Body-in-the-Cloud: Towards Visualization-driven Optimization and Stabilization for Cloud-integrated Body Area Networks

Yi Cheng-Ren
Dept. of Computer Science
University of Massachusetts
Boston, Boston, MA, USA
yiren001@cs.umb.edu

Junichi Suzuki
Dept. of Computer Science
University of Massachusetts
Boston, Boston, MA, USA
jxs@cs.umb.edu

Ryuichi Hosoya
OGIS International, Inc.
San Mateo, CA 94404, USA
hosoya@ogis-international.com

ABSTRACT

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Management, Performance

Keywords

Cloud computing, software-defined networks, body sensor networks, multiobjective optimization, evolutionary game theory

1. INTRODUCTION

An emerging era of the Internet of Things faces *manageability* and *configurability* issues due to accelerated proliferation of pervasive sensor and actuation devices. This paper addresses these issues in Body-in-the-Cloud (BitC), which is a tiered architecture for cloud-integrated body area networks (BSNs). BitC is designed to facilitate biomedical and activity monitoring in home healthcare with two types of sensor networks (SNs): (1) body sensor networks (BSNs), each of which is a per-patient wireless network of on/in-body sensors such as ECG and accelerometers and (2) ambient sensor networks (ASNs), which wirelessly connect ambient sensors such as RFID sensors attached on the wall and foot pressure measurement mats on the floor.

BitC approaches manageability and configurability issues with the notion of *software-defined sensor networks* (SDSNs) and a set of optimization algorithms, respectively. An SDBSN is a network of *virtual sensors*, which are virtualized counterparts (or software counterparts) of physical sensors in a cloud computing platform(s). It represents physical SNs in the cloud and allows human administrators and cloud-based services to configure, deploy, optimize and monitor SNs. BitC's optimization algorithms allow SNs to adapt and stabilize their configurations (e.g., sampling rates and data transmission intervals) according to operational conditions (e.g., data request patterns). This paper describes the design and implementation of BitC and evaluates it through simulation and empirical studies.

2. RELATED WORK

To address the quality of life and economic issues in medical center and home healthcare, various research efforts have been made for developing *body sensor networks* (BSNs), each of which is a per-patient wireless network of on/in-body sensors. For example, heart rate, blood pressure, oxygen saturation, body temperature respiratory rate, blood coagulation, galvanic skin response and fall detection [1,3,4]. BSNs can be used to remotely and continuously perform physiological and physical activities monitoring for patients. In another hand ASNs are able to provide not just physical activities monitoring, but also geographical location detection services by using RFID sensors.

This paper envisions putting together BSNs and ASNs to obtain an accurate and reliable location based healthcare monitoring with *cloud-integrated* system BitC, which virtualize on/in-body sensors in the clouds. BitC is seamlessly integrated with cloud for patients healthcare by taking advantage of cloud computing features such as pay-per-use billing and scalability in data storage and processing. And also availability through multi-regional application deployment and accessibility through universal communication protocols (e.g., MQTT, HTTP/REST).

3. DESIGN AND IMPLEMENTATION

BitC consists of three layers, *sensor*, *edge* and *cloud* (Fig. 2). The sensor layer is a collection of sensor nodes in BSNs or/and ASNs. Each BSN or ASN operates one or more sensor nodes, each of which is equipped with one or several sensor(s). In BSN 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. And in ASN sensors nodes are wirelessly connected to a central computer (e.g., local server machine) which acts as a *sink* node.

The edge layer consists of sink nodes, which collect sensor data from sensor nodes in BSNs and ASNs. The cloud layer consists of cloud environments that host *virtual sensors*, which are virtualized counterparts (or software coun-

terparts) of physical sensors in BSNs and ASNs. 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.

BitC configures both BSNs and ASNs by tuning sensing intervals and sampling rates for sensors as well as data transmission intervals for sensor and sink nodes. Two properties are considered in configuring BSNs and ASNs:

- *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 and ASNs. Each BSN and ASN maintains a set (or a population) of configuration strategies. BitC theoretically guarantees 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 to operate at equilibria by using an evolutionarily stable strategy to configure BSNs and ASNs in a deterministic (i.e., stable) manner.

Simulation results verify this theoretical analysis. BSNs and ASNs seek equilibria to perform adaptive and evolutionarily stable configuration strategies. This paper evaluates BitC algorithm and to compares it with NSGA-II and NSGA-III, two well-known multiobjective genetic algorithm. BitC outperforms these two existing well-known genetic algorithm in the quality, stability and computational cost in configuring BSNs and ASNs.

