Journal of Systems Architecture 126 (2022) 102489

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

BANSIM: A new discrete-event simulator for wireless body area networks with deep reinforcement learning in Python[☆]

Beom-Su Kim[a], Ki-Il Kim[a,*], Babar Shah[b]

[a] Department of Computer Science and Engineering, Chungnam National University, Daejeon, 34134, Republic of Korea
[b] College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Discrete-event network simulator
Deep reinforcement learning
SimPy
BANSIM
Wireless body area networks

ABSTRACT

Many studies have investigated machine learning algorithms to improve the performance of wireless body area networks (WBANs). However, it was difficult to evaluate algorithms in a network simulator because of missing interfaces between the simulators and machine learning libraries. To solve the problem of compatibility, some researchers have attempted to interconnect existing network simulators and artificial intelligence (AI) frameworks. For example, ns3-gym is a simple interface between ns-3 (in C++) and the AI model (in Python) based on message queues and sockets. However, the most essential part is the implementation of an integrated event scheduler, which is left to the user. In this study, we aim to develop a new integrated event scheduler. We present BANSIM, a discrete-event network simulator for WBAN in standard Python that supports deep reinforcement learning (DRL). BANSIM provides an intuitive and simple DRL development environment with basic packet communication and BAN-specific components, such as the human mobility model and on-body channel model. Using BANSIM, users can easily build a WBAN environment, design a DRL-based protocol, and evaluate its performance. We experimentally demonstrated that BANSIM captured a wide range of interactions that occurred in the network. Finally, we verified the completeness and applicability of BANSIM by comparing it with an existing network simulator.

1. Introduction

A wireless body area network (WBAN) is a specialized network that supports on-body communication between medical sensors. In a WBAN, each node collects physical data and forwards them to the coordinator. The IEEE 802.15.6 task group established the following technical requirements for WBAN applications [1]: (1) A WBAN should support a link throughput of up to tens of kilobytes per second. (2) Reliability, latency, and jitter should be guaranteed for each node according to the application type. (3) The application lifetime should range from 24 h to several years of autonomous operation. However, frequent changes in the on-body channel and different quality-of-service (QoS) requirements of medical sensors make it difficult to satisfy these technical requirements.

To achieve such requirements, adaptive resource management schemes employing machine learning (ML) have been proposed. As a striking feature of these schemes, the coordinator can learn the optimal scheduling policy through trial and error in a dynamic WBAN environment. Many researchers have demonstrated that DRL-based prediction models can jointly improve the performance metrics in dynamic network environments; however, they encounter difficulties in evaluating their ML algorithms in existing network simulators owing to a lack of interfaces for interworking with ML libraries.

For example, conventional network simulators such as OPNET, OMNeT++, and QualNet provide diverse standard communication models; however, they do not support WBAN-specific reference models. Although these simulators provide a high scalability, it is not easy for individual researchers to implement a new communication model, owing to the complexity of the source code. To solve this problem, the Castalia framework was proposed, which is an extended simulator based on the OMNeT++ platform providing a WBAN simulation environment. However, these network simulators are implemented in C/C++; thus, they cannot directly use the ML libraries in Python. Although `tensorflow` C++ exists, conventional network simulators have poor scalability with external programs, which causes complex binding problems. Moreover, users cannot use wrapper libraries such as Keras or Pytorch in C/C++ development environments.

The ns-3 network simulator provides an IEEE 802.15.4-based WBAN simulation environment. Although Python bindings in ns-3 provide

☆ Code repository: https://bitbucket.org/bumsou10/bansim.
* Corresponding author.
E-mail address: kikim@cnu.ac.kr (K.-I. Kim).

https://doi.org/10.1016/j.sysarc.2022.102489
Received 23 December 2021; Received in revised form 11 March 2022; Accepted 26 March 2022
Available online 2 April 2022

support for importing ns-3 models as Python modules, they are limited to running ns-3 models in Python instead of providing a real-time interaction between ns-3 models and Python modules. Specifically, the DRL technique requires a real-time interaction between the agent in Python and the WBAN environment in ns-3. However, it is impossible to control the event flow between two separate files through Python bindings.

To overcome the limitations of Python bindings, several frameworks [2–4] have been proposed for real-time interworking with existing network simulators and Python libraries. These frameworks interconnect network simulators (in C++) and ML libraries (in Python) through text files or message queues. Specifically, they translate simulation events or ML function calls into messages and transfer them through inter-process communication. This approach can build a DRL environment in existing network simulators through real-time interactions between two separate modules. However, owing to the lack of an integrated approach between two separate event schedulers, it is difficult to configure an integrated simulation environment. In addition, interprocess communication increases the overhead and complexity of the simulator.

For this reason, most DRL-based studies [5–11] on WBANs have built a simulation environment in Python instead of using existing network simulators. However, they did not cover the details of diverse events in the network. In addition to WBANs, although most DRL-based network protocols [12–17] in other research fields use Python for a performance evaluation, they also omit the details of the network simulation program. The lack of details in the simulation program can degrade the reliability of the simulation results. Hence, it is important to develop an integrated simulation program to help researchers conducted accurate performance evaluations using Python.

Motivated by this problem, in this study, we aimed to develop a lightweight simulation program in Python. Specifically, we present BANSIM, which is a DRL-enabled discrete-event network simulator for WBANs. Because BANSIM is implemented in Python, it provides users with a simple and intuitive DRL development environment. BANSIM not only supports basic packet communication, it also provides BAN-specific components, such as a human mobility model and an on-body channel model.

Compared with existing well-known network simulators, as a unique advantage of BANSIM, it is suitable for evaluating the conceptual design of ML models for resource optimization in WBANs. Using BANSIM, researchers no longer need to worry about the compatibility between existing network simulators and machine-learning libraries. We also enable users to extend BANSIM to other networks by providing essential lower-layer components and design strategies for translating packet streams into discrete simulation events.

The main contributions of this study are summarized as follows:

- We use the SimPy component model [18] to develop a discrete-event network simulator in Python. We provide an example code for using SimPy and translate packet streams into discrete simulation events (code repository: https://bitbucket.org/bumsou10/bansim).
- We implement basic network models for packet communication and BAN-specific components, such as the human mobility model and on-body channel model. We experimentally demonstrate that BANSIM can capture a wide range of interactions that occur in the network.
- To improve the reliability of our system, BANSIM is modeled using the network architecture of ns-3. We evaluate the applicability and accuracy of BANSIM by comparing it with ns-3.
- To evaluate whether BANSIM can support DRL-based protocol modeling and simulation, we present an example of deep Q-network (DQN) training. The simulation results prove that users can easily build a DQN environment and accurately evaluate performance using BANSIM.
- Although BANSIM is designed for WBAN environments, it supports fine-grained scalability by providing inter-layer interaction and frame transaction models.

2. Background and motivation

Popular network simulators such as OPNET, QualNet, OMNeT++ (Castalia Framework), and ns-3 allow researchers to build a simulation environment for WBANs. However, it is difficult to evaluate DRL algorithms in existing C/C++ network simulators because researchers cannot directly import machine-learning libraries available for Python. Although `tensorflow` C++ already exists, researchers have encountered various build errors when running `tensorflow` C++ on existing network simulators because of compatibility problems.

