2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00020

Jarvis: Moving Towards a Smarter Internet of Things

Anand Mudgerikar
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
amudgeri@purdue.edu

Elisa Bertino
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
bertino@purdue.edu

***Abstract*—The deployment of Internet of Things (IoT) combined with cyber-physical systems is resulting in complex environments comprising of various devices interacting with each other and with users through apps running on computing platforms like mobile phones, tablets, and desktops. In addition, the rapid advances in Artificial Intelligence are making those devices able to autonomously modify their behaviors through the use of techniques such as reinforcement learning (RL). It is clear however that ensuring safety and security in such environments is critical. In this paper, we introduce `Jarvis`, a constrained RL framework for IoT environments that determines optimal devices actions with respect to user-defined goals, such as energy optimization, while at the same time ensuring safety and security. `Jarvis` is scalable and context independent in that it is applicable to any IoT environment with minimum human effort. We instantiate `Jarvis` for a smart home environment and evaluate its performance using both simulated and real world data. In terms of safety and security, `Jarvis` is able to detect 100% of the 214 manually crafted security violations collected from prior work and is able to correctly filter 99.2% of the user-defined benign anomalies and malfunctions from safety violations. For measuring functionality benefits, `Jarvis` is evaluated using real world smart home datasets with respect to three user required functionalities: energy use minimization, energy cost minimization, and temperature optimization. Our analysis shows that `Jarvis` provides significant advantages over normal device behavior in terms of functionality and over general unconstrained RL frameworks in terms of safety and security.**

***Index Terms*—IoT, Safety, Security, Functionality, Reinforcement Learning, Smart Home**

I. Introduction

The development of communication protocols for Internet of Things (IoT) devices, such as 6LowPAN, CoAp, and Zigbee, and the progresses in AI have made it possible to interconnect different IoT devices and achieve smart autonomous IoT systems. In the consumer market, IoT technology is mostly synonymous with products pertaining to the concept of "smart home", covering devices and appliances, such as lighting fixtures, thermostats, home security systems and cameras, that support one or more common ecosystems consisting of sensors of different kinds, such as motion, sound, light, heat, and touch, that can be controlled via devices associated with the ecosystems, such as smartphones.

A widely used approach for providing intelligent IoT services is through apps. These apps can support simple functionality such as "opening the door as an authorized user approaches the building" or more complex functionality like "navigating the car through a certain area autonomously". The apps communicate through API calls from the managing device to the IoT devices (sensors/actuators) through edge or cloud computing. Trigger-action apps platforms, like IFTTT [1], Zapier [2], and Apiant [3], allow 3rd parties to develop apps for these platforms. Such a strategy promotes the creation of communities of developers that build apps catering to different environments, devices, and protocols.

However, smart IoT-based systems require interconnection and inter-operation among devices, apps, and users. Such a complex environment of IoT devices and apps dynamically interacting with each other is prone to security/safety issues [4]–[6]. Another issue is that, in terms of functionality, each app has specific individual goals (e.g, "turn on the heater if the temperature is below the user requirement"). However, when selecting the actions to be executed, the app does not take into consideration the global view of the environment where the IoT device is deployed. Such a lack of awareness can lead to decisions and actions that are not globally optimal in terms of user requirements or goals. Evidently, there is a need for intelligent monitoring systems to ensure safety and maximize functionality with a global view of all devices and apps and their interactions. Previous work has focused on building RL based solutions to optimize IoT environment with specific goals, like energy management [7], [8], optimal resource allocation [9], and minimization of energy prices [10]. The drawback of those RL frameworks is that they focus on optimizing certain goals but do not take into account safety and security.

To address such drawbacks, in this paper we introduce `Jarvis`, a novel constrained RL framework for autonomously predicting 'optimal' and 'safe' decisions in an IoT ecosystem. An agent explores a simulated RL environment in order to find the optimal device actions according to the user's goals, such as saving energy and minimizing cost. Examples include turning off appliances to save energy, using devices during the off-peak hours to minimize electricity costs, etc. At the same time, the exploration of the agent is constrained by security and safety policies identified for the environment. For example, actions such as unlocking doors when user is not home or sleeping, turning off heater during winters, etc. are not permitted to the agent.

We design the framework to be context-independent and applicable to any IoT environment. By observing the specific IoT environment, `Jarvis` dynamically builds a simulated environment in terms of devices states and actions. An agent, constrained by security policies, can traverse the simulated environment in multiple episodes of specific time periods and find the optimal safe actions in terms of functionality requirements provided by the user. A Deep Q learning network (DQN) is used to determine the highest rewarding (quality) actions for each environment state and *time instance*. A Q learning approach is ideal for such an environment/ecosystem where, for each state-action pair, we can determine its quality through a cumulative reward for the time period in terms of the user goals. We train the agent using a deep neural network (DNN) to maximize the cumulative reward and thus generate the optimal quality function.

One of the main challenges in the design of our framework is that safety and security policies have to be specified by users and vary across different environments, deployed devices, and apps. Therefore, full manual specification of policies is not a viable approach. In order to minimize human effort in defining safety and security policies, `Jarvis` is designed to identify benign trigger-action (T/A) behavior in IoT environments by observing events occurring naturally in the environment or 'how the environment would function without machine intervention'. We define trigger-action (T/A) behavior based on enterprise platforms [1] and formal models [11] expressed in terms of device states and actions. Our assumption for safety and security is that such natural behavior is safe. Any unnatural or anomalous behavior is considered unsafe or malicious. However, in practice benign device malfunctions and human errors are common. So, an artificial neural network (ANN) is trained to filter these benign anomalies using back propagation. `Jarvis` observes the state transitions of the environment during a specified learning phase and dynamically builds the white-list of "safe policies" or safe T/A behavior using the ANN. After the learning phase is completed, the RL agent uses the white-list to constrain the RL environment and thus prevent unsafe/insecure state transitions. We use the terms ANN and DNN to refer to neural networks, with a single hidden layer trained by back propagation and multiple hidden layers trained by reinforcement learning, respectively. Formally, `Jarvis` can be defined as a model-based RL framework based on the Dyna-Q framework [12], where: (i) the model of the environment in terms of safety/security policies is learnt from actual user experiences through supervised learning, and (ii) the optimal policies for the learnt model are learnt by the RL agent from simulated experiences.

To summarize, we make the following contributions:

1) A context independent RL framework `Jarvis` for IoT environments.
2) A novel approach to constrain RL using safety and security policies.
3) An approach for dynamically learning safety/security policies in terms of triggers and corresponding actions in IoT environments.
4) An instantiation of `Jarvis` for a smart home environment and an extensive evaluation of the security and safety of the system using simulated data from manually crafted safety violations collected from prior work [4], [5] and user defined anomalous activities from the SIMADL project [13].
5) An analysis of the functionality benefits of the system using real world data in terms of three functional requirements: energy conservation, cost minimization, and temperature optimization.

The rest of the paper is organized as follows. In Section II, we briefly review the general IoT architecture and the Deep Q Learning framework. Next, we formally define our system model and challenges, and formulate the functionality optimization goal as a Markov Decision problem (MDP) in Section III. In Section IV, we define our deep RL based Q learning framework to solve the problem optimally. Next in Section V, we instantiate `Jarvis` for a smart home environment, provide design details, and analyze the benefits of `Jarvis` in a small smart home example. We quantitatively evaluate the instantiation in terms of safety and functionality in Section VI. Finally, we discuss related work in Section VII and outline a few conclusions in Section VIII.

II. Background

A. IoT Architecture

All popular IoT platforms, like Samsung Smartthings, Apple HomeKit, and OpenHAB, rely on the concept of separation of intelligence from devices. Smart applications can then be built by combining the intelligence layer with the functions and data provided by the devices. At a higher level, an IoT architecture can be considered as organized into four components: devices (sensors, actuators, appliances, etc.), edge (hub, router, connecting devices), cloud (cloud services, databases, analytics etc.), and control devices (mobile phones, desktops etc.). All IoT devices are standardized to provide certain 'capabilities' which allow devices to alter certain 'attributes' through 'commands'. Actuators like smart locks, lights, appliances have capabilities like lock/unlock, power on/off, increase temperature etc. Sensors for motion, sound, heat etc. have capabilities such as motion detected/no motion, high pitch sound/no sound/noise, optimal/high/low temperature etc.

