Configuring Cloud-integrated Body Sensor Networks with Evolutionary Algorithms

Yi Cheng-Ren
Department of Computer Science
Univ. of Massachusetts, Boston
Boston, MA, 02125, USA
yiren001@cs.umb.edu

Junichi Suzuki
Department of Computer Science
Univ. of Massachusetts, Boston
Boston, MA, 02125, USA
jxs@cs.umb.edu

Dũng H. Phan
Department of Computer Science
Univ. of Massachusetts, Boston
Boston, MA, 02125, USA
phdung@cs.umb.edu

Shigo Omura
OGIS International, Inc.
San Mateo, CA 94402, USA
omura@ogis-international.com

Ryuichi Hosoya
OGIS International, Inc.
San Mateo, CA 94402, USA
hosoya@ogis-international.com

ABSTRACT

This paper investigates a few evolutionary game theoretic algorithms to configure cloud-integrated body sensor networks (BSNs) in an adaptive and stable manner with a multi-tier architecture called Body-in-the-Cloud (BitC). BitC allows BSNs to adapt their configurations (sensing intervals and sampling dates as well as data transmission intervals) to operational conditions (e.g., data request patterns) with respect to multiple conflicting performance objectives such as resource consumption and data yield. BitC theoretically guarantees that each BSN performs an evolutionarily stable configuration strategy, which is an equilibrium solution under given operational conditions. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies. BitC outperforms an existing well-known genetic algorithm in the quality, stability and computational cost in configuring BSNs.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Design, Algorithms

1. INTRODUCTION

This paper considers *cloud-integrated* body sensor networks (BSNs), which virtualize physical on/in-body sensors into clouds, formulates a configuration problem in a multi-tier architecture called Body-in-the-Cloud (BitC) and investigates a configuration problem in BitC with a few evolutionary algorithms. BitC consists of the *sensor*, *edge* and *cloud* layers. The sensor layer is a collection of sensors and sensor nodes in BSNs. Each BSN operates sensor nodes, each of which is equipped with sensors and wirelessly connected to a dedicated per-patient device or a patient's computer (e.g., smartphone or tablet machine) that serves as a *sink* node. The edge layer consists of sink nodes, which collect sensor data from sensor nodes in BSNs. The cloud layer consists of cloud environments that host *virtual sensors*, which are virtualized counterparts (or software counterparts) of physical sensors in BSNs. Virtual sensors collect sensor data from sink nodes in the edge layer and store those data for future use. The cloud layer also hosts various applications that obtain sensor data from virtual sensors and aid medical staff (e.g., clinicians, hospital/visiting nurses and caregivers) to share sensor data for clinical observation and intervention.

BitC performs *push-pull hybrid communication* between its three layers. Each sensor node periodically collects data from sensors attached to it based on sensor-specific sensing intervals and sampling rates and transmits (or pushes) those collected data to a sink node. The sink node in turn forwards (or pushes) incoming sensor data periodically to virtual sensors in clouds. When a virtual sensor does not have sensor data that a cloud application requires, it obtains (or pulls) that data from a sink node or a sensor node. This push-pull communication is intended to make as much sensor data as possible available for cloud applications by taking advantage of push communication while allowing virtual sensors to pull any missing or extra data anytime in an on-demand manner. For example, when an anomaly is found in pushed sensor data, medical staff may pull extra data in a higher temporal resolution to better understand a patient's medical condition. Given a sufficient amount of data, they may perform clinical intervention, order clinical cares, dispatch ambulances or notify family members of patients.

This paper focuses on configuring BSNs in BitC by tuning sensing intervals and sampling rates for sensors as well as data transmission intervals for sensor and sink nodes, and studies two properties in configuring BSNs:

- *Adaptability*: Adjusting BSN configurations accord-

BODYNETS 2014, September 29-October 01, London, Great Britain

DOI 10.4108/icst.bodynets.2014.257098

ing to operational conditions (e.g., data request patterns placed by cloud applications and availability of resources such as bandwidth and memory) with respect to performance objectives such as bandwidth consumption, energy consumption and data yield.

- *Stability*: Minimizing oscillations (non-deterministic inconsistencies) in making adaptation decisions.

BitC leverages an evolutionary game theoretic approach to configure BSNs. Each BSN maintains a set (or a population) of configuration strategies. BitC theoretically guarantees that, through a series of evolutionary games between BSN configuration strategies, the population state (i.e., the distribution of strategies) converges to an evolutionarily stable equilibrium regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an *evolutionarily stable strategy*.) In this state, no other strategies except an evolutionarily stable strategy can dominate the population. Given this theoretical property, BitC allows each BSN to operate at equilibria by using an evolutionarily stable strategy to configure BSNs in a deterministic (i.e., stable) manner. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies. This paper evaluates three algorithmic variants in BitC and compares them with NSGA-II, a well-known multiobjective genetic algorithm. BitC outperforms an existing well-known genetic algorithm in the quality, stability and computational cost in configuring BSNs.

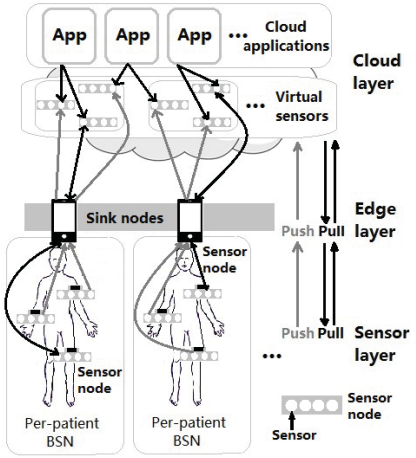


Figure 1: A Push-Pull Hybrid Communication in BitC

2. SYSTEM ARCHITECTURE

BitC consists of the following three layers (Fig. 1).