In real experiment settings, the algorithm runs in the cloud server and it could be triggered by changes happened in the BitC environment such as registration of new patients, patients discharged from hospital or recovered from rehabilitation, changes in the requests pattern. If one of these environment changes occurs, then BitC reruns the algorithm in the back end system to adapt the new environment and getting the newest configuration parameters as result. Later these new configuration parameters are propagated towards each BSN and ASN's sink node (i.e. smart phone, tablets, local server, ...) to configure sensor nodes and sensors. There are four type of parameters to configure each BSN or ASN.

- Sink node transmission time interval: It specifies the time interval that a sink node pushes the stored data to the cloud layer.
- Sensor node transmission time interval: It refers to the interval of time that a sensor node pushes the sensing data to the edge layer (Sink node).
- Sensor sensing time interval: It is the time interval that a sensor collects the sensing samples.
- Sensor sampling rate: It indicates the number of samples collected every time by a sensor.

To maintain a high accuracy, each sample is encoded using 16 bits (2 bytes). Each time the client side device (sink node devices) is turned on, it will automatically issue a request to the back end server (cloud server) to get the latest configuration parameters and it will keep doing this in period of one hour to update its parameters. Users can get their updated BSNs configuration parameter manually by performing actions like pressing button or simply swiping down the touch screen. Users also have the full control of starting and stopping their own BSNs sensing and communication system. Up on sensing started, they are able to monitor their health status (i.e. heart rate, blood pressure, etc ...) and physical activities in their own sink node devices.

Sensors collect the data periodically based on their own sensing time interval and sampling rate. After the sensing data are transmitted to the sink node based on the sensor node transmission interval and then are pushed to the cloud server followed the interval of time configured in the sink node. Communication protocol between Edge layer and Cloud layer could be HTTP or/and MQTT, depending on the sink node types. In case of BSNs, sink nodes are smart phones or tablets and MQTT would handle to publish and to subscribe sensors data to the cloud. MQTT is designed to be battery and bandwidth efficient which making it well suited to be used in environments like smart phone and tablets. When it comes to ASNs, HTTP would be used to handle data transferring between the local server and cloud server (Fig. 1).

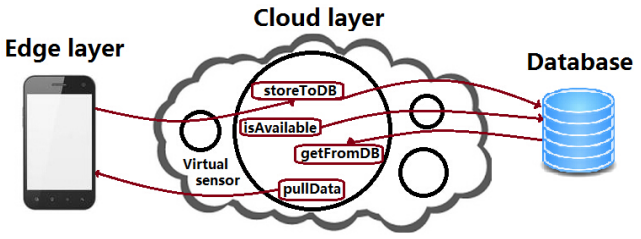


Figure 1: BitC communication diagram

Each sensor has its own ID to identify its type, owner, and the facilities where it belongs. The sensor ID is composed

by four fields separated with dot. The first two characters identifies the facilities entity, followed by three digits which is the BSNs index number and then the next field is about the sensor type, and the last two digits indicates the sensor's index number (i.e. NW.001.ACC.01). Each sensor ID is preconfigured by the medical doctor assigning to a specific BSN. All sensing data are headed with its own ID. Once data are pushed to the cloud, the back end system is able to identify the data source by its ID stored in the header.

The back end system is implemented using Node.js and the optimizer algorithm is written in Java. Virtual sink node, virtual sensor node and virtual sensors in the cloud are JavaScript classes or functions that are created to be the counterpart of each real component in the BSN and ASN. Theses virtual nodes offers different features like storing and retrieving data from database, issuing pull requests to Edge layer, getting result and replying to end users, etc.... Data stored in database are raw data, which are used later to analyze patients health status and physical activities using different data mining tools available in the cloud server.

3.1 System Architecture

BitC consists of the following three layers (Fig. 2).

Sensor Layer: operates one or more BSNs on a per-patient basis (Fig. 2). 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).

Virtual sensors are JavaScript nodes running in the server on the cloud layer. These virtual sensor nodes are predefined by medical doctors. 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 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 method. If the data is available then it retrieves the data by calling getFromDB 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 method. Once virtual sensors get the desired data, it backs to user.

