Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation

Junyang Shi and Mo Sha, *State University of New York at Binghamton;*
Xi Peng, *University of Delaware*

https://www.usenix.org/conference/nsdi21/presentation/shi

This paper is included in the
Proceedings of the 18th USENIX Symposium on
Networked Systems Design and Implementation.
April 12–14, 2021

978-1-939133-21-2

Open access to the Proceedings of the
18th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by NetApp

Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation

Junyang Shi
State University of New York at Binghamton

Mo Sha
State University of New York at Binghamton

Xi Peng
University of Delaware

Abstract

Recent years have witnessed the rapid deployments of wireless mesh networks (WMNs) for industrial automation, military operations, smart energy, etc. Although WMNs work satisfactorily most of the time thanks to years of research, they are often difficult to configure as configuring a WMN is a complex process, which involves theoretical computation, simulation, and field testing, among other tasks. Simulating a WMN provides distinct advantages over experimenting on a physical network when it comes to identifying a good network configuration. Unfortunately, our study shows that the models for network configuration prediction learned from simulations cannot always help physical networks meet performance requirements because of the simulation-to-reality gap. In this paper, we employ deep learning based domain adaptation to close the gap and leverage a teacher-student neural network to transfer the network configuration knowledge learned from a simulated network to its corresponding physical network. Experimental results show that our method effectively closes the gap and increases the accuracy of predicting a good network configuration that allows the network to meet performance requirements from 30.10% to 70.24% by learning robust machine learning models from a large amount of inexpensive simulation data and a few costly field testing measurements.

1 Introduction

Recent years have witnessed rapid deployments of wireless mesh networks (WMNs) for industrial automation [38, 95], military operations [57], smart energy [80], etc. For instance, IEEE 802.15.4-based industrial WMNs, also known as wireless sensor-actuator networks (WSANs), are gaining rapid adoption in process industries over the past decade due to their advantage in lowering operating costs [51]. Battery-powered wireless modules easily and inexpensively retrofit existing sensors and actuators in industrial facilities without the need to run cables for communication and power. Industrial standard organizations such as HART [31], ISA [38], IEC [37], and ZigBee [104] are actively pushing the real-world implementations of WSANs for industrial automation. For example, more than 54,835 WSANs that implement the WirelessHART standard [95] have been deployed globally by Emerson Process Management to monitor and control industrial processes [24].

Although WMNs work satisfactorily most of the time thanks to years of research, they are often difficult to configure as configuring a WMN is a complex process, involving theoretical computation, simulation, and field testing, among other tasks. Simulating a WMN provides distinct advantages than experimenting on a physical network when it comes to identifying a good network configuration: a simulation can be set up in less time, introduce less overhead, and allow for different configurations to be tested under exactly the same conditions. Significant efforts have been made to investigate the characteristics of wireless communication in the literature. For instance, there has been a vast array of research that empirically studied the low-power wireless links with different platforms, under varying experimental conditions, assumptions, and scenarios [6]. Decades of research have gathered precious knowledge and produced a set of mathematical models that capture the characteristics of wireless links, interference, etc, and enable the development of wireless simulators, such as TOSSIM [44, 84], Cooja [17, 65], OMNeT++ [63, 89], and NS-3 [61].

However, it is still very challenging to date to set up a simulation that captures extensive uncertainties, variations, and dynamics in real-world WMN deployments. Our study shows that the models for network configuration prediction learned from simulations cannot always help physical networks meet performance requirements because of the *simulation-to-reality gap*; therefore the advantages of using simulations to reduce experimental overhead, improve flexibility, and enhance repeatability come at the expense of questionable credibility of the results. On the other hand, data collection from many WMN deployments, which include the ones in industrial facilities, is costly; therefore it is difficult to obtain sufficient information to train a good model or identify an optimal policy for network configurations by relying solely on field testing.