Several frameworks have been proposed to improve compatibility of existing network simulators with machine learning tools. For example, OMNeT++ (in C++) and Keras (in Python) have been integrated by implementing a method [2] that allows them to communicate using text files. The simulation program periodically generates files that describe the current state of simulation. Upon receiving the simulation data, the DRL program generates a text file that describes the action to be executed in the simulator. However, both programs should implement a new character-based format to read and write text files. In addition, this mechanism is highly dependent on the logic of the user program and has a significant overhead.

ns3-gym [3] and ns3-ai [4] are examples of middleware that interconnects ns-3 (in C++) and an AI framework (in Python). These programs are responsible for transferring the current state of the simulation program and the resulting action of the reinforcement learning (RL) agent. Specifically, ns3-gym translates the simulation events or RL function calls into messages and transfers them through the ZMQ socket. Similarly, ns3-ai supports process communication based on shared memory. However, internal process communication increases the overhead and complexity of the simulator. Moreover, the difficulty of implementing the simulation program is significantly increased.

For these reasons, researchers have started to implement their DRL algorithms in Python to avoid interoperability and compatibility problems. Performance of most DRL-based methods for WBANs [5–11] was evaluated in Python. However, these studies focused on the conditions for building a DRL environment and did not provide a detailed description of the network simulator. Similarly, performance of most DRL-based network protocols [12–17] in other research fields was also evaluated in Python, but details of the simulation program were omitted; in some cases, it was simply mentioned that SimPy had been used.

As a network simulator implemented in Python has not yet been proposed, we should determine the specifications and capabilities of the network simulation program. The lack of details may reduce the reliability of the simulation results. Therefore, the right simulation program is essential to help researchers accurately evaluate their DRL algorithms in Python. In this study, we developed a discrete-event network simulator for a WBAN using SimPy. SimPy is a process-based discrete-event simulation framework for Python. To the best of our knowledge, this is the first study to develop a discrete-event network simulator using SimPy.

3. SimPy

SimPy is a process-based discrete-event simulation framework for Python. The SimPy component model allows the design of a network protocol stack and a series of frame transactions. As shown in Fig. 1, we can create events and execute them using SimPy. We can set the priority and delay for each event using the `env.schedule()` method, and these options inspired us to develop a discrete-event network simulator.

Most simulation frameworks, including SimPy, are based on single-threaded deterministic libraries. A single-threaded simulation system works as follows: if concurrent tasks are not handled as events, all tasks on other nodes are not performed until the current task has been

```
import simpy

def example(event)
    print('Hello world')

env = simpy.Environment()              # Create event queue
event = env.event()                    # Generate an event
event._ok = True                       # Activate the event
event.callbacks.append(example)        # Define event action
env.schedule(event, priority, delay)   # Register the event in the event queue
env.run(until)                         # Execute the contents of the event queue
```

Fig. 1. Event registration code using SimPy.

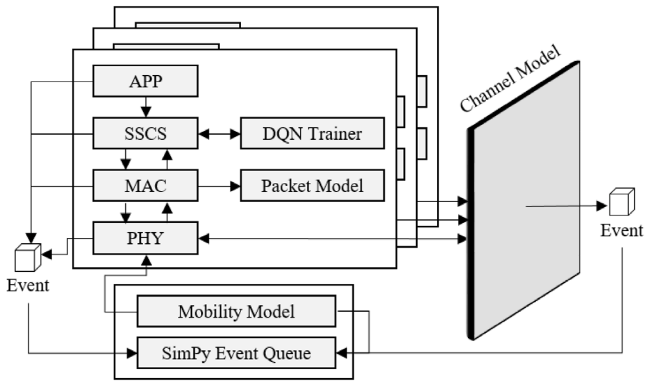


Fig. 2. BANSIM architecture.

completed. Therefore, all tasks that can be executed simultaneously should be treated as events.

For example, events in the real world (e.g., wireless LAN) occur at slightly different times but may appear to occur simultaneously in the simulation system. In SimPy, if events are scheduled at the same time, they are processed sequentially according to the reservation order or priorities of events. In addition, the start time of events registered in the event queue is based on the delay option set for each event, not the system clock time. Similarly, the simulation time passes between sequential events.

4. BANSIM

BANSIM is a DRL-enabled discrete-event network simulator for WBANs implemented in Python 3.7. The simulator allows us to simulate network models, run custom protocol stacks, and communicate by sending packets over a wireless link. BANSIM is a simple and lightweight simulation program that implements minimal network functions but provides a useful test environment. The main advantage of BANSIM is that it does not require a framework or middleware to interact with a DRL development environment.

Fig. 2 shows the architecture of BANSIM. It consists of four network models (i.e., APP, SSCS, MAC, and PHY) and four supplementary components (i.e., DQN trainer, channel model, mobility model, and packet model). The role of each component is described in the following subsections.

In BANSIM, network protocols are designed using a class model. Each class defines its functional operation and provides interface methods for communicating with the adjacent layers. For interaction between two protocols, for example, an upper-layer protocol (e.g., medium access control (MAC) layer) directly calls an interface method of a lower-layer protocol (e.g., physical (PHY) layer) or a lower-layer protocol (e.g., PHY) calls back an interface method of an upper-layer protocol (e.g., MAC). To implement a hierarchical communication structure, BANSIM adopts a call-by-reference structure.

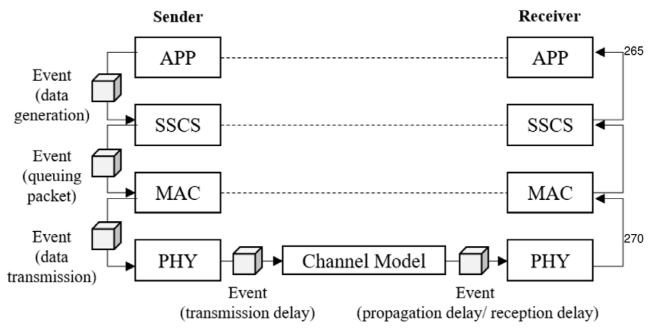


Fig. 3. List of events during packet transmission.

The root class (i.e., node or agent) creates instances of each protocol model and passes them to each instance so that they can directly call the interface methods defined in other class models.

4.1. Event list

In BANSIM, all concurrent tasks are handled as events. Thus, the simulation time does not pass until all events scheduled at the same time have been processed. In addition, we can generate events to reflect network delays, such as transmission and propagation delays, in the simulation time. As shown in Fig. 3, BANSIM translates the packet streams into discrete simulator events.

The list of events generated by each component is as follows:

- **Data generation and packet queuing**: Nodes belonging to the same BAN generate data packets according to their sampling rates and store the packets in the data queue using SSCS. These procedures are concurrent tasks and, therefore, are treated as events.
- **Packet transmission**: Each node accesses a wireless channel based on time division multiple access (TDMA). In the node, MAC checks the data queue in the scheduled allocation interval for each beacon period. If there is a packet in the data queue, MAC creates an event for sending the packet.
- **Transmission and reception delays**: Transmission delay is the time until a packet is completely sent from the network interface to the wireless channel, which depends on the packet size and data rate. PHY creates an event for sending a packet over a wireless channel (i.e., PHY creates an event that calls the interface method of the channel model), and then it sets the transmission delay to the delay option of the event. Reception delay is the time required for a packet to be completely sent from the wireless channel to the network interface. The channel model generates an event to call the interface method of the receiver PHY and sets the reception delay to the delay option of the event.
- **Propagation delay**: Propagation delay is the time a packet leaves the network interface on the sender-side and arrives at the network interface on the receiver-side, which depends on the distance between devices and the speed of the medium. The channel model generates an event to call the interface method of receiver PHY and sets the propagation delay to the delay option of the event.

