Adaptive and Stable Integration of Energy Harvesting Aware Body Sensor Networks with Clouds

Yi Cheng-Ren[1], Junichi Suzuki[1], Dung H. Phan[1], Shingo Omura[3] and Ryuichi Hosoya[3]

[1] University of Massachusetts, Boston
Boston, MA 02125-3393, USA
Email: {yiren001,jxs,phdung}@cs.umb.edu
[3]OGIS International, Inc.
San Mateo, CA 94402, USA
Email: {omura,hosoya}@ogis-international.com

This paper considers a multi-tier architecture for cloud-integrated body sensor networks (BSNs), called Body-in-the-Cloud (BitC), which is intended to support home healthcare with on-body energy harvesting devices (e.g., piezoelectric and thermoelectric generators) as well as on-body physiological and activity monitoring sensors. This paper formulates a configuration problem in BitC and approaches the problem with an evolutionary game theoretic algorithm to configure BSNs in an adaptive and stable manner. BitC allows BSNs to adapt their configurations (i.e., sensing intervals and sampling rates 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. In BitC, evolutionary multiobjective games are performed on configuration strategies (i.e., solution candidates) with an aid of local search mechanisms. 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. This paper evaluates five algorithmic variants of BitC under various settings and demonstrates that BitC allows BSNs to successfully leverage harvested energy to balance their performance in different objectives such as resource consumption and data yield.

***Index Terms*—Body sensor networks, Cloud computing, Multiobjective optimization, Evolutionary game theory**

I. INTRODUCTION

BODY sensor networks (BSNs) are expected to aid pervasive healthcare with on-body sensors by remotely and continuously performing physiological and activity monitoring for patients [1], [2]. This paper envisions an architecture for cloud-integrated BSNs, called Body-in-the-Cloud (BitC), which virtualizes per-patient BSNs onto clouds by taking advantage of cloud computing features such as pay-per-use billing, scalability in data storage and processing, availability through multi-regional application deployment and accessibility through universal communication protocols (e.g., HTTP and REST). BitC assumes energy harvesting aware BSNs, each of which operates on-body energy harvesting devices (e.g., piezoelectric and thermoelectric generators) as well as on-body sensors for, for example, heart rate, oxygen saturation, body temperature and fall detection.

BitC consists of the *sensor*, *edge* and *cloud* layers (Fig. 1). The sensor layer is a collection of sensor nodes in BSNs. Each BSN operates one or more sensor nodes, each of which is equipped with a sensor(s) and an energy harvester(s). Sensor nodes are 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 monitor patients and 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 a sensor(s) 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 adjusting four types of parameters (i.e., 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 according 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. This paper considers stability as the reachability to at least one of equilibrium solutions in decision making. A lack of stability results in making inconsistent adaptation decisions in different attempts/trials with the same problem settings.

BitC leverages an evolutionary game theoretic algorithm to configure BSNs in an adaptive and stable manner. This paper describes the design of BitC and evaluates its adaptability and stability. In BitC, each BSN maintains a set (or a population) of configuration strategies (solution candidates), each of which specifies a set of configuration parameters for that BSN. 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, which is always converged to regardless of the initial state. (A dominant strategy in the evolutionarily stable population state is called an *evolutionarily stable strategy* (ESS).) In this state, no other strategies except an ESS can dominate the population. Given this theoretical property, BitC allows each BSN to operate at equilibrium by using an ESS in a deterministic (i.e., stable) manner.

Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies and adapt their configuration parameters to given operational conditions subject to given constraints. This paper evaluates five algorithmic variants of BitC under various settings and demonstrates that BitC allows BSNs to successfully leverage harvested energy to balance their performance with respect to multiple objectives such as resource consumption and data yield. BitC's performance is evaluated in comparison to a well-known multiobjective evolutionary optimization algorithm, NSGA-II [3], while maintaining 37% higher stability (lower oscillations) in performance across different simulation runs.

II. An Architectural Overview of BitC

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 node in a certain topology (e.g., tree, star or mesh topology). This paper assumes the star topology. Each sensor node is equipped with a sensor(s) and an energy harvester(s). It is assumed to be battery-operated. It supplies a limited amount of energy to a sensor(s) attached to it and receives power supply from an attached energy harvester(s) as it/they harvest energy. It maintains a sensing interval and a sampling rate for each attached sensor. 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.

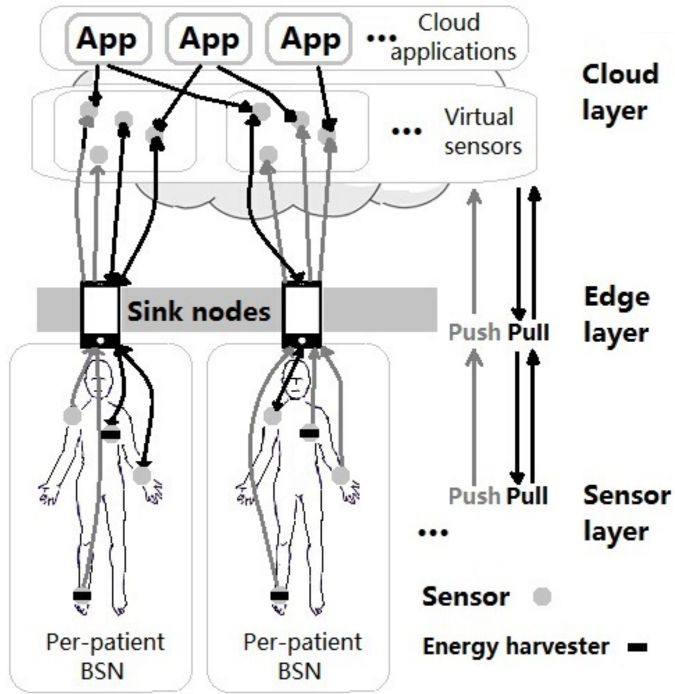

Fig. 1: A Push-Pull Hybrid Communication in BitC

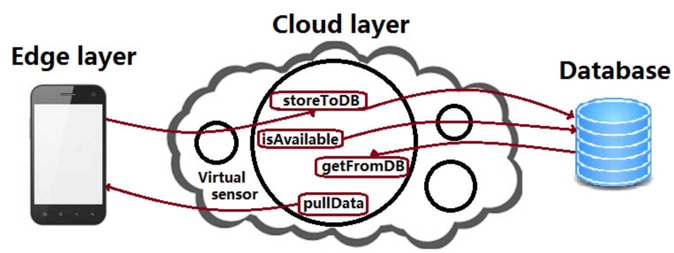

Fig. 2: Virtual sensor communication diagram

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 haves ensor 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 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 data and receiving that data (or an error message).