Sensor Layer: operates one or more BSNs on a per-patient basis (Fig. 1). Each BSN contains one or more sensor nodes in a certain topology. This paper assumes the star topology. Each sensor node is equipped with different types of sensors. It is assumed to be battery-operated. (It has limited energy supply.) It maintains a sensing interval and a sampling rate for each sensor attached to it. Upon a sensor reading, it stores collected data in its own memory space. Given a data transmission interval, it periodically flushes all data stored in its memory space and transmits the data to a sink node.

Edge Layer: consists of sink nodes, each of which participates in a certain BSN and receives sensor data periodically from sensor nodes in the BSN. A sink node stores incoming sensor data in its memory space and periodically flushes stored data to transmit (or push) them to the cloud layer. It maintains the mappings between physical and virtual sensors. In other words, it knows the origins and destinations of sensor data. Different sink nodes have different data transmission intervals. A sink node's data transmission interval can be different from the ones of sensor nodes in the same BSN. Sink nodes are assumed to have limited energy supplies through batteries.

In addition to pushing sensor data to a virtual sensor, each sink node receives a pull request from a virtual sensor when the virtual sensor does not have data that a cloud application(s) requires. If the sink node has the requested data in its memory, it returns that data. Otherwise, it issues another pull request to a sensor node that is responsible for the requested data. Upon receiving a pull request, the sensor node returns the requested data if it has the data in its memory. Otherwise, it returns an error message to a could application.

Cloud Layer: operates on clouds to host applications that allow medical staff to place continuous sensor data requests on virtual sensors in order to monitor patients. If a virtual sensor has data that an application requests, it returns that data. Otherwise, it issues a pull request to a sink node. While push communication carries out a one-way upstream travel of sensor data, pull communication incurs a round trip for requesting sensor data and receiving that data (or an error message).

3. PROBLEM STATEMENT

This section describes a BSN configuration problem for which BitC seeks equilibrium solutions. Each BSN configuration consists of four types of parameters (i.e., decision variables): sensing intervals and sampling rates for sensors as well as data transmission intervals for sensor and sink nodes. The problem is stated with the following symbols.

- $B = \{b_1, b_2, ..., b_i, ..., b_N\}$ denotes the set of N BSNs, each of which operates for a patient.
- Each BSN b_i consists of a sink node (denoted by m_i) and M sensor nodes: $b_i = \{h_{i1}, h_{i2}, ..., h_{ij}, ..., h_{iM}\}$. Each sensor node h_{ij} has L sensors: $h_{ij} = \{s_{ij1}, s_{ij2}, ..., s_{ijk}, ..., s_{ijL}\}$. o_{ijk} is the data transmission interval for h_{ij} to transmit sensor data collected from s_{ijk}. p_{ijk} and q_{ijk} are the sensing interval and sampling rate for s_{ijk}. Sampling rate is defined as the number of sensor data samples collected in a unit time. Each sensor node stores collected sensor data in its memory space until its next push transmission. If the memory becomes full, it performs FIFO (First-In-First-Out) data replacement. In a push transmission, it flushes and sends out all data stored in its memory.
- o_{m_i} denotes the data transmission interval for m_i to forward (or push) sensor data incoming from sensor nodes in b_i In between two push transmissions, m_i stores sensor data from b_i in its memory. It performs FIFO data replacement if the memory becomes full. In a push transmission, it flushes and sends out all data stored in the memory.
- $R_{ijk} = \{r_{ijk1}, r_{ijk2}, ..., r_{ijkr}, ..., r_{ijk|R_{ijk}|}\}$ denotes the set of sensor data requests that cloud applications issue to the virtual counterpart of s_{ijk} (s'_{ijk}) during the time period of W in the past. Each request r_{ijkr} is

characterized by its time stamp (t_{ijkr}) and time window (w_{ijkr}). It retrieves all sensor data available in the time interval $[t_{ijkr} - w_{ijkr}, t_{ijkr}]$. If s'_{ijk} has at least one data in the interval, it returns those data; otherwise, it issues a pull request to m_i.

- $R^m_{ijk} \in R_{ijk}$ denotes the set of sensor data requests for which a virtual sensor s'_{ijk} has no data. $|R^m_{ijk}|$ indicates the number of pull requests that s'_{ijk} issues to m_i. In other words, $R_{ijk} \setminus R^m_{ijk}$ is the set of sensor data requests that s'_{ijk} fulfills regarding s_{ijk}.

- $R^s_{ijk} \in R^m_{ijk} \in R_{ijk}$ denotes the set of sensor data requests for which m_i has no data. $|R^s_{ijk}|$ indicates the number of pull requests that m_i issues to h_{ij} for collecting data from s_{ijk}. $R^m_{ijk} \setminus R^s_{ijk}$ is the set of sensor data requests that m_i fulfills regarding s_{ijk}.

This paper considers four performance objectives: bandwidth consumption between the edge and cloud layers (f_B), energy consumption of sensor and sink nodes (f_E), request fulfillment for cloud applications (f_R) and data yield for cloud applications (f_D). The first two objectives are to be minimized while the others are to be maximized.