4.2. Overview of BANSIM

BANSIM can build a WBAN environment with one agent (i.e., coordinator) and n sensor nodes. The agent and nodes are interconnected in a one-hop star topology. Users can configure the BAN environment without any limit on the number of nodes and determine the sampling rate of each node. BANSIM supports TDMA as a channel access

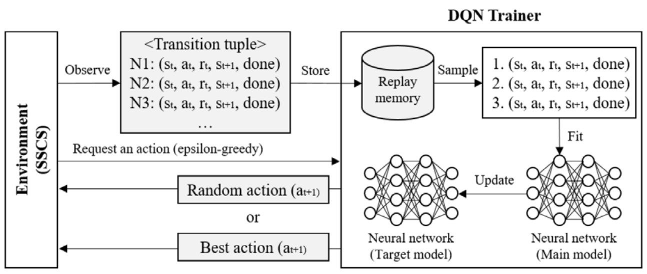

DQN Trainer
Environment (SSCS)
Observe
<Transition tuple>
N1: (st, at, rt, st+1, done)
N2: (st, at, rt, st+1, done)
N3: (st, at, rt, st+1, done)
Store
Replay memory
Sample
1. (st, at, rt, st+1, done)
2. (st, at, rt, st+1, done)
3. (st, at, rt, st+1, done)
Fit
Request an action (epsilon-greedy)
Random action (at+1)
or
Best action (at+1)
Update
Neural network (Target model)
Neural network (Main model)

Fig. 4. DQN training procedure between SSCS and the DQN trainer in the agent.

mechanism, and users can customize the beacon structure and beacon period.

BANSIM supports three types of frames: management, control, and data frames. Users can customize the frame header and frame body according to the type of frame. BANSIM provides a beacon frame as a management frame and an ACK frame as a control frame.

BANSIM provides three types of human mobility models: standing, sitting, and walking. Users can obtain a body-shadowing factor and path loss according to their body movements. BANSIM assumes an error-free channel but a constant noise value. The receiver calculates the reception power using the transmission power, path loss, and noise value. If the reception power is lower than the reception sensitivity, the packet is discarded.

In addition, users can establish a DRL-based resource allocation policy using the `DQN trainer`. In the agent, users can define the state, action, and reward using spectral parameters, such as transmission power, reception power, and path loss. Although BANSIM is designed for WBAN environments, it has fine-grained scalability. Components of BANSIM are described in the following subsections.

4.3. SSCS

In the agent, a service-specific convergence sublayer (SSCS) provides a service access point between the DQN trainer and MAC. As shown in Fig. 4, SSCS serves as an environment from the perspective of the DQN trainer. Specifically, SSCS queries the DQN trainer for the resource allocation policy suitable for the current state, and the DQN trainer returns an optimal action. Thus, SSCS can learn a resource allocation policy by itself by interacting with the DQN trainer.

SSCS allows users to customize two types of resource allocation algorithms: uplink scheduling and transmission power control (TPC). Users can easily implement their ideas in SSCS. The assignment information is passed to MAC and encapsulated in the form of a management-type frame (i.e., beacon frame). In summary, users can develop algorithms to address the following two major problems in WBANs.

- **Uplink scheduling**: In BANSIM, the agent provides assignment information on the number of time slots, slot duration, and transmission power level to nodes using a beacon frame. In WBAN, to prevent transmission failure or retransmission owing to body movements [19], the agent should dynamically determine the appropriate number of time slots and slot duration in every beacon period. In SSCS, users can devise scheduling parameters and develop adaptive uplink scheduling algorithms.
- **TPC**: Transmission power required for successful transmission depends on the current body posture. To increase reliability, it is important to determine the appropriate transmission power level. In addition, energy efficiency can be improved if the optimal transmission power is calculated by considering the path loss and body shadowing. In SSCS, users can develop appropriate TPC algorithms and verify their performance.

4.4. DQN trainer

Deep Q-network (DQN) is one of the most widely used DRL algorithms. As shown in Fig. 4, DQN components are implemented in the DQN trainer, and users can customize the DQN environment. Because BANSIM runs in Python, users can easily import Python packages for machine learning, such as NumPy, Keras, and Pytorch.

4.4.1. Basic operations

In the agent, the DQN trainer interacts with the SSCS to determine the optimal resource allocation policy. The DQN trainer returns an action to the environment (i.e., SSCS) on an on-demand basis by observing the reward and new state from the SSCS. To interact with the SSCS, the DQN trainer provides the following interface methods: `get_action()` and `set_observation()`. In addition, the DQN trainer defines internal methods for DQN training: `create_model()`, `train()`, and `update_replay_memory()`.

The experiences are tuples of $[s_t, a_t, r_t, s_{t+1}, done]$, which maintain the transitions obtained from the environment. For example, when the SSCS requests an action from the DQN trainer (i.e., SSCS calls `get_action()` in the DQN trainer), the DQN trainer observes the current state s_t from the SSCS and returns immediate action a_t based on the epsilon-greedy algorithm.

SSCS allocates radio resources to node i based on the received action. If the SSCS receives a data packet from node i, the reward r_t and new state s_{t+1} are updated by the DQN trainer (i.e., the SSCS calls `set_observation()` in the DQN trainer). If the SSCS allocates radio resources to node i, but does not receive any response from node i until an acknowledgment (ACK) timeout occurs, the $done$ flag is enabled.

Each node constituting the BAN has different link conditions and properties. To improve the policy π in various cases, the DQN trainer should not assign the same action to each node. To avoid such assignment, the SSCS should maintain a status block $b_i = [s_t, a_t, r_t, s_{t+1}, done steps]_{i\in N}$ for each node. For example, when the SSCS requests an action for node i from the DQN trainer, it passes the status block b_i to the DQN trainer, which returns the immediate action a_t for the current state s_t of b_i. When a data packet arrives from node i, the SSCS updates r_t, s_{t+1}, $done$, and $steps$ of b_i, and creates a transition tuple for node i. If the $done$ flag is enabled or the value of $steps$ in b_i exceeds the threshold, the current training episode for node i ends. All nodes share the same number of episodes, and users can modify the maximum number of training episodes. If the number of training episodes is exceeded, the DQN trainer always returns the best action.

The DQN trainer uses replay memory for training the DQN, which stores the transitions observed by SSCS in replay memory (i.e., `update_replay_memory()`) and reuses these data later. The transitions that constitute a batch are decorrelated by randomly sampling the transitions from replay memory. This task stabilizes and improves the DQN training process. The DQN trainer has two types of neural network, i.e., the target network and the main network. These neural networks have the same structure but different weights. For a stable convergence of the predicted Q-value, the target network is periodically updated with the weights of the main network as the training episode progresses.

4.4.2. Hyperparameter tuning

In the DQN trainer, users can simply create a neural network using the sequential model or configure a complex neural network (e.g., feed-forward neural network (FFNN)) using the functional application programming interface (API) (e.g., `Input`, `Dense`, and `Model`) provided by Keras. For beginners, BANSIM provides the sample code and templates in the DQN trainer (i.e., `create_model()` and `train()`) to configure the two types of neural network models. In addition, users can tune the hyperparameters, such as the number of epochs, batch size, learning rate, and number of hidden layers in the built-in neural network model.