Virtual sensors are java nodes running in the server on the cloud layer. These virtual sensor nodes are predefined by medical doctors. Each sensor is associated with a four digit identification number, the first two digit indicate the index of the BSN and the last two digit indicate the index of the sensor. Once data are pushed to the cloud layer, corresponding virtual sensor node responds to the arrived data by checking the identification number. Virtual sensors call storeToDB java method which issues a sql query that store the received data to the corresponding table in the database. Every time a request come in virtual sensors first check whether the data is available by calling isAvailable java method. If the data is available then it retrieves the data by calling getFromDB java method that issues a sql query to the corresponding table in the database, if not virtual sensors issues a pull request to Edge layer by calling pullData java method. Once virtual sensors get the desired data, it backs to user.

III. BSN Configuration Problem in BitC

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 sensor nodes 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 sensors: $b_i = \{s_{i1}, s_{i2}, ..., s_{ij}, ..., s_{iM}\}$. o_{ij} is the data transmission interval for s_{ij} to transmit sensor collected data. p_{ij} and q_{ij} are the sensing interval and sampling rate for s_{ij}. Sampling rate is defined as the number of sensor data samples collected in a unit time. Each sensor 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.
- $R_{ij} = \{r_{ij1}, r_{ij2}, ..., r_{ijr}, ..., r_{ij|R_{ijk}|}\}$ denotes the set of sensor data requests that cloud applications issue to the virtual counterpart of s_{ij} (s'_{ij}) during the time period of W in the past. Each request r_{ijr} is characterized by its time stamp (t_{ijr}) and time window (w_{ijr}). It retrieves all sensor data available in the time interval $[t_{ijr} - w_{ijr}, t_{ijr}]$. If s'_{ij} has at least one data in the interval, it returns those data; otherwise, it issues a pull request to m_i.
- $R^m_{ij} \in R_{ij}$ denotes the set of sensor data requests for which a virtual sensor s'_{ij} has no data. $|R^m_{ij}|$ indicates the number of pull requests that s'_{ij} issues to m_i. In other words, $R_{ij} \setminus R^m_{ij}$ is the set of sensor data requests that s'_{ij} fulfills regarding s_{ij}.
- $R^s_{ij} \in R^m_{ij} \in R_{ij}$ denotes the set of sensor data requests for which m_i has no data. $|R^s_{ij}|$ indicates the number of pull requests that m_i issues to h_{ij} for collecting data from s_{ij}. $R^m_{ij} \setminus R^s_{ij}$ is the set of sensor data requests that m_i fulfills regarding s_{ij}.

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.

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

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_{ij} denotes the number of sensor data that s_{ij} generates and sink nodes in turn push to the cloud layer during W. d_{ij} denotes the size of each sensor data (in bits) that s_{ij} generates. It is currently computed as: $q_{ij} \times 16$ bits/sample. ϕ_{ijr} denotes the number of sensor data that a pull request $r \in R^m_{ij}$ can collect from sink nodes ($\phi_{ijr} = |R^m_{ij} \setminus R^s_{ij}|$). 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_{ijr} is the number of sensor data that a pull request $r \in R^s_{ij}$ 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.

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

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:

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

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

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.

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

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 $I_B = 0$.

IV. Background: 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 [4]. This section overviews key elements in evolutionary games: evolutionarily stable strategies (ESS) and replicator dynamics.

A. 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*

$$U(k,\ x\ell+(1-x)k) > U(\ell,\ x\ell+(1-x)k) \quad (5)$$

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

If the payoff function is linear, Equation 5 derives:

$$(1-x)U(k,k) + xU(k,\ell) > (1-x)U(\ell,k) + xU(\ell,\ell) \quad (6)$$

If x is close to zero, Equation 6 derives either

$$U(k,k) > U(\ell,k),\ or$$
$$U(k,k) = U(\ell,k)\ and\ U(k,\ell) > U(\ell,\ell) \quad (7)$$

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.

B. Replicator Dynamics

The replicator dynamics describes how population shares associated with different strategies evolve over time [5]. 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.

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

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* $x_k(t)_{t\rightarrow\infty} \rightarrow 0$.

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 IV-A, 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.

V. 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_{ij} and q_{ij}) as well as data transmission intervals for sink and sensor nodes (o_{m_i} and o_{ij}).

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

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 III.

In each generation (g), a series of games are carried out on every population (Lines 4 to 28). 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 III (Lines 7 to 9). The winner is replicated to increase its population share and mutated with polynomial mutation (Lines 10 to 18) [3]. 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 12 to 14). Then a game is performed between loser and the mutated winner (Line 16). Elitism concept is applied here to select the best two among strategies (winner, loser and mutated winner), and the worst strategy disappears in the population.

Once all strategies play games in the population, BitC identifies a feasible strategy whose population share (x_s) is the highest and determines it as a dominant strategy (d_i) (Lines 20 to 24). After a dominant strategy is determined, BitC performs *local search* to improve the dominant strategy (Line 26). In the end, BitC configures a BSN with the parameters contained in the dominant strategy (Line 27).

A game is carried out based on the superior-inferior relationship between given two strategies and their feasibility (c.f. `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 five schemes to select the winner.