MongoDB is chosen to be used as the main storage scheme in BitC due to its high scalability and integrity with NodeJS. The data structure is described in the Fig. **??** and explained in the Section 5.2.

4. CONFIGURATOR 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.

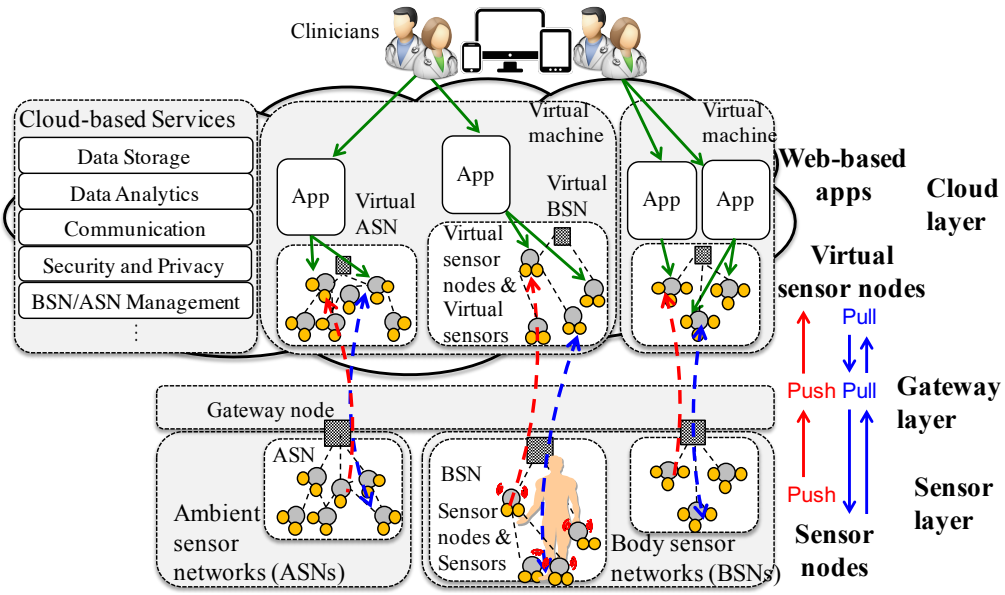


Figure 2: The Architecture of Body-in-the-Cloud (BitC)

- $R^m_{ij} \in R_{ij}$ 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.

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

The first and second terms indicate the bandwidth consumption by one-way push communication from the edge layer to the cloud layer and two-way pull communication between the cloud and edge layers, respectively. c_{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.

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

In BitC, optimization objectives conflict with each other. For example, the data yield objective conflicts with the other two objectives. Maximizing data yield means increasing data transmission intervals for sensor and sink nodes. This increases bandwidth consumption and energy consumption. Similarly, the energy consumption objective conflicts with the data yield objective. Minimizing energy consumption means reducing data transmission intervals for sensor nodes. This can reduce data yield. Given these conflicting objectives, this paper seeks the optimal trade-off (i.e., Pareto-optimal) configurations for data transmission intervals in BitC.

4.1 A Design of Evolutionary Game in CIELO

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}).

$$s(b_i) = \bigcup_{j \in 1..M} (o_{m_i}, o_{ij}, p_{ij}, q_{ij}) \quad 1 < i < N \quad (5)$$

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 27). 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 4 (Lines 7 to 9). The winner is replicated to increase its population share and mutated with polynomial mutation (Lines 10 to 18) [2]. 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 the loser of the game is replaced by the winner's replica (Line 17). 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). In the end, BitC uses the dominant strategy to adjust the configuration parameters for a BSN in question (Line 25).

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 their hypervolume value.

Hypervolume (HV) metric [5] measures the volume that a given strategy s dominates in the objective space:

$$HV(s) = \Lambda\left(\bigcup\{x' | s \succ x' \succ x_r\}\right) \quad (6)$$

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.

