BLE Can See: A Reinforcement Learning Approach for RF-based Indoor Occupancy Detection

Md Fazlay Rabbi Masum Billah, Nurani Saoda, Jiechao Gao, Bradford Campbell
University of Virginia, VA, USA
{masum,saoda,jg5ycn,bradjc}@virginia.edu

Abstract

The emergence of radio frequency (RF) dependent device-free indoor occupancy detection has seen slow acceptance due to its high fragility. Experimentation shows that an RF-dependent occupancy detector initially performs well in the room to be sensed. However, once the physical arrangement of objects changes in the room, the performance of the classifier degrades significantly. To address this issue, we propose BLECS, a Bluetooth-dependent indoor occupancy detection system which can adapt itself in the dynamic environment. BLECS uses a reinforcement learning approach to predict the occupancy of an indoor environment and updates its decision policy by interacting with existing IoT devices and sensors in the room. We tested this system in five different rooms for 520 hours in total, involving four occupants. Results show that, BLECS achieves 21.4% performance improvement in a dynamic environment compared to the state-of-the-art supervised learning algorithm with an average F1 score of 86.52%. This system can also predict occupancy with a maximum 89.23% F1 score in a completely unknown environment with no initial trained model.

CCS Concepts

• **Computer systems organization → Embedded systems, Sensor networks**.

Keywords

Reinforcement learning, BLE, DQN, Occupancy detection

ACM Reference Format:
Md Fazlay Rabbi Masum Billah, Nurani Saoda, Jiechao Gao, Bradford Campbell. 2021. BLE Can See: A Reinforcement Learning Approach for RF-based Indoor Occupancy Detection. In *Information Processing in Sensor Networks (IPSN' 21), May 18–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3412382.3458262

IPSN' 21, May 18–21, 2021, Nashville, TN, USA

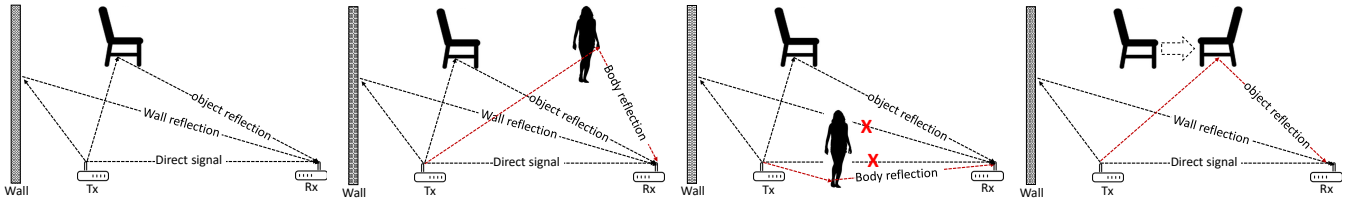
ACM ISBN 978-1-4503-8098-0/21/05…$15.00
https://doi.org/10.1145/3412382.3458262

1 Introduction

Indoor occupancy detection is a difficult problem, yet a reliable solution can yield a wide range of applications including home automation, energy savings, optimized ventilation, and pet monitoring. Until now, several approaches have been proposed to solve the device-free indoor occupancy detection challenge. The most common and intuitive solution is to use motion sensors. However, motion sensor-dependent systems often exhibit poor performance as they provide false predictions when the occupant is not moving. Another popular approach proposes installing radar at the zone transition point (i.e. doors) where the system counts the number of people entering or exiting the room [13, 14, 16–18, 22]. This scheme often cannot differentiate between a near-door event and a real crossing, also confuses the count when a group of people walks through the door in conjunction. Other alternative solutions use environmental data of a room such as CO_2, humidity, or temperature to infer occupancy [2, 3, 34]. However, environmental data changes slowly with respect to human presence and as such this system fails to make correct prediction instantaneously.

Recent advances in wireless sensing techniques provide a new solution to infer occupancy from the radio signal distortion caused by human presence [5, 12, 23, 27, 29, 31, 32, 36–38]. The intuition behind this technique is that human presence impacts the wireless signal through body reflection which reduces the similarity of the signal pattern between occupied room and unoccupied room. A signal processing algorithm or a machine learning model trained to identify the pattern of an empty room and the occupied room could detect human presence instantaneously.

A common limitation of this approach is that, to identify human presence in all kinds of indoor environment it requires a large database of every occupied scenario in different indoor environments. In practice, this is not possible as human behavior and movements are very random. As such, existing RF-based schemes suffer from high false positive



(a) Silent profile 1 (b) Noise profile 1 (c) Noise profile 2 (d) Silent profile 2

Figure 1: An illustration of RF-based occupancy paradigm. (a) Signals propagating in an empty room follows a certain multipath pattern. (b) The presence of a human can add additional multipath effect. (c) The presence of human can also alter existing multipath effect. (d) Movement of objects can alter the pattern as well.

rates when tested in a new room. Moreover, over time, performance of the system decreases in the room where it is initially calibrated due to the change of physical arrangements of objects. The system fails to identify the new multipath component caused by a shifted object (i.e. chair) and tends to characterize the reflection from the object as a reflection from a person (treating the empty room as occupied). This limitation is problematic as it means RF-based approaches can only work in a stationary environment (no alteration of objects) where the model is initially trained and would fail in a dynamic environment.

In recent years, there have been a proliferation of smart devices in homes and offices. These smart devices interact with humans frequently and as a result often correlate with occupancy. In this paper, we propose taking advantage of these devices for occupancy detection by blending them with a reinforcement learning agent. We introduce *BLE Can See* (*BLECS*), an RF-based occupancy detection system that can predict the occupancy of a dynamic environment using off-the-shelf energy efficient Bluetooth devices. BLECS design requires multiple BLE devices to communicate with each other in a room to generate RF features. Receiving those features in a wireless fashion, a reinforcement learning (RL) agent predicts the occupancy of the room. To update the decision policy and to adapt with the dynamic environment, the RL agent can take certain actions (e.g. control a smart appliance or turn off a light), depending on the application it serves. If an occupant reacts to the action by interacting with a smart appliance, the agent gets a feedback from that appliance that the room is occupied and can update its policy accordingly. However, it might degrade the user experience if the agent keeps turning off a light to see reaction. As such, in *BLECS* design, the RL agent also gets feedback from environmental sensors to update the policy for the unoccupied scenario where no user action is needed. Moreover, *BLECS* lets smart appliances to voluntarily send feedback to the RL agent, which allows the agent to update its decision policy in an online-learning fashion.

Realizing BLECS requires addressing two technical challenges:

(1) RF Feature extraction: To make the system energy frugal, BLECS extracts RF features: time of flight (ToF), received signal strength indicator (RSSI), and packet drop effect (PDE) from a Bluetooth signal rather than the conventionally used WiFi signals. However, unlike commercial WiFi chipsets, BLE devices lack channel state information (CSI) which makes it difficult to extract some of the RF features. Also, BLE devices do not have synchronized clocks and there is no straightforward approach to measure the ToF. To solve this issue, BLECS adopts a round-trip time (RTT) based ToF measurement technique. To account for the packet drop caused by human presence, BLECS introduces a delay-based mechanism.

(2) Feedback generation: To maintain the reinforcement learning scheme, the RL agent requires feedback from the environment. This feedback mechanism allows the agent to adapt its decision policy with the dynamic environment or even learn a completely new policy for an unknown environment. BLECS orchestrates this mechanism by receiving feedback from smart devices whenever an occupant interacts with a device. However, we observe that, if the agent gets feedback from only smart devices it gets biased towards predicting the room is occupied. To reduce this bias and to accelerate the policy update frequency, BLECS takes feedback from environmental sensors (i.e. CO_2 sensor) as well. Nevertheless, feedback from environmental sensors could be erroneous since they cannot immediately identify human presence. To reduce the impact of erroneous feedback, BLECS uses an exploration-exploitation strategy.

BLECS includes many desirable features for an indoor occupancy sensing scheme. It is **(1) dynamic:** the system can quickly adapt if the environment changes while remaining accurate; **(2) scalable:** can obtain high performance in rooms which it has never experienced; and **(3) energy efficient:** the system is very energy-frugal. Specifically, we make the following key contributions in this paper:

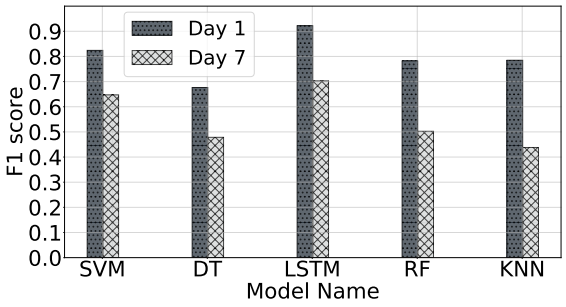


Figure 2: Five classifiers were trained with RF features to predict indoor occupancy. Over a seven-day period their performance dropped significantly.