These components inter-operate based on an event publish-subscribe architecture. The devices interact with the edge through device specific handlers which parse device specific messages (open/close, on/off, heat/cool, brew/do not brew) to/from the device and relay normalized edge-readable events (door opened, device turned on, heating temperature reached value x, etc.) to the edge. All publications of an event can be seen by the apps that have subscribed to that particular event.

B. Deep Q Learning

A Reinforcement learning (RL) framework [14] (see Figure 1) is an environment where for every state transition or state-action pair (s, a), a reward function $R(s, a)$ determines a reward value r. A RL agent traverses the environment according to a policy $\pi_\theta(s,a)$ for a time period θ and receives a total reward value accrued over all the state transitions involved for a given model of state transition probabilities. In a Q learning framework, the goal is to find the optimal policy which maximizes the total reward through exploration in such a RL environment using a Q function which determines the quality or cumulative reward for all state-action pairs.

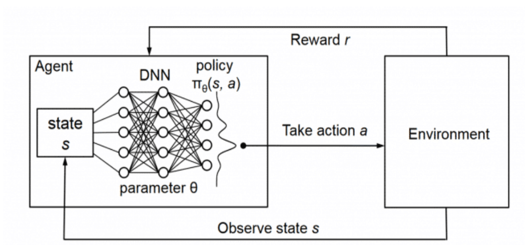

Fig. 1: Deep Q Learning Environment

In a deep Q learning system, a Deep Neural Network (DNN) is used to determine the optimal Q function using a temporal difference equation defined as follows:

$Q_t(s,a) = Q_{t-1}(s,a) + \alpha[R(s,a) + \gamma Max_{a'}\{Q_t(s',a')\} - Q_{t-1}(s,a)]$

where $Q_t(s,a)$ is the current Q function and $Q_{t-1}(s,a)$ is the previous Q function for the environment. The estimated next state and action are denoted by s' and a', respectively. The learning rate (α) determines to what extent newly acquired information overrides old information. The discount factor (γ) determines the importance of future rewards.

III. System Model and Problem Formulation

In this section, we first discuss modelling an IoT environment as a finite state model (FSM). After which, we define the state transitions of the environment as "episodes" and formulate the functionality optimization goal of `Jarvis` as a MDP. Then, we discuss the challenges associated with applying such a model to cyber-physical systems.

A. IoT Environment

The IoT environment FSM consists of k devices $\{D_1, D_2, .., D_k\}$, η users $\{U_0, U_1, ..., U_\eta\}$, and m apps $\{ap_0, ap_1, ..., ap_m\}$. By convention, manual operations in the model are denoted by a pseudo app ap_0. Each device in the environment is modelled by a set of device-states and a set of device-actions. At any point of time, device D_i can be in one of a set with a number i_{ss} of device-states: $\{p_{i_0}, p_{i_1}, ...p_{i_{ss}}\}$. At a *time instance* t, a device-action can be executed on device D_i from a set with a number i_{as} of device-actions: $a_i^t \in \{a_{i_0}, a_{i_1}, \ldots, a_{i_{as}}\}$. Specifically for IoT platforms, device capabilities and device attributes can be translated to device actions and device-states respectively.

Each device D_i has a transition function δ_i which is the link between a device-action and a device-state. For a device D_i in state p_{i_x} and having device action a_{i_y} take place on it, $\delta_i(p_{i_x}, a_{i_y}) = p_{i_{x'}}$ gives the new state of the device. Along with this, each device D_i has a dis-utility function $\omega_i(p_{i_x}, a_{i_y})$ which represents the dis-utility per *time instance* that results if the execution of device-action a_{i_y} is delayed in state p_{i_x}. A device can exist in different locations and thus have varying contexts in terms of accessibility, user permissions etc. To model this in our framework, we follow the container based approach followed by most IoT platforms. Each container acts as a boundary between the devices; the containers are organized hierarchically according to: user accounts, locations and groups. Therefore, device D_i can only be accessed by a set of authorized users u_i, $u_i \subseteq \{U_0, U_1, ..., U_\eta\}$, depending on its location l_i and its group g_i, and corresponding device and app subscription policies.

B. State Transition Model

For the overall environment state S_t, at the next *time instance* $t+1$, a set of authorized users $U^t \subseteq \{U_0, U_1, ..., U_\eta\}$ can use a set of apps $AP^t \subseteq \{ap_0, ap_1, ..., ap_m\}$ to perform an *action* A_t on a set of devices $D \subseteq \{D_1, D_2, .., D_k\}$, to get the new state of the environment S_{t+1}. So the state transition of the environment is represented as the current state $S_t = (s_0, s_1, ., s_i, ., s_k)$ plus a set of at most k (one per each device) device-actions. $A_t = \{a_0^t, a_1^t, ..., a_k^t\}$ is the set of actions taken at *time instance* t by a set of users U^t through a set of apps AP^t resulting in the next state S_{t+1} at *time instance* $t+1$. The next state is computed using the transition function for each device and corresponding action on the device such that $S_{t+1} = (\delta_0(s_0, a_0^t), \delta_1(s_1, a_1^t), ..., \delta_k(s_k, a_k^t) = \Delta(S_t, A_t)$ where Δ is the overall transition function of the environment.

Definition 1. *A FSM consists of tuple (SS, AS, Δ) where: $SS = \bigcup_{i=0}^{\nu} S_i$ is the state space with $\nu = \prod_{i=0}^{k} i_{ss}$; $AS = \bigcup_{i=0}^{\upsilon} A_i$ is the action space with $\upsilon = \prod_{i=0}^{k} i_{as}$; and Δ is the overall state transition function. The overall state of the FSM at time instance t is defined as $S_t = (s_0, s_1, ., s_i, ., s_k)$, where s_i is the state of the i-th device such that $s_i \in \{p_{i_0}, p_{i_1}, ...p_{i_{ss}}\}$.*

We monitor state transitions in the IoT environment in terms of "episodes". We define two configuration parameters for an episode: time period T and interval I. The state transitions occur every I seconds until the timestamp reaches T seconds, after which the state is reset to the initial state and marks the end of an episode. An episode basically consists of T/I *time instances* at which the state transitions of the environment are recorded. For example, for $\{T, I\} = \{60, 1\}$ minutes,the episodes are an hour long with state transitions every minute.

The following constraints apply to all state transitions:

1) Only one action per device per interval is allowed.
2) Only authorized users U^t at *time instance* t can access the app according to app subscription policies.
3) Only authorized app AP^t at *time instance* t for the device can take actions on the device according to device subscription policies.
4) Only one app can take action on the device per interval. Conflicts are resolved on a first come first serve basis.
5) In a single interval, each device can only change its state at most one time.

We following definition defines our model of the IoT environment state transitions in terms of episodes.

Definition 2. *An episode is defined as a tuple (N, S_0, T, I). $N = \{S_0, S_1, ., S_t, ..S_n\}$ is an ordered list of states reached in the episode where each next state $S_{t+1} = \Delta(S_t, A_t)$ for an action A_t and $0 < t \leq n$; $n = \lceil T/I \rceil$; S_0 is the initial state of the episode; T is the time period; I is the interval.*

C. Problem Definition

We formulate the functionality optimization goal of `Jarvis` as a MDP. A MDP is a sequential decision making problem where outcomes are under the control of an agent. The agent's goal in this case is to maximize functionality as specified by the user by choosing a sequence of actions for the upcoming episode of the environment. In our model, functionality requirements defined by the user are measured through a reward function. The functionality requirement specified determines the utility ($F()$) gained by the user which is one part of the reward function in the environment. The other part derives from the dis-utility ($D()$) caused to the user in terms of delays, waiting time, and discomfort. The general structure of the reward function is defined as follows: $R(S, S', t) = F(S, S', t) - D(S, S', t)$ where S is the current state, S' is the next state of the environment, and t is the current *time instance* of the episode. The goal of `Jarvis` is to maximize the cumulative reward at the end of the episode which is a MDP problem defined formally as follows.