In this paper, we formulate the network configuration prediction into a machine learning problem, use the configurations of a WirelessHART network [95] as an example to illustrate the simulation-to-reality gap, and then employ deep learning based domain adaptation to close the gap. Specifically, this paper makes the following contributions:

- We present the simulation-to-reality gap in network configurations and show that the models for network configuration prediction learned from simulations cannot always help physical networks meet performance requirements;
- We develop a teacher-student neural network[1] that learns robust machine learning models for network configuration prediction from a large amount of inexpensive simulation data and a few costly physical measurements; to our knowledge, our work represents the first systematic study of the effectiveness of domain adaptation in closing the simulation-to-reality gap in network configurations;
- We implement our method, evaluate it using four simulators and a physical testbed, and repeat our evaluation with different network topologies under various wireless conditions. Experimental results show that our method can significantly improve the prediction accuracy and help physical networks meet performance requirements.

The remainder of our paper is organized as the following sections. Section 2 reviews the related work. Section 3 introduces the background of WirelessHART networks. Section 4 presents our problem formulation, our feature selection study, the simulation-to-reality gap, and our method that closes the gap. Section 5 shows the design of our teacher-student neural network. Section 6 evaluates our method. Section 7 concludes the paper.

2 Related Works

The current practices in network configurations rely largely on experience and rules of thumb that involve a coarse-grained analysis of network loads or dynamics during a few field trials. For example, the WirelessHART standard specifies the use of all available channels after a human operator manually blacklists noisy ones [95], and Emerson Process Management [22] recommends using a constant value (60% in general or 70% for control and high speed monitoring) as the packet reception ratio (PRR) threshold to select links for routing [23]. Unfortunately, recent studies show that such specifications are problematic, because using more channels or a fixed PRR threshold is not always desirable in WirelessHART networks [30, 75, 76]. In the literature, significant research efforts have been made to model the characteristics of wireless networks and optimize network configurations through mathematical techniques such as convex optimization [52], game theory [2], and meta heuristics [73]. For instance, the characteristics of low-power wireless links have been studied empirically with different platforms, under varying experimental conditions, assumptions, and scenarios [6]. Runtime adaptation methods have been developed to improve the performance of wireless sensor networks (WSNs) by adapting a few parameters in the physical and media access control (MAC) layers [20, 25, 70, 90, 105]. Those methods are not directly applicable to configure a network with many interplaying parameters.

As wireless deployments become increasingly hierarchical, heterogeneous, and complex, a breadth of recent research has reported that resorting to advanced machine learning techniques for wireless networking presents significant performance improvements compared to traditional methods. Deep learning has been used to handle a large number of network parameters and automatically uncover correlations that would otherwise have been too complex to extract by human experts [5, 14, 42, 54, 97, 101] and reinforcement learning has been employed to enable network self-configurations [18, 32, 36, 45, 47, 53, 56, 59, 60, 67, 72, 74, 91, 93, 96, 98–100, 103]. The key behind the remarkable success of those data-driven methods is the capability of optimizing a huge number of free parameters [33, 35] to capture extensive uncertainties, variations, and dynamics in real-world wireless deployments, which not only yield complex features, such as communication signal characteristics, channel quality, queuing state of each device, and path congestion situation, but also have many control targets, such as resource allocation, queue management, and congestion control.

However, data collection from many wireless deployments that are not easily accessible (e.g., the ones in industrial facilities) is costly; therefore it is difficult to obtain sufficient information to train a good model or identify an optimal policy for network configurations. In such scenarios, the benefits of employing learning-based methods that require much data are outweighed by the costs. Industry has consequently shown a marked reluctance to adopt them. To address this limitation, there has been increasing interest in using simulations to identify good network configurations [7, 40, 46, 49, 75, 76, 79, 82]. Unfortunately, our study shows that a straightforward deployment of a model learned from simulations results in poor performance in a physical network due to the simulation-to-reality gap.