- We design an adaptive scheme that can retain high performance in a dynamic environment. This scheme can also predict the occupancy of a room which it has never experienced without any pretrained model. This in effect, reduces the burden of labor-intensive offline training.
- We design BLECS with a working prototype involving ubiquitous commodity IoT devices, environmental sensors, and Bluetooth devices. We performed extensive evaluation in typical bedrooms, living rooms, and office rooms involving four occupants for up to 7 days in each room. Results indicate that, on average BLECS can achieve 75.54%–86.52% F1 score in a dynamic environment and up to 89.23% F1 score without a pretrained model in an unknown environment. Compared to the state-of-the-art ML-based approach, BLECS gains 21.4% performance improvement in dynamic environments.
- Unlike traditional RF-based occupancy detectors which use Wi-Fi signals, BLECS extracts features from Bluetooth signals. Our results estimate that on average *BLECS* requires only 20.5 mJ power to perform a prediction.

2 Background and Motivation

Figure 1 illustrates the underlying mechanism of the RF-based human sensing solution. To begin, in an empty room, a transmitter device sends signals to a receiver device in a continuous fashion and the receiver receives either the direct signal or a reflected signal (Figure 1a). The hypothesis is that, if everything in that room remains unchanged, over time, these received signals would build a *silent profile* (i.e. similar angle of arrival (AoA), ToF, RSSI, or PDE). When a person enters the room, she could create a *noise profile* by adding new multipath components (Figure 1b) or alter existing multipath components (Figure 1c). With a sufficient silent profile and noise profile, a supervised classification model could infer the occupancy of that room instantaneously.

One particular problem associated with this approach is that once the physical arrangement of objects changes in the room, the initially learned silent profile no longer exists. A new silent profile gets created due to shifted objects (Figure 1d) and the classifier needs to be re-trained with this new silent profile. A similar problem occurs when the position of the transmitter and the receiver changes. These devices need to be in exact same position where data collection was performed for model training, otherwise the silent profile would not be the same as the learned one. Other RF based approaches which use signal processing approach or threshold mechanism instead of classifiers can face a similar problem. The signal processing scheme that would have been developed for one room would only work in that room due to the variability in room shape, object orientation, and human activity.

Figure 2 illustrates how much the performance of supervised machine learning models drops in a seven-day period. We trained these supervised learning models using RF-features obtained from communication between several wireless BLE devices and deployed the trained model into a bedroom. Initially, these models performed very well since there was little change in the environment. As days passed, performance degraded due to silent profile alteration caused by object reorientation. For instance, at day 1, the best performing model LSTM was operating with an average F1 score of 92.21%. This performance gradually decreased to 70.25% at day 7. We observe similar performance degradation with SVM, Decision Tree, Random Forest, and KNN classifiers.

This demands a scheme that can adapt itself not only in a dynamic environment but also in an environment which it has never experienced. Reinforcement learning is one of the branches of machine learning where the learning model is not required to have a labelled dataset and can learn through experience. As such, in this paper we investigate the potential of reinforcement learning to solve this key challenge of RF-based occupancy detection.

3 Overview of BLECS

As shown in Figure 3, the BLECS design has five key components: the environment, an RF feature extractor module, a data preprocessor module, a reward generator, and an RL agent. These components interact to ultimately predict if a room is occupied or not. BLECS starts by receiving BLE radio signals from BLE devices in the environment and computing features based on those signals. The data preprocessor then removes outliers and combines features to create a state for the reinforcement learning (RL) agent. The RL agent uses this state and some history to estimate if the room is occupied. The RL agent can optionally actuate the environment (e.g. by turning off the lights in an unoccupied room), and receives feedback both from an occupant potentially overriding its actuation as well from other smart sensors that may exist in the

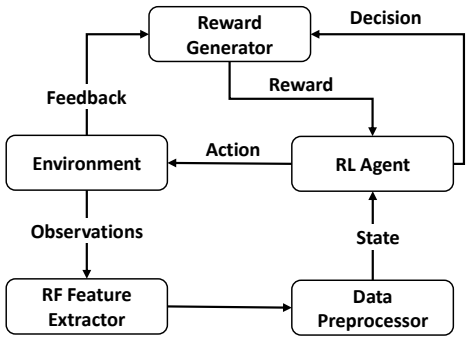


Figure 3: Workflow of BLECS

room. The reward generator uses this feedback to compute a reward which the agent uses to update its learning policy.

Reinforcement learning allows an agent to observe an environment, take an action in the environment, receive a reward for the action taken, and improve the correctness of its action to maximize its overall cumulative reward. This ongoing process allows RL agent to improve over time, and to adapt to changing conditions. BLECS leverages reinforcement learning to adapt to both different deployment rooms, as well as natural changes in those rooms over time.

Necessary for this process, however, is the ability to observe the environment and a mechanism for extracting feedback from the environment. We observe that as IoT devices become more prolific, BLECS can use already deployed BLE devices, environmental sensors, and smart devices to observe the environment. BLE devices and their wireless signals enable the agent to sense the current state of the environment, while smart IoT devices and environmental sensors provide feedback to the agent. We define these devices as the environment.

To interpret the wireless signals, *BLECS* computes RSSI, ToF, and PDE features from received BLE packets. ToF corresponds to the time taken for a signal to propagate from the sender to the BLECS receiver, and we use a RTT-based measurement technique to calculate ToF [11]. In this technique the BLE transmitter starts a timer and sends a message to the receiver. Upon reception, the second device responds, and when the original transmitter receives the reply it stops its timer and calculates the ToF considering propagation delay and hardware-specific delay. We also observe that when a human is in the line-of-sight path of any of the BLE devices this often leads to BLE message drop, suggesting that packet-drop information has a correlation with human presence. The transceiver calculates the PDE metric by timing the interval between BLE packet reception. It is essential to mention that, although, ToF and PDE are both dependent on time they are not related. The ToF value is very small (nanoseconds) given that wireless signals propagate at the speed of light. Whereas PDE can be quite large (few seconds) if a person blocks any of the devices for a while.

The data preprocessor module performs outlier filtering, feature augmentation and feedback sanitization on the computed features. Raw ToF data occasionally contains outliers and random noise which this module eliminates. To improve the performance of BLECS, this module also performs feature augmentation by concatenating multiple feature vectors coming from the BLE transceiver before feeding the augmented feature to the RL agent. Additionally, the preprocessor module sanitizes the feedback coming from the smart devices before sending the feedback to the RL agent.

Once the RL agent receives the RF features, it uses a deep Q-network (DQN) algorithm to predict the occupancy of the room. After the RL agent gets feedback from the feedback generator module it updates its decision policy accordingly. The working principle of the RL agent is discussed in detail in Algorithm 1.

BLECS generates rewards using the feedback from the building (e.g. a light switch), IoT devices (e.g. a power meter), environmental sensors (e.g. carbon dioxide sensors), and the most recent prediction of the RL agent. If the feedback suggests that the agent has made a correct prediction it receives a positive reward. On the other hand, the agent gets a negative reward for a wrong prediction.

3.1 BLECS Feedback Generation Mechanism

Whenever a human-actuated device (e.g. voice assistant, smart trash can, smart coffee machine, light switch, television, or refrigerator) in the room being monitored reports activity, BLECS is certain that the room is occupied. Certain smart devices can directly report when they are used, and BLECS can collect this signal as a sign of human presence. Other devices are not capable of reporting, but instead do change their power draw when actively used. To monitor these, a smart-socket IoT device periodically reports power draw information and the reward generator module analyzes this information to identify human presence. If human presence is detected, the reward generator module calculates a reward considering the prediction of the agent and forwards the reward to the RL agent.

In addition to the devices humans interact with, BLECS takes feedback from nearby environmental sensors (e.g. CO_2, humidity). The intuition is that if the level of CO_2 or humidity of a room does not change for a while, there is no human in that room. As such, if the level remains near-constant for a certain duration, these sensors send feedback that there is no one in the room. However, as mentioned earlier, these environmental sensors can provide false prediction since the level of CO_2 or humidity does not change instantaneously with human presence. To reduce the effect of false prediction, BLECS uses an exploration-exploitation strategy where 95%

of the time, the agent receives the actual feedback and 5% of the time the feedback is reversed by the reward generator.

Combining feedback from both IoT devices and environmental sensors is essential for successful reinforcement learning in BLECS. We empirically observed that using feedback only from devices occupants interact with, causes BLECS to achieve a high precision score but a very low recall score over time. This occurs because these devices only provide feedback when the room is occupied, and the RL agent gets positive reward for correctly classifying the room as occupied, but no negative reward for misclassifying the room as occupied. Hence, with time the agent is inclined to predict the room as occupied more frequently, rather than predicting the room as unoccupied. Involving environmental sensors solves this problem as they can provide feedback when the room is unoccupied and the RL agent can get negative reward for misclassification. Involving environmental sensors also increases the feedback generation frequency since, unlike IoT devices, environmental sensors do not depend on the human interaction. However, using only environmental sensors for feedback generation has its own problem. The feedback from environmental sensors are often erroneous and we observe that the agent fails to achieve a good performance with the absence of user-focused devices.