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

```
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, loser} ← performGame(s_1, s_2)
10:      replica ← replicate(winner)
11:      for each parameter v in replica do
12:        if random() ≤ P_m then
13:          replica ← mutate(replica, v)
14:        end if
15:      end for
16:      P_i \ {s_1, s_2}
17:      P'_i ∪ {winner, replica}
18:    end for
19:    P_i ← P'_i
20:    d_i ← argmax_{s∈P_i} x_s
21:    while d_i is infeasible do
22:      P_i \ {d_i}
23:      d_i ← argmax_{s∈P_i} x_s
24:    end while
25:    Use d_i to adjust the parameters for a BSN in question.
26:  end for
27:  g = g + 1
28: end while
```

5. SIMULATION AND EMPIRICAL EVALUATION

5.1 Simulation configuration and results

This section evaluates BitC through simulations and studies how BitC adapts BSN 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 real experiment parameters shown in Table 1. This paper assumes a nursing home where senior residents/patients live. BitC supports implementation of ASNs and BSNs, however ASNs are not taken into account in this paper's experiment implementation due to limitation of available resources. A small-scale and a larger-scale simulations are carried out with 3 and 100 residents, respectively. The large-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. Mutation rate is set to $1/V$ where V is the number of parameters in a strategy. Every simulation result is the average with 5 independent simulation runs.

Table 1: BitC Parameters

	BitC S	BitC L
Number of generations (G_{max})	20	100
Number of BSNs (N)	3	100
Number of simulation runs	5	5
Number of requests	1000	1000
Duration of a simulation (W)	3 hrs	3 hrs
Population size ($\|\mathcal{P}_i\|$)	50	50
Mutation rate (P_m)	$1/V$	$1/V$

Fig. 3a shows how hypervolume increases through generations. At each generation, hypervolume is measured with a set of dominant strategies taken from individual populations. Hypervolume starts from 0.872 and reaches 0.948 at the last generation. Fig. 3b and Fig. 3c are maximum, minimum and average objectives values of the dominant strategy in the last generation from 5 different simulation runs represented in box plots.

Fig. 4 shows how BitC improves its performance through generations. Figs. 4a to 4d show the changes of objective values over generations. Results illustrate that BitC improves its objective values by balancing the trade-offs among conflicting objectives. For example, in Fig. 4a and Figs. 4b, BitC improves both request fulfillment and bandwidth consumption through generations while the two objectives conflict with each other.

The optimization configurator supports the integration of any optimization algorithm. In this paper BitC is compared with two well known genetic algorithms NSGA II and its newer version NSGA III. Both genetic algorithms run with the same configuration parameters as BitC shown in Table 1. Two different simulation scale are performed to evaluate each algorithm's execution time under small and large environment settings. Results in Table 2 show that BitC outperforms NSGA II and NSGA III in execution time for both small and large environment scale. Optimization algorithms are running in the cloud back end system, in this paper Amazon EC2 Ubuntu free instance is used as cloud computing environment.

Table 2: Execution time per run

	BitC	NSGA II	NSGA III
Exec time S	5m 54s	7m 44s	24m 5s
Exec time L	2h 49m	4h 18m	5h 05m

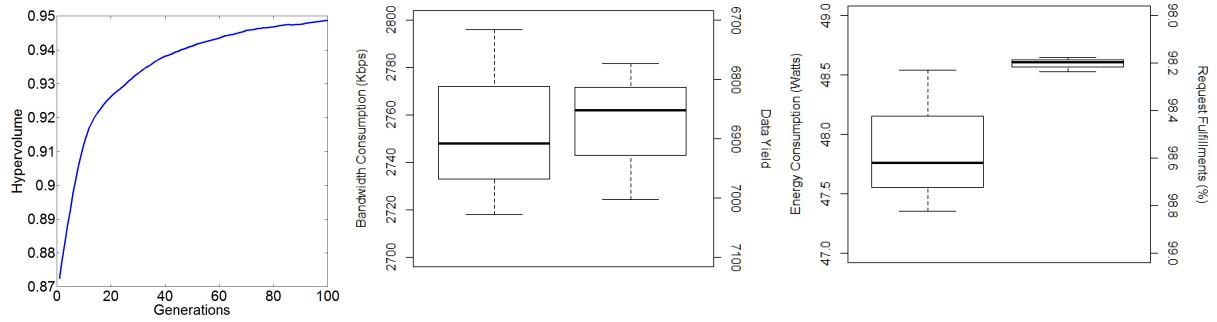
5.2 Empirical experiment evaluation

In this section empirical experiment are built to evaluate the real communication delay as well as the time taken to push and to store sensor data into the database. In the experiment a nexus 5 smartphone is used as sink node device, it is configured to simulate the collection of four sensors data in one BSN (one blood pressure, two accelerometer and one ECG). It starts issuing a pull request to get the latest configuration parameters from Cloud and to use it to configure its own BSN (sink node transmission interval, sensor node transmission interval, sensors sensing interval and sensors sampling rate). And It will send the getting configuration request every hour to keep update its BSN's configuration parameters. These configuration parameters have a fix size of 512 bits, the response time from cloud to sink node takes less than 1 ms.