The bandwidth consumption objective (f_B) is defined as the total amount of data transmitted per a unit time between the edge and cloud layers. This objective impacts the payment for bandwidth consumption based on a cloud operator's pay-per-use billing scheme. It also impacts the lifetime of sink nodes. f_B is computed as follows.

$$\begin{aligned} f_B =& \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}(c_{ijk}d_{ijk}) \\ &+ \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|}(\phi_{ijkr}d_{ijk} + d_r) \\ &+ \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_r(|R^s_{ijk}| - \eta_{ijkr}) \end{aligned} \quad (1)$$

The first and second terms indicate the bandwidth consumption by one-way push communication from the edge layer to the cloud layer and two-way pull communication between the cloud and edge layers, respectively. c_{ijk} denotes the number of sensor data that s_{ijk} generates and sink nodes in turn push to the cloud layer during W. d_{ijk} denotes the size of each sensor data (in bits) that s_{ijk} generates. It is currently computed as: $q_{ijk} \times 16$ bits/sample. ϕ_{ijkr} denotes the number of sensor data that a pull request $r \in R^m_{ijk}$ can collect from sink nodes ($\phi_{ijkr} = |R^m_{ijk} \setminus R^s_{ijk}|$). d_r is the size of a pull request transmitted from the cloud layer to the edge layer. The third term in Eq. 1 indicates the bandwidth consumption by the error messages that sensors generate because they fail to fulfill pull requests. η_{ijkr} is the number of sensor data that a pull request $r \in R^s_{ijk}$ can collect from sensor nodes. e_r is the size of an error message.

The energy consumption objective (f_E) is defined as the total amount of energy that sensor and sink nodes consume for data transmissions during W. It impacts the lifetime of sensor and sink nodes. It is computed as follows.

$$\begin{aligned} f_E =& \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\frac{W}{o_{ijk}}e_t d_{ijk} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_t\eta_{ijkr}(d_{ijk} + d'_r) \\ &+ \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\frac{W}{o_{m_i}}e_t d_{ijk} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|} e_t\phi_{ijkr}(d_{ijk} + d_r) \\ &+ 2\times\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|} e_t e_r(|R^s_{ijk}| - \eta_{ijkr}) \end{aligned} \quad (2)$$

The first and second terms indicate the energy consumption by one-way push communication from the sensor layer to the edge layer and two-way pull communication between the edge layer and the sensor layer, respectively. e_t denotes the amount of energy (in Watts) that a sensor or sink node consumes to transmit a single bit of data. d'_r denotes the size of a pull request from the edge layer to the sensor layer. The third and fourth terms indicate the energy consumption by push and pull communication between the edge and cloud layer, respectively. The fifth term indicates the energy consumption for transmitting error messages on sensor and sink nodes.

The request fulfillment objective (f_R) is the ratio of the number of fulfilled requests over the total number of requests:

$$f_R = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R_{ijk}|} I_{R_{ijk}}}{|R_{ijk}|} \times 100 \quad (3)$$

$I_{R_{ijk}} = 1$ if a request $r \in R_{ijk}$ obtains at least one sensor data; otherwise, $I_{R_{ijk}} = 0$.

The data yield objective (f_Y) is defined as the total amount of data that cloud applications gather for their users. This objective impacts the informedness and situation awareness for application users. It is computed as follows.

$$f_Y = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|}\phi_{ijkr} + \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^s_{ijk}|}\eta_{ijkr} + c_{ijk} \quad (4)$$

BitC considers four constraints. The first constraint (C_E) is the upper limit for energy consumption: $f_E < C_E$. A violation for the constraint (g_E) is computed as $g_E = I_E \times (f_E - C_E)$ where $I_E = 1$ if $f_E > C_E$; otherwise $I_E = 0$.

The second constraint (C_Y) is the lower limit for data yield: $f_Y > C_Y$. A constraint violation (g_Y) is computed as $g_Y = I_Y \times (C_Y - f_Y)$ where $I_Y = 1$ if $f_Y < C_Y$; otherwise $I_Y = 0$.

The third constraint (C_R) is the lower limit for request fulfillment: $f_R > C_R$. The constraint violation in request fulfillment (g_R) is computed as $g_R = I_R \times (C_R - f_R)$ where $I_R = 1$ if $f_R < C_R$; otherwise $I_R = 0$.

The fourth constraint (C_B) is the upper limit for bandwidth consumption: $f_B < C_B$. A violation for this constraint (g_B) is computed as $g_B = I_B \times (f_B - C_B)$ where $I_B = 1$ if $f_B > C_B$; otherwise $I_B = 0$.

4. EVOLUTIONARY GAME THEORY

In a conventional game, the objective of a player is to choose a strategy that maximizes its payoff in a single game. In contrast, evolutionary games are played repeatedly by players randomly drawn from a population [15]. This section overviews key elements in evolutionary games: evolutionarily stable strategies (ESS) and replicator dynamics.

4.1 Evolutionarily Stable Strategies (ESS)

Suppose all players in the initial population are programmed to play a certain (incumbent) strategy k. Then, let a small population share of players, $x \in (0, 1)$, mutate and play a different (mutant) strategy ℓ. When a player is drawn for a game, the probabilities that its opponent plays k and ℓ are $1-x$ and x, respectively. Thus, the expected payoffs for the player to play k and ℓ are denoted as $U(k, x\ell+(1-x)k)$ and $U(\ell, x\ell + (1-x)k)$, respectively.

Definition 1. *A strategy k is said to be evolutionarily stable if, for every strategy $\ell \neq k$, a certain $\bar{x} \in (0, 1)$ exists, such that the inequality*

$$U(k,\ x\ell + (1-x)k) > U(\ell,\ x\ell + (1-x)k) \tag{5}$$

holds for all $x \in (0, \bar{x})$.