4 Learning Algorithm

In this section, we describe the learning methodology of BLECS. We start by explaining the underlying DQN algorithm. We then describe the enhancement of the basic DQN algorithm which enables BLECS to train an agent using radio-frequency parameters. Table 1 represents the notation used in our study.

4.1 The DQN Algorithm

Reinforcement learning is a framework where at each time step t an agent observes a state s_t in an environment, takes an action a_t based on the observation, and receives positive or negative reward r_t for the action taken. The objective of the RL agent is to find an action policy π that would maximize the expected cumulative reward $[\sum_{t=0}^{\infty} \gamma^t r_t]$, where γ is the discounted rate. $\gamma \rightarrow 0$ means immediate reward maximization is preferred, and $\gamma \rightarrow 1$ means far-sighted reward maximization is preferred.

Deep Q-network (DQN) is one of the approaches to find the optimal action policy. In this approach, a deep neural network (policy network) is used, which for a given environment, accepts a state s_t as input and gives $Q(s_t, a_t)$ as output (Figure 4). The agent takes the action a_t that satisfies $\max_{a_t} Q(s_t, a_t)$ and receives a reward $r(s_t, a_t)$. The objective of the policy network is to find a policy π that will approximate the optimal Q-function $Q^*(s_t, a_t)$. This optimal Q-function

Table 1: Notation

Symbol	Description
π	Policy
$\mathbf{S}, s$	State space and state
$\mathbf{A}, a$	Action space and action
$\mathbf{F}, f$	Feedback space, feedback
γ	Discounted rate
r	Reward
t	Time
Q	Q-function; output of the network
τ	Temporal difference error
D	Replay memory
M	Replay memory size
e	Replay memory tuple
n	Number of states concatenated
$\overrightarrow{tf_t}, \overrightarrow{rs_t}, \overrightarrow{pd_t}$	State parameters ToF, RSSI, PDE
ϵ	Exploration probability
$\mathcal{L}$	Loss function
θ_p, θ_t	Network parameters
N	Total steps required to sync. θ_p and θ_t
B	Batch size
α	Learning rate
E	Total epochs
k	Number of transmitters
T	Model update interval
U	Samples collected during T

maximizes the expected cumulative reward for each possible (s, a) tuple. In other words,

$$Q^*(s_t, a_t) = \max_{\pi} Q_\pi(s, a); \text{ for all } s \in \mathbf{S} \text{ and } a \in \mathbf{A}(s) \quad (1)$$

where, $\mathbf{S}$ represents set of all states and $\mathbf{A}(s)$ represents set of all possible actions for state s.

A key property of $Q^*(s_t, a_t)$ is that it always satisfies Bellman's optimality equation. That is,

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \cdot \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \quad (2)$$

where, $Q^*(s_{t+1}, a_{t+1})$ is the next-step's optimal Q value. In order to calculate $Q^*(s_{t+1}, a_{t+1})$ we do a second pass to the policy network using the next state s_{t+1} as input. From the output of the second pass $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ is calculated.

Once we know the optimal Q-value, Q^*, we subtract the inferred Q value from Q^* and calculate the temporal difference error (τ) incurred for the state-action pair (s_t, a_t).

$$\tau = r(s_t, a_t) + \gamma \cdot \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (3)$$

After the error is calculated, the weights within the policy network are optimized using gradient descent and back-propagation.

Target network: At each iteration over the dataset we first pass state s_t to the policy network in order to retrieve $Q(s_t, a_t)$. Then we do a second pass to the network using s_{t+1}

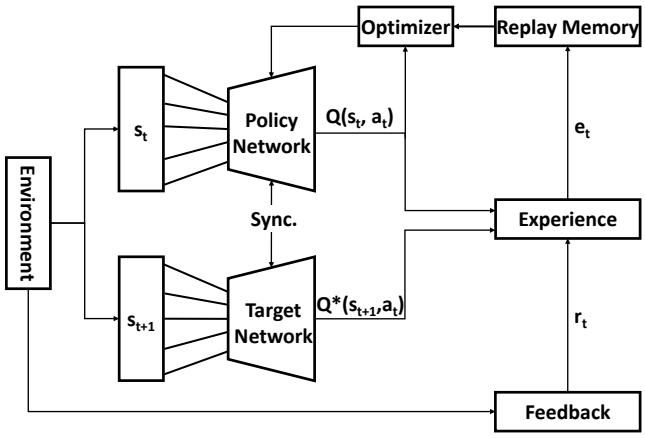


Figure 4: The DQN architecture

to find the target value $Q^*(s_{t+1}, a_{t+1})$. Finally, the gradient descent attempts to move $Q(s_t, a_t)$ closer to $Q^*(s_t, a_t)$. However, when we do the second pass we use the same weights of the network used in the first pass. This makes $Q^*(s_t, a_t)$ to move at the same direction as $Q(s_t, a_t)$. As a result, the loss function always finds a moving target and the optimization becomes unstable.

In order to solve this, a target network is used in DQN. The policy network is used to calculate $Q(s_t, a_t)$ while the target network includes all updates in the training. The policy network parameters θ_p and the target network parameters θ_t are synchronized after performing N updates over the dataset.

Experience Replay: In DQN a technique called *experience replay* is often used. With experience replay, we store the agents experiences at each timestep in a dataset called the replay memory (D). At time t, the agent's experience is defined as the tuple $e_t = (s_t, a_t, r_t, s_{t+1})$. This tuple gives us the summary of the agents experience at time t which we store in the replay memory. In practice, we set the size of the replay memory with some constant M. Therefore, it only stores last M experiences in the memory. From this replay memory dataset we randomly sample a batch to optimize the network. The reason we choose to optimize the network from random samples from the replay memory rather than providing the network with the sequential experiences as they occur in the environment is to break the correlation between consecutive experiences. If a network is trained only from the consecutive samples of experiences as they occur sequentially in the environment, the samples will be highly correlated and would therefore lead to inefficient learning. Taking random samples from replay memory breaks this correlation.

4.2 Training Methodology for BLECS

BLECS uses a state-of-the-art DQN algorithm explained in Section 4.1 for training purpose. Below we describe the specifics of this model.

Inputs: After the collection of RF features from BLE devices BLECS forwards those features to the network s_t. Here, the state input $s_t = (\overrightarrow{tf_t}, \overrightarrow{rs_t}, \overrightarrow{pd_t})$, given that $\overrightarrow{tf_t}$ is the measurement of ToF, $\overrightarrow{rs_t}$ is the RSSI, and $\overrightarrow{pd_t}$ is the packet drop effect. The dimension of these feature vectors are $n \times k$, where, n represents number of consecutive signals concatenated and k is number of transmitter BLE devices used.

Action space: After receiving s_t, BLECS's RL agent computes the Q-values and takes the application specific action a_t, where

$$a_t = \begin{cases} 0, & \text{room unoccupied, turn off light} \\ 1, & \text{room occupied, turn on light} \end{cases} \tag{4}$$

Reward function: The RL agent receives a reward $r(s, t)$ for each action taken at time t and, based on this reward, the agent updates the weights of its policy network. In BLECS, the agent can receive positive rewards in two scenarios: first, the agent predicts the room is occupied ($a_t = 1$) and the feedback from environment says the room is occupied ($f_t = 1$). Second, the agent predicts the room is unoccupied ($a_t = 0$) and the feedback supports this prediction ($f_t = 0$). In these two cases the agent receives a positive reward (+1). On the other hand, if the agent makes a wrong prediction it gets a negative reward (-1). In our system, there could be another case when there is no feedback from sensors and IoT devices. In this case the feedback is $f_t = -1$ and the agent gets no reward for the prediction, hence the model stays stable. Equation 5 defines our reward function.

$$r(s,t) = \begin{cases} 1, & (a_t = 1 \cap f_t = 1) \cup ((a_t = 0 \cap f_t = 0)) \\ -1, & (a_t = 1 \cap f_t = 0) \cup ((a_t = 0 \cap f_t = 1)) \\ none, & f_t = -1 \end{cases} \tag{5}$$

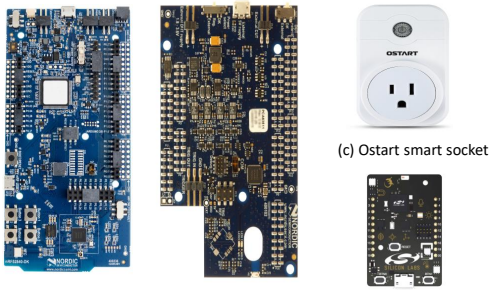
Exploration and exploitation: To find a balance between exploring and exploiting the environment BLECS uses an Epsilon-Greedy method. This method uses a decaying probability value ϵ which helps the agent to choose between a random action (explore) and a greedy action based on the existing knowledge (exploit). At the outset of the training, we set the $\epsilon = 1$, meaning the agent uniformly considers all possible actions. However, as the training advances, we slowly anneal ϵ to 0.001 and the agent tends to take the optimal action more frequently compared to a random action. As such, the agent performs more exploration at the beginning and slowly switches to exploitation.