- Pareto dominance (PD): This scheme is based on the notion of *dominance* [6], in which a strategy s_1 is said to dominate another strategy 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.

  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 (HV) metric [7]. 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 dominance 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 with each other, the hypervolume (HV) comparison is used to select the winner. If they still tie with the hypervolume metric, the winner is randomly selected.
- Maxmin (MM): This scheme is based on the maxmin (MM) metric [8]. It measures how distant (i.e., better) a given strategy s is from the other strategies in a population ($s^{'} \in \mathcal{P}_i$).

  $$MM(s) = \max_{s^{'} \in \mathcal{P}_i \backslash \{s\}} \left\{ \min_k \left(s_k, s_k^{'} \right) \right\} \tag{11}$$

  s_k denotes the k-th objective value of the strategy s. Given two strategies, the one with a higher maxmin value wins a game. If both have the same maxmin value, the winner is randomly selected.

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

If both strategies are infeasible in a game, they are compared based on their constraint violation. An infeasible strategy s_1 wins a game over another infeasible 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.

Algorithm 1 Evolutionary Process in BitC

g = 0
Randomly generate the initial N populations for N BSNs: $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_N\}$
while $g < G_{max}$ **do**
  for each population $\mathcal{P}_i$ randomly selected from $\mathcal{P}$ **do**
    $\mathcal{P}'_i \leftarrow \emptyset$
    for $j = 1$ to $|\mathcal{P}_i|/2$ **do**
      $s_1 \leftarrow$ randomlySelect($\mathcal{P}_i$)
      $s_2 \leftarrow$ randomlySelect($\mathcal{P}_i$)
      $\{winner, loser\} \leftarrow$ performGame(s_1, s_2)
      $replica \leftarrow$ replicate($winner$)
      for each parameter v in $replica$ **do**
        if random() $\leq P_m$ **then**
          $replica \leftarrow$ mutate($replica$, v)
        end if
      end for
      $winner' \leftarrow$ performGame($loser$, $replica$)
      $\mathcal{P}_i \setminus \{s_1, s_2\}$
      $\mathcal{P}'_i \cup \{winner, winner'\}$
    end for
    $\mathcal{P}_i \leftarrow \mathcal{P}'_i$
    $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
    while d_i is infeasible **do**
      $\mathcal{P}_i \setminus \{d_i\}$
      $d_i \leftarrow argmax_{s \in \mathcal{P}_i} x_s$
    end while
    $d_i \leftarrow$ localSearch(d_i)
    Configure a BSN in question based on d_i.
  end for
  $g = g + 1$
end while

This paper studies three different local search mechanisms (c.f. `localSearch()` in Algorithm 1). They are all based on polynomial mutation. Algorithm 2 shows the first mechanism, Tabu Local Search (TLS). TLS creates Q mutants of a given strategy (d_i) using polynomial mutation and identifies the best of $Q + 1$ strategies (d_i and its Q mutants). As mutants are created, TLS updates a *tabu list* T to record which parameters have been mutated so that a new mutant is never be created by altering *taboo parameters* (i.e., the parameters in T).

The second local search mechanism is called Greedy Local Search (GLS) (Algorithm 3). Similar to TLS, GLS creates Q mutants iteratively; however, it replaces the original strategy (e.g., d_i) with a mutant if the mutant wins over the original in a game (Line 7). Though Q iterations, GLS keeps the best mutant discovered so far and mutates it when mutation occurs.

Algorithm 2 Tabu Local Search (`localSearch()`)

Input: d_i: Dominant strategy to improve
Output: Improved dominant strategy
$T \leftarrow \emptyset$
for each parameter $v \in d_i$ and $v \notin T$ **do**
  if random() $\leq P_m$ **then**
    $replica \leftarrow$ mutate(d_i, v)
    $T \leftarrow T \cup \{v\}$
  end if
end for
for $i = 1$ to $Q - 1$ **do**
  for each parameter $v \in d_i$ and $v \notin T$ **do**
    if random() $\leq P_m$ **then**
      $replica' \leftarrow$ mutate(d_i, v)
      $T \leftarrow T \cup \{v\}$
    end if
  end for
  $replica \leftarrow$ performGame($replica$, $replica'$)
end for
$best \leftarrow$ performGame($replica$, d_i)
return $best$

Algorithm 3 Greedy Local Search (`localSearch()`)

Input: d_i: Dominant strategy to improve
Output: Improved dominant strategy
1: **for** $i = 1$ to Q **do**
2:   **for each** parameter v in $replica$ **do**
3:     **if** random() $\leq P_m$ **then**
4:       $replica \leftarrow$ mutate(d_i, v)
5:     **end if**
6:   **end for**
7:   $d_i \leftarrow$ performGame($replica$, d_i)
8: **end for**
9: **return** d_i

Algorithm 4 shows the third local search mechanism, Greedy Tabu Local Search (GTLS). It customizes GLS with a tabu list T. It avoids taboo parameters in T when it performs mutation.

VI. 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. The winner of a game earns a higher fitness than the loser.
- $p^s_k = x_k \cdot \phi(F_s - F_k)$ 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:

Algorithm 4 Greedy Tabu Local Search (`localSearch()`)

Input: d_i: Dominant strategy to improve
Output: Improved dominant strategy
$T \leftarrow \emptyset$
for $i = 1$ to Q **do**
  for each parameter $v \in d_i$ and $v \notin T$ **do**
    if random() $\leq P_m$ **then**
      $replica \leftarrow$ mutate(d_i, v)
      $T \leftarrow T \cup \{v\}$
    end if
  end for
  $d_i \leftarrow$ performGame($replica$, d_i)
end for
return d_i

$$\begin{aligned} \dot{x}_s &= \sum_{k \in S^*, k \neq s} \{x_s p_k^s - x_k p_s^k\} \\ &= x_s \sum_{k \in S^*, k \neq s} x_k \{\phi(F_s - F_k) - \phi(F_k - F_s)\} \end{aligned} \quad (12)$$