4.5. MAC

IEEE 802.15.6 supports three types of communication modes: (1) beacon mode with superframes, (2) nonbeacon mode with superframes, and (3) nonbeacon mode without superframes. In BANSIM, a beacon mode with superframes was implemented. In this mode, the coordinator transmits beacon frames in the active superframes. The superframe structure is divided into a random access phase (RAP) and managed access phase (MAP). Users can customize the superframe structures.

BANSIM provides three key functions of MAC layer: radio transceiver control, flow control, and event tracing. For radio transceiver control, BANSIM supports two types of transmission modes: TDMA and carrier-sense multiple access with collision avoidance (CSMA/CA).

4.5.1. TDMA

As illustrated in Fig. 5, MAC is responsible for changing the state of the radio transceiver. After the node is initialized, the state becomes `RX_ON`. When the node starts to receive the beacon frame, the state is set to `BUSY_RX`. A collision may occur when a new packet arrives when the state is `BUSY_RX`. If MAC successfully receives the beacon frame, it checks the assignment information included in the beacon frame. The assignment information contains scheduled uplink information (i.e., interval start and interval end) and the transmission power level.

Upon receiving the beacon frame, MAC generates an event that calls `check_queue()`, a private method that sends packets from the data queue, and it sets the interval start to the delay option of the event. PHY attributes (i.e., transmission power level) are passed from MAC to PHY by calling `set_phy_attribute_request()`, which is an interface method of PHY. In `check_queue()`, the state of the radio transceiver is changed to `TX_ON` by calling `set_phy_trx_state_request()`. Then, the state of the radio transceiver is set to `BUSY_TX` until PHY has completely forwarded the packet over the wireless channel.

After MAC successfully transmits the packet, it registers an ACK timeout event and changes the state of the radio transceiver to `ACK_PENDING` until an ACK frame arrives at the receiver. When an ACK timeout occurs or an ACK frame is successfully received, MAC prepares for the next transmission by calling the `check_queue()`. If the remaining time slot is less than the expected transmission time, MAC abandons the next transmission; otherwise, it transmits the next packet by changing the state of the radio transceiver to `TX_ON`. If the allocated time slot is over, MAC changes the state to `RX_ON`.

4.5.2. CSMA/CA

BANSIM provides a contention-free channel access mechanism based on TDMA. To increase the scalability of BANSIM, we also support the CSMA/CA mechanism. Users can set a RAP in the superframe, and then nodes acquire a channel based on CSMA/CA. To support CSMA/CA, we implement two interface methods in the PHY model: clear channel assessment (CCA) and energy detection (ED). The CSMA/CA model uses these methods to check whether the channel is busy or idle, and it attempts to access the channel based on the exponential backoff algorithm. MAC receives channel information from the CSMA/CA model through a callback function and generates an event that calls `check_queue()` when the channel is idle.

4.6. PHY

IEEE 802.15.6 supports three types of PHYs, i.e., ultra-wideband (UWB), human body communications (HBC), and narrowband (NB) PHYs. In BANSIM, the NB PHY is implemented and is responsible for the following functions: (1) radio transceiver control, (2) CCA, and (3) data transmission and reception.

As shown in Table 1, PHY supports three types of modulation schemes according to the frequency bands. The data and symbol rate are determined based on the type of modulation. Users can add new PHY attributes, such as the frequency band, modulation, data rate, and symbol rate. PHY supports four types of radio transceiver options: `TX_ON`, `RX_ON`, `BUSY_TX`, and `BUSY_RX`. PHY provides an interface method that allows MAC to control the radio transceiver.

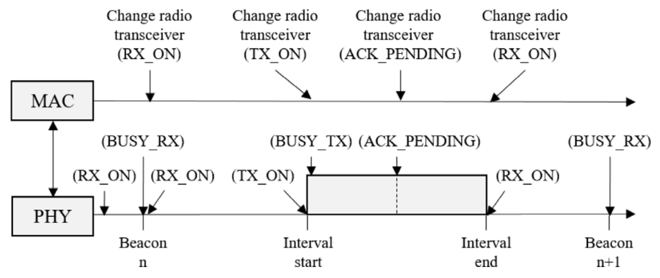

Change radio transceiver (RX_ON)
Change radio transceiver (TX_ON)
Change radio transceiver (ACK_PENDING)
Change radio transceiver (RX_ON)
MAC
PHY
(BUSY_RX)
(RX_ON)
(RX_ON)
(TX_ON)
(BUSY_TX)
(ACK_PENDING)
(RX_ON)
(BUSY_RX)
Beacon n
Interval start
Interval end
Beacon n+1

Fig. 5. Example of radio transceiver control.

Table 1
Frequency bands and modulation schemes supported by PHY.

Frequency band	Modulation	Data rate	Symbol rate	Number of symbols
868 MHz	BPSK	20 Kbps	20 Ksps	(32, 8, 8)
915 MHz	BPSK	40 Kbps	40 Ksps	(32, 8, 8)
868 MHz	ASK	250 Kbps	12.5 Ksps	(2, 1, 0.4)
915 MHz	ASK	250 Kbps	50 Ksps	(6, 1, 1.6)
868 MHz	QPSK	100 Kbps	25 Ksps	(8, 2, 2)
915 MHz	QPSK	250 Kbps	62.5 Ksps	(8, 2, 2)
2.4 GHz	QPSK	250 Kbps	62.5 Ksps	(8, 2, 2)

PHY interacts with the channel model to handle the packet transmission and reception. Upon receiving a packet (i.e., PSDU) from MAC, PHY forwards the packet to the channel model. At this time, instead of adding the PLCP header to the packet, the transmission overhead of the PLCP header is added to the transmission delay.

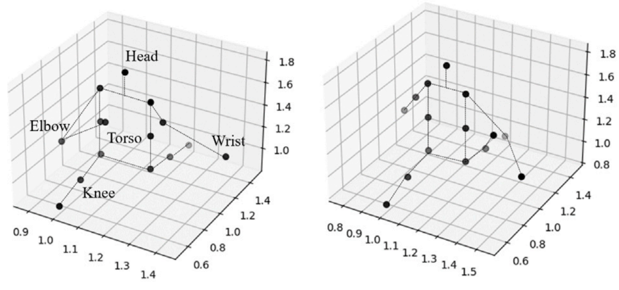
For example, PHY generates an event that calls `start_tx()`, an interface method of the channel model, to load a packet into the wireless channel. The delay of this event is the transmission delay, which is set as the packet size divided by the data rate. The channel model maintains a list of PHY instances belonging to the same BAN. If the channel model receives a packet, it generates an event that calls an interface method (i.e., `start_rx()`) for all PHY instances in the list. The delay of this event is the propagation delay, which is set as the distance between the sender and the receiver divided by the speed of the medium (i.e., light).

4.7. Channel model

In BANSIM, the channel model serves as an interface for communication between the sender and receiver. In this model, users can implement various types of propagation loss models. BANSIM supports two types of such models: the Friis free-space propagation model and the BAN-specific path-loss model.