Loss function: In BLECS, we have used the Huber Loss function which acts quadratic for small errors and linear for large errors. This prevents the network from having a dramatic changes while processing outliers. At each iteration, we sample a batch B from the replay memory dataset and use the temporal difference error measured in Equation 3 to calculate the loss function.

$$\mathcal{L} = \frac{1}{B}\sum_{e_t \in B} \mathcal{L}(\tau); \text{ where } \mathcal{L}(\tau) = \begin{cases} \frac{1}{2}\tau^2, & \text{for } |\tau| \leq 1 \\ |\tau| - \frac{1}{2}, & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 1: Training algorithm

Initialize:
$\gamma, N, M, \epsilon, B, \alpha, E, T, U$
Replay memory $D[M]$
if θ_p *exists* **then**
  θ_p = load(θ_p);
  $\theta_t = \theta_p$;
while *true* **do**
  Initialize: S, F, A;
  for $j \leq U$ **do**
    Fetch $s_j = [\overrightarrow{tf_t}, \overrightarrow{rs_t}, \overrightarrow{pd_t}]$;
    With probability ϵ:
      a_j = random(0,1);
      or, $a_j = \text{argmax}_a\, Q(s_j, a; \theta_p))$;
    takeAction(a_j);
    f_j = getFeedback();
    A.append(a_j), ;
    S.append(s_j);
    F.append(f_j);
  end
  for *epoch=1,...,E* **do**
    for *t=1,...,U-1* **do**
      $a_t = A[t]$;
      $s_t = S[t]$;
      $f_t = F[t]$;
      Compute r_t from f_t, a_t;
      $s_{t+1} = S[t+1]$;
      Store $e = [s_t, a_t, r_t, s_{t+1}]$ in D;
      From D, sample a random batch
       $B = [\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{r}_k, \boldsymbol{s}_{k+1}]$;
      for *each sample in B* **do**
        Calculate target Q value,
         $y_i = r(s_i, a_i) + \gamma \cdot \max_{a'} Q(_{i+1}, a', ; \theta_t)$;
        Calculate error, $\tau_i = y_i - Q(s_i, a_i; \theta_p)$;
        Calculate loss, $\mathcal{L} = \frac{1}{B}\sum_{e_t \in B} \mathcal{L}(\tau)$;
        Update network parameters,
         $\theta_p \leftarrow \theta_p - \alpha \nabla \theta_p \mathcal{L}(\theta_p)$
      end
      In every N steps, set $\theta_t = \theta_p$;
    end
  end
end



(a) Nordic nRF52840 BLE DK (b) Nordic nRF52 power profiler (c) Ostart smart socket (d) UG313: Thunderboard Sense 2

Figure 5: Devices used for prototyping

Optimization: Once the loss is calculated after applying an action, the policy network parameters, θ_p, are updated. To do that, at first the gradient step on the loss function with respect to θ_p is performed.

$$\nabla\theta\mathcal{L}(\theta_p) = -\tau \cdot \nabla\theta_p Q(s_t, a_t; \theta_p) \tag{7}$$

Once the gradient is measured, the parameters of the policy network are updated as follows:

$$\theta_p \leftarrow \theta_p - \alpha\nabla\theta_p\mathcal{L}(\theta_p) \tag{8}$$

where, α is the learning rate. After performing N steps over the dataset, the target network parameters θ_t is updated as $\theta_t = \theta_p$.

Overall algorithm: Algorithm 1 shows the overall proposed scheme. After initializing hyperparameters, the algorithm loads any pretrained agent if it exists. After that, the agent starts a lifelong approach to sense the environment and update the model. To prevent the agent from spending most of its resources on model updates, the BLECS algorithm updates its model every T hours. Suppose the agent receives U inputs, takes U actions, and collects U feedback over this period. After that, it enters the model update loop. In each step of the loop, the agent computes the reward and stores the experience tuple e in the replay memory. Then the algorithm fetches a random batch from the memory. For each sample, the agent calculates the loss and updates the policy network parameter. After performing N steps, it synchronizes target network parameters with the policy network parameters.

5 Implementation

To implement BLECS, we deployed up to seven BLE devices inside a single room, where one BLE device acted as a transceiver BLE and the rest acted as transmitter BLE. For prototyping, we used seven nRF52840 BLE boards, an Intel Core-i7 laptop as gateway, two Ostart smart sockets to get feedback from smart devices, one Thunderboard Sense 2 BLE as a CO_2 sensor, and an nRF52 power profiler kit to measure the current draw of BLE boards (Figure 5).

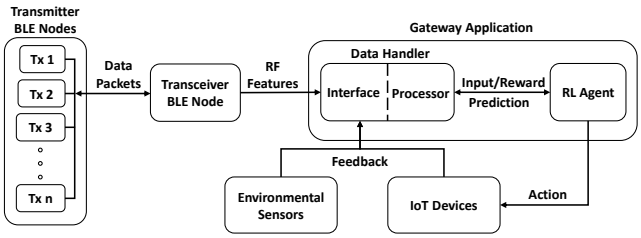


Figure 6: BLECS implementation overview

Figure 6 depicts how transceiver, transmitters, smart devices, environment sensors, and the RL agent inter-operate to predict the occupancy and update the agents decision policy. At the beginning, each of the transmitter BLEs establish a wireless connection with the transceiver BLE. In a round-robin fashion, the transceiver BLE communicates with each of the transmitter BLE devices and collects ToF, RSSI, and PDE. Upon receiving packets from all transmitter BLE devices and computing the features, the transceiver BLE concatenates them and forwards them to a gateway application.

The gateway application contains two modules: the data handler module and the RL agent. The data handler module hides the complex communication protocol from the RL agent. This module contains a JavaScript sub-module (interface) and a Python sub-module (processor). Each sub-module contains an MQTT broker which facilitates the connectivity among them. The interface captures all the messages broadcasted by the transceiver BLE in a wireless fashion and publishes to the processor. The interface is also responsible for capturing feedback generated by IoT devices and the CO_2 sensor wirelessly. A smart socket attached to an IoT device periodically advertises the current draw of its associated device, and the interface, upon receiving it, analyzes if there is a sudden rise or fall of the current from its previous recorded value. If a sudden change is detected the interface tells the processor that the room is occupied.

To measure feedback from the CO_2 sensor, the interface sub-module analyzes CO_2 concentration of last three minutes. If the difference between maximum concentration and minimum concentration over that period is not more than 20 ppm the interface decides the room have been unoccupied. In the course of our experimentation, we empirically selected a three minute monitoring window and 20 ppm concentration-difference because it gave fairly accurate prediction. Although there are sophisticated techniques to detect human presence from CO_2 concentration [34] we found that BLECS performs well with our straightforward technique. However, in the future we intend to incorporate techniques [19] that could provide good performance in different window/door opening settings.

Table 2: Network Hyperparameters

Hyperparameter	Values
Model update interval, T	3 hours
Discounted rate γ	$0.9 \sim 0.99$
Learning rate α	10^{-3}
Replay memory size, M	25,000
Target network synchronization step, N	5,000
Epochs, e	10

Upon receiving the RF features, the processor sub-module performs feature augmentation by aggregating consecutive signals and forwards them to the RL agent. Using those features as input, the RL agent makes a prediction on room occupancy and takes an action. In our prototype, we have used switching a light on or off as an action taken by the agent. If the agent turns off the light predicting the room is unoccupied, while it is not, the occupant turns it on and the feedback is sent to the processor. Using this feedback and the prediction of the agent, the processor module calculates a reward and forwards to the agent. Vice versa, if the agent turns on the light of an unoccupied room, feedback from environmental sensors helps to correct the agent. The BLECS RL agent does not need to completely rely on the reaction of the occupant (or the action of the agent) to calculate a reward since the processor is able to generate a reward using the feedback received from an occupants spontaneous interaction with a device.

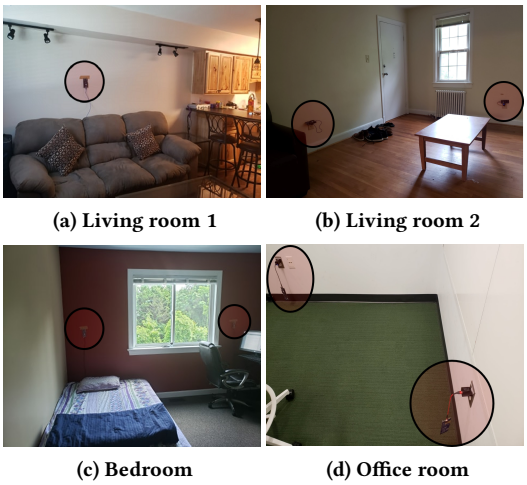
The underlying policy neural network which the RL agent uses consists of an input layer, three hidden layers, and an output layer. The input layer has $3 \times k \times n$ neurons as each of the k transmitter BLE devices provides 3 RF features and the processor sub-module aggregates n signals. Results from this input layer are forwarded to hidden layers each having 100 neurons followed by a ReLU function. The output layer consists of two neurons providing Q-values of the occupied and unoccupied state. The target network follows the same NN structure. To implement the neural network architecture we leveraged a Python-based deep learning API, Chainer [1].