If the payoff function is linear, Equation 5 derives:

$$(1-x)U(k,k) + xU(k,\ell) > (1-x)U(\ell,k) + xU(\ell,\ell) \tag{6}$$

If x is close to zero, Equation 6 derives either

$$U(k,k) > U(\ell,k) \text{ or } U(k,k) = U(\ell,k) \text{ and } U(k,\ell) > U(\ell,\ell) \tag{7}$$

This indicates that a player associated with the strategy k gains a higher payoff than the ones associated with the other strategies. Therefore, no players can benefit by changing their strategies from k to the others. This means that an ESS is a solution on a Nash equilibrium. An ESS is a strategy that cannot be invaded by any alternative (mutant) strategies that have lower population shares.

4.2 Replicator Dynamics

The replicator dynamics describes how population shares associated with different strategies evolve over time [13]. Let $\lambda_k(t) \geq 0$ be the number of players who play the strategy $k \in K$, where K is the set of available strategies. The total population of players is given by $\lambda(t) = \sum_{k=1}^{|K|} \lambda_k(t)$. Let $x_k(t) = \lambda_k(t)/\lambda(t)$ be the population share of players who play k at time t. The population state is defined by $X(t) = [x_1(t), \cdots, x_k(t), \cdots, x_K(t)]$. Given X, the expected payoff of playing k is denoted by $U(k, X)$. The population's average payoff, which is same as the payoff of a player drawn randomly from the population, is denoted by $U(X, X) = \sum_{k=1}^{|K|} x_k \cdot U(k, X)$. In the replicator dynamics, the dynamics of the population share x_k is described as follows. $\dot{x}_k$ is the time derivative of x_k.

$$\dot{x}_k = x_k \cdot [U(k, X) - U(X, X)] \tag{8}$$

This equation states that players increase (or decrease) their population shares when their payoffs are higher (or lower) than the population's average payoff.

Theorem 1. *If a strategy k is strictly dominated, then $x_k(t)_{t\to\infty} \to 0$.*

A strategy is said to be strictly dominant if its payoff is strictly higher than any opponents. As its population share grows, it dominates the population over time. Conversely, a strategy is said to be strictly dominated if its payoff is lower than that of a strictly dominant strategy. Thus, strictly dominated strategies disappear in the population over time.

There is a close connection between Nash equilibria and the steady states in the replicator dynamics, in which the population shares do not change over time. Since no players change their strategies on Nash equilibria, every Nash equilibrium is a steady state in the replicator dynamics. As described in Section 4.1, an ESS is a solution on a Nash equilibrium. Thus, an ESS is a solution at a steady state in the replicator dynamics. In other words, an ESS is the strictly dominant strategy in the population on a steady state.

BitC maintains a population of configuration strategies for each BSN. In each population, strategies are randomly drawn to play games repeatedly until the population state reaches a steady state. Then, BitC identifies a strictly dominant strategy in the population and configures a BSN based on the strategy as an ESS.

5. BODY-IN-THE-CLOUD

BitC maintains N populations, $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$, for N BSNs and performs games among strategies in each population. Each strategy $s(b_i)$ specifies a particular configuration for a BSN b_i using four types of parameters: sensing intervals and sampling rates for sensors (p_{ijk} and q_{ijk}) as well as data transmission intervals for sink and sensor nodes (o_{m_i} and o_{ijk}).

$$s(b_i) = \bigcup_{j\in 1..M} \bigcup_{k\in 1..L} (o_{m_i}, o_{ijk}, p_{ijk}, q_{ijk}) \quad 1 < i < N \tag{9}$$

Algorithm 1 shows how BitC seeks an evolutionarily stable configuration strategy for each BSN through evolutionary games. In the 0-th generation, strategies are randomly generated for each of N populations $\{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$ (Line 2). Those strategies may or may not be feasible. Note that a strategy is said to be feasible if it violates none of four constraints described in Section 3.

In each generation (g), a series of games are carried out on every population (Lines 4 to 25). A single game randomly chooses a pair of strategies (s_1 and s_2) and distinguishes them to the winner and the loser with respect to performance objectives described in Section 3 (Lines 7 to 9). The loser disappears in the population. The winner is replicated to increase its population share and mutated with polynomial mutation [4] (Lines 10 to 17). Mutation randomly chooses a parameter (or parameters) in a given strategy with a certain mutation rate P_m and alters its/their value(s) at random (Lines 11 to 15).

Once all strategies play games in the population, BitC identifies a feasible strategy with the highest population share (x_s) and determines it as a dominant strategy (d_i) (Lines 20 to 24). BitC configures a BSN with parameters contained in the dominant strategy (Line 25).

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (`performGame()` in Algorithm 1). If a feasible strategy and an infeasible strategy participate in a game, the feasible one always wins over its opponent. If both strategies are feasible, they are compared with one of the following three schemes to select the winner.

- Pareto dominance (PD): This scheme is based on the notion of *dominance* [12], in which a strategy s_1 is said to dominate another strategy s_2 (denoted by $s_1 \succ s_2$) if both of the following conditions hold:
  - s_1's objective values are superior than, or equal to, s_2's in all objectives.
  - s_1's objective values are superior than s_2's in at least one objectives.

Algorithm 1 Evolutionary Process in BitC

```
g = 0
Randomly generate the initial N populations for N BSNs: P = {P_1, P_2, ..., P_N}
while g < G_max do
  for each population P_i randomly selected from P do
    P_i' ← ∅
    for j = 1 to |P_i|/2 do
      s_1 ← randomlySelect(P_i)
      s_2 ← randomlySelect(P_i)
      winner ← performGame(s_1, s_2)
      replica ← replicate(winner)
      for each parameter in replica do
        if random() ≤ P_m then
          replica ← mutate(winner)
        end if
      end for
      P_i \ {s_1, s_2}
      P_i' ∪ {winner, replica}
    end for
    P_i ← P_i'
    d_i ← argmax_{s∈P_i} x_s
    while d_i is infeasible do
      P_i \ {d_i}
      d_i ← argmax_{s∈P_i} x_s
    end while
    Configure a BSN in question based on d_i.
  end for
  g = g + 1
end while
```

The dominating strategy wins a game over the dominated one. If two strategies are non-dominated with each other, the winner is randomly selected.