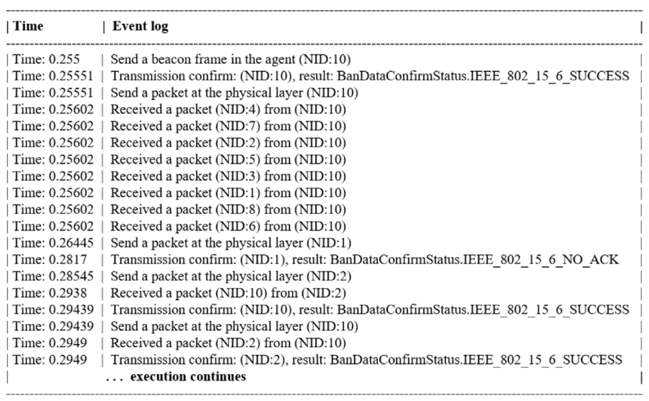
The Friis free-space propagation model is used to predict the reception power when the line-of-sight (LOS) between the sender and receiver is guaranteed. In this model, the reception power is calculated using the transmission power, antenna gain, wavelength, and distance between the sender and receiver. However, frequent body movements in WBANs cause non-line-of-sight (NLOS) propagation. Therefore, the propagation loss model should consider the body-shadowing factor according to body movements to build a realistic WBAN channel environment.

The BAN-specific path-loss model predicts the reception power using the loss function extracted from the real-WBAN testbed. Based on the distance between the sender and receiver, several path-loss models were tested in [20,21]. As a result, a BAN-specific path-loss model was



(a) Vector positions at time t (b) Vector positions at time $t+1$

Fig. 6. Movement of position vectors over time in the walking model.



```
| Time            | Event log                                                                              |
| Time: 0.255     | Send a beacon frame in the agent (NID:10)                                              |
| Time: 0.25551   | Transmission confirm: (NID:10), result: BanDataConfirmStatus.IEEE_802_15_6_SUCCESS     |
| Time: 0.25551   | Send a packet at the physical layer (NID:10)                                           |
| Time: 0.25602   | Received a packet (NID:4) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:7) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:2) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:5) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:3) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:1) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:8) from (NID:10)                                                |
| Time: 0.25602   | Received a packet (NID:6) from (NID:10)                                                |
| Time: 0.26445   | Send a packet at the physical layer (NID:1)                                            |
| Time: 0.2817    | Transmission confirm: (NID:1), result: BanDataConfirmStatus.IEEE_802_15_6_NO_ACK       |
| Time: 0.28545   | Send a packet at the physical layer (NID:2)                                            |
| Time: 0.2938    | Received a packet (NID:10) from (NID:2)                                                |
| Time: 0.29439   | Transmission confirm: (NID:10), result: BanDataConfirmStatus.IEEE_802_15_6_SUCCESS     |
| Time: 0.29439   | Send a packet at the physical layer (NID:10)                                           |
| Time: 0.2949    | Received a packet (NID:2) from (NID:10)                                                |
| Time: 0.2949    | Transmission confirm: (NID:2), result: BanDataConfirmStatus.IEEE_802_15_6_SUCCESS      |
|                 ... execution continues                                                                  |
```

Fig. 7. Sample execution.

extracted and showed the best fit with the measurement data. This model can be expressed as follows:

$$PL(d) = PL(d_0) + 10n\log_{10}(d/d_0) + S, \tag{1}$$

where d denotes the distance between the sender and receiver. $PL(d_0)$ is the path loss at a reference distance d_0, and n is the path-loss exponent. S denotes the body-shadowing factor. If the LOS between the sender and receiver is guaranteed, the body-shadowing factor is not considered when calculating the path loss. The LOS or NLOS between the sender and receiver can be calculated using the mobility model.

4.8. Mobility model

BANSIM provides three types of human mobility models: standing, sitting, and walking. Standing and sitting models are static positions, whereas the walking model is dynamic.

The mobility model has 15 position vectors for describing body movements. As illustrated in Fig. 6, each position vector represents a part of the human body (e.g., torso, knee, and wrist), which periodically moves or rotates according to the human mobility model. For example, the rotation of a position vector in a walking model can be expressed as follows:

$$vector_x = random(0,1), \tag{2}$$

$$vector_y = \cos(-DegreesToRadians(D)) \times C, \tag{3}$$

$$vector_z = \sin(-DegreesToRadians(D)) \times C, \tag{4}$$

where $vector_x$ denotes the direction of movement, and `random()` generates a random integer within a given range (i.e., 0 indicates the direction toward the left and 1 indicates the direction toward the right). In addition, $vector_y$ and $vector_z$ are the coordinates representing the rotation of the position. Here, D denotes the current degree and C is a constant value. Based on the above equations, the mobility model implements a rotation of the arms and legs.

Specifically, we designated the joints where the arms and legs connect as anchor points. To implement the walking model, we represent each body part as a node and connect the arms and legs to each anchor point. For example, the node at the anchor point is the root node, and the child nodes (i.e., arms and legs) rotate based on the position of the parent node. To implement the standing and sitting model, we used the relative distance between the coordinator and nodes configured in the actual WBAN testbed [21] as a reference model.

The mobility model generates a random seed by using the NumPy package to create a movement pattern. This ensures different performance results each time the user runs a simulation. In addition, the mobility model provides utility methods for calculating the distance or LOS/NLOS between position vectors. The PHY contains an instance of the mobility model and refers to the position vector corresponding to the desired body position. Furthermore, the channel model calculates the path loss between the sender and receiver using the mobility model.

4.9. Packet model

The packet model defines the MAC header and MAC frame body specified in the IEEE 802.15.6 standard. When SSCS creates a packet and determines its purpose, MAC adds the MAC header and MAC frame body to the packet using the packet model. As previously described, the PLCP header is not added to the packet, but the transmission overhead of the PLCP header is calculated in PHY. To simplify the interface of the simulation program, the packet model contains additional information (e.g., spectral parameters). For example, a packet contains the following information:

- **Packet size**: Users can determine the packet size (i.e., MSDU length) when generating a packet in SSCS. It is used to calculate transmission delay and throughput.
- **MAC header**: A packet includes the MAC header specified in the IEEE 802.15.6 standard. The MAC header contains the following fields: frame control, BAN ID, sender ID, and recipient ID. The frame control field includes various control information such as frame-type, frame-subtype, ACK policy, and sequence number. The packet model defines three types of frame structures: management-type frame, control-type frame, and data frame. Users can define a new MAC header or modify the existing frame structure.
- **MAC frame body**: The MAC frame body is initialized according to the frame-subtype field. The packet model provides three types of MAC frame bodies: beacon frame, ACK frame, and data frame. Users can define a new MAC frame body or modify the existing frame structure.
- **Spectral parameters**: A packet contains spectral parameters such as transmission power, reception power, and path loss. This information is initialized or updated as the packet passes through each layer. Users can easily access the spectral parameters without any additional interface or storage space.

4.10. Sample execution

Fig. 7 shows a sample execution that demonstrates the operation of BANSIM. In the sample execution, a BAN consists of one agent and eight nodes, and the agent (node ID (NID): 10) broadcasts a beacon frame every 255 ms (time: 0.255). When the agent successfully transmits the beacon frame, the status information is printed (time: 0.25551). Each node simultaneously receives the beacon frame (time: 25602) and then sends data packets to the agent in the scheduled allocation interval.