Table 2 represents the hyperparameters we use while training the network. Throughout our experimentation we keep all of these hyperparameters unchanged. We did not perform any sophisticated tuning on the hyperparameters mentioned in the Table 2 as we noticed BLECS performs pretty well for a wide range of these hyperparameter values.

6 Evaluation

In this section, we describe the extensive experimentation we conducted to evaluate BLECS. Our experimentation answers the following questions:

- How does BLECS perform compared to the state-of-the-art supervised learning algorithms in a dynamic environment? We show that BLECS outperforms state-of-the-art baseline models.



(a) Living room 1 (b) Living room 2

(c) Bedroom (d) Office room

Figure 7: BLECS deployment in different settings

- Can BLECS provide accurate result in a completely unknown environment without a pretrained agent? We show that BLECS is able to achieve an 89.23% F1 score with no initial trained model.
- How sensitive is BLECS to different parameter settings including number of BLE devices, number of occupants, and changes in the environment? We observe that BLECS performance remains high in different settings.
- What is the overhead of providing a single prediction? Our experimentation tells that on average BLECS requires 1.29 milliseconds to make one prediction with average power consumption of 20.5 mJ.

6.1 Methodology

Evaluation metric: We use F1 score to evaluate the performance of BLECS. F1 score is the harmonic mean of precision and recall. In our case, precision is how often the agent correctly predicts the room is occupied, whereas recall means the number of times the agent correctly identifies the room as occupied out of all "occupied" decisions. Throughout the evaluation we determine the ground-truth label from our direct observation. Additionally, we evaluate the average energy consumption of associated BLE devices and average execution time for one prediction.

Benchmark: To evaluate BLECS, we develop four baseline models. All of these models are RF dependent and make use of different supervised learning classifiers to predict indoor occupancy. We train these models with the RF features collected using our system and fine tune them with optimal parameters obtained through experimentation.

The comparisons are 1) PADS: Qian et. al. [27] propose a human detection technique using an RF metric extracted and shaped from CSI measurement. From the physical layer of 802.11n standard this scheme at first gets the raw CSI and then extracts features from amplitude and phase information. Later they train a SVM classifier to predict human presence. 2) FreeSense: Xin et. al. [33] propose FreeSense where they use time and frequency domain information of Wi-Fi signals and trains a k-nearest neighbor (KNN) classifier to perform human identification. 3) FreeDetector: This scheme obtains channel state information from the PHY layer of commodity WiFi devices, analyzes the CSI variations and feeds them to a random forest classifier [40] to sense human presence. 4) WiWho: This model monitors the variation in the CSI data and collects walk segment feature of a person. Later, this model trains a decision tree classifier to identify a person [35]. 5) LSTM: We also implement a LSTM model to compare with BLECS. The RF features BLECS collects are sequential and LSTM is known to perform better on sequential data compared to other traditional classifiers.

Testbed: We conduct experiments in five rooms including two office rooms, one bedroom, and two living rooms (Figure 7), and with up to four individuals. All of these rooms are equipped with regular furniture. The experiments include four different room states: 1) empty, 2) occupant with activity: occupant walked around or exercised in the room, 3) occupant with fine movement: occupant was sitting still but doing activities like writing, typing, or eating, and 4) still occupant: occupant was sitting still or lying down with no movement. In the bedroom and living rooms, we evaluated BLECS for seven days each, whereas in the office we were limited by COVID-19 restrictions and evaluated for up to three hours.

Dynamic environment creation: To create the dynamic environment we either added, rearranged, or removed objects from the room under test in a controlled fashion. We evaluated the system robustness by changing the arrangement of small objects, medium objects, and large objects (i.e. cups, chairs, and beds, respectively). Furthermore, we changed the orientation of transceiver BLE and transmitter BLE devices by putting them on different walls of the room while the system was running. We also tested the system by covering some of the BLE devices with large objects (i.e. bookshelf).

Structured test and unstructured test: In our experimentation we performed two kinds of tests: structured and unstructured. The structured test was done in the bedroom for 10 hours in a very controlled fashion (e.g. controlled occupant activity, dynamic environment creation in a systemic fashion). The purpose of this test was to tune BLECS with optimal parameters. Once we found the optimal parameters we started the unstructured test. This test was performed for up to seven days in five different rooms. The unstructured test demonstrates how BLECS performs in a more realistic deployment.

Table 3: Structured test stages

Step	Activity	Duration
1	The room was kept empty	20 min
2	A person entered the room and performed following four activities in order: Sitting, Lying, Walking, and Exercising	5 min each
3	Admitted a refrigerator into the room, then the room was kept empty	20 min
4	A person entered the room and performed activities described in step 2	20 min
5	Removed a chair from the room, then the room was kept empty	20 min
6	A person entered the room and performed activities described in step 2	20 min
7	Changed the orientation of table and chairs, covered a transmitter BLE with bookshelf, then the room was kept empty	20 min
8	A person entered the room and performed activities described in step 2	20 min
9	Changed the orientation of BLE devices, then the room was kept empty	20 min
10	A person entered the room and performed activities described in step 2	20 min

6.2 BLECS Parameter Sensitivity

To understand how BLECS performs under different parameters such as the number of BLE transmitters and the number of aggregated signals, we performed a structured test in the bedroom. We deployed six transmitter BLE devices and one transceiver BLE device in the room and collected RF features for 600 minutes. The first 200 minutes involved 10 stages each having a time span of 20 minutes with one occupant. Table 3 describes these stages. During the course of the next 200 minutes we followed the same stages, but with two occupants. In the last 200 minutes we had three occupants. While collecting the data, we labeled each sample with the ground truth so that once the agent takes an action it can be given a reward using the label. The performance of our RL agent over this dataset is described next.

Number of BLE devices: As we collected RF features from transmitter BLE devices in a round robin fashion the dataset we created contains features from BLE nodes in a sequential manner. To vary the number of BLE devices, at each iteration we left out the sample which came from a distinct transmitter. Figure 8 represents the effectiveness of changing number of transmitter BLE devices. As shown, the prediction performance improves with increasing BLE devices within 10 prediction epochs. For instance, BLECS has a 83% F1 score with only one transmitter BLE device. With

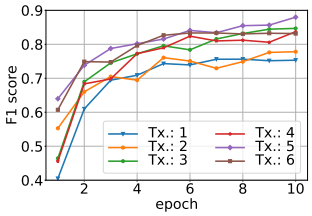


Figure 8: Effect of number of transmitters used. Increasing transmitters provides increased performance. Three transmitters are sufficient.

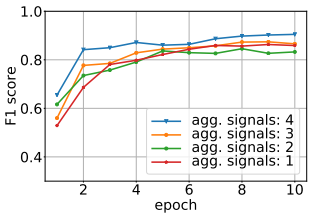


Figure 9: Effect of signal aggregation. Aggregating four rounds of signals provides 9.5% performance improvement.

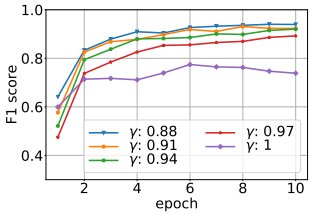


Figure 10: Effect of discounted rate γ. With γ = 0.88 BLECS performs optimally.

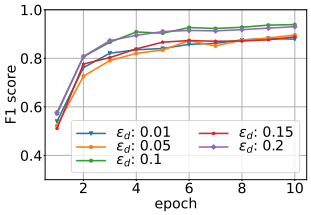
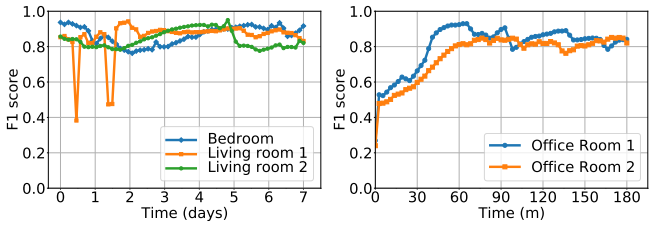


Figure 11: Effect of ϵ decrease rate. ϵ_d = 0.1 provides the optimal performance.

three transmitter BLE devices the performance increases up to 90%. However, performance did not improve much having more than three transmitter BLE devices. As a result, for all of the following evaluation steps we used three transmitter BLE devices.