Definition 3. *A MDP consists of a tuple (F, R, P, T, I, S_0). F is the FSM of the environment; R is the reward function; P is the state transition probability table; T is the time period; I is the interval, and S_0 is the initial state of the environment. The goal of the agent is to find a strategy of actions in accordance with P, maximizing the total value of R of the next upcoming episode, for the environment in state S_i in F where $0 \leq i \leq \lceil T/I \rceil$.*

D. Challenges

There are two key challenges when applying our model to cyber physical systems:

1) Unknown Reward Function: Quantifying the exact value of "utility" or "reward" gained or lost for any action performed is a challenge. It is worth noting that the $R()$ function described is not strictly Markovian in practice. In practice, $F()$ and $D()$ directly depend on inherently non-Markovian components, like environment dependant variables (electricity prices, temperature variations etc.) and user behavior, respectively. So, the value of the reward function R for a given state, action and *time instance* is not exactly known to the agent.

2) Safety/security of state transitions: Since the model is used to control a cyber physical environment, some state transitions can result in safety or security threats to the user specifically or to the general environment. Therefore, using a uniform state transition probability is impractical and state transitions must be context and environment dependant. The state transition probabilities of such unsafe state transitions should be zero. So, environment specific safety policies have to be identified before setting the model's state transition probability table P.

IV. RL BASED SOLUTION

In this section, we propose a RL based solution to the problem in Definition 3 subject to the aforementioned challenges. We propose algorithms to: 1) learn the safety/security policies in the environment and estimate the safe state transition probability table P_{safe}, and 2) estimate the reward function R_{smart} and learn the optimal and safe behavior in terms of user required functionality.

A. Safe State Transition

We identify state transitions which occur naturally in the environment during a specified learning phase as safe state transitions. The learning phase can either occur during setup of the environment or during a period defined by the user. During this learning phase, the user must approve every action taken as safe or manually perform it. This gives assurance that the corresponding state transition is 'natural' and the safety/security of the user is not violated. All state transitions are learnt in the form of trigger-action (T/A) behavior defined as:

T: Current State $S_t \rightarrow$ **A: Next Action** A_{t+1}

The T/A behavior from the learning phase is recorded to form the training dataset TD. However during the learning phase, there is the possibility of benign device malfunctions or human errors. To avoid learning this benign anomalous activity as unsafe, we filter TD to remove these anomalous state transitions using a feed forward ANN. The ANN is trained by back-propagation using environment specific user labelled benign anomalous activities. The labelling of benign anomalies can be done offline by the user or through user prompts in real time according to the environment and user preferences.

All filtered state transitions in the learning phase and their instance counts are stored in memory. Finally, only state transitions having instance counts greater than the environment specific threshold $Thresh_{Env}$ have a uniform probabilistic distribution in the model. All other state transitions probabilities are assigned null values to prevent unsafe situations. The details of the approach are outlined in Algorithm 1.

B. Reward Function Estimation

To address the challenge of unknown reward function, we estimate the reward function using user input and previous experiences. The estimated (smart) reward function for environment state S_t, action A, and *time instance* t is defined as:

$$R_{smart}(S, A, t) = \sum_{j=0}^{\kappa}(f_j)F_j(s, a, t) - \frac{I}{kT}\sum_{i=0}^{k}\omega_i(s_i, a)(t - t')$$

where the user inputs define the κ individual functionality requirements in terms of normalized functionality reward functions F_j. Examples of functionality rewards are: energy consumed, electricity costs, difference in optimal and current

Algorithm 1: Learning Safe State Transitions

```
Input: Training Dataset $TD$, Environment Context
       Variables $\{T, I, Thresh_{env}\}$
Output: State Transition Probability Table $P_{safe}$
Initialize: $P_{safe}[:,:] = 0, Count[:,:] = 0$
$Mem \leftarrow Filter_{ANN}(TD)$
while $d \leftarrow 0 : SizeOf(Mem)$ do
    while $t \leftarrow 0 : T$ do
        $(S, A) \leftarrow get(Mem, d)$
        $Count[S, A] \leftarrow Count[S, A] + 1$
        $SafeMem \leftarrow (S, A, Count[S, A])$
        $t \leftarrow t + I$
    $d \leftarrow d + 1$
while $(S, A, Count) \in SafeMem$ do
    if $Count > Thresh_{Env}$ then
        $S' \leftarrow \Delta(S, A)$
        $P_{safe}[S, S'] \leftarrow 1$
```

temperature, network usage etc. f_j are the weights associated with each functionality reward according to the user. The weights instill the concept of machine 'smartness' into the system, where the system is able to learn hypothetical strategies according to a combination of requirements or goals of the user. The user can alter weights to give more preference to one goal over the other but the essential strategy choices are made by the system keeping in mind the entire environment.

The second part in the above expression is the sum of all the estimated dis-utility caused by each device according to the their state and their pre-defined dis-utility values in episodes with time period T and interval size I. By dis-utility, we mean the discomfort or waiting time that might be caused to the user. This is an estimate obtained from the past behavior of the user, i.e., higher dis-utility means that the current state change is highly different from previous user behavior and vice-versa. t' is the closest preferred *time instance* for the state-action according to past behavior. The higher the difference in t and t', the higher the estimated dis-utility is. The normalized ω_i function corresponds to the cost of dis-utility for the specific device. The reason for including the dis-utility in the reward function is to make sure that the agent in the virtual environment does not take actions purely for functionality optimization. For example, the agent deciding not to operate any of the appliances when power conservation is a functionality goal. It would give maximum functionality results (0 power consumption) but at a high cost with respect to user convenience. The reward function parameters f_j and ω_i are chosen to balance utility and dis-utility rewards according to a utility-disutility ratio $\chi = \frac{kT \sum_{j=0}^{k} f_i}{I \sum_{i=0}^{k} \omega_i}$. Ideally, the value of χ should be chosen according to user preferences and environment configuration.

Algorithm 2: Learning Optimal State Action Quality

```
Input: Simulated Environment $Env$; Estimated
       Reward Function $R_{smart}() : \{SS, AS\} \rightarrow \mathbb{Z}$;
       Maximum Episodes $EP$; Exploration (Rate ,
       Min, Decay) = $(\epsilon, \epsilon_{min}, \epsilon_{decay})$; Safe State
       Transition Table $P_{safe}[S, S']$; Batch Size
       $BSize$; Discount Rate $\gamma$; Preferable Loss $L_p$
Output: State Action Quality Function
       $Q : \{SS, n\} \rightarrow AS$
Initialize: $Q[:,:] = 0$
while $ep \leftarrow 0 : EP$ do
    $S_{curr} = Env.Init()$
    $S_{next} = S_{curr}$
    while $t \leftarrow 0 : T$ do
        if $Random() \leq \epsilon$ then
            ▷ Exploration:
            while $P_{safe}[S_{curr}, S_{next}] \neq 0$ do
                $A_{curr} = Random(AS)$
                $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$
        else
            ▷ Exploitation:
            $count = 0$
            while $P_{safe}[S_{prev}, S_{curr}] \neq 0$ do
                $A_{curr} = Max(Q[S_{curr}, t], c)$
                $S_{next} \leftarrow \Delta(S_{curr}, A_{curr})$
                $c \leftarrow c + 1$
        $R_{curr} = R_{smart}(S_{curr}, A_{curr}, t)$
        $Mem \leftarrow (S_{curr}, A_{curr}, R_{curr}, t)$
        if $Mem > Bsize$ then
            $Q = Replay(Bsize)$
        $t \leftarrow t + I$
        $S_{curr} \leftarrow S_{next}$
    $ep \leftarrow ep + 1$
    procedure REPLAY($Bsize$, $L_p$)
    $MiniBatch \leftarrow Sample(Mem, Bsize)$
    while $(S, A, R, t) \in MiniBatch$ do
        $S_{next} \leftarrow \Delta(S, A)$
    $count = 0$
    while $P_{smart}[S_{next}, \Delta(S_{next}, A_{next})] \neq 0$ do
            $A_{next} = Max(Q[S_{next}, t + 1], count)$
        $count \leftarrow count + 1$
        $R_{cum} = R + \gamma R_{smart}(S_{next}, A_{next}, t)$
    $MiniBatch[(S, A, R, t)] \leftarrow (S, A, R_{cum}, t)$
        $Loss \leftarrow DNN_{Train}(Q, MiniBatch)$
    if $\epsilon \geq \epsilon_{min}$ $AND$ $Loss \leq Loss_p$ then
        $\epsilon \leftarrow \epsilon * \epsilon_{decay}$
        return $Q$
    end procedure
```