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 $x_1 = 1 - x_2 - \cdots - x_{|S^*|}$

$$\dot{z}_s = z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|s^*|} z_i \cdot c_{si}], \quad s, k = 2, ..., |S^*| \quad (13)$$

where $c_{sk} \equiv \phi(F_s - F_k) - \phi(F_k - F_s))$ and $Z(t) = \{z_2(t), z_3(t), \cdots, z_{|S^*|}(t)\}$ 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 $s, k = 2, ..., |S^*|$.

$$\begin{aligned} J_{sk} &= \left[\frac{\partial \dot{z}_s}{\partial z_k}\right]_{|Z = Z_{eq}} \\ &= \left[\frac{\partial z_s[c_{s1}(1 - z_s) + \sum_{i=2, i \neq s}^{|S^*|} z_i \cdot c_{si}]}{\partial z_k}\right]_{|Z = Z_{eq}} \end{aligned} \quad (14)$$

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

$$J = \begin{bmatrix} c_{21} & 0 & \cdots & 0 \\ 0 & c_{31} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{|S^*|1} \end{bmatrix} \quad (15)$$

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

VII. Simulation Evaluation

This section evaluates BitC through simulations and discusses how BitC allows BSNs to adapt their configurations to given operational conditions (e.g., data request patterns placed by cloud applications and memory space availability in sink and sensor nodes) Simulations are configured with the parameters shown in Table I.

TABLE I: Simulation Settings

Parameter	Value
Duration of a simulation (W)	10,800 secs (3 hrs)
Number of simulation runs	10
Number of BSNs (N)	20 and 100
Number of sensor nodes in a BSN (M)	4
Memory space in a sensor node	2 GB
Memory space in a sink node	16 GB
Total number of data requests from cloud apps	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
Blood pressure request time window	[0, 1000 secs]
Acelerometer request time window	[0, 1800 secs]
ECG request time window	[0, 600 secs]
Number of generations (G_{max})	300
Number of local search iterations (Q)	20
Population size ($\|\mathcal{P}_i\|$)	100
Mutation rate (P_m)	1/V

This paper assumes a nursing home where senior residents/patients live. A small-scale and a larger-scale simulations are carried out with 20 and 100 residents, respectively. The small-scale setting is used unless otherwise noted. Each resident is simulated to wear four sensors: a blood pressure sensor, an ECG sensor and two accelerometers (Fig. 1).

Cloud applications issue 1,000 data requests during three hours. Data requests are uniformly distributed over virtual sensors. 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 600 seconds for an ECG sensor (Table I). Mutation rate is set to 1/V where V is the number of parameters in a strategy. Every simulation result is the average with 10 independent simulation runs.

TABLE II: Energy Harvesting Configurations

Category	Energy source	Harvested energy in 3 hrs	Total harvested energy in 3 hrs
Very healthy	Piezo (2.0 Hz) Thermo	18.27 W 0.10 W	18.37 W
Healthy	Piezo (1.38 Hz) Thermo	12.56 W 0.068 W	12.63 W
Rehabilitation	Piezo (0.25 Hz) Thermo	2.28 W 0.0124 W	2.29 W
Wheelchair	Piezo (0.0 Hz) Thermo	0.0 W 0.0 W	0.0 W

This paper assumes four types of residents (Table II). 25% of residents are simulated to be in each category. Each resident wears two energy harvesters: piezoelectric energy generator (PEG) and thermoelectric generators (TEG). A PEG and a TEG are assumed to be embedded in a shoe and attached to the skin, respectively (Fig. 1). A PEG generates energy (piezoelectricity) from walking activities of a resident [9]. A TEG generates energy (thermoelectricity) from a resident's body temperature [10].

The amount of harvested energy is computed based on a set of daily activities assumed for each type of residents. For example, a very healthy resident is assumed to have a scheduled walk and an excercise session with, for example, a treadmill under the average walking step frequency of 2 hertz. A PEG and a TEG is assumed to generate 11 mW per step and 0.06 mW per second [9], [10].

TABLE III: Constraint Combinations

Constraint Combination	C_E (W)	C_Y	C_R (%)	C_B (Kbps)
C_∞	∞	0	0	∞
C_{VL}	450	16	90	30
C_L	350	17	93	25
C_M	200	18	95	20
C_S	150	19	97	10
C_{VS}	100	20	99	7
C_{EN}	50	16	90	30
C_{DY}	450	25	90	30

This paper simulates eight different combinations of constraints (Table III): no constraints (C_∞), very lightweight (C_{VL}), lightweight (C_L), moderate (C_M), stringent (C_S), very stringent (C_{VS}), very stringent for energy consumption (C_{EN}) and very stringent for data yield (C_{DY}). C_∞ is used unless otherwise noted.

Comparative performance study is carried out for BitC's five variants (i.e., PD, HV, PD-HV, MM and PD-MM in Section V). BitC-PD is used unless otherwise noted. BitC is also compared with NSGA-II, which is a well-known multiobjective evolutionary algorithm [3]. BitC and NSGA-II use the same parameter settings shown in Table I. All other NSGA-II settings are borrowed from [3]. Both BitC and NSGA-II are implemented with jMetal [11].

Table IV examines how a mutation-related parameter, called distribution index (η_m in [3]), impacts the performance of BitC. This parameter controls how likely a mutated strategy is similar to its original. (A higher distribution index makes a mutant more similar to its original.) In Table IV, the performance of BitC is evaluated with the hypervolume measure that a set of dominant strategies yield in the 300th generation. The hypervolume metric indicates the union of the volumes that a given set of solutions dominates in the objective space [7]. A higher hypervolume means that a set of solutions is more optimal. As shown in Table IV, BitC yields the best performance with the distribution index value of 60. (Local search is not used to obtain this result.) Thus, this parameter setting is used in all successive simulations.