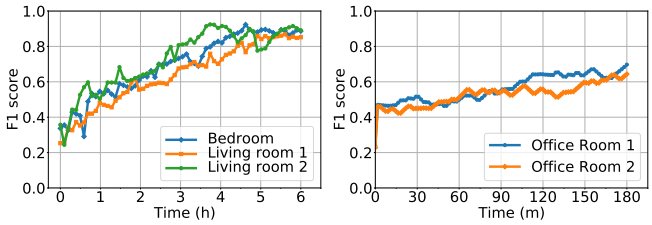
Signal aggregation: As mentioned earlier, BLECS receives signals from all transmitter BLE devices in a round robin fashion. After receiving a round of signals it stacks the RF features of all involved transmitter BLE devices. A common technique in RL algorithms is to stack consecutive samples and represent them as one state so that the state can contain temporal information. This is why we further stack more rounds of signals in order they are received by the transceiver BLE and represents them as one state. Figure 9 shows the effectiveness of signal aggregation. By stacking four rounds of signals, BLECS achieves a 9.5% performance improvement compared to having one round of signal as the input state.

Discounted rate, γ: Figure 10 represents the effect of the value of the discounted rate γ. When γ is set to one, future rewards are considered as important as immediate reward. When γ is decreased immediate rewards are more important to the agent than future reward. Figure 10 tells us that the BLECS agent performs better when γ is set to 0.88. This means the agent does not need to look very far to obtain maximum performance. This is understandable, since the environment the agent is working in is very dynamic, it can receive a negative reward for the same state it received a



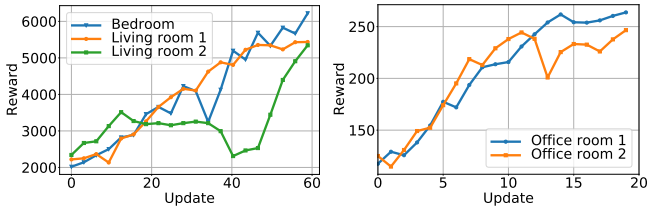
(a) Living rooms and bedroom (b) Office rooms

Figure 12: Testing BLECS in known environments. Results indicate that BLECS can maintain high performance over time. (a) Agent was pretrained in each room for five hours before deployment. (b) Agent was pretrained in each room for two hours before deployment.



(a) Living rooms and bedroom (b) Office rooms

Figure 13: Testing BLECS in unknown environments. Results suggest that BLECS can quickly adapt itself in an unknown environment.

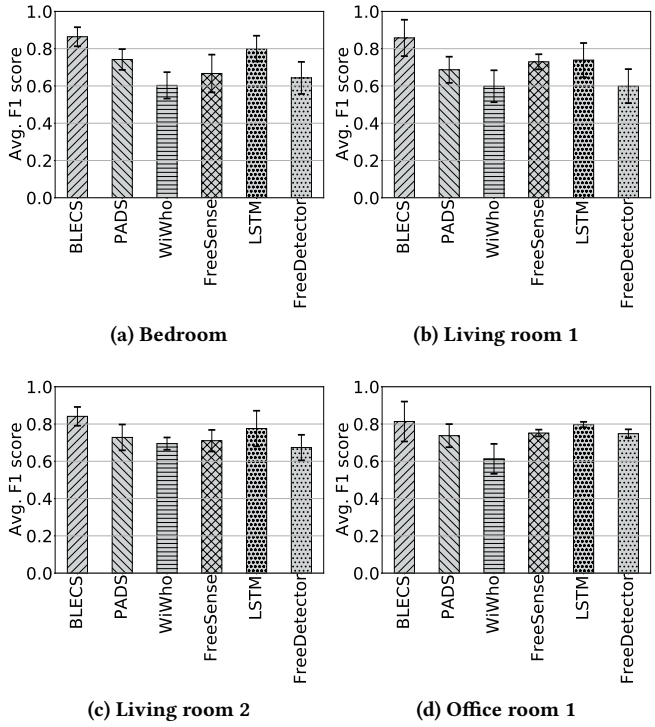


(a) Living rooms and bedroom (b) Office rooms

Figure 14: Total reward per update. The agent improves itself with time. (a) Model was updated in every three hours. (b) Model was updated in every 10 min.

positive reward for earlier. This being the case, the agent performs better when immediate rewards are preferred.

ϵ decrease rate: At each step of the learning process, the RL agent of the BLECS explores a new environment with a decaying probability ϵ. Initially, ϵ is set to one and the agent randomly explores new actions. However, with time ϵ decays and the agent focuses more on optimal action rather than exploring. Our evaluation tells that the decaying rate of ϵ has an impact of BLECS performance. Figure 11 depicts the performance of BLECS with varying decay rate. We observe that, with decaying rate 0.1, BLECS performance reaches maximum.



(a) Bedroom (b) Living room 1

(c) Living room 2 (d) Office room 1

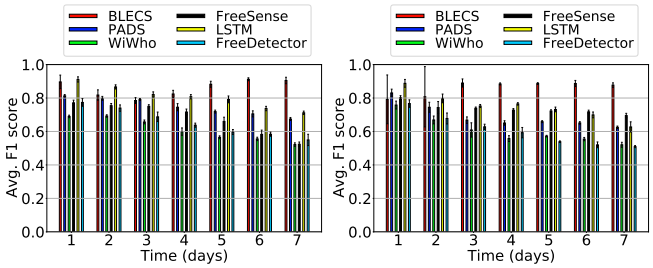
Figure 15: Average F1 score of BLECS and baseline models over a seven-day period in different environments.

6.3 Field Test

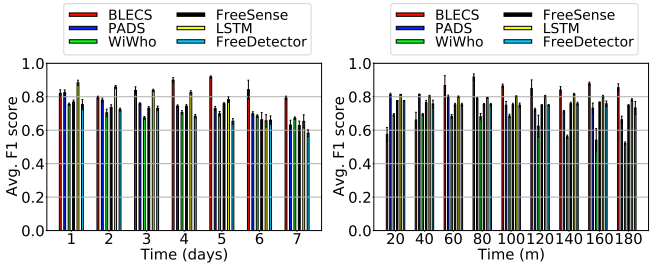
After fine tuning BLECS with optimal parameters we evaluated the system for seven days in the bedroom, living room 1, and living room 2. In office room 1 and 2 we evaluated BLECS for three hours. Here, we discuss our findings.

Performance in known environment: To evaluate how BLECS performs in known environment we pretrained the agent in the bedroom, living room 1, and living room 2 by collecting five hours worth of data from each room. To pretrain the agent in the office room 1 and 2 we collected samples for two hours from each room. Figure 12 depicts the system performance in different rooms with a pretrained agent. It can be seen that, over the course of the whole evaluation period BLECS was able to maintain high performance in every room with an average F1 score of 86.52% in the bedroom, 85.76% in living room 1, 84.14% in living room 2, 80.93% in office room 1, and 75.54% in office room 2.

Performance in unknown environment: To monitor BLECS performance in unknown environment we initialized the RL agent with random network parameters before deployment. Figure 13 illustrates BLECS performance in unknown environment. We can see that, within six hours BLECS was able to achieve maximum F1 score of 89.23% in the bedroom, 86.71% in living room 1, and 89.03% in living room 2. In office



(a) Bedroom (b) Living room 1



(c) Living room 2 (d) Office room 1

Figure 16: Average F1 score per day of BLECS and baseline models. BLECS retains high performance over time but performance of baseline models degrades with time.

room 1 and 2 BLECS was able to achieve maximum F1 score of 70.39% and 64.85% within three hours.

Average reward: We monitored total reward the pre-trained agent receives for each model update over the whole period of evaluation. Figure 14 illustrates that the reward trend-line follows an upward direction in all rooms. In the bedroom and living rooms the agent updated its learning model every 3 three hours, while in the office rooms the agent updated its model every 10 minutes.

Comparison with baseline models: To evaluate how BLECS performs compared to other supervised learning models, we developed PADS[27], FreeSense[33], FreeDetector[40], WiWho[35], and an LSTM classifier. We trained these models with our radio frequency features and fine tuned them with optimal parameters found by our empirical study. Figure 15 illustrates the performance of BLECS compared to these models in different environment settings. We deployed BLECS along with these baseline models in four different rooms and monitored their performance for seven days in the bedroom and living rooms. In the office room we monitored these model's performances for three hours. Results indicate that BLECS achieves 86.52% average F1 score over this seven days period in the bedroom. The next best model LSTM achieves 79.04% average F1 score during this period. In living room 1, BLECS achieves 85.76% F1 score on average, whereas the best performing model FreeSense achieves 72.95% F1 score on average. In the living room 2, BLECS performs with 84.14%

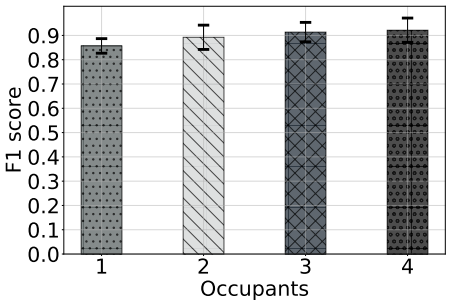
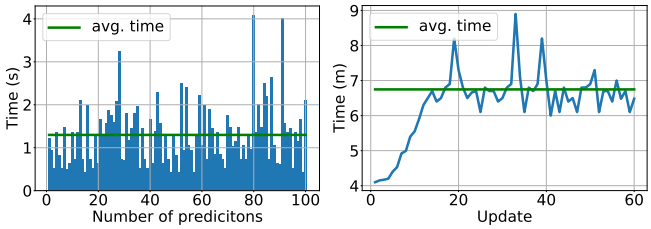


Figure 17: Average accuracy BLECS obtains over a seven day period with respect to the number of occupants. Results indicate that with increasing number of occupants performance of BLECS improves.