C. Q Learning Algorithm

We use a RL based approach to find the optimal quality function for the environment in terms of the estimated reward function R_{smart} as detailed in Algorithm 2. The FSM of the IoT environment is used to build a simulated environment where an agent can run multiple episodes to find the optimal and safe device actions for upcoming episodes. The agent balances exploration and exploitation according to the exploration rate ϵ. The exploration of the agent is constrained by security and safety policies at each step by using the safe state transition table P_{safe} learnt from Algorithm 1. The $Max(Q, c)$ function returns the c highest quality action for the given state and time stamp. Random batches from the agents prior experiences are selected and replayed to learn cumulative rewards according to the discount factor γ and batch size $Bsize$. Finally, the random batch with cumulative rewards is used to further train the DNN in order learn optimal Q table values for each state action pair and timestamp of the episode.

It is important to note that in our current formulation of the MDP, optimal actions are chosen with respect to the current state and timestamp of the episode. It is possible that in complex IoT environments, a more sophisticated policy identification in terms of higher order temporal trajectories is required. In this case, a MDP model relying on the immediate previous state would be a limitation. However, such a limitation can be overcome by incorporating temporal parameters for the episode in the state definition of the model. Such an approach would result in more fine-grained `Jarvis`'s optimization policies but at the cost of a higher number of state spaces and computation cost. The identification of ideal temporal parameters and specifics of the DNN, like number of hidden layers, activation functions, optimizers, network organization (feed forward/recurrent/convolutional), are beyond the scope of this work and should be chosen according to the device configurations and user requirements in the specific IoT environment.

V. Instantiation for a Smart Home Environment

In this section, we instantiate `Jarvis` for a smart home environment. We implement a prototype of `Jarvis` on the popular Samsung SmartThings [15] IoT platform. In what follows, we first provide design details about the key components of `Jarvis`. Next, we qualitatively analyze the benefits of `Jarvis` using a small smart home example.

A. Design Details

The main components of the architecture are discussed in what follows.

1) Logging System: To capture all device related events, this component uses a logger app (see Figure 2) which subscribes to all device capabilities and attribute changes associated with these capabilities. More specifically, an attribute change on the device results in the creation of an event which triggers the logger app because of its subscriptions, after which the logger app proceeds to store the log. The logs in JSON format are shown as follows:

(Event.date, Event.data, User.info, App.info, Group.info, Location.info, Device.label, Capability.name, Attribute.name, Attribute.value, Capability.command)

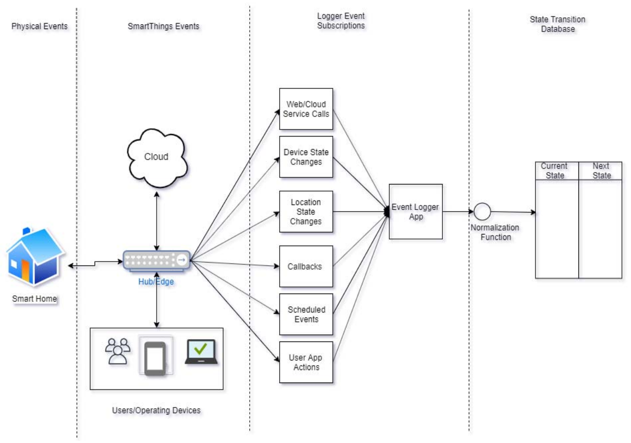

Fig. 2: Logging and State Modelling for the SmartThings Architecture

2) Log Parser: The logs are parsed through a normalization function to generate the state model of the environment. The normalization function quantifies the device *Attribute.Value* to discrete device states and *Capability.command* to discrete device actions. Capabilities and attributes for a device can be numbers, strings, vectors, enum values etc., so we have manually developed device specific normalization functions. For all devices in the logs, their unique device states and actions are modelled as the FSM of the environment. In our prototype, we choose the time period value T as 1 day, interval value I as 1 min, and learning phase time L as 1 week. We choose these values intuitively as typically users have periodic behavior in terms of days/weeks and generally do not require demand response times below a minute. During the learning period, state transitions are recorded as *learning episodes* according to T, I, L values.

3) Security Policy Learner: The security policy learner (SPL) is responsible for learning the safe state transitions table (P_{safe}) from *learning episodes* using Algorithm 1. By using results of user studies like [13] we manually generate labelled *benign anomalous* state transitions applicable to the smart home environment. Examples of such behavior, such as leaving fridge/oven door open, TV/oven on for short periods etc., are shown in Appendix A. The training set TD for the ANN is generated by randomly choosing state transitions from *learning episodes* and *benign anomalous* state transitions. The $Thresh_{env}$ should ideally be 0 as safety is critical in a smart home. A feed-forward multi-layer perceptron ANN with a single hidden layer is trained using back-propagation of TD samples.

4) Smart Reward Function: The smart reward function R_{smart} is estimated from environment specific normalized functionality reward functions F_j, the reward weights f_j, and the dis-utility functions ω_i provided by the user. The reward functions are generally scalar values available to the user in the environment through power meters, smart grids, and sensors. For example, if energy conservation is a functionality requirement, F_0 is directly proportional to power consumed in all device state transitions for the particular time interval which can be monitored by power meters. ω_i's are chosen by the user according to the environment configurations. For example, appliances which require immediate action and do not consume a lot of power are devices with high dis-utility, like lights, door bells, locks etc., have very high values of ω_i. Low dis-utility devices, like HVACs, washing machines, dish washers etc., have high power costs and no immediate action requirements. The dis-utility value of R_{smart} for *time instance* t is generated from the delay in state transitions $t' - t$, when compared to ideal *time instance* t' in *learning episodes* and the device specific dis-utility ω_i. The definition and discretization of ω_i depend on the device and environment configurations, and thus we manually define ω_i values for each device. However, it is important to note this is just a one-time offline cost after which the devices can be used in other smart home contexts.

5) RL Environment: `Jarvis` dynamically builds a RL Q learning environment using the FSM of the smart home and the functionality requirements specified by the user. It builds a simulated virtual environment using the openAI gym [16]. We have developed a general environment in the openAI gym which can be used for various IoT setups and device configurations. An agent can explore all device states and take any action on any of the devices at every *time instance* with interval time I for an *episode* of time T.