TABLE IV: Impacts of Distribution Index Values on Hypervolume (HV)

Distribution Index	**HV**	**Distribution Index**	**HV**
45	0.886	50	0.910
55	0.912	60	0.917
65	0.903		

Fig. 3 studies how different local search mechanisms impact the performance of BitC. It verifies that all of them can improve BitC's performance. Among them, greedy local search (GLS) is the most effective in both convergence speed and hypervolume. Thus, GLS is used in all successive simulations.

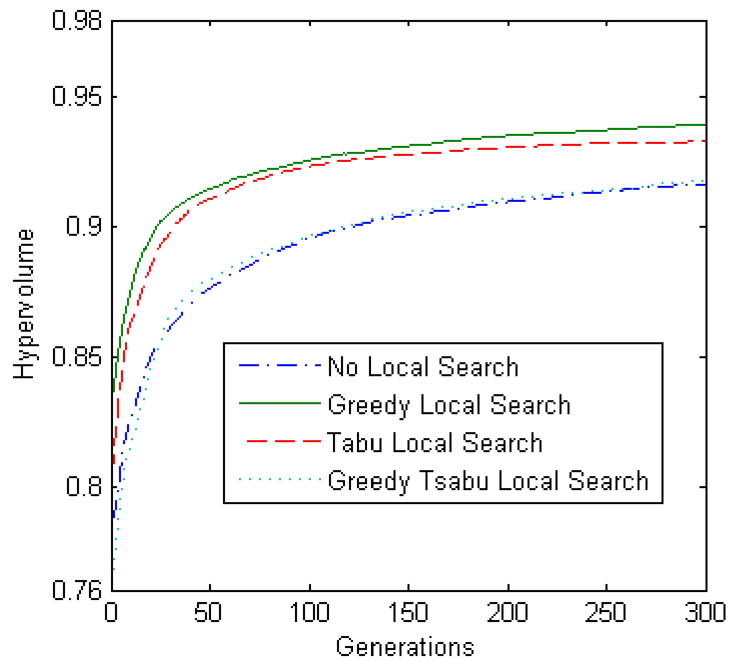

Fig. 3: Local Search Comparison

Table V illustrates the hypervolume that each BitC variant yields at the last generation. As shown in this table, BitC-HV yields the highest hypervolume value among five variants. Therefore, the variant is used in all successive simulations.

TABLE V: Comparison of BitC's Variants in Hypervolume

Algorithm	Hypervolume (HV)
BitC PD	0.9247
BitC HV	0.9394
BitC PD-HV	0.9056
BitC MM	0.9071
BitC PD-MM	0.9143

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 Table VI, the BitC solution is evaluated against an NSGA-II individual that is closest to the solution in the objective space. Table VI compares BitC-HV and NSGA-II based on three 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, 2].)

As shown in Table VI, BitC-HV is non-dominated with NSGA-II with respect to four objectives. BitC-HV outperforms NSGA-II and it gets very close to the performance bounds in three objectives (request fulfillment, bandwidth consumption and energy consumption) while NSGA-II outperforms BitC-HV in data yield. BitC-HV yields a slightly higher (2% higher) hypervolume value than NSGA-II. Table VI demonstrates that BitC-HV slightly outperforms NSGA-II on an a solution-to-solution basis. The performance bounds are given by running NSGA-II with single objective. Euclidean and Manhattan distances are used as metrics. In both metrics, a shorter distance means a given solution is closer to the performance bounds. BitC is closer to the bounds than NSGA-II by 29% and 34% in Euclidean and Manhattan distances, respectively.

TABLE VI: Comparison of BitC-HV and NSGA-II

Objective	NSGA-II	BitC-HV	Bounds
Request Fulfillment: f_R (%)	97.6	98.35	99.00
Bandwidth: f_B (Kbps)	10.45	7.41	7.32
Energy consumption: f_E (Watts)	178.89	129.26	126.91
Data Yield: f_Y	37.92	14.54	44.72
Hypervolume	0.922	0.9394	-
Euclidean distance	0.152	0.108	-
Manhattan distance	0.232	0.153	-

Table VII shows the variance of objective values that BitC-HV and NSGA-II yield at the last generation in 10 different simulation runs. A lower variance means higher stability (or higher similarity) in objective value results (i.e., lower oscillations in objective value results) among different simulation runs. BitC-HV maintains significantly higher stability than NSGA-II in all objectives except energy consumption. On average, BitC's stability is 36.75% higher than NSGA-II's. This result exhibits BitC's stability property (i.e. reachability to at least one equillibira), which NSGA-II does not have.

TABLE VII: Stability of Objective Values in BitC-HV and NSGA-II

Objectives	BitC-HV	NSGA-II	Diff (%)
Request Fulfillment: f_R	0.05	0.5	90%
Bandwidth: f_B	0.17	0.22	22.72%
Energy Consumption: f_E	3.38	3.01	-12.33%
Data Yield: f_Y	0.42	0.97	56.70%
Average Difference (%)	–	–	36.75%

Fig. 4 shows two three-dimensional objective spaces that plot a set of dominant strategies obtained from individual populations at each generation. Each blue dot indicates the average objective values that dominant strategies yield at a particular generation in 10 simulation runs. The trajectory of blue dots illustrates a path through which strategies evolve and improve objective values. Gray and red dots represent 10 different sets of objective values at the first and last generation in 10 simulation runs, respectively. While initial (gray) dots disperse (because the initial strategies are generated at random), final (red) dots are overlapped in a particular region. Consistent with Table VII, Fig. 4 verifies BitC's stability: reachability to at least one equilibria regardless of the initial conditions.