- Hypervolume (HV): This scheme is based on the hypervolume metric [17]. It measures the volume that a given strategy (s) dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \tag{10}$$

Λ denotes the Lebesgue measure. x_r is the reference point placed in the objective space. A higher hypervolume means that a strategy is more optimal. Given two strategies, the one with a higher hypervolume value wins a game. If both have the same hypervolume value, the winner is randomly selected.

- Hybrid of Pareto comparison and hypervolume (PD-HV): This scheme is a combination of the above two schemes. First, it performs the Pareto dominance (PD) comparison for given two strategies. If they are non-dominated, the hypervolume (HV) comparison is used to select the winner. If they still tie with the hypervolume metric, the winner is randomly selected.

If both strategies are infeasible in a game, they are compared based on their constraint violation. A strategy s_1 wins a game over another strategy s_2 if both of the following conditions hold:

- s_1's constraint violation is lower than, or equal to, s_2's in all constraints.
- s_1's constraint violation is lower than s_2's in at least one constraints.

6. STABILITY ANALYSIS

This section analyzes BitC's stability (i.e., reachability to at least one of Nash equilibrium) by proving the state of each population converges to an evolutionarily stable equillibrium. The proof consists of three steps: (1) designing a set of differential equations that describe the dynamics of the population state, (2) proving an strategy selection process has equilibria, and (3) proving the the equilibria are asymptotically (or evolutionarily) stable. The proof uses the following symbols:

- S denotes the set of available strategies. S^* denotes a set of strategies that appear in the population.
- $X(t) = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}$ denotes a population state at time t where $x_s(t)$ is the population share a strategy $s \in S$. $\sum_{s\in S^*}(x_s) = 1$.
- F_s denotes the fitness of a strategy s. It is a relative value that is determined in a game against an opponent based on the dominance relationship between them (Algorithm **??**). The winner of a game earns a higher fitness than the loser.
- $p_k^s = x_k \cdot \phi(F_s - F_k)$ denotes the probability that a strategy s is replicated by winning a game against another strategy k. $\phi(F_s - F_k)$ is the probability that the fitness of s is higher than that of k.

The dynamics of the population share of s is described as:

$$\begin{aligned}\dot{x}_s &= \sum_{k\in S^*, k\neq s}\{x_s p_k^s - x_k p_s^k\} \\ &= x_s \sum_{k\in S^*, k\neq s} x_k\{\phi(F_s - F_k) - \phi(F_k - F_s)\}\end{aligned} \tag{11}$$

Note that if s is strictly dominated, $x_s(t)_{t\to\infty} \to 0$.

THEOREM 2. *The state of a population converges to an equilibrium.*

PROOF. It is true that different strategies have different fitness values. In other words, only one strategy has the highest fitness among others. Given Theorem 1, assuming that $F_1 > F_2 > \cdots > F_{|S^*|}$, the population state converges to an equilibrium: $X(t)_{t\to\infty} = \{x_1(t), x_2(t), \cdots, x_{|S^*|}(t)\}_{t\to\infty}$ $= \{1, 0, \cdots, 0\}$. □

THEOREM 3. *The equilibrium found in Theorem 2 is asymptotically stable.*

PROOF. At the equilibrium $X = \{1, 0, \cdots, 0\}$, a set of differential equations can be downsized by substituting $x_1 = 1 - x_2 - \cdots - x_{|S^*|}$

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{i=2, i\neq s}^{|s^*|} z_i \cdot c_{si}], \quad s, k = 2, ..., |S^*| \tag{12}$$

where $c_{sk} \equiv \phi(F_s - F_k) - \phi(F_k - F_s))$ and $Z(t) = \{z_2(t), z_3(t), \cdots, z_{|S^*|}$ denotes the corresponding downsized population state. Given Theorem 1, $Z_{t\to\infty} = Z_{eq} = \{0, 0, \cdots, 0\}$ of $(|S^*| - 1)$-dimension.

If all Eigenvalues of Jaccobian matrix of $Z(t)$ has negative real parts, Z_{eq} is asymptotically stable. The Jaccobian matrix J's elements are described as follows where $s, k = 2, ..., |S^*|$.

$$J_{sk} = \left[\frac{\partial \dot{z}_s}{\partial z_k}\right]_{|Z=Z_{eq}} = \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{i=2, i\neq s}^{|S^*|} z_i \cdot c_{si}]}{\partial z_k}\right]_{|Z=Z_{eq}} \tag{13}$$

Therefore, J is given as follows, where $c_{21}, c_{31}, \cdots, c_{|S^*|1}$ are J's Eigenvalues.

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{|S^*|1} \end{bmatrix} \tag{14}$$

$c_{s1} = -\phi(F_1 - F_s) < 0$ for all s; therefore, $Z_{eq} = \{0, 0, \cdots, 0\}$ is asymptotically stable. □

7. SIMULATION EVALUATION

This section evaluates BitC through simulations and discusses how BitC configures BSNs under given operational conditions (e.g., data request patterns placed by cloud applications and memory space availability in sink and sensor nodes). Comparative performance study is carried out with BitC's three variants (PD, HV and PD-HV; c.f., Section 5) and NSGA-II, which is a well-known multiobjective genetic algorithm (EMOA) [4]. All four algorithms are implemented with jMetal [5].