Domain adaptation aims to learn from one or multiple source domains and produce a model that performs well on a related target domain; the assumption is that the source and

[1] To eliminate ambiguity, we use the word "network" to denote a wireless network and use the word "neural network" to represent a deep learning model in this paper.

target domains are associated with the same label space. It has been successfully used in computer vision [69, 92], natural language processing [66], and building occupancy estimation [3, 102]. Studies have shown that domain adaptation can mitigate the harmful effects of domain discrepancy by optimizing the representation to minimize some measures of domain shift, such as maximum mean discrepancy [13] or correlation distances [27]. Compared to fine-tuning the deep learning model, which is pre-trained using simulation data, employing domain adaptation is expected to close the gap between the simulated network (source) domain and the physical network (target) domain with fewer costly physical measurements. Recent work has focused on transferring deep neural network (DNN) representations from a labeled source dataset to a target domain where labeled data is sparse or non-existent. The main strategy is to guide feature learning via minimizing the difference between the source and target feature distributions. The maximum mean discrepancy (MMD) has been successfully used for domain adaptation, which computes the norm of the difference between two domain means [29, 86]. Several methods employed an adversarial loss to minimize domain shift and learned a representation that is simultaneously discriminative of source labels while not being able to distinguish between domains [19,26]. Despite the extensive literature on domain adaptation, little work has been done to investigate whether it can be applied to close simulation-to-reality gap in network configurations.

3 Background of WirelessHART Networks

To meet the stringent reliability and real-time requirements of industrial applications, WirelessHART networks [95] made a set of specific design choices that distinguish themselves from traditional WSNs designed for best effort services [51]. A WirelessHART network is managed by the centralized network manager, a software module, which is responsible for managing the entire network that includes generating routes, scheduling all transmissions, and selecting network parameters. Network devices include a set of field devices (sensors and actuators) and multiple access points. Each network device is equipped with a half-duplex omnidirectional radio transceiver compliant with the IEEE 802.15.4 standard [1].

WirelessHART networks adopt the time-slotted channel hopping (TSCH) technique [85], which combines time-slotted medium access, channel hopping, and multi-channel communication to provide time-deterministic packet deliveries[2]. Under TSCH, time is divided into 10*ms* time slots, each of which can be used to transmit a packet and receive an acknowledgment between a pair of devices. The network uses up to 16 channels in the 2.4 GHz ISM band and performs channel hopping in every time slot to combat narrow band interference. WirelessHART networks support two types of routing: source routing and graph routing. Source routing provides a single directed path from each data source to its destination. Graph routing is designed to enhance network reliability by providing redundant routes between field devices and access points. A packet may be transmitted through the backup routes if the links on the primary path fail to deliver it.

[2]Packets must be delivered along the data flow (from a sensor to an access point and then to an actuator) by the specified time deadline.

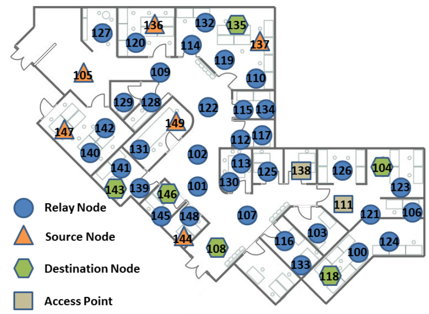


Figure 1: Device deployment on our testbed. The device ID ranges from 100 to 149.

4 Methodology

In this section, we first describe our experimental setup and data collection method. Then we formulate the network configuration prediction as a machine learning problem and present our feature selection study and the simulation-to-reality gap. Finally, we introduce our deep learning based domain adaptation method, which closes the gap.

4.1 Experimental Setup and Data Collection

We adopt the open-source implementation of WirelessHART networks provided by Li et al. [94] and configure six data flows on our testbed, which consists of 50 TelosB motes [81]. Figure 1 shows the device deployment on our testbed and Table 1 lists the source node (sensor), the destination node (actuator), the data generation interval (period), and the priority of each data flow. We employ the rate monotonic scheduling [48], an optimal fixed-priority policy, to generate the transmission schedule, set the data delivery deadline of each data flow to its period, and configure the devices with ID 111 and 138 to serve as two access points.

We consider three configurable network parameters, which include (i) the PRR threshold for link selection R, (ii) the number of channels used in the network C, and (iii) the number of transmission attempts scheduled for each packet