Table VIII evaluates how different constraint combinations impact on the performance of BitC in objective values. The table shows the average, maximum and minimum objective values at the last generation subject to eight constraint combinations listed in Table III. BitC successfully satisfies four

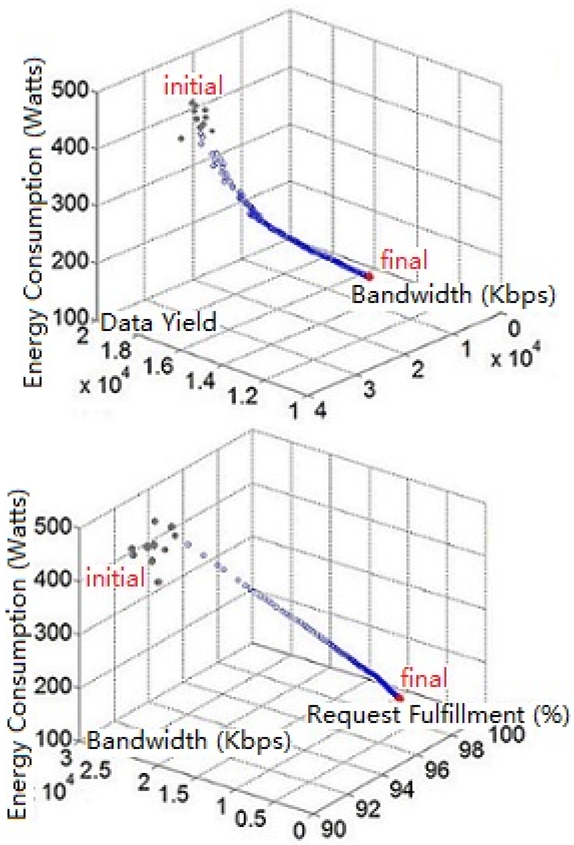

Fig. 4: Three-dimensional Objective Spaces

constraint combinations (C_{VL},C_L, C_M and C_S). Under C_{EN} and C_{DY}, BitC fails to satisfy a constraint in each case although it satisfies three other constraints. And BitC fails to satisfy all the objectives subject to C_{VS} constraints, since the constraints setting is very stringent. The best performance is produced under C_S. In most of the cases BitC fails to satisfy the data yield constraint due to it is a conflicting objective with other three, and BitC tries to balance the trade off among all the four objectives. Comparing the results of C_S and C_∞, and the results of C_S and C_{VS}, Table VIII demonstrates that BitC satisfies the data yield constraint by trading three other objectives for a global better performance. Given this result, C_S is used in all successive simulations.

Fig. 5 shows how BitC improves its performance through generations with 20 BSNs (i.e., 20 patients) and 100 BSNs. It shows the changes of objective values over generations. All four constraints are satisfied at the last generation. The two figures illustrate that BitC improves its objective values subject to given constraints by balancing the trade-offs among conflicting objectives. For example, BitC improves both request fulfillment and bandwidth consumption through generations while the two objectives conflict with each other.

Results are qualitatively similar comparing both results with 20 BSNs and 100 BSNs, although BitC yields a slightly lower performance with a larger number of BSNs. It is harder to satisfy given constraints in a larger-scale setting. (All four constraints are satisfied at the last generation.) BitC 100 BSNs's performance decreases a little respect to 20 BSNs in

TABLE VIII: Objective Values of BitC-HV under Different Constraint Combinations

Constraint Combination		f_B (Kbps)	f_E (W)	f_R(%)	f_Y
C_∞	maximum	11.38	195.81	97.7	30.25
	Avg	11.07	190.28	97.57	27.33
	minimum	10.88	186.96	97.4	25.54
C_{VL}	maximum	12.11	208.96	97.3	17.64
	Avg	11.8	203.77	97.27	16.62
	minimum	11.53	198.94	96.9	15.64
C_L	maximum	12.45	214.72	97.1	14.48
	Avg	12.11	209.72	97.04	14.32
	minimum	11.87	205.84	97	14.19
C_M	maximum	9.13	159.93	97.85	17.53
	Avg	9.04	157.98	97.81	16.39
	minimum	8.90	155.14	97.7	15.61
C_S	maximum	7.69	133.37	98.4	14.86
	Avg	7.41	129.26	98.35	14.54
	minimum	7.25	126.68	98.3	14.02
C_{VS}	maximum	9.34	161.09	97.9	15.45
	Avg	9.24	159.48	97.88	14.95
	minimum	9.05	156.81	97.85	14.59
C_{EN}	maximum	12.08	208.63	96.95	16.50
	Avg	11.81	204.66	96.94	15.87
	minimum	11.58	201.50	96.9	15.00
C_{DY}	maximum	11.89	206.14	97.45	15.41
	Avg	11.32	195.72	97.32	14.75
	minimum	10.93	188.75	97.2	13.91

bandwidth consumption and energy consumption. It happens due to the increasing number of BSNs, BitC needs more bandwidth and energy to satisfy all BSNs needs. And as consequence the data yield increases, but 20 BSNs reaches slightly higher request fulfillment because it is easier for BitC to satisfy greater number of requests for smaller number 20 BSNs rather than 100 BSNs. Fig. 5 demonstrates that BitC scales well against the number of BSNs.

At the last generation BitC reaches hypervolume value 0.936 and 0.922 for 20 BSNs and 100 BSNs respectively.

Table IX examines how BitC and NSGA-II maintain the lifetime of BSNs and yield data yield performance with energy harvesting enabled and disabled. Both BitC and NSGA-II utilize harvested energy to extend the lifetime of BSNs and in turn improve data yield performance. With energy harvesting enabled under a stringent set of constraints (C_S), BitC extends the BSN lifetime by 4.6%. BitC maintains 39% longer lifetime than NSGA-II. Table IX demonstrates that BitC successfully leverages energy harvesting to improve its performance.