Table 1: Simulation Settings

Parameter	Value
Duration of a simulation (W)	10,800 seconds (3 hours)
Number of BSNs (N)	10
Number of simulation runs	20
Number of nodes in a BSN (M)	3
Number of sensors in a node (L)	4
Memory space in a sensor node	2 GB
Memory space in a sink node	16 GB
Total number of data requests from cloud applications	1,000
Size of a data request (d_r and d'_r)	100 bytes
Size of an error message (e_r)	250 bytes
Energy consumption for a single bit of data (e_t)	0.001 Watt
Time window for a data request to a body temp sensor	[0, 60 secs]
Time window for a data request to an oximeter	[0, 60 secs]
Time window for a data request to an acelerometer	[0, 1,800 secs]
Time window for a data request to an ECG sensor	[0, 600 secs]
Number of generations (G_{max})	400
Population size ($\|\mathcal{P}_i\|$)	100
Mutation rate (P_m)	$1/v$
Upper limit of bandwidth consumption (C_B)	100 Kbps
Lower limit of data yield (C_Y)	6000
Upper limit of energy consumption (C_E)	1 KWatts
Lower limit of request fulfillment (C_R)	95%

Simulations are configured with the parameters shown in Table 1. Cloud applications issue 1,000 data requests during three hours. Requests are uniformly distributed over virtual sensors. Each sensor node contains four sensors: body temperature sensor, oximeter, accelerometer and ECG sensor. A time window is randomly set for each request to a sensor. For example, it is set with the uniform distribution in between 0 and 60 seconds for an oximeter. Mutation rate is set to $1/v$ where v is the number of parameters in a strategy. BitC and NSGA-II are equally configured except that NSGA-II performs one-point crossover. (BitC does not have the notion of crossover.) For example, BitC and NSGA-II maintain the same population size (100) and perform polynomial mutation in the same way. Every simulation result is the average with 20 independent simulation runs.

Figure 2 shows how BitC variants (PD, HV and HV-PD) and NSGA-II improve objective values change over generations. Four constraints (C_B, C_Y, C_E and C_R in Table 1) are enabled (Table 1). BitC variants and NSGA-II satisfy all four constraints at the last generation. Figure 2 illustrates that BitC and NSGA-II improve objective values subject to given constraints by balancing the trade-offs among conflicting objectives. For example, in Figures 2e and 2g, BitC and NSGA-II improve both bandwidth consumption (f_B) and request fulfillment (f_R) through generations while the two objectives conflict with each other.

Table 2 compares BitC variants and NSGA-II with the notion of Pareto dominance. (See Section 5 for the definition of the Pareto dominance.) It shows how many of 100 NSGA-II individuals dominate, are non-dominated with and are dominated by each variant of BitC at the last generation. 22 and 18 NSGA-II individuals dominate PD and PD-HV, respectively. 78 NSGA-II individuals are non-dominated with HV. 21 and 23 NSGA-II individuals are dominated by PD and PD-HV, respectively. Given these results, PD-HV yields the highest performance among three BitC variants and outperforms NSGA-II.

Table 2: Dominance Comparison of BitC Variants and NSGA-II

	PD	HV	PD-HV
# of NSGA-II individuals that dominate:	22	16	18
# of NSGA-II individuals that are non-dominated with:	57	78	59
# of NSGA-II individuals that are dominated by:	21	6	23

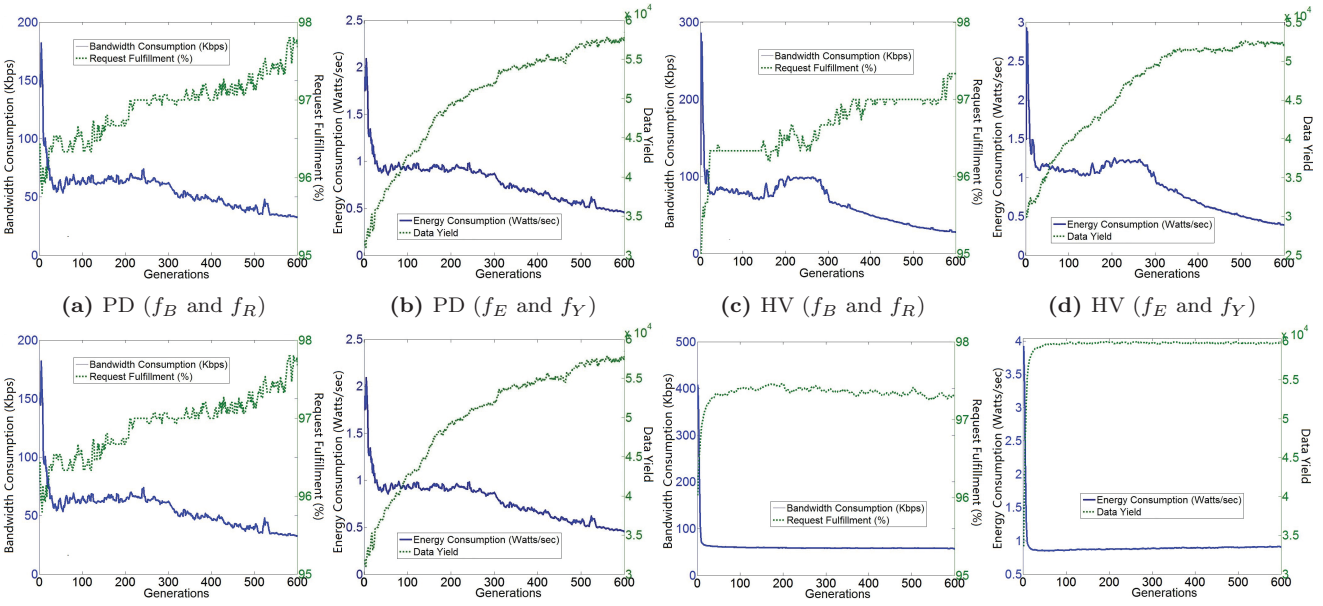
Table 3 compares BitC variants and NSGA-II with the hypervolume metric. (See Section 5 for the definition of hypervolume.) It shows the hypervolume that each BitC variant yields at the last generation as well as the average hypervolume that 100 NSGA-II individuals yield at the last generation. Hypervolume is computed with each objective normalized to [0, 1]. (The value range of hypervolume is [0, 1].) BitC variants produce higher hypervolume values than NSGA-II (0.84). Among those variants, the HV variant yields the highest hypervolume (0.912).