(a) Execution time to make a prediction (b) Execution time to make an update

Figure 18: End-to-end time requirement for execution. On average BLECS requires (a) 1.29 seconds to make a prediction, (b) 6.74 minutes to update the model

average F1 score compared to the 77.5% F1 score of LSTM. Finally, in office room 1, BLECS gets 80.93% F1 score, whereas LSTM gets 78.71% F1 score.

To truly understand the contribution of BLECS, we need to analyze the system's performance in later periods. Figure 16 illustrates the day by day performance of BLECS and baseline models in different environment settings. Our evaluation shows that over time all of baseline models performance decrease due to their inability to adapt with dynamic environment. In contrast, BLECS is able to adapt with changing environment and maintain high performance over time. For instance, at day seven in the bedroom BLECS maintains a F1 score of 91.66%, whereas the performance of PADS, FreeSense, FreeDetector, WiWho, LSTM drops to 64.70%, 43.82%, 50.25%, 64.70%, and 70.25% respectively. Compared to the best performing model LSTM, BLECS achieves a 21.4% performance improvement at the last day of evaluation. In living room 1, BLECS was able to keep up its high F1 accuracy with 19.35% performance improvement compared to the best performing model FreeSense, whereas in living room 2 BLECS gains 19.12% performance improvement compared to LSTM. In office room 1, BLECS on average achieves 1.65% performance improvement compared to LSTM and 11.6% compared to PADS over the whole duration of the evaluation.

Table 4: Exploring related research

Theme	Reference	General Idea	Limitations
Fingerpr-inting	[7, 29, 30, 33, 35, 40, 41]	Recognize dissimilarities between RF measurements	applicable in stationary environment only
Threshold	[23, 27, 28, 31]	Compare RF measurements with a predefined threshold	Low performance when occupant is still
Respiratory	[15, 25, 31]	Development of RF profile from chest motion	Requires a dense link of networks
Environ-mental Sensors	[2, 3, 6, 10, 34]	Measure CO_2, temperature of humidity	Not instantaneous
Ultrasound, RFID, Infrared, Camera	[4, 8, 20, 21, 24, 26, 39]	Monitor a space to detect change in sensor data	Privacy violation or need a controlled environment
Door sensor	[9, 13, 14, 16–18, 22]	Monitor entrance of a space	Low accuracy in near door event

Impact of number of occupants: To identify the impact of number of occupants we monitored BLECS over a seven-day period in the bedroom. Figure 17 shows the average accuracy BLECS achieves with respect to the number of occupants present in the room. As can be seen, with increasing number of occupants performance of BLECS increases to some degree. This is due to the fact that an increasing number of occupants creates more disruption over the noise profile and as a result the RL agent can differentiate between the silent profile and noise profile with higher accuracy.

Execution time: Figure 18a shows the time required for BLECS to make one prediction. On average BLECS requires 1.29 seconds to collect data from three transmitter BLE devices, aggregate four consecutive signals, preprocess the sample and make a prediction with the sample. Figure 18b illustrates the required time to update the model. Initially, when the replay memory is not full it requires less time to update the model. Once the memory is full BLECS requires on average 6.74 minutes to update the learning model.

Energy consumption: Using the nRF52 power-profiler test kit, we monitored how much energy the transceiver BLE and transmitter BLE consumes. From our analysis, the transceiver BLE node draws only 20.5 mJ power on average to collect data from six transmitter BLE devices, process the data to infer RF features and forward the features to the gateway application. Each transmitter BLE draws less power (3.0 mJ on average) over one cycle of data collection since its only task is to communicate with the transceiver BLE device. Our system design facilitates *BLECS* to predict occupancy in a continuous fashion. We believe, by controlling the prediction frequency, the average energy cost could be further reduced. We intend to incorporate such control mechanism in our future work.

7 Related Work

Recent years have seen many efforts to device-free human sensing using wireless signals. Unobtrusive, wireless occupancy detection systems works via analyzing the human impact on propagated signals, with no device attached to human body. These techniques can be broadly grouped into two categories: RF-based techniques and Non RF-based techniques. In Table 4 we summarize the existing occupancy detection techniques.

7.1 RF-Based Techniques

This approach can be bundled into three groups: a fingerprinting based approach, a threshold-based approach, and a respiration detection approach. The fingerprinting-based approach attempts to recognize dissimilarities between the CSI measurements caused by human presence in the occupied room and unoccupied room. Soltanaghaei et.al. proposed PeriFi which can detect people with no movement by analyzing multipath reflections of WiFi signals [29]. Rapid[7], proposed by Chen et.al. is another framework that analyzes CSI and acoustic information for robust person identification. FreeSense[33] captures the human influence over CSI measurement by performing a series of operations including principal component analysis, dynamic time wrapping, and discrete wavelet transform. WiWho[35] is another device-free sensing scheme that analyzes WiFi signals to find characteristics which can distinguish a person from a group of people. Other alternate approaches [30, 40, 41] use commodity WiFi routers and analyze the variations in RF measurements to predict human presence.

Threshold-based wireless sensing approaches compare RF features such as RSSI measurement with a predefined threshold [23, 27, 28, 31]. This approach is able to provide prediction with high accuracy when the occupant is moving, however, often fails when then occupant is stationary.

A common limitation of fingerprinting-based and threshold-based approaches is that they can only operate in a stationary environment where they are initially trained [28, 33, 35, 40]. We notice that when the system is stationary these systems achieve up to 91% accuracy, but if the environment is dynamic accuracy falls to 70%. Moreover, the energy cost and

deployment complexity of these schemes are very high due to involvement of WiFi routers. Our work, on the other hand, can profile the dynamic nature of environment through periodic updates to the the learning model, is easily to deploy, and draws minimal current.Respiration detection is another popular approach for human sensing, where, a particular RF profile is developed caused by chest motion during breathing [15, 25, 31]. Using commodity WiFi devices this approach detects a certain breathing pattern and detects human presence. Although this scheme performs well in certain scenarios, it requires a dense network created by many transceivers.

7.2 Non RF-based Techniques

There exists various approaches which use environmental sensors for human sensing. By measuring CO_2, temperature, and humidity [2, 3, 6, 10, 34] variation in the room these schemes determine indoor occupancy. Although these approaches are fairly accurate when the room is occupied or unoccupied for long period, they are incapable to provide instant prediction since the alteration of CO_2 or temperature is slow with respect to human presence. Moreover, this approach requires complex tuning for different window/door opening setting.

Other approaches involve door sensors [9, 13, 14, 16–18, 22] to detect human entrance or exit. This approach often fails to decide the occupancy situation of the room as it confuses its count if there is a near door event or if multiple people walk through the door simultaneously. Some pioneering solutions use ultrasonic sound [20, 24], infrared [4, 8], Camera [26], RFID [39], and electric field sensor [21] for occupancy detection. Deployment of camera or the infrared sensor increases the localization and occupancy detection capability however with a cost of increased privacy violation. Ultrasound sensors, RFID and electric field sensors on the other hand involves complex signal processing steps operate in an extremely controlled environment.

8 Limitations and Future Directions

Edge cases limitation: In this paper, we primarily focused on enabling RF-based sensing to work in changing spaces. We have not evaluated BLECS in a few corner cases such as multiple people entering or exiting the room, and people walking through the doorway. In future work, we intend to test and facilitate BLECS to work in these scenarios.

Robust feedback system: We have used a straightforward approach to provide feedback using CO_2 concentration level. We would like to incorporate advanced CO_2 based occupancy detection techniques that can work in different ventilation system to make the feedback system more robust. An interesting alternative would be to see how a motion sensor-based feedback system would work in lieu of CO_2-based feedback system.

Counting occupants: We have designed BLECS to predict whether the room is occupied or not. We believe, by modifying our existing CO_2 based feedback mechanism we could extend BLECS to count the number of people inside a room.

Incorporating AoA and AoD: A major feature added in new commodity BLE devices is the capability of direction finding. Our future endeavour would be to find angle of arrival (AoA) and angle of departure (AoD) using existing BLE beacons which we believe could offer improved performance.

Multi-agent reinforcement learning: The BLECS RL agent works in a single-agent environment. An exciting extension of this platform would be in a smart-building where multiple agents deployed in various rooms could collaborate for better prediction.