TABLE IX: Comparison of BitC-HV and NSGA-II in BSN Lifetime and Data Yield with Energy Harvesting (EH) Enabled and Disabled

Algorithms		Lifetime (hrs)	Data yield (f_Y)
C_∞	BitC-HV w/o EH	2.36	21,505
	BitC-HV w/ EH	2.47 (+4.66%)	22,507
	NSGA-II w/o EH	0.66	4,702
	NSGA-II w/ EH	0.69 (+4.55%)	4,916
C_S	BitC-HV w/o EH	3.48	16,866
	BitC-HV w/ EH	3.64 (+4.60%)	17,641
	NSGA-II w/o EH	2.51	31,727
	NSGA-II w/ EH	2.62 (+4.38%)	33,117

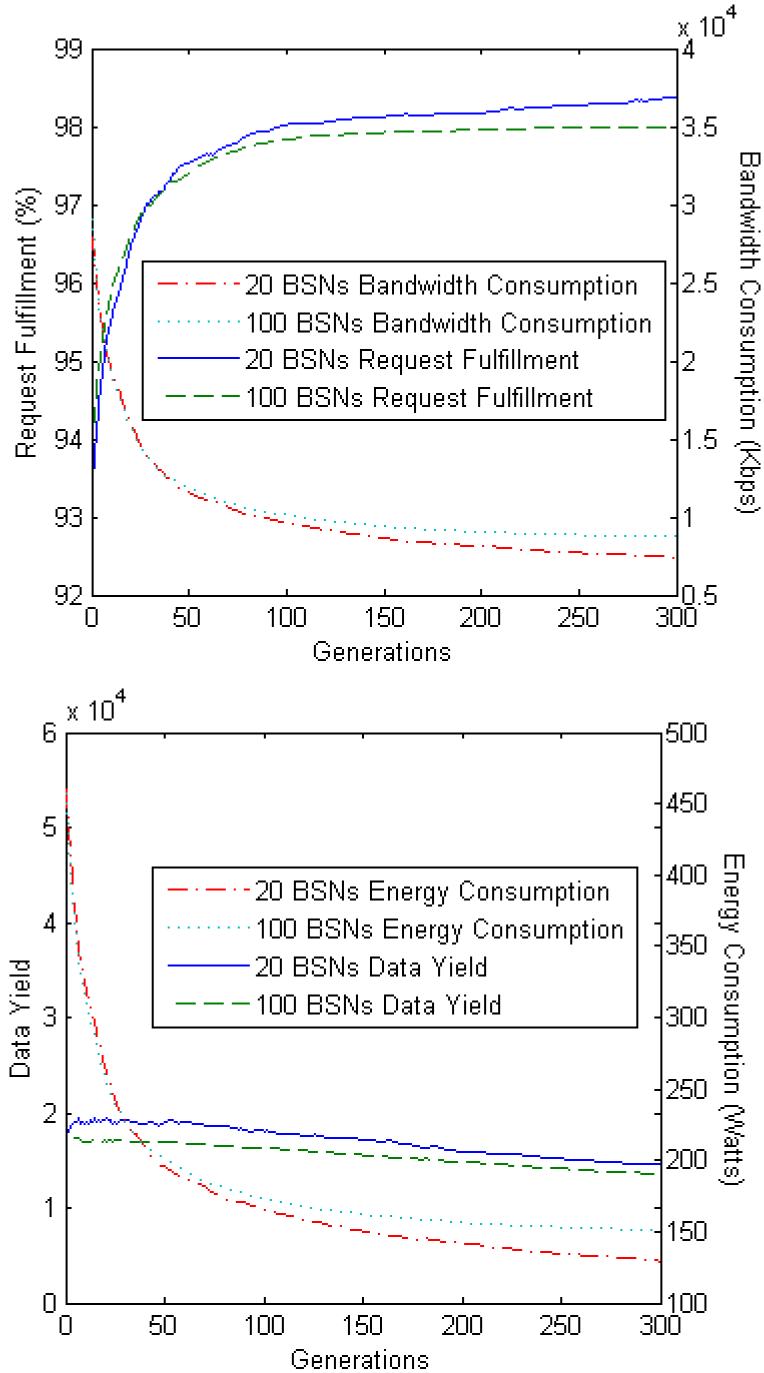

Fig. 5: 20 BSNs and 100 BSNs performance comparison

VIII. Related Work

This paper extends a prior work of the authors [12]. A similar BSN configuration problem is considered in [12]; however, this paper extends the problem by incorporating energy harvesters and investigates how to leverage harvested energy to improve and balance the performance of BSNs by adjusting their configuration parameters. This paper also makes algorithmic extensions; it examines three local search operators as well as five different schemes to carry out games. All of these are not studied in [12]. Moreover, this paper

TABLE X: Notation table

Notation	Description	Notation	Description
B	set of BSNs	P_i	population size
N	number of BSNs	Q	number of local search iterations
M	number of sensors	f_R	request fulfillment objective
m	sink node	f_B	bandwidth objective
O	data transmission interval	f_E	energy consumption objective
p	sensing interval	f_Y	data yield objective
q	sampling rate	C_R	request fulfillment constraint
R	data requests	C_B	bandwidth constraint
K	set of strategies	C_E	energy consumption constraint
G	number of generations	C_Y	data yield constraint
P_m	mutation rate		

conducts larger-scale simulations with up to 100 BSNs while only 10 BSNs are simulated in [12].

Various architectures and research tools have been proposed for cloud-integrated sensor networks including BSNs [13]–[27]. Many of them, [13]–[22], 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. [23], Rollin et al. [24] and Chung et al. [26] propose a two-tier architecture that consists of the sensor and cloud layers. The architectures proposed by Yuriyama et al. and Fortino et al. [27] 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 relevant work do not consider adaptive/stable configurations of sensor networks as BitC does [14]–[27].

Push-pull hybrid communication has been studied in sensor networks [28]–[31]. 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.