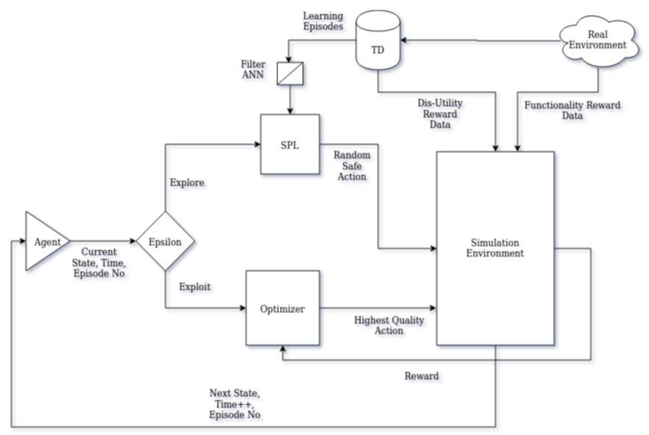

Fig. 3: `Jarvis` RL Framework

6) Optimizer: The optimizer deploys a RL agent to traverse the RL environment in EP episodes to optimize the Q function according to Algorithm 2 (see Figure 3). A batch processing DNN, developed using TensorFlow [17] with two hidden layers and a learning rate of 0.001, is used to determine the optimal Q function values using first-order gradient-based optimization. The drawback of using the DNN arbitrarily is that it does not remember experiences from the previous episodes.

To address such problem, we defined the concept of 'replay' in Algorithm 2, where the agent remembers the actions and corresponding cumulative rewards, for all previous 'replays' of prior episodes and uses the cumulative rewards to optimize the Q function. The system stores all the experiences of the episode and 'exploits' this knowledge during a new episode (see Figure 3). The agent uses a random batch from the past experiences to further train the neural network after every new episode.

7) Practical Deep Learning: The problem of action space explosion occurs when building the DNN as the action space increases exponentially. Consider an environment where each device can exist in only 2 states: on and off. Then even in this minimalist model, there can be a total of 2^k actions and we see that the action space grows exponentially with the number of devices. Maintaining a quality function of that magnitude for even small environments would be infeasible. To address this problem, we break up an action into mini-actions. The input to the DNN is the state of the environment. The output is an array of rewards for each 'mini-action' instead of a whole environment action. A 'mini-action' is defined as an intermediate action performed on only one device in an interval of the episode. In an asynchronous system where the state of the global system is characterized by the composition of the internal state of each individual device, a parallel execution can always be serialized into the execution of the individual device mini-actions. There can be at most k mini-actions possible in one interval of the episode, i.e one per each device. This allows the neural network to learn quality values efficiently for the mini-action space (grows linearly) rather than total action space (grows exponentially). There can only be k mini-actions for each trigger. So, in a single interval, the agent can take a maximum of k mini-actions to form a complete action.

B. Example: Analysis and Discussion

The example smart home consists of 5 devices: a smart lock, a door touch sensor, a smart light, a smart thermostat controller and a temperature sensor. The FSM of the environment is shown in Table I.

1) Safety: To demonstrate the effectiveness of our approach in learning safe state transitions, we install five common IFTTT [1] apps in the example smart home (see Table II). For instance, App 1 opens the door when a user arrives home. In terms of T/A behavior, the action (A) here is 'opening the door' and the trigger (T) is 'the door touch sensor or camera identifying the authorized user at the door'. In our model, this trigger would be represented as the door sensor being in "Auth. User" or in p_{1_1} state.

If no safety policies were identified by `Jarvis`, the T/A behavior of the apps would be as described in columns 4-5 of Table II. 'X' indicates that the device can be in any state (or take any action) and is not directly affected by the app. 'O' indicates that no action should be taken on the device. We see that these triggers and actions are specified without taking into account the environment context and only consider the individual app and the devices associated with it. They also do not take into account the security or safety of the environment. For example, in the context of App 1, it is not

Device Type	Device	p_{i_0}	p_{i_1}	p_{i_2}	p_{i_3}	a_{i_0}	a_{i_1}	a_{i_2}	a_{i_3}
Lock	D_0	locked(outside)	unlocked	off	locked(inside)	Lock	Unlock	Power Off	Power On
Door Sensor	D_1	Sensing	Auth. User	Unauth. user	-	Power Off	Power On	-	-
Light	D_2	off	On	-	-	Power Off	Power On	-	-
Thermostat	D_3	Heat	Cool	Off	-	Increase Temp.	Decrease Temp.	Power Off	Power On
Temp. Sensor	D_4	Above Opt. Temp	Below Opt. Temp	Optimum	Fire Alarm	Power Off	Power On	-	-

TABLE I: Smart Home Environment FSM

App	Description	Devices Involved	Trigger	Action	Safe Triggers	Safe Actions
1	Door unlocks when authenticated user arrives at the door	D_0, D_1	$(p_{0_0}, p_{1_1}, X, X, X)$	(a_{0_1}, X, X, X, X)	$(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_0})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_1})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_2})$	$(a_{0_1}, O, a_{2_1}, X, X)$
2	Maintain optimal temperature in the house	D_3, D_4	(X, X, X, X, p_{4_0}), (X, X, X, X, p_{4_1})	(X, X, X, a_{3_1}, X), (X, X, X, a_{3_0}, X)	$(p_{0_1}, p_{1_0}, X, X, p_{4_0})$ $(p_{0_1}, p_{1_0}, X, X, p_{4_1})$	(O, O, O, a_{3_1}, X), (O, O, O, a_{3_0}, X)
3	Lights turn on when user arrives home	D_0, D_1, D_2	$(p_{0_0}, p_{1_1}, X, X, X)$	(X, X, a_{2_1}, X, X)	$(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_0})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_1})$ $(p_{0_0}, p_{1_1}, p_{2_0}, p_{3_2}, p_{4_2})$	$(a_{0_1}, O, a_{2_1}, X, O)$
4	Door is opened/lights turned on when fire alarm is raised	D_0, D_2, D_4	(X, X, X, X, p_{4_3})	$(a_{0_1}, X, a_{2_1}, X, X)$	-	-
5	Thermostat/lights turned off when user leaves the house	D_0, D_1, D_3	$(p_{0_0}, p_{1_0}, X, X, X)$	$(X, X, a_{2_0}, a_{3_2}, X)$	$(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_0})$ $(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_1})$ $(p_{0_0}, p_{1_0}, p_{2_0}, X, p_{4_2})$	$(O, O, a_{2_0}, a_{3_2}, O)$

TABLE II: Comparison of normal vs safe T/A behavior

safe for the door to be unlocked when the user is not at home or sleeping. Similarly, turning the thermostat on when the user is not home or setting the temperature to unsafe high values in App 2's context. Turning off temperature and door sensors can also lead to safety issues. Next in columns 6-7, we show the safe T/A behavior learnt from Algorithm 1. We can see that `Jarvis` is able to learn the *natural progression* of device states, from which it automatically learns the security policies. The only exception occurs in the case of emergency situations raised by certain sensors like smoke alarms, fire sensors etc. The safe functioning of these devices cannot be determined from natural progression as such scenarios occur only in rare situations. So, we have to adjust our model to add security/safety policies for such devices manually as our security policy learner does not in these situations. In Section VI, we discuss using active learning approaches to learn *safe non-natural behavior* and make such adjustments autonomously.

2) Effectiveness of Constrained Exploration: We specify $k = 3$ example user required functionalities in the smart home environment: Energy Conservation, Electricity Cost Minimization, and House Temperature Optimization.

We compare the action quality learnt without any security constrains (unconstrained exploration) to action quality learnt with `Jarvis` (constrained exploration) in terms of the given functionalities in Table III using 8 common T/A behaviors. For instance, turning off the lights and thermostat when the user leaves the house. An unconstrained optimizer with a goal of conserving energy would turn off all the devices which can cause safety concerns like turning off essential sensory devices like fire-alarms, door lock sensors and temperature sensors. In the case of the single goal of temperature optimization, the unconstrained exploration would lead to turning on the thermostat even when the user is not home which is a safety as well as energy waste issue. We see that unconstrained optimization of actions according to functionality leads to unsafe situations, which are avoided using the constrained exploration of the `Jarvis` RL environment.

C. *Dis-utility vs Safety.*

It is important to notice the difference between the dis-utility in the R_{smart} reward function and the learnt safe transitions P_{safe} from the SPL component. High dis-utility and unsafe behavior might share some common patterns but they have different definitions in our model. While both are probabilistic and are learnt during *learning episodes*, they differ in their enforcement: the former is 'strict' as it represents user safety in the model, while the latter is 'lapse' as it represents user inconvenience.