For example, if a packet sent by the node (NID: 1) is lost owing to body shadowing, an ACK timeout occurs (time: 0.2817). If the agent successfully receives the packet sent from the node (NID: 2), it returns an ACK frame (time: 0.29439), after which the transmission is

Table 2
Simulation parameters.

Parameter	Value
Frequency band	868 MHz/915 MHz
Modulation	ASK/QPSK
Transmission power	0 dBm
Reception sensitivity	−82 dBm
Rx energy	4.5 mA
Channel access mechanism	TDMA
Access mode	Beacon with superframes
Beacon period	255 ms
Packet size	50–250 bytes
ACK policy	I-ACK
pAllocationSlotLength	20
pAllocationSlotResolution	500 us
pAllocationSlotMin	500 us
pExtraIFS/ pMIFS/ pSIFS	10 us/20 us/75 us
Mobility model	Walking model

completed successfully (time: 0.2949). This sample execution demonstrates that BANSIM supports basic packet communication, and users can easily implement their ideas without in-depth code analysis.

4.11. Extension of BANSIM

A wireless network based on a one-hop star topology, such as a WBAN, assumes that all nodes are connected to a coordinator. This assumption makes the network configuration easier by eliminating the exchange of control messages required for network maintenance. However, a network whose topology changes frequently, such as a mobile ad hoc network (MANET), requires reliable topology control at the routing layer. To extend BANSIM to other networks, we provide users with an abstract class to define a new upper layer and service primitives to interwork with a lower layer.

4.11.1. Interface for new upper layers

Users who wish to extend BANSIM to other networks can configure an 802.11-based communication environment by setting CSMA/CA as the default channel access model in the MAC. In addition, users can construct a routing table by defining a new upper layer (i.e., routing layer) using an abstract class (i.e., `UpperLayer`). Specifically, users must add topology information to the payload of the packet model and then define a set of transaction models to process control packets.

4.11.2. Two-hop star topology extension

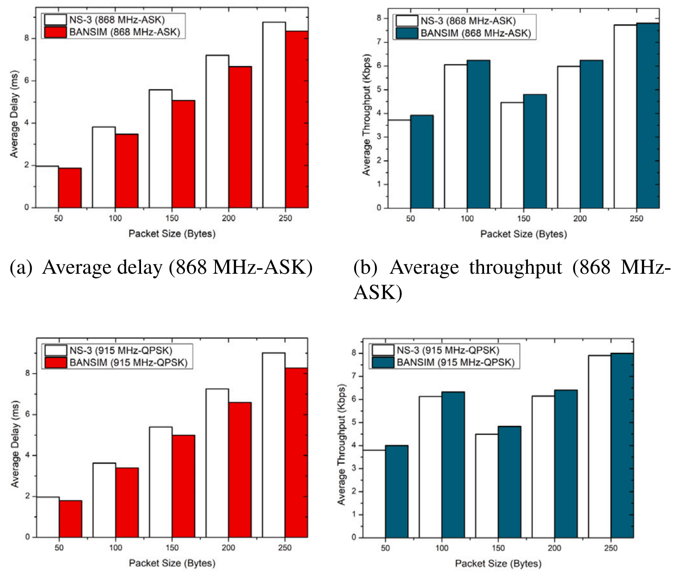
The IEEE 802.15.6 standard supports a two-hop star topology extension. Basically, although BANSIM is implemented based on a one-hop star topology, users who want to extend the two-hop star topology can define a new frame transaction model in the SSCS. Specifically, BANSIM supports the three types of frame structures specified in the IEEE 802.15.6 standard; however, detailed operations according to the bit fields of each frame header have yet to be implemented. Hence, users who want to extend the two-hop star topology must implement core functions of the IEEE 802.15.6 standard, such as an association and two-hop star topology extension.

5. Experiment

To validate the reliability of our system, BANSIM should show similar performance as existing network simulators in the same network environment. BANSIM is modeled based on the network architecture of ns-3; therefore, we first evaluate the applicability and accuracy of BANSIM by comparing it with ns-3. We compare BANSIM with the IEEE 802.15.6 standard program developed in our previous study [22]. The simulator code is publicly available at https://bitbucket.org/bumsou10/bansim.

Table 3
Node specifications.

NID	Priority	Sampling rate	Position
N1	0	$\leq$ 2.5 Kbps	Left elbow
N2	0	$\leq$ 2.5 Kbps	Left wrist
N3	0	$\leq$ 2.5 Kbps	Right elbow
N4	0	$\leq$ 2.5 Kbps	Right wrist
N5	0	$\leq$ 2.5 Kbps	Left knee
N6	0	$\leq$ 2.5 Kbps	Left ankle
N7	0	$\leq$ 2.5 Kbps	Right knee
N8	0	$\leq$ 2.5 Kbps	Right ankle



(a) Average delay (868 MHz-ASK)
(b) Average throughput (868 MHz-ASK)
(c) Average delay (915 MHz-QPSK)
(d) Average throughput (915 MHz-QPSK)

Fig. 8. Comparison of ns-3 and BANSIM: average delay and throughput.

Table 4
Comparison of ns-3 and BANSIM: system performance.

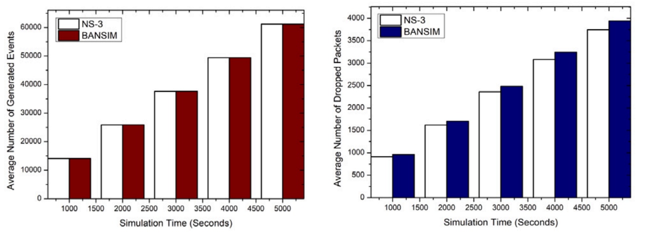
Parameter	ns-3	BANSIM
CPU	18%	14%
Memory	180 MB	130 MB
Build time	10 s	2 s
Simulation time	30 s	25 s

5.1. Simulation setup

The test network consists of one agent and eight nodes, which are interconnected based on a one-hop star topology. The agent broadcasts a beacon frame every 255 ms. The beacon frame includes the scheduled link information (i.e., each node is assigned 20 time slots, and the slot duration is 1 ms) and transmission power level. Each node moves periodically according to the walking model, and the channel model causes a packet loss based on the link quality between the sender and receiver. After receiving the beacon frame, each node sends data packets in the scheduled allocation interval and outputs a transmission result. Tables 2 and 3 list the common simulation parameters and node specifications, respectively. We conducted the simulation 50times with a 95% confidence interval.

5.2. Comparison with ns-3

Table 4 shows the system performance of ns-3 and BANSIM under the same simulation scenario. The biggest difference between them is the build time, the reason for which is that BANSIM builds only the



(a) Average number of generated events (b) Average number of dropped packets

Fig. 9. Comparison of ns-3 and BANSIM: average numbers of generated events and dropped packets.

essential models for configuring the WBAN environment; by contrast, ns-3 must build even unused source files owing to the dependency between communication models. These results prove that BANSIM can provide a lightweight simulation environment compared to ns-3. System variables used in this experiment can be found in the code repository.

As shown in Fig. 8, we derive the transmission delay and throughput according to the frequency band and modulation scheme. The data rate and symbol rate for each modulation scheme are listed in Table 1. By deriving the basic MAC performance parameters from both simulators, we can prove that they exhibit similar performances in the same network conditions.