Xu et al. propose a three-tier architecture called CEB (Cloud, Edge and Beneath), which is similar to BitC, and investigate a mechanism to adapt data transmission rates between layers according to a given pattern of data requests [32]. CEB runs two optimization algorithms collaboratively: OPT-1 and OPT-2, which optimize data transmission rates between the cloud and edge layers and between the edge and sensor layers, respectively. Optimization is carried out on a sensor node by sensor node basis with respect to a single objective: energy consumption. In contrast, BitC considers sensing intervals and sampling rates for sensors as well as data transmission rates for nodes and runs a single algorithm for the entire group of sensor and sink nodes with respect to multiple conflicting objectives including energy consumption.

Kumrai [33] proposes a novel incentive mechanism for participatory sensing based on the evolutionary algorithm. It considers energy consumption optimization problem similar to BitC, however it does not consider multi objective performance scenario.

SC-iPaaS (Sensor-Cloud Integration Platform as a Service) is similar to BitC in that both consider three-tier architecture for cloud-integrated BSNs and caries out a single algorithm for the entire group of sensor and sink nodes with respect to multiple conflicting objectives including energy consumption. SC-iPaaS uses an evolutionary game theoretic algorithm that retains stability (i.e. reachability to at least one Nash equilibria) as well as adaptability in configuring BSNs while a genetic algorithm is used in SC-iPaaS [34]. As stochastic global search algorithms, genetic algorithms lack stability.

IX. Conclusion

This paper considers a layered push-pull hybrid communication for cloud-integrated BSNs and formulates a BSN configuration problem to seek adaptive and stable solutions. An evolutionary game theoretic algorithm is used to approach the problem. A theoretical analysis proves that the proposed algorithm allows each BSN to operate at an equilibrium by using an evolutionarily stable configuration strategy in a deterministic (i.e., stable) manner. Simulation results verify this theoretical analysis; BSNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies.

References

[1] M. Chen, S. Gonzalez, A. V. Vasilakos, H. Cao, and V. C. Leung, "Body area networks: A survey," *Mobile Netw. Appl.*, vol. 16, no. 2, 2011.

[2] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of Neuroengineering and Rehabilitation*, vol. 9, no. 21, 2012.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans Evol. Computat.*, vol. 6, no. 2, 2002.

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

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

[6] N. Srinivas and K. Deb, "Multiobjective function optimization using nondominated sorting genetic algorithms," *Evol. Computat.*, vol. 2, no. 3, 1995.

[7] 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.

[8] C. A. Coello Coello, "Using the min-max method to solve multiobjective optimization problems with genetic algorithms," in *Progress in Artificial Intelligence–IBERAMIA 98*, 1998.

[9] J. A. Paradiso and T. Starner, "Energy scavenging for mobile and wireless electronics," *Pervasive Computing*, vol. 4, no. 1, p. 1827, 2005.

[10] D. Salerno, "Ultralow voltage energy harvester uses thermoelectric generator for battery-free wireless sensors," *LT Journal of Analog Innovation*, vol. 20, no. 3, 2010.

[11] J. Durillo, A. Nebro, and E. Alba, "The jMetal framework for multi-objective optimization: Design and architecture," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010.

[12] Y. Cheng-Ren, J. Suzuki, D. H. Phan, S. Omura, and R. Hosoya, "Configuring cloud-integrated body sensor networks with evolutionary algorithms," in *Proc. 9th Int'l Conference on Body Area Networks*, 2014.

[13] M. Dong, H. Li, K. Ota, L. Yang, and H. Zhu, "Multicloud-based evacuation services for emergency management," *Cloud Computing, IEEE*, vol. 1, no. 4, pp. 50–59, Nov 2014.

[14] 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.

[15] 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.

[16] 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.

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

[18] N. Suryadevara, M. T. Quazi, and S. Mukhopadhyay, "Intelligent sensing systems for measuring wellness indices of the daily activities for the elderly," in *8th Int'l Conference on Intelligent Environments*, 2012.

[19] A. Ambrose, M. Cardei, and I. Cardei, "Patient-centric hurricane evacuation management system," in *29th IEEE Int'l Performance Computing and Communications Conference*, 2010.

[20] M. Boulmalf, A. Belgana, T. Sadiki, S. Hussein, T. Aouam, and H. Harroud, "A lightweight middleware for an e-health wsn based system using android technology," in *Int'l Conference on Multimedia Computing and Systems*, 2012.

[21] K. E. U. Ahmed and M. A. Gregory, "Integrating wireless sensor networks with cloud computing," in *7th International Conference on Mobile Ad-hoc and Sensor Networks*, 2011.

[22] P. Zhang, Z. Yan, and H. Sun, "A novel architecture based on cloud computing for wireless sensor network," in *2nd International Conference on Computer Science and Electronics Engineering*, 2013.

[23] 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.

[24] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi, and G. S. Salvador, "A cloud computing solution for patient's data collection in health care institutions," in *Proc. the 2nd IARIA Int'l Conference on eHealth, Telemedicine and Social Medicine*, 2010.

[25] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proc. the 6th ACM Int'l Conference on Embedded Network Sensor Systems*, 2008.

[26] W.-Y. Chung, P.-S. Yu, and C.-J. Huang, "Cloud computing system based on wireless sensor network," in *Federated Conference on Computer Science and Information Systems*, 2013.

[27] G. Fortino, D. Parisi, V. Pirrone, and G. D. Fatta, "BodyCloud: A SaaS approach for community body sensor networks," *Future Generation Computer Systems*, vol. 35, no. 6, pp. 62–79, 2014.

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

[29] 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.

[30] 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.

[31] 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.

[32] Y. Xu, S. Helal, M. Thai, and M. Scmalz, "Optimizing push/pull envelopes for energy-efficient cloud-sensor systems," in *Proc. the 14th ACM Int'l Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2011.

[33] T. Kumrai, K. Ota, M. Dong, and P. Champrasert, "An incentive-based evolutionary algorithm for participatory sensing," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, Dec 2014, pp. 5021–5025.

[34] D. H. Phan, J. Suzuki, S. Omura, K. Oba, and A. Vasilakos, "Multi-objective 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*, 2014.