Table 3: Comparison of BitC Variants and NSGA-II in Hypervolume

Algorithms	Hypervolume
BitC-PD	0.870
BitC-HV	0.912
BitC-PD-HV	0.903
NSGA-II	0.840

BitC yields a single set of objective values with dominant strategies at each generation while NSGA-II yields 100 sets of objective values with 100 individuals at each generation. Therefore, in Tables 4 to 6, the BitC solution is evaluated against an NSGA-II individual that is closest to the solution in the objective space. Tables 4 to 6 compare each BitC variant and NSGA-II based on three different metrics: objective values, hypervolume and Euclidean distance. For NSGA-II, objective values are measured with an individual that minimizes the Euclidean distance to the BitC solution at the last generation. Hypervolume is measured with the NSGA-II individual. Distance is measured in between the NSGA-II individual and the BitC solution. Distance is computed with each objective normalized to [0, 1]. (The value range of distance is $[0, \sqrt{2}]$.)

As shown in Tables 4 to 6, each of BitC variants is non-dominated with NSGA-II. NSGA-II yields slightly higher,



(a) PD (f_B and f_R) **(b)** PD (f_E and f_Y) **(c)** HV (f_B and f_R) **(d)** HV (f_E and f_Y)

(e) HV-PD (f_B and f_R) **(f)** HV-PD (f_E and f_Y) **(g)** NSGA-II (f_B and f_R) **(h)** NSGA-II (f_E and f_Y)

Figure 2: Objectives Values through Generations with BitC Variants and NSGA-II

yet very close, hypervolume measures than BitC variants. (NSGA-II yields 0.2%, 0.1% and 0.4% higher hypervolume than BitC's PD, HV and PD-HV variants, respectively.) The distance between BitC and NSGA-II is minimum (0.04) with its HV variant. Given these results, BitC and NSGA-II tie in a solution-to-solution basis.

Table 4: Comparison of BitC-PD and NSGA-II

Objectives	NSGA-II	BitC-PD
Bandwidth consumption (Kbps)	33.89	42.28
Energy consumption (W/sec)	0.51	0.58
Request fulfillment (%)	98	97.3
Data yield	58,014	58,139
Hypervolume	0.89	0.87
Euclidean distance		0.23

Table 5: Comparison of BitC-HV and NSGA-II

Objectives	NSGA-II	BitC-HV
Bandwidth consumption (Kbps)	28.49	27.491
Energy consumption (W/sec)	0.396	0.3874
Request fulfillment (%)	98	97.3
Data yield	52,910	52,356
Hypervolume	0.913	0.912
Euclidean Distance		0.04

Table 6: Comparison of BitC-PD-HV and NSGA-II

Objectives	NSGA-II	BItC-PD-HV
Bandwidth consumption (Kbps)	32.91	32.21
Energy consumption (W/sec)	0.41	0.44
Request fulfillment (%)	98	97.6
Data yield	56,497	58,479
Hypervolume	0.907	0.903
Euclidean Distance		0.08

Table 7 shows the variance of objective values that BitC variants and NSGA-II yield at the last generation in 20 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillation in objective value results) among different simulation runs. BitC variants maintain higher stability than NSGA-II. BitC's HV variant yields 38.15% higher stability than NSGA-II. Table 7 exhibits BitC's stability property (i.e. its ability to seek evolutionarily stable strategies), which NSGA-II does not have.

Table 7: Comparison in Performance Variance

Objectives	NSGA-II	PD	HV	PD-HV
Bandwidth consumption (Kbps)	3.12	3.00 (4%)	3.25 (-4%)	3.7 (-15.6%)
Energy consumption (W/sec)	0.115	0.06 (47.8%)	0.074 (35.6%)	0.045 (60.8%)
Request fulfillment (%)	0.61	0.58 (3%)	0.35 (42.5%)	0.3 (50.8%)
Data yield	3958	1070 (73%)	849 (78.5%)	2746 (30.62%)
Average difference		31.95%	38.15%	31.65%

Table 8 shows the time required for BitC variants and NSGA-II to execute a single simulation of 600 generations. Simulations were carried out with a Java VM 1.7 on a Windows 8.1 PC with a 3.6 GHz AMD A6-5400K APU and 6 GB memory space. Among BItC's variants, its PD variant is fastest, and its PD-HV variant is slowest. All variants run faster than NSGA-II. The PD variant is 72.4% faster than NSGA-II.