As explained in the Section 3.1 MongoDB is used in BitC architecture as data storage system. Table. 3 and 4 shows the database implementation of Both collection BSN and Blood pressure sensor. Table. 3 illustrates the data structure of one BSN, each BSN has its own unique identification number generated automatically by MongoDB and it is stored in the field *_id*. The rest of data fields are divided by each sensor starting with the abbreviation of the sensor type (BP: Blood pressure, ACC: accelerometer, ECG: electrocardiogram sensor). Each sensor have five data fields which are four configuration parameters (*_tx* denotes the sink note transmission interval, *_tx_interval* denotes the sink note transmission interval, *_sen_interval* denotes the sensor sensing interval and *_samp_rate* denotes the sensor sampling rate) plus sensor unique identification code stored in the field *_code*. Table. 4 shows the data structure of a blood pressure sensor, the collection is named with sensor's unique identification code. Each document in the collection stores the sensor data collected per day indicated in the field *date*. *p_id* indicates the BSN's unique id which the sensor belongs to, *s_data* stores the raw sensor data pushed from sink node (fake data are used in the experiment), and *type* shows the sensor type.

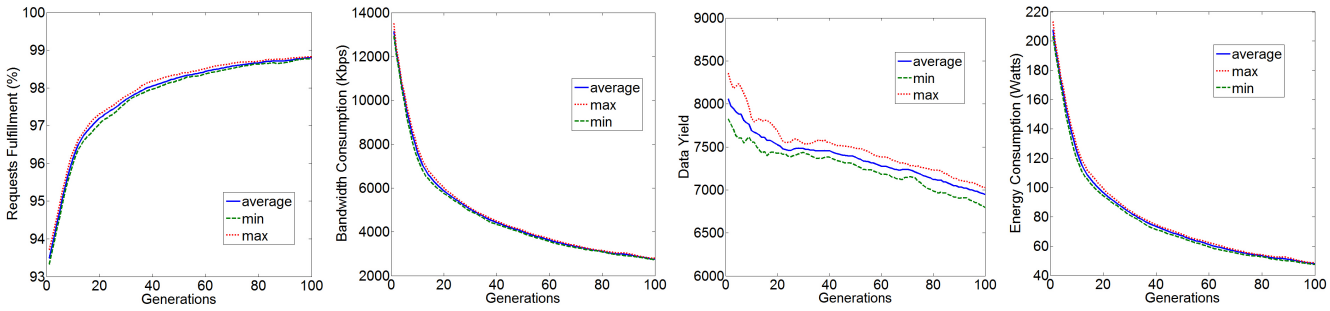
Sink node collects the amount of sensor data based on the each sensor sensing interval and sampling rate, then it pushes to the cloud periodically following the configuration parameters got from BitC optimization algorithm. Table 5 shows the average, maximum and minimum number of samples for each sensor type that are taken into account in the experiment setting to evaluate the transmission latency in different possible scenarios. These average, maximum and minimum values indicate the amount of samples that are pushed each time to the cloud by different sensor type.

Empirical experiment results are shown in Table 6 and Fig 5, 3G/Wifi both network are used in the experiment. The amount of sensor data are calculated from Table 5, where the average, maximum and minimum data amount per each sensor are taken into account. Also two extreme cases are evaluate, 2 bytes is the minimum data unit and 16 Mbytes is the limitation of BSON data structure. These two



(a) Hypervolume **(b)** Bandwidth & Data Yield **(c)** Energy & Request Fulfillment

Figure 3: BitC Objective Values in the last generation presented in Boxplot and BitC Hypervolume value through generations



(a) Request Fulfillment **(b)** Bandwidth Consumption **(c)** Data Yield **(d)** Energy Consumption