Figs. 8(a) and (b) show the average delay and average throughput when the ASK modulation scheme is applied in the 868 MHz frequency band. The delay and throughput depend on the packet size, data rate, link quality, and distance between the sender and the receiver. Both simulators have the same protocol architecture, simulation parameters, and event handling mechanism. As a result, even if the packet size increases, the difference in performance between both simulators is less than 10%. The 10% difference is determined by system parameters. Importantly, the performance of both simulators changes at the same ratio as the packet size increases. These results prove that no additional dominant factors affect the delay and throughput other than the simulation parameters covered by BANSIM.

Comparing Figs. 8(c)–(d) with Figs. 8(a)–(b), the performance slightly increases as a result of changing the PHY attributes (i.e., QPSK modulation is applied). Similarly, the performance of both simulators changes at the same ratio as the packet size increases. These results prove that BANSIM uses the same simulation parameters as ns-3 and supports the essential components required for packet communication.

In particular, as shown in Figs. 8(b) and (d), the average throughput when sending a 100 byte packet is the same as the average throughput when the packet size is set to 200 bytes. The reason is that each node is allocated a total of 20 ms time slots for each beacon period, and a 100 byte packet can be sent twice within a given time. That is, considering the given PHY attributes, each node can only send a single packet within a given time when it sends a 150 byte packet or more. This phenomenon occurs in both simulators. These results show that both simulators have the same PHY attributes and transport mechanisms.

In addition, we verified that BANSIM could capture a wide range of interactions that occurred in the network by comparing the number of events generated by MAC and PHY. Fig. 9(a) shows that the number of events generated by both simulators is the same because both simulators have the same event-handling mechanism (i.e., both simulators translate packet streams into discrete simulator events).

To evaluate the accuracy of the channel model, we compared the number of packets lost in the PHYs of both simulators under the condition that used the same walking model. As shown in Fig. 9(b), the number of packets lost in both simulators differs by less than 10%.

Table 5
DQN hyperparameters.

Hyperparameter	Value
Neural network model	Sequential model
Number of layers	2
Number of neurons per layer	24
Input dimension	2
Activation function	ReLU
Optimizer	RMSProp
Maximum number of episodes	4000
Maximum number of steps	300
Minimum epsilon/epsilon	0.1/1.0
Exploration ratio	0.5
Minimum replay memory size	1000
Replay memory size	100000
Update frequency of target-net	5
Reward discount factor	0.99
Batch size	64

Notably, the number of lost packets over the simulation time increased at the same rate in both simulators. Therefore, we demonstrated that the channel model provided by BANSIM performed the same operation as ns-3 without additional factors.

5.3. Use case: DQN training for TPC

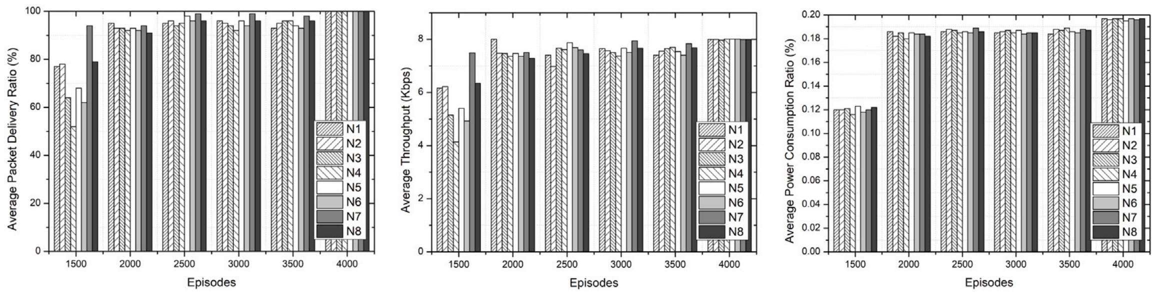
To evaluate whether BANSIM supports DRL-based protocol modeling and simulation, we present an example of a DQN training. In this example, we created a DQN model for TPC optimization. This example used the network parameters listed in Table 2 and assumed the same network environment and scenario as specified in the previous subsection. The packet size was fixed at 250 bytes.

In the agent, the DQN trainer interacts with SSCS to determine the optimal TPC policy π. SSCS requests an action (i.e., transmission power level) for node i from DQN trainer with the status block b_i. DQN trainer returns the immediate action a_t for current state s_t of b_i based on the epsilon-greedy algorithm. Then, SSCS notifies node i of the transmission power level by sending a beacon frame.

If the agent receives a data packet from node i, SSCS updates s_t, r_t, a_t, s_{t+1}, $done$, and $steps$ of b_i, and it creates a transition tuple for node i. DQN trainer stores the transition tuple observed from SSCS in the replay memory. If the size of the replay memory is larger than the minimum size, the DQN trainer trains the DQN model by sampling the transition tuples. Here, we define the state, action set, and reward as follows:

- **State**: The state is defined as $s = [P_{rx}, d]$. Here, P_{rx} indicates the signal strength when the agent receives a data packet from a node, and d denotes the distance between the agent and node. The agent is expected to find the optimal TPC policy according to the reception power and distance.
- **Action set**: The action set is defined as $A = [-25, -24, -23, -22, -21, -20, -18, -16, -14, -12, -10, -8, -6, -4, -2]$, where each value represents the transmission power level (dBm).
- **Reward**: Sending a packet with a low transmission power can save energy. The agent can earn from 1 to 15 points as a reward for the action. For example, if a node successfully sends a packet with the lowest transmission power (i.e., -25 dBm), the agent receives 15 points as a reward. In contrast, if a node successfully sends a packet with the highest transmission power (i.e., -2 dBm), the agent receives 1 point. If the transmission fails, the agent receives -1 points and activates the $done$ flag.

If the $done$ flag is enabled or the value of $steps$ exceeds the maximum number of steps, the current training episode ends. Moreover, when the current number of training episodes exceeds the maximum number of episodes, the DQN training procedure is completed. The DQN trainer



(a) Average packet delivery ratio (b) Average throughput (c) Average power consumption ratio

Fig. 10. Performance variation as the training episode progresses.

always returns the best action with the highest Q-value. The details on DQN hyperparameters are listed in Table 5.

Given s_t, the agent randomly selects a_t from A or selects a_t with the highest Q-value. The exploration rate was relatively high at the beginning of the training. Thus, the agent attempts to find the optimal policy π by trial and error. As a result, as shown in Fig. 10(a), the fluctuation of the average packet delivery ratio is the highest at the beginning of the training (i.e., the episode number is 1500). This phenomenon can also be observed in Fig. 10(b). The fluctuation in the average throughput was the highest at the beginning of the training. In addition, Fig. 10(c) shows that the average power consumption ratio is the lowest because the agent randomly selects the transmission power level at the beginning of the training.

However, as the training episode progresses, the exploration rate decreases. Hence, the probability of selecting the transmission power level suitable for the current state gradually increases over time rather than finding a new TPC policy by selecting various actions. As a result, according to Figs. 10(a), (b), and (c), the variation in performance gradually decreases as the episode progresses.

As shown in Figs. 10(a) and (b), at the end of the training (i.e., the episode number is 4001), each node shows an average packet delivery ratio close to 100% and the maximum average throughput achievable for a given bandwidth. In addition, each node sends a data packet with a minimum transmission power. Thus, the agent has learned the optimal TPC policy for the current state, which is composed of the reception power and the distance between the sender and receiver. Specifically, the agent finds the TPC policy optimized for the current mobility pattern. Therefore, the simulation results prove that BANSIM is capable of DRL-based network modeling and simulation. Users can easily build a DQN environment using the given DQN example code.