8. RELATED WORK

This paper extends a prior work on cloud-integrated BSNs [11]. Compared to [11], this paper formulates a more realistic problem (Section 3) than the one in [11] by accommodating parameters configurable in Simmer's sensor nodes[1]. Moreover, this paper uses an evolutionary game theoretic algo-

[1]http://www.shimmersensing.com/

Table 8: Comparison in Execution Time

Algorithms	Execution Time
BitC-PD	1 hour 38 minutes (27.6%)
BitC-HV	1 hour 50 minutes (31%)
BitC-PD-HV	2 hours 39 minutes (44.9%)
NSGA-II	5 hours 54 minutes

rithm that possesses stability (i.e. reachability to at least one Nash equilibria) in configuring BSNs while a genetic algorithm is used in [11]. As stochastic global search algorithms, genetic algorithms lack stability.

Various architectures and research tools have been proposed for cloud-integrated sensor networks including BSNs [8, 1, 7, 3, 16, 6]. Hassan et al. [8], Aberer et al. [1], Gaynor et al. [7] and Boonma et al. [3] assume three-tier architectures similar to BitC and investigate publish/subscribe communication between the edge layer to the cloud layer. Their focus is placed on push communication. In contrast, BitC investigates push-pull hybrid communication between the sensor layer and the cloud layer through the edge layer. Yuriyama et al. propose a two-tier architecture that consists of the sensor and cloud layers [16]. The architectures proposed by Yuriyama et al. and Fortino et al. [6] are similar to BitC in that they leverage the notion of virtual sensors. However, they do not consider push-pull (nor publish/subscribe) communication. All the above-mentioned work do not consider adaptive/stable configurations of sensor networks as BitC does.

Push-pull hybrid communication has been studied in sensor networks [14, 2, 9, 10]. However, few efforts exist to study it between the edge and cloud layers in the context of cloud-integrated sensor networks. Unlike those relevant work, this paper formulates a sensor network configuration problem with cloud-specific objectives as well as the ones in sensor networks and seeks adaptive/stable solutions for the problem.

9. CONCLUSION

This paper considers a layered push-pull hybrid communication for cloud-integrated BSNs and formulates a BSN configuration problem to seek equilibrium solutions. Evolutionary game theoretic algorithms are used to approach the problem. A theoretical analysis proves that the proposed algorithms allow each BSN to operate at an equilibrium by using an evolutionarily stable configuration strategy in a deterministic (i.e., stable) manner. Simulation results verify that BitC configures BSNs in an adaptive and stable manner. BitC outperforms an existing well-known genetic algorithm in the quality, stability and computational cost in configuring BSNs.

10. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Proc. the 8th IEEE Int'l Conference on Mobile Data Management*, 2007.

[2] P. Boonma, Q. Han, and J. Suzuki. Leveraging biologically-inspired mobile agents supporting composite needs of reliability and timeliness in sensor applications. In *Proc. IEEE Int'l Conf. on Frontiers in the Convergence of Biosci. and Info. Tech.*, 2007.

[3] P. Boonma and J. Suzuki. TinyDDS: An interoperable and configurable publish/subscribe middleware for wireless sensor networks. In A. Hinze and A. Buchmann, editors, *Principles and Apps. of Dist. Event-Based Systems*, chapter 9. IGI Global, 2010.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evol. Computat.*, 6(2), 2002.

[5] J. Durillo, A. Nebro, and E. Alba. The jMetal framework for multi-objective optimization: Design and architecture. In *Proc. IEEE Congress on Evol. Computat.*, 2010.

[6] G. Fortino, D. Parisi, V. Pirrone, and G. D. Fatta. BodyCloud: A SaaS approach for community body sensor networks. *Future Generation Computer Systems*, 35(6):62–79, 2014.

[7] M. Gaynor, M. Welsh, S. Moulton, A. Rowan, E. LaCombe, and J. Wynne. Integrating wireless sensor networks with the grid. *IEEE Internet Computing*, July/August 2004.

[8] M. M. Hassan, B. Song, and E.-N. Huh. A framework of sensor-cloud integration opportunities and challenges. In *Proc. the 3rd ACM Int'l Conference on Ubiquitous Info. Mgt. and Comm.*, 2009.

[9] S. Kapadia and B. Krishnamachari. Comparative analysis of push-pull query strategies for wireless sensor networks. In *Proc. International Conference on Distributed Computing in Sensor Systems*, 2006.

[10] M. Li, D. Ganesan, and P. Shenoy. PRESTO: Feedback-driven data management in sensor networks. In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2006.

[11] D. H. Phan, J. Suzuki, S. Omura, K. Oba, and A. Vasilakos. Multiobjective communication optimization for cloud-integrated body sensor networks. In *Proc. IEEE/ACM Int'l Workshop on Data-intensive Process Management in Large-Scale Sensor Systems: From Sensor Networks to Sensor Clouds, In conjunction with IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing*, May 2014.

[12] N. Srinivas and K. Deb. Multiobjective function optimization using nondominated sorting genetic algorithms. *Evol. Computat.*, 2(3), 1995.

[13] P. Taylor and L. Jonker. Evolutionary stable strategies and game dynamics. *Elsevier Mathematical Biosciences*, 40(1), 1978.

[14] H. Wada, P. Boonma, and J. Suzuki. Chronus: A spatiotemporal macroprogramming language for autonomic wireless sensor networks. In N. Agoulmine, editor, *Autonomic Network Mgt. Principles: From Concepts to Applications*, chapter 8. Elsevier, 2010.

[15] J. W. Weibull. *Evolutionary Game Theory*. MIT Press, 1996.

[16] M. Yuriyama and T. Kushida. Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing. In *Proc. the 13th Int'l Conf. on Network-Based Info. Sys.*, 2010.

[17] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms: A comparative study. In *Proc. Int'l Conf. on Parallel Problem Solving from Nature*, 1998.