Figure 4: BitC Objective Values through Generations

Table 3: BSN data structure

Data field	Description
_id	BSN unique identification number
BP_code	Blood pressure sensor unique identification code
BP_tx	Blood pressure sink node transmission interval
BP_tx_interval	Blood pressure sensor transmission interval
BP_sen_interval	Blood pressure sensing interval
BP_samp_rate	Blood pressure sensor sampling rate
ACC1_code	Accelerometer 1 sensor unique identification code
ACC1_tx	Accelerometer 1 sink node transmission interval
ACC1_tx_interval	Accelerometer 1 sensor transmission interval
ACC1_sen_interval	Accelerometer 1 sensing interval
ACC1_samp_rate	Accelerometer 1 sensor sampling rate
ACC2_code	Accelerometer 2 sensor unique identification code
ACC2_tx	Accelerometer 2 sink node transmission interval
ACC2_tx_interval	Accelerometer 2 sensor transmission interval
ACC2_sen_interval	Accelerometer 2 sensing interval
ACC2_samp_rate	Accelerometer 2 sensor sampling rate
ECG_code	Electrocardiogram sensor unique identification code
ECG_tx	Electrocardiogram sink node transmission interval
ECG_tx_interval	Electrocardiogram sensor transmission interval
ECG_sen_interval	Electrocardiogram sensing interval
ECG_samp_rate	Electrocardiogram sensor sampling rate

Table 4: Sensor data structure

Data field	Description
_id	Sensor data unique identification number
s_data	Sensor raw data
type	Sensor type
p_id	BSN unique identification number
date	Date

Table 5: Number of samples pushed each time by different sensor type

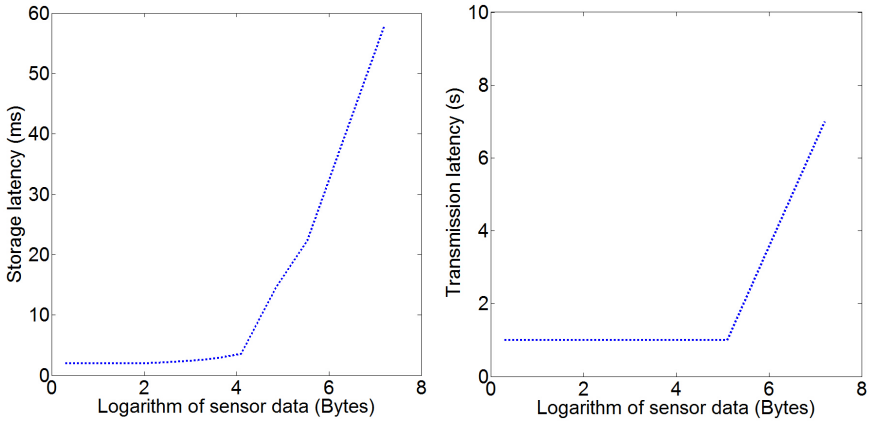
	Blood pressure	Accelerometer	ECG
minimum	40	100	60
average	645	1350	2530
maximum	1250	2500	5000

values are just chosen for evaluation purposes, in real BitC system none of these two values could be reached. Transmission time is calculated as the time taken from sending the data to the moment that cloud receive completely the pushed data in seconds, as results show that the transmission time remain constant 1 second and it does not increase except the amount of data pushed reaches to 16 Mbytes. BitC is configured and implemented as an energy and bandwidth efficient system, the amount of sensor data need to be pushed by each BSN is relatively very small in comparison with the large bandwidth offered by 3G and Wifi. Storage time is computed as the time taken from receiving the data from sink node to the moment that the total amount of data is ensured to be stored in the database. Results show that the storage time increases very smoothly and in the extreme case 16 Mbytes it just takes about 58 ms. All the results are calculated as the average of 10 experiment runs.

6. CONCLUSION

7. REFERENCES

[1] M. Chen, S. Gonzalez, A. V. Vasilakos, H. Cao, and V. C. Leung. Body area networks: A survey. *Mobile Netw. Appl.*, 16(2), 2011.



(a) storage latency **(b)** transmission latency

Figure 5: Transmission and Storage latency

Table 6: Transmission and storage latency

Sensor data (bytes)	Transmission time (s)	Storage time (ms)
2	1 s	2 ms
80	1 s	2 ms
120	1 s	2 ms
200	1 s	2 ms
1250	1 s	2 ms
2500	1 s	3 ms
5000	1 s	3 ms
10000	1 s	3 ms
20000	1 s	4 ms
125000	1 s	5 ms
16000000	7 s	58 ms

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

[3] Y. Hao and R. Foster. Wireless body sensor networks for health-monitoring applications. *Physiological Measurement*, 29(11):R27–56, 2008.

[4] 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*, 9(21), 2012.

[5] 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.