An Evolutionary Game Theoretic Approach for Configuring Cloud-integrated Body Sensor Networks

Yi Cheng-Ren*, Junichi Suzuki*, Dung H. Phan*, Shingo Omura‡ and Ryuichi Hosoya‡
* University of Massachusetts, Boston
Boston, MA 02125-3393, USA
Email: {yiren001,jxs,phdung}@cs.umb.edu
‡OGIS International, Inc.
San Mateo, CA 94402, USA
Email: {omura,hosoya}@ogis-international.com

***Abstract*—This paper considers a cloud-integrated architecture for body sensor networks (BSNs), called Body-in-the-Cloud (BitC), and studies an evolutionary game theoretic approach 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 for nodes) to operational conditions (e.g., data request patterns) with respect to multiple conflicting objectives such as resource consumption and data yield. Moreover, BitC allows each BSN to perform an evolutionarily stable configuration strategy, which is an equilibrium solution under given operational conditions. Simulation results show that BitC effectively configures BSNs by seeking the trade-offs among conflicting objectives.**

I. Introduction

Home healthcare is the most rapidly growing segment of the U.S. healthcare system since 1990s for both acute and chronic cares. In 2012, it recorded the highest spending growth (5.1%) among all healthcare segments in the US [1].

To address the quality of life and economic issues in home healthcare, various research efforts have been made for the development of *body sensor networks* (BSNs), each of which is a per-patient wireless network of on-body and/or in-body sensors for, for example, heart rate, blood pressure and fall detection [2]. BSNs can be used to remotely and continuously perform physiological and activity monitoring for homebound patients. This paper envisions *cloud-integrated* BSNs, which virtualize on/in-body sensors with clouds for home healthcare by taking advantage of cloud computing features such as pay-per-use billing, scalability in data storage and processing, and availability through multi-regional application deployment.

This paper proposes an architecture for cloud-integrated BSNs, called Body-in-the-Cloud (BitC), which 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 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.

BitC leverages an evolutionary game theoretic approach to configure BSNs. Each BSN maintains a set (or a population) of configuration strategies. It is theoretically proven that, through a series of evolutionary games between 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 an equilibrium by using an evolutionarily stable strategy. Simulation results show that BitC effectively configures BSNs to maintain and improve their performance by seeking the trade-offs among conflicting objectives with evolutionary stable strategies.

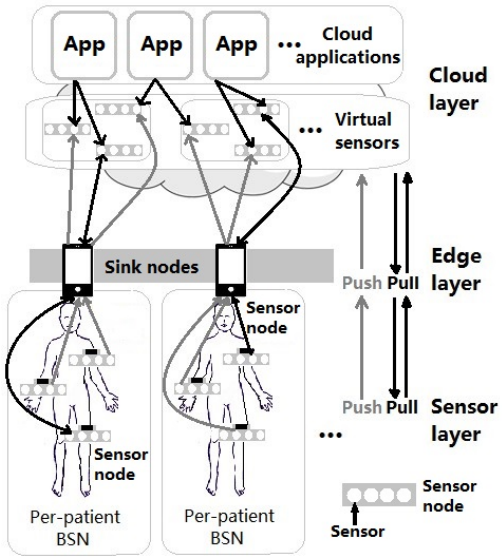


Fig. 1: A Push-Pull Hybrid Communication in BitC

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 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).

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 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.

$$f_B = \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}(c_{ijk}d_{ijk}) + \frac{1}{W}\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{L}\sum_{r=1}^{|R^m_{ijk}|}(\phi_{ijkr}d_{ijk}+d_r) \tag{1}$$

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_{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 a sink node ($\phi_{ijkr} = |R^m_{ijk} \setminus R^s_{ijk}|$). d_r is the size of a pull request transmitted from the cloud layer to the edge layer. It is constant for all sensor nodes and sensors.

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.

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

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 single bit of data consumes to travel in between a sensor node and a sink node. η_{ijkr} is the number of sensor data that a pull request $r \in R^s_{ijk}$ can collect from a sensor node. d'_r denotes the size of a pull request from the edge layer to the sensor layer. It is constant for all sensor nodes and sensors. The third and fourth terms indicate the energy consumption by push and pull communication between the edge and cloud layer, respectively.

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

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

$I_{R_{ijk}} = 1$ 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.

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

IV. Background: Evolutionary Game Theory

In a conventional game, the objective of a player is to choose a strategy that maximizes its payoff. In contrast, evolutionary games are played repeatedly by players randomly drawn from a population [3]. 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*

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

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.

B. Replicator Dynamics

The replicator dynamics describes how population shares associated with different strategies evolve over time [4]. 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 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.

In evolutionary game theory, it has been theoretically proved that, through a series of games, each population reaches a steady state regardless of the initial state [3]. This reachability to at least one of Nash equilibria guarantees to obtain an ESS from a population in a stable (or deterministic) manner.

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_{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). In each generation (g), a series of games are carried out on every population (Lines 4 to 24). 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 loser disappears in the population. The winner is replicated to increase its population share and mutated with a certain mutation rate P_m (Lines 10 to 15). Mutation randomly chooses one of sensor node in the winner and alters its o_{ijk},p_{ijk} and q_{ijk} values at random (Line 12).

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 18 to 22). A strategy is said to be feasible if it violates none of four constraints described in Section III. BitC configures a BSN with parameters contained in the dominant strategy.