6. Conclusion

In this study, we developed BANSIM, a simple and intuitive DRL-based simulation environment in Python. BANSIM supports basic network models for packet communication and BAN-specific components. As a result, researchers can use BANSIM to address major research problems in WBAN, such as uplink scheduling or TPC. The sample execution demonstrated that BANSIM can capture a wide range of interactions that occur in networks. Using the DQN training example, we proved that BANSIM could support DRL-based network modeling and simulation. In future work, we plan to add supplementary features to BANSIM, such as multi-channel model and ETSI SmartBAN [23]. We hope that BANSIM becomes a useful WBAN simulation environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by an Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2019-0-01343, Training Key Talents in Industrial Convergence Security) and Research Cluster Project, R20143, by Zayed University Research Office.

References

[1] D. Lewis, IEEE P802.15.6/D0 Draft Standard for Body Area Network, Tech. rep., 15-10-0245-06, 2010.
[2] F. De Vita, D. Bruneo, A. Puliafito, G. Nardini, A. Virdis, G. Stea, A deep reinforcement learning approach for data migration in multi-access edge computing, in: Proceedings of the ITU Kaleidoscope: Machine Learning for a 5G Future, ITU K, IEEE, 2018, pp. 1–8.
[3] P. Gawłowicz, A. Zubow, Ns-3 meets OpenAI gym: The playground for machine learning in networking research, in: Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2019, pp. 113–120.
[4] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, X. Hei, ns3-ai: Fostering artificial intelligence algorithms for networking research, in: Proceedings of the 2020 Workshop on Ns-3, 2020, pp. 57–64.
[5] R. Kazemi, R. Vesilo, E. Dutkiewicz, R. Liu, Dynamic power control in wireless body area networks using reinforcement learning with approximation, in: Proceedings of the 22th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, IEEE, 2011, pp. 2203–2208.
[6] R. Kazemi, R. Vesilo, E. Dutkiewicz, R.P. Liu, Reinforcement learning in power control games for internetwork interference mitigation in Wireless Body Area networks, in: Proceedings of the International Symposium on Communications and Information Technologies, ISCIT, IEEE, 2012, pp. 256–262.
[7] A. Chowdhury, S.A. Raut, A QoS alert scheduling based on Q-learning for medical wireless body area network, in: Proceedings of the International Conference on Bioinformatics and Systems Biology, BSB, IEEE, 2018, pp. 53–57.
[8] Y.-H. Xu, J.-W. Xie, Y.-G. Zhang, M. Hua, W. Zhou, Reinforcement learning (RL)-based energy efficient resource allocation for energy harvesting-powered wireless body area network, Sensors 20 (1) (2020) 44.
[9] G. Chen, Y. Zhan, G. Sheng, L. Xiao, Y. Wang, Reinforcement learning-based sensor access control for WBANs, IEEE Access 7 (2018) 8483–8494.
[10] E.M. George, L. Jacob, Interference mitigation for coexisting wireless body area networks: Distributed learning solutions, IEEE Access 8 (2020) 24209–24218.
[11] L. Wang, G. Zhang, J. Li, G. Lin, Joint optimization of power control and time slot allocation for wireless body area networks via deep reinforcement learning, Wirel. Netw. (2020) 1–10.
[12] R.I. Tinini, M.R.P. dos Santos, G.B. Figueiredo, D.M. Batista, 5GPy: A simpy-based simulator for performance evaluations in 5G hybrid cloud-fog RAN architectures, Simul. Model. Pract. Theory 101 (2020) 102030.
[13] D. Ghosal, S. Shukla, A. Sim, A.V. Thakur, K. Wu, A reinforcement learning based network scheduler for deadline-driven data transfers, in: Proceedings of the IEEE Global Communications Conference, GLOBECOM, IEEE, 2019, pp. 1–6.
[14] K. Menda, Y.-C. Chen, J. Grana, J.W. Bono, B.D. Tracey, M.J. Kochenderfer, D. Wolpert, Deep reinforcement learning for event-driven multi-agent decision processes, IEEE Trans. Intell. Transp. Syst. 20 (4) (2018) 1259–1268.
[15] K. Qu, W. Zhuang, X. Shen, X. Li, J. Rao, Dynamic resource scaling for VNF over nonstationary traffic: A learning approach, IEEE Trans. Cogn. Commun. Netw. 7 (2) (2020) 648–662.
[16] R.M. Sandoval, A.-J. Garcia-Sanchez, J. Garcia-Haro, Optimizing and updating lora communication parameters: A machine learning approach, IEEE Trans. Netw. Serv. Manag. 16 (3) (2019) 884–895.

[17] P. Gazori, D. Rahbari, M. Nickray, Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach, Future Gener. Comput. Syst. 110 (2020) 1098–1115.
[18] K. Muller, T. Vignaux, Simpy: Simulating systems in python, 2003, Available: https://pypi.python.org/pypi/simpy.
[19] M.O. Munoz, R. Foster, Y. Hao, Exploring physiological parameters in dynamic WBAN channels, IEEE Trans. Antennas and Propagation 62 (10) (2014) 5268–5281.
[20] K. Yazdandoost, Channel Model for Body Area Network (BAN), IEEE 802.15-08-0780-05-0006, 2009, p. 91.
[21] G. Dolmans, A. Fort, Channel Models Wban-Holst Centre/imec-nl, IEEE 802.15–08-0418-01-0006, 2008, p. 53.
[22] B.-S. Kim, T.-E. Sung, K.-I. Kim, An ns-3 implementation and experimental performance analysis of IEEE 802.15.6 standard under different deployment scenarios, Int. J. Environ. Res. Public Health 17 (11) (2020) 4007.
[23] M. Hämäläinen, T. Paso, L. Mucchi, M. Girod-Genet, J. Farserotu, H. Tanaka, W.H. Chin, L.N. Ismail, Etsi TC SmartBAN: Overview of the wireless body area network standard, in: Proceedings of the 9th International Symposium on Medical Information and Communication Technology, ISMICT, IEEE, 2015, pp. 1–5.

Beom-Su Kim, received his B.S and M.S degree in software engineering from Gyeongsang National University, Korea. He is currently working toward Ph.D degree at Chungnam National University. His research interests include ad hoc networks, wireless body area networks and QoS.

Ki-Il Kim received the M.S. and Ph.D. degrees in computer science from the Chungnam National University, Daejeon, Korea, in 2002 and 2005, respectively. He is with Department of Computer Science and Engineering, Chungnam National University, Daejeon, Korea. Prior to joining, he has been with the Department of Informatics at Gyeongsang National University since 2006. His research interests include machine learning for networks, wireless/mobile networks, fog computing, MANET, QoS in wireless networks, multicast and sensor networks.

Babar Shah is Associate Professor at the College of Technological Innovation, Zayed University, UAE. Dr. Babar received MS degree (2007) from University of Derby, Ukand Ph.D. (2014) from Gyeongsang National University, South Korea. Dr. Babar professional services includes but are not limited to - Guest Editorships, University Services, Workshops Chair, Technical Program Committee Member, and reviewer for several international journals and conferences. He has authored over 50 scientific research articles in prestigious international journals and conferences. His Research interests include WSN, WBAN, IoT, Churn Prediction, Security, Real-time communication and Mobile P2P networks.