Younis, "A survey of routing protocols in wireless sensor networks," *Elsevier Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, 2005.

[2] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann, "Wireless sensor networks - new challenges in software engineering," in *Proc. of IEEE Emerging Technologies and Factory Automation*, September 2003.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Elsevier J. of Computer Networks*, vol. 38, pp. 393–422, 2002.

[4] P. Rentala, R. Musunuri, S. Gandham, and U. Sexena, "Survey on sensor networks," in *Proc. of the 7th ACM Int'l Conf. on Mobile Computing and Networking*, 2001.

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

[6] J. B. Free and I. H. Williams, "The role of the nasonov gland pheromone in crop communication by honey bees," *Int'l J. of Behavioural Biology, Brill Publishing*, vol. 41, 1972.

[7] J. Faruque, K. Psounis, and A. Helmy, "Analysis of gradient-based routing protocols in sensor networks," in *Proc. of IEEE/ACM Int'l Conf. on Distributed Computing in Sensor Systems*, 2005.

[8] J. Faruque and A. Helmy, "RUGGED: Routing on fingerprint gradients in sensor networks," in *Proc. of IEEE Int'l Conf. on Pervasive Services*, 2004.

[9] C. L. Fok, G. C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proc. of 25th IEEE Int'l Conf. on Distributed Computing Systems*, June 2005.

[10] P. Boonma and J. Suzuki, "BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks," *Elsevier J. of Computer Networks*. doi:10.1016/j.comnet.2007.06.006, 2007.

[11] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates, "PESA-II: Region-based selection in evolutionary multiobjective optimization," in *Proc. of Genetic and Evolutionary Computation Conference*, 2001.

[12] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan., "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. of INRIA Parallel Problem Solving from Nature*, 2000.

[13] M. Britton, V. Shum, L. Sacks, and H. Haddadi, "A biologically inspired approach to designing wireless sensor networks," in *Proc. of The 2nd European Workshop on Wireless Sensor Networks*, 2005.

[14] R. Khanna, H. Liu, and H. Chen, "Self-organisation of sensor networks using genetic algorithms," *Inderscience Int'l J. of Sensor Networks*, vol. 1, no. 3, pp. 241–252, 2006.

[15] S. Hussain and A. W. Matin, "Hierarchical cluster-based routing in wireless sensor networks," in *Proc. of the 5th Int'l Conf. on Info. Processing in Sensor Nets*, 2006.

[16] S. Jin, M. Zhou, and A. S. Wu, "Sensor network optimization using a genetic algorithm," in *Proc. of Multiconf. on Systemics, Cybernetics and Informatics*, 2003.

[17] K. P. Ferentinos and T. A. Tsiligiridis, "Adaptive design optimization of wireless sensor networks using genetic algorithms," *Elsevier J. of Computer Nets.*, vol. 51, no. 4, 2007.

[18] J. Hauser and C. Purdy, "Sensor data processing using genetic algorithms," in *Proc. of the 43rd IEEE Midwest Symposium on Circuits and Systems*, 2000.

[19] 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*, vol. 1, no. 4, 2006.

[20] H. Y. Guo, L. Zhang, L. L. Zhang, and J. X. Zhou, "Optimal placement of sensors for structural health monitoring using improved genetic algorithms," *Smart Materials and Structures*, vol. 13, no. 3, pp. 528–534, 2004.

[21] 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.

[22] D. B. Jourdan and O. L. de Weck, "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*, 2004.

[23] 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.

[24] A. M. Raich and T. R. Liszkai, "Multi-objective genetic algorithm methodology for optimizing sensor layouts to enhance structural damage identification," in *Proc. of the 4th Int'l Workshop on Structural Health Monitoring*, 2003.

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

[26] R. Rajagopalan, P. K. Varshney, K. G. Mehrotra, and C. K. Mohan, "Fault tolerant mobile agent routing in sensor networks: A multi-objective optimization approach," in *Proc. of the 2nd IEEE Upstate New York Workshop on Communication and Networking*, 2005.