We illustrate the difference using the smart home example with energy conservation as the functionality goal. A user leaves his house meticulously at 8 am and locks the door during the learning phase. Then while exploring the RL environment, the agent learns that it gains a higher reward when the lights and thermostat are turned off exactly at 8am because of the low dis-utility in the reward function. So the agent learns that for the trigger T_1 ="locking the door at 8am", the action A_1="turning the thermostat and lights off" is the highest quality action. Now suppose that the user is not so meticulous and leaves sometimes between 6am-8am. Now the agent finds that the reward for taking A_1 is lower as the utility part is the same but the dis-utility part for this action is higher as the user does not leave exactly at 8am. So, we see that although the reward is lower, if the user were to leave the house at 6am instead of 8am, the optimizer will still predict the optimal action A_1 even if the reward is lower. A security policy on the other hand is strict and does not allow any unsafe action. In the same example, for the trigger T_1, the action A_2 = "power off the lock" would cause a security concern. The SPL component learns that for T_1, the action A_2 must never taken place. So, in the RL environment this T/A behavior would never occur unlike the high dis-utility case seen before.

VI. Evaluation

We evaluate the instantiation of `Jarvis` for a smart home environment quantitatively in terms of two metrics: security and functionality. We first describe the simulated test-bed

Function	Trigger	Trigger Description	High Quality Action	High Quality Safe Action	Action Description (quality vs safety)
Energy Conservation	$(p_{0_0}, p_{1_0}, X, X, X), t$	User leaves the house and locks the door	$(a_{0_2}, a_{1_0}, a_{2_0}, a_{3_2}, a_{4_0}), t$	$(O, O, a_{2_0}, a_{3_2}, O), t$	Turn off the lights and thermostat only not all appliances
Energy Conservation	$(X, X, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn the thermostat off
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_0}), t$	Temperature drops below optimum and user at home	$(O, O, O, a_{3_0}, O), t_p$	$(O, O, O, a_{3_0}, O), t'$	Power on the heater at closest off peak hour t' instead of waiting for optimal non-peak hour t_p
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_1}), t$	Temperature goes above optimum and user at home	$(O, O, O, a_{3_1}, O), t_p$	$(O, O, O, a_{3_1}, O), t'$	Power on the cooler at closest off peak hour t' instead of waiting for optimal non-peak hour t_p
Electricity Cost Minimization	$(p_{0_2}, p_{1_0}, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn the thermostat off
Temperature Optimization	$(X, X, X, X, p_{4_0}), t$	Temperature drops below optimum	$(O, O, O, a_{3_0}, O), t$	$(O, O, O, a_{3_0}, O), t''$	Power on the heater to optimize the temperature but only at t'' when the user arrives home
Temperature Optimization	$(X, X, X, X, p_{4_1}), t$	Temperature goes above optimum	$(O, O, O, a_{3_0}, O), t$	$(O, O, O, a_{3_0}, O), t''$	Power on the cooler to optimize the temperature but only at t'' when the user arrives home
Temperature Optimization	$(X, X, X, X, p_{4_2}), t$	Optimal temperature is reached	$(O, O, O, a_{3_2}, O), t$	$(O, O, O, a_{3_2}, O), t$	Turn thermostat off

TABLE III: Comparison of action quality for Unconstrained vs Constrained Exploration

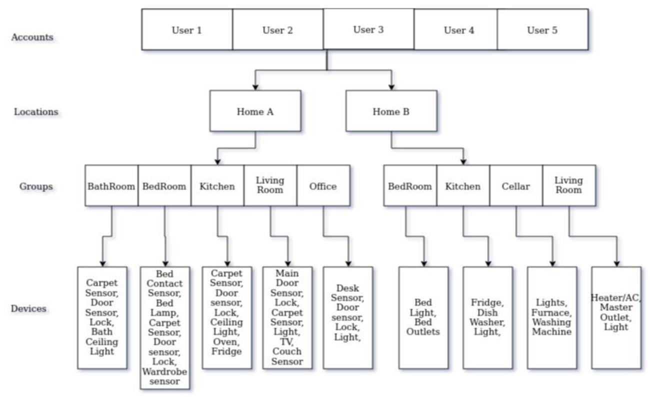
Accounts
Locations
Groups
Devices
User 1
User 2
User 3
User 4
User 5
Home A
Home B
BathRoom
BedRoom
Kitchen
Living Room
Office
BedRoom
Kitchen
Cellar
Living Room
Carpet Sensor, Door Sensor, Lock, Bath Ceiling Light
Bed Contact Sensor, Bed Lamp, Carpet Sensor, Door sensor, Lock, Wardrobe sensor
Carpet Sensor, Door sensor, Lock, Ceiling Light, Oven, Fridge
Main Door Sensor, Lock, Carpet Sensor, Light, TV, Couch Sensor
Desk Sensor, Door sensor, Lock, Light,
Bed Light, Bed Outlets
Fridge, Dish Washer, Light,
Lights, Furnace, Washing Machine
Heater/AC, Master Outlet, Light

Fig. 4: Overview of the evaluation setup

for our evaluation. Next, we evaluate the safety and security policies learnt by the SPL component. Then, we analyze the effectiveness of the Optimizer in terms of three user defined functionalities.

A. Testbed

We build a virtual testbed (see Figure 4). It consists of five users and two locations: Home A and Home B. We use the open smart home simulator (OpenSHS) [18] to build data sets for home A using simulated daily user activities [19]. The Home B's datasets are simulated from real world data collected by user studies [19]. We use 55,156 samples of user generated benign anomalies [13] for building the training dataset TD for the testbed.

B. Safety and Security

Building Malicious Data-sets: There has been significant research on identification and analysis of safety and security violations that occur in smart homes [4], [5], [20]. After reviewing prior work, we define the following six types of malicious activities:

1) Type 1: T/A safety violations
2) Type 2: Integrity/access control violations
3) Type 3: General security/conflicting actions/race condition violations
4) Type 4: Safety violations by malicious apps
5) Type 5: Insider attacks

We give examples of each type and details of how violations from prior work are reproduced for our testbed in Appendix B. In total, we develop 214 security violation instances with a breakdown as follows: Type 1 (114), Type 2 (40), Type 3 (40), Type 4 (10), and Type 5 (10).

Security Analysis: To evaluate safety provided by the SPL, we manually engineer each of the 214 malicious state transitions in random episodes of the RL environment to generate 21,400 *malicious episodes*. We then play these malicious episodes in the RL environment after the *learning episodes* of the SPL component are complete. We find that the SPL is able to flag 100% of all the malicious state transitions in the *malicious episodes*.

C. False Positives

We evaluate the SPL in terms of false positives resulting from benign user anomalies that are classified as malicious state transitions. We use data-sets [13] that have samples of anomalous activity collected from user studies. The participants of the studies were asked to define anomalous activities themselves and then perform additional simulations of these defined anomalies. These data-sets have activities for a period of one month in smart home A with interval size of one minute. We engineer instances of benign anomalous state transitions collected from [13] in random episodes after the *learning episodes* of the SPL are complete for smart home A to generate 18,120 *benign anomalous episodes*.

The benign anomalies are detected and filtered by the ANN used by the SPL. We find that 99.2% of the *benign anomalous episodes* are correctly classified by the ANN and the false positives are 0.8%. The ROC graph is shown in Figure 5.