A strategy s_1 is said to dominate another strategy s_2 if both of the following conditions are 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

```
1:  g = 0
2:  Randomly generate the initial N populations for N BSNs: P = {P_1, P_2, ..., P_N}
3:  while g < G_max do
4:      for each population P_i randomly selected from P do
5:          P'_i ← ∅
6:          for j = 1 to |P_i|/2 do
7:              s_1 ← randomlySelect(P_i)
8:              s_2 ← randomlySelect(P_i)
9:              winner ← performGame(s_1, s_2)
10:             replica ← replicate(winner)
11:             if random() ≤ P_m then
12:                 replica ← mutate(winner)
13:             end if
14:             P_i \ {s_1, s_2}
15:             P'_i ∪ {winner, replica}
16:         end for
17:         P_i ← P'_i
18:         d_i ← argmax_{s∈P_i} x_s
19:         while d_i is infeasible do
20:             P_i \ {d_i}
21:             d_i ← argmax_{s∈P_i} x_s
22:         end while
23:         current patient transmits data based on d_i.
24:     end for
25:     g = g + 1
26: end while
```

VI. Simulation Evaluation

This section evaluates BitC through simulations and discusses how BitC allows BSNs to 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) and improve their performance. Simulations are configured with the parameters in Table I.

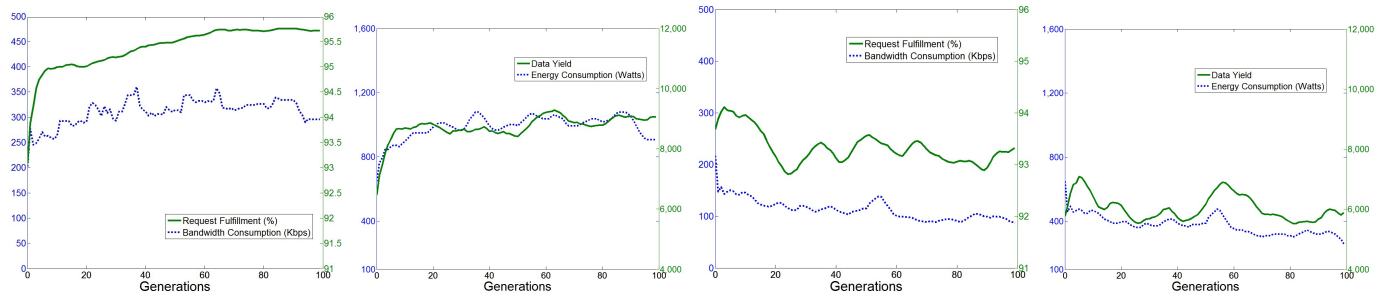
TABLE I: Simulation Settings

Parameter	Value
Duration of a simulation (W)	10,800 seconds (3 hours)
Number of BSNs (N)	5
Number of sensor nodes in a BSN (M)	3
Number of sensors per a sensor 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 apps	1,000
Maximum request windows size ($\max w_{ijk}$)	60 seconds
Number of generations (G_{max})	100
Population size ($\|\mathcal{P}_i\|$)	100
Mutation rate (P_m)	0.01

Cloud applications issue 1,000 sensor data requests during three hours. Sensor data requests are uniformly distributed over sensors. Every simulation result is the average with 20 independent simulation runs.

Fig. 2 shows how BitC evolves BSN configuration strategies through generations and improves the performance of BSNs. In Figs. 2a and 2b, the request fulfillment objective is considered. In Figs. 2c and 2d, all four objectives are considered simultaneously.

In Fig. 2a, request fulfillment (f_R) gradually increases through generations because it is considered as the objective. Its average value reaches 96.5% at the last generation. The



(a) f_B and f_R with the Request Fulfillment Objective Enabled **(b)** f_E and f_Y with the Request Fulfillment Objective Enabled **(c)** f_B and f_R with All Objectives Enabled **(d)** f_E and f_Y with All Objectives Enabled

Fig. 2: Objective Values through Generations

increase in request fulfillment contributes to the increase in data yield (f_Y) to some extent (Fig. 2b). Bandwidth consumption (f_B) and energy consumption (f_E) deteriorate over time because they are not considered as objectives and they conflict with the request fulfillment objective. Figs. 2a and 2b demonstrate that BitC allows BSNs to successfully adapt their performance with respect to a given objective.

Figs. 2c and 2d illustrates that, when multiple conflicting objectives are considered simultaneously, BitC allows BSNs to maintain their performance subject to given constraints by balancing the trade-offs among those objectives.

VII. Related Work

This paper extends a prior work on cloud-integrated BSNs [5]. This paper formulates a more realistic problem (Section III) than the one in [5] by accommodating decision variables available in Simmer's sensor nodes[1]. Moreover, this paper leverages an evolutionary game theoretic algorithm that possesses stability (i.e. reachability to at least one Nash equilibria) in configuring BSNs while a genetic algorithm is used in [5]. As stochastic global search algorithms, genetic algorithms lack stability.

Various architectures and research tools have been proposed for cloud-integrated sensor networks including BSNs [6]–[11]. Hassan et al. [6], Aberer et al. [7], Gaynor et al. [8] and Boonma et al. [9] 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 [10]. The architectures proposed by Yuriyama et al. and Fortino et al. [11] 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.

VIII. Conclusion

This paper proposes a cloud-integrated BSN architecture, called BitC, which hosts virtualized sensors in clouds and operates physical sensors through their virtual counterparts. BitC performs push-pull hybrid communication between three layers: cloud, edge and sensor layers. This paper formulates a BSN configuration problem for BitC to seek equilibrium solutions and approaches the problem with an evolutionary game theoretic algorithm. Simulation results demonstrate that BitC allows BSNs to adapt their configurations to given operational conditions and improve their performance with respect to multiple objectives.

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

References

[1] U.S. Department of Health and Human Services, Centers for Medicare and Medicaid Services, *National Health Expenditure Data*, 2013.

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

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

[5] D. H. Phan, K. O. J. Suzuki, S. Omura, 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.