9 Conclusion

Accurately detecting when a room is occupied has proven to be a thorny challenge, with a system that is inexpensive, accurate, and privacy-preserving remaining elusive. However, BLECS revisits this challenge in light of several trends: i) promising wireless sensing techniques, ii) a wealth of machine learning approaches, and iii) an explosion of IoT devices. By sacrificing some capability yet reducing the power requirements of wireless sensing, BLECS can accurately detect occupancy without privacy-invasive sensors (e.g. cameras), is easily deployable, and adapts over time to changes in the environment. Central to extracting the necessary signal from the wireless channel measurements is a new reinforcement learning-based technique that learns from existing IoT devices. Importantly, this is not a black-box approach and our algorithm is stable with changes to its hyperparameters.

After implementing and deploying BLECS, we demonstrate the accuracy of this approach with a maximum 89.23% F1 score for occupancy detection in an unknown environment after six hours. Requiring only four BLE-based devices, BLECS can be retrofitted in existing buildings to enable the responsive, occupant-driven applications smart buildings require. Further, BLECS provides an example for how matching the correct machine learning approach to a new data stream (in this case wireless sensing with BLE) can produce new solutions to long-standing sensing challenges.

Acknowledgement

We sincerely thank the anonymous reviewers and the Shepherd for their valuable suggestions and feedback. This work is supported in part by the National Science Foundation under grant CNS-1823325.

References

[1] 2020. Chainer. https://chainer.org//. Accessed: 06-05-2020.

[2] M. Abedi and F. Jazizadeh. 2019. Deep-learning for Occupancy Detection Using Doppler Radar and Infrared Thermal Array Sensors. In *Proceedings of the IAARC*.

[3] R. Adeogun. 2019. Indoor occupancy detection and estimation using machine learning and measurements from an IoT LoRa-based monitoring system. In *Global IoT Summit (GIoTS)*.

[4] Y. Agarwal, B. Balaji, S. Dutta, and R. Gupta. 2011. Duty-cycling buildings aggressively: The next frontier in HVAC control. In *10th ACM/IEEE IPSN*.

[5] M. Billah and B. Campbell. 2019. Unobtrusive Occupancy Detection with FastGRNN on Resource-Constrained BLE Devices. In *Proceedings of the 1st ACM International Workshop on Device-Free Human Sensing*.

[6] C. Brennan, G. Taylor, and P. Spachos. 2018. Designing learned CO2-based occupancy estimation in smart buildings. *IET Wireless Sensor Systems* (2018).

[7] W. Chen, Y.and Dong, Y. Gao, and X. Liu. 2017. Rapid: A multimodal and device-free approach using noise estimation for robust person identification. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*.

[8] Robert H Dodier, Gregor P Henze, Dale K Tiller, and Xin Guo. 2006. Building occupancy detection through sensor belief networks. *Energy and buildings* 38, 9 (2006), 1033–1043.

[9] H. Elkhoukhi, Y. NaitMalek, and A. Berouine. 2018. Towards a real-time occupancy detection approach for smart buildings. *Procedia computer science* (2018).

[10] Alessandro Franco, Francesco Leccese, and Lorenzo Marchi. 2019. Occupancy modelling of buildings based on CO2 concentration measurements: an experimental analysis. In *Journal of Physics: Conference Series*, Vol. 1224. IOP Publishing, 012016.

[11] D. Giovanelli and E. Farella. 2018. Rssi or time-of-flight for bluetooth low energy based localization? an experimental evaluation. In *11th IFIP Wireless and Mobile Networking Conference (WMNC)*.

[12] L. Gong and Z. Yang, W.and Zhou. 2016. An adaptive wireless passive human detection via fine-grained physical layer information. *Ad Hoc Networks* (2016).

[13] E. Griffiths, A. Kalyanaraman, J. Ranjan, and K. Whitehouse. 2017. An Empirical Design Space Analysis of Doorway Tracking Systems for Real-World Environments. *ACM Transactions on Sensor Networks (TOSN)* (2017).

[14] T. Hnat, E. Griffiths, R. Dawson, and K. Whitehouse. 2012. Doorjamb: unobtrusive room-level tracking of people in homes using doorway sensors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*.

[15] O. Kaltiokallio, H. Yiğitler, and R. Jäntti. 2014. Non-invasive respiration rate monitoring using a single COTS TX-RX pair. In *13th International Symposium on IPSN*.

[16] A. Kalyanaraman, D. Hong, E. Soltanaghaei, and K. Whitehouse. 2017. Forma track: tracking people based on body shape. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2017).

[17] A. Kalyanaraman, E. Soltanaghaei, and K. Whitehouse. 2019. Doorpler: A Radar-Based System for Real-Time, Low Power Zone Occupancy Sensing. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

[18] N. Khalil, D. Benhaddou, O. Gnawali, and J. Subhlok. 2016. Nonintrusive occupant identification by sensing body shape and movement. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*.

[19] W. Liang and M. Qin. 2017. A simulation study of ventilation and indoor gaseous pollutant transport under different window/door opening behaviors. In *Building Simulation*.

[20] W. Mao, M. Wang, and L. Qiu. 2018. AIM: Acoustic imaging on a mobile. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*.

[21] Kevin C Mccarthy, Lori L Burgner, David W Taylor, Niall R Lynam, Eugenie V Uhlmann, Mitchell J Hourtienne, and Kenneth Schofield. 2004. Vehicle compartment occupancy detection system. US Patent 6,768,420.

[22] S. Munir. 2017. Real-time fine grained occupancy estimation using depth sensors on ARM embedded platforms. In *IEEE Real-Time and Embedded Technology and Applications Symposium*.

[23] S. Palipana, P. Agrawal, and D. Pesch. 2016. Channel state information based human presence detection using non-linear techniques. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*.

[24] Ashish Pandharipande and David Caicedo. 2011. Daylight integrated illumination control of LED systems based on enhanced presence sensing. *Energy and Buildings* 43, 4 (2011), 944–950.

[25] N. Patwari, J. Wilson, and S. Ananthanarayanan. 2013. Monitoring breathing via signal strength in wireless networks. *IEEE Transactions on Mobile Computing*.

[26] Ioannis Pavlidis, Peter F Symosek, and Bernard S Fritz. 2004. Near-infrared disguise detection. US Patent 6,718,049.

[27] K. Qian, C. Wu, Z. Yang, and Y. Liu. 2018. Enabling contactless detection of moving humans with dynamic speeds using CSI. *ACM Transactions on Embedded Computing Systems (TECS)* (2018).

[28] K. Qian, Z. Wu, C.and Yang, and Y. Liu. [n.d.]. PADS: Passive detection of moving targets with dynamic speed using PHY layer information. In *2014 20th ICPADS*.

[29] E. Soltanaghaei, A. Kalyanaraman, and K. Whitehouse. 2017. Peripheral wifi vision: Exploiting multipath reflections for more sensitive human sensing. In *Proceedings of the 4th International on Workshop on Physical Analytics*.

[30] J. Wang, N. Tse, and J. Chan. 2019. Wi-Fi based occupancy detection in a complex indoor space under discontinuous wireless communication: A robust filtering based on event-triggered updating. *Building and Environment* (2019).

[31] C. Wu. 2015. Non-invasive detection of moving and stationary human with wifi. *IEEE Journal on Selected Areas in Communications* (2015).

[32] Dan Wu, Daqing Zhang, Chenren Xu, Hao Wang, and Xiang Li. 2017. Device-free WiFi human sensing: From pattern-based to model-based approaches. *IEEE Communications Magazine* 55, 10 (2017), 91–97.

[33] Tong Xin, Bin Guo, Zhu Wang, Mingyang Li, Zhiwen Yu, and Xingshe Zhou. 2016. Freesense: Indoor human identification with Wi-Fi signals. In *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.

[34] S. Zemouri, Y. Gkoufas, and J. Murphy. 2019. A Machine Learning Approach to Indoor Occupancy Detection Using Non-Intrusive Environmental Sensor Data. In *Proceedings of the 3rd International Conference on Big Data and Internet of Things*.

[35] Y. Zeng, P. Pathak, and P. Mohapatra. 2016. WiWho: wifi-based person identification in smart spaces. In *15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[36] R. Zhou, X. Lu, P. Zhao, and J. Chen. 2017. Device-free presence detection and localization with SVM and CSI fingerprinting. *IEEE Sensors Journal* (2017).

[37] Z. Zhou, Z. Yang, and C. Wu. 2013. Towards omnidirectional passive human detection. In *Proceedings IEEE INFOCOM*.

[38] Z. Zhou, Z. Yang, C. Wu, L. Shangguan, and Y. Liu. 2013. Omnidirectional coverage for device-free passive human detection. *IEEE*

Transactions on Parallel and Distributed Systems (2013).

[39] H. Zou, L. Xie, and H. Jia, Q.and Wang. 2014. Platform and algorithm development for a rfid-based indoor positioning system. *Unmanned Systems* (2014).

[40] H. Zou, Y. Zhou, J. Yang, and C. Spanos. 2017. Freedetector: Device-free occupancy detection with commodity wifi. In *2017 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*.

[41] H. Zou, Y. Zhou, J. Yang, and C. Spanos. 2018. Device-free occupancy detection and crowd counting in smart buildings with WiFi-enabled IoT. *Energy and Buildings* (2018).