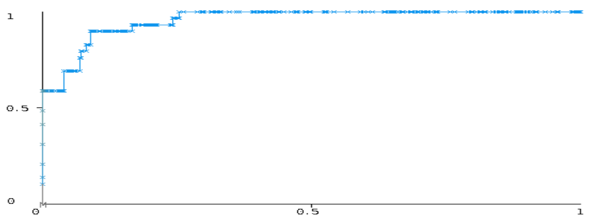

Fig. 5: ROC curve for filtering accuracy of the SPL

D. Functionality

For analysing the functionality optimizer component, we take $\kappa = 3$ hypothetical goals of the user: energy optimization, energy cost minimization, and ideal temperature maintenance. We define normalized reward functions for each as: F_0 - meter readings of power usage (energy usage), F_1 - electricity costs for power usage according to DAM (day-ahead-market) prices [21] (energy cost), and F_3 - temperature difference between day ahead forecasted temperature and HVAC readings. We manually define the values of the dis-utility reward functions (ω_i) for $k = 11$ devices such that total dis-utility and total utility rewards are equally balanced: $\chi = 1$ and $f_1 + f_2 + f_3 = 1$. This makes sure that the optimized actions never cause more dis-utility than functionality when traversing the environment.

We perform experiments for a range of rewards weights $f_1, f_2, f_3 \in [0.1, 0.9]$ for 30 random days from the dataset from [19]. For each day, `Jarvis` produces the optimal strategy or Q table based on the f_j values. We compare the normal user behavior with `Jarvis` optimized behavior; the results of the comparison are reported in Figures 6, 7, and 8 for each functionality, respectively. We can see that the optimized actions provide an advantage over normal behavior in the range from $f_j = 0.1$ to $f_j = 0.9$. The shaded region in the graphs is defined as the *safe benefit space* of the environment for each individual functionality. It is important to note the user may take some actions of the day manually and depend on `Jarvis` for other actions. In this case, `Jarvis` still suggests the best possible action from the *safe benefit space* for whichever state the environment has reached because of user actions.

E. Analysis of the Benefit Space

We can see that `Jarvis` learns to take 'smart' decisions by exploring the *safe benefit space*. With a variation of f_j and χ values, `Jarvis` is able to learn a variety of hypothetical environment specific ethics for state space exploration. For instance, configuration values $f_1 = 0.9, f_2 = 0.05, f_3 = 0.05$ result in `Jarvis` having highly energy usage conscious ethics. On the other hand, for values $f_1 = 0.2, f_2 = 0.2, f_3 = 0.6$, `Jarvis` learns more balanced ethics which emphasizes maintaining house temperature at the expense of energy usage and costs.

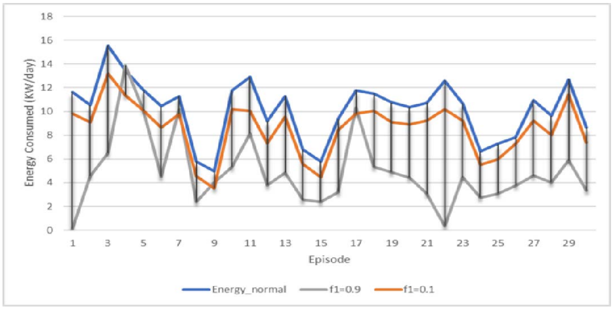

Fig. 6: Energy Conservation

Learning hypothetical human objectives or ethics to model the reward function is a hard problem because of the intractability of value of information to the user for a given environment. It is the focus of various active learning schemes [22], [23]. `Jarvis` provides an effective way of producing hypotheticals along the *safe benefit space* for each functionality defined by the user. The reward function model can then be learnt from user feedback on these hypothetical behaviors.

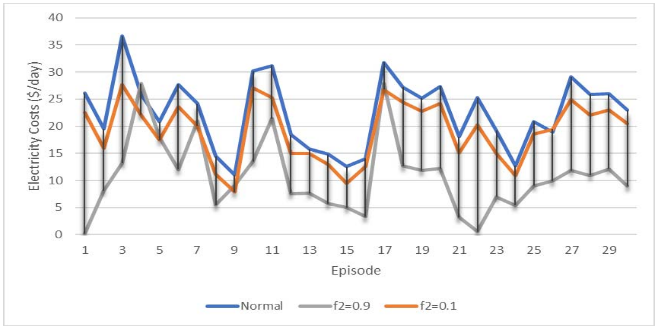

Fig. 7: Energy Price Minimization

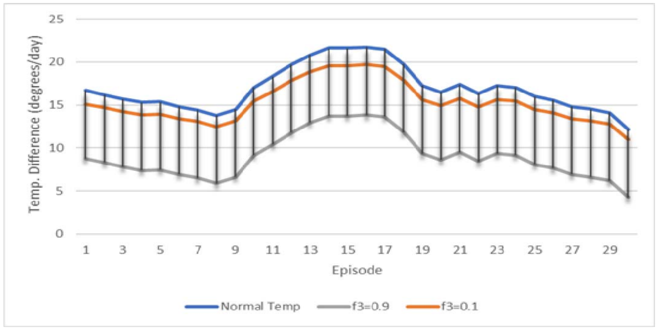

Fig. 8: Temperature Difference Optimization

F. Limitations of Unconstrained Exploration

In order to see the benefits and drawbacks of unconstrained exploration, we perform an experiment to compare the behavior of `Jarvis` with and without the SPL component. The results are reported in Figure 9. We can see that unconstrained exploration promises higher rewards but at the same time has an average of 32 safety violations per episode. We refer to the grey region of the graph as the overall *unsafe benefit space* of the environment (the overall *safe benefit space* is shown in orange). Since the SPL is probabilistic and prone to false positives, of the 32 violations, there might be some which are benign and should be allowed. In this regard, `Jarvis` has a bias towards 'safer' solutions based on Occam's razor [24]. This is a valid assumption in IoT environments especially in smart homes where user safety is paramount. However, there may be violations which are benign (i.e., false positives) or provide enough functionality benefits such that they are acceptable to the user in certain scenarios. User feedback on these actions in the *unsafe benefit space* can be used to correctly classify these actions as benign or malicious. Using such active learning methods to utilize the *unsafe benefit space* is a promising future direction of research.

VII. Related Work

Approaches for securing IoT environments are based on vetting apps by building static or dynamic smart app models [4]–[6], [11], [25]–[27]. The models are generated using a combination of device capabilities, user prompts, app permissions, and event subscriptions. After which, some form of forward symbolic execution [28] is used to explore all

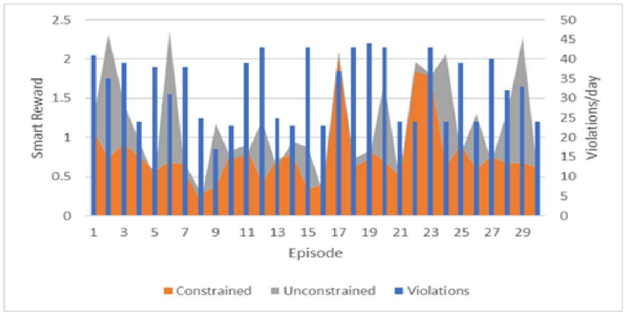

Fig. 9: Unconstrained vs Constrained Exploration Benefit Space

feasible paths in the built graphs. In order to build multi-app models, a set of the apps which share device events and graph causality are merged together to form a union of the app's state models. Then a general purpose model checking, like SPIN [29], is used to detect policy violations in the generated app models. These app-based modelling approaches are prone to state space and path explosion problems especially with large numbers of heterogeneous apps in terms of functionalities, dependencies, and attribute values. These systems work for small dynamic models built by limiting app-interactions, interleaving or using just a specific set of apps and devices triggered in a particular order. On the other hand, the `Jarvis` state model is constructed only from device states and actions and hence not affected by the number or variety of apps running in the environment. Also, the safety and/or security of the decisions is assessed from pre-defined policies set by users [4], [5]. Defining such policies for different devices and app configurations requires significant human effort. An approach for the automated learning of such safety policies in a smart home by observing natural behavior of the devices is proposed in [30]. Our system is similar in this regard as it learns security policies by observing prior T-A behavior during the learning phase.

The other drawback of those approaches is that they only detect unsafe/insecure states and do not consider the functionality requirements or goals of the user. Approaches have been proposed for building optimization models for individual functionality requirements like energy usage [31], [32], electricity prices [10], [33], and thermal comfort [34], [35] for smart homes and smart grids using various statistical and machine learning tools. In [8], a RL based system is used to manage the energy in a residential setting with a smart grid. For smart homes, a deep RL based framework is developed using energy and user discomfort as opposing rewards [7]. Other learning systems are used to maintain the state changes of power intensive HVAC, washing machines, industrial appliances etc. [36]. However, these systems are designed to have simplistic state models with specific set of devices, thermal dynamics and simple environments. These cannot be directly used for the complex and heterogeneous IoT environment with multiple apps and devices. Also, those approaches do not address security and safety.

VIII. Conclusion

In this paper we have proposed `Jarvis`, a RL framework for IoT environments that learns device actions to optimize user defined goals but whose exploration is constrained by dynamically identified safety and security policies. We have instantiated `Jarvis` for a smart home environment and have shown that it provides considerable benefits in terms of functionality and safety. We plan to extend the framework to other IoT environments like vehicular networks. Another interesting area of future work is using active learning methods to better utilize the *safe* and *unsafe benefit spaces*.

Acknowledgements

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Ifttt, "Ifttt." [Online]. Available: https://ifttt.com/

[2] Zapier, "The easiest way to automate your work." [Online]. Available: https://zapier.com/

[3] "Hybrid integration platform (hip)." [Online]. Available: https://apiant.com/

[4] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 USENIX Annual Technical Conference*, 2018, pp. 147–158.

[5] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot." in *NDSS*, 2019.

[6] S. Bauer and D. Schreckling, "Data provenance in the internet of things," in *EU Project COMPOSE, Conference Seminar*, 2013.

[7] D. O'Neill, M. Levorato, A. Goldsmith, and U. Mitra, "Residential demand response using reinforcement learning," in *First IEEE International Conference on Smart Grid Communications*. IEEE, 2010, pp. 409–414.

[8] L. Yu, W. Xie, D. Xie, Y. Zou, D. Zhang, Z. Sun, L. Zhang, and T. Jiang, "Deep reinforcement learning for smart home energy management," *arXiv preprint arXiv:1909.10165*, 2019.

[9] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for iot content-centric services," *Applied Soft Computing*, vol. 70, pp. 12–21, 2018.

[10] K. M. Tsui and S.-C. Chan, "Demand response optimization for smart home scheduling under real-time pricing," *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 1812–1821, 2012.

[11] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman, "Iotsat: A formal framework for security analysis of the internet of things (iot)," in *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2016, pp. 180–188.

[12] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[13] T. Alshammari, N. Alshammari, M. Sedky, and C. Howard, "Simadl: Simulated activities of daily living dataset," *Data*, vol. 3, no. 2, p. 11, 2018.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[15] "Smartthings." [Online]. Available: https://www.smartthings.com/

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[17] Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation OSDI*, 2016, pp. 265–283.
[18] N. Alshammari, T. Alshammari, M. Sedky, J. Champion, and C. Bauer, "Openshs: Open smart home simulator," *Sensors*, vol. 17, no. 5, p. 1003, 2017.
[19] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, J. Albrecht *et al.*, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, no. 112, p. 108, 2012.
[20] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 832–846.
[21] "Day-ahead market." [Online]. Available: http://www.ercot.com/mktinfo/dam
[22] S. Reddy, A. D. Dragan, S. Levine, S. Legg, and J. Leike, "Learning human objectives by evaluating hypothetical behavior," *arXiv preprint arXiv:1912.05652*, 2019.
[23] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.
[24] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information processing letters*, vol. 24, no. 6, pp. 377–380, 1987.
[25] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms." in *NDSS*, 2017.
[26] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *Proceedings of the 26th USENIX Security Symposium*, 2017, pp. 361–378.
[27] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 636–654.
[28] S. Khurshid, C. S. Păsăreanu, and W. Visser, "Generalized symbolic execution for model checking and testing," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2003, pp. 553–568.
[29] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.
[30] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Helion: Enabling a natural perspective of home automation," *arXiv preprint arXiv:1907.00124*, 2019.
[31] V. Pilloni, A. Floris, A. Meloni, and L. Atzori, "Smart home energy management including renewable sources: A qoe-driven approach," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 2006–2018, 2016.
[32] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied energy*, vol. 235, pp. 1072–1089, 2019.
[33] F. De Angelis, M. Boaro, D. Fuselli, S. Squartini, F. Piazza, and Q. Wei, "Optimal home energy management under dynamic electrical and thermal constraints," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1518–1527, 2012.
[34] L. Yu, T. Jiang, and Y. Zou, "Online energy management for a sustainable smart home with an hvac load and random occupancy," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 1646–1659, 2017.
[35] M. Franceschelli, A. Pilloni, and A. Gaspam, "A heuristic approach for online distributed optimization of multi-agent networks of smart sockets and thermostatically controlled loads based on dynamic average consensus," in *2018 European Control Conference (ECC)*. IEEE, 2018, pp. 2541–2548.
[36] M. Collotta and G. Pau, "An innovative approach for forecasting of energy requirements to improve a smart home management system based on ble," *IEEE Transactions on Green Communications and Networking*, vol. 1, no. 1, pp. 112–120, 2017.

Appendix

A. Benign User Anomaly Examples

1) Common benign anomalous user activities:
   a) Leaving Fridge Door Open
   b) Leaving oven on for a long time
   c) Leaving Main Door open
   d) Leaving lights on for a long time
   e) Leaving TV on
   f) Leaving wardrobe open

B. Safety Violation Examples

1) Type 1: Trigger-Action Application specific Safety Violations
   a) The main door is unlocked when user is not home
   b) The main door is unlocked when user is sleeping
   c) The lights of the room are off when motion is detected
   d) The lights of the room are off when a user unlocks the door
   e) The door is opened when there is no motion outside/inside the house
   f) Lights are on when the user is sleeping
   g) Appliances like oven,TV etc. are on when user is sleeping/away from home
   h) Appliances like fridge/security system/fire alarm are off
   i) Appliances interfering with each other eg: turning on heater when ac is on and vice-versa
2) Type 2: Integrity/Access Control Violations
   a) Unauthorized Users performing actions
   b) Users performing actions in unauthorized groups/locations
3) Type 3: General security/conflicting actions/race condition violations
   a) A single user performing multiple actions repeatedly resulting in conflicting states on a device
   b) Multiple users performing the same action on the device multiple times
   c) Multiple users performing conflicting actions on the device
   d) User performing an action which acts as a trigger to perform an action on the same device
4) Type 4: Malicious apps causing safety violations
   a) App turns off lights when motion is detected (e.g., to prevent brightening of the path outside the house)
   b) App turns lights/devices off after user specified time to save energy
   c) App turns lights to dim mode when user leaves home and door is unlocked after some time
   d) App turns on devices when user is home and turns them off when the user is not
   e) App 1 turns on lights when alarm sounds, App 2 assumes user is home when lights turn on and locks the door
   f) App turns off all devices when user is sleeping
5) Type 5: Insider attacks
   a) Heater is set to unsafe temperature and door is locked by insider
   b) Oven is left turned on for long time by insider when user is sleeping/not home
   c) Door is unlocked by an insider when no user is at home/sleeping

Constructing Malicious Data-sets: By simulating instances of these activities in smart home B, we generate *malicious* labelled state transitions. We build the malicious data-set for each type as follows. For type 1, we manually go through the trigger-action application specific security policies defined in [4], [5] and identify activities applicable to the devices in our testbed. For example, R.1 in [5] is applied to five devices in smart home A, R.14 is applied to one device in smart home B, and to sixteen devices in smart home A. In total, we identify 114 such violations by applying policies R1-R30 to our testbed. For type 2, we survey prior work on access control in IoT environments [25] to construct 40 unauthorized access instances which violate various security policies in smart homes A and B. For type 3, we construct violation instances from the general security (G1-G4) rules defined in [5]. For type 4, we use the malicious app interactions identified in [4] to build ten instances in smart home A. For type 5, we emulate common real world insider attacks resulting in thefts, breaches and safety issues to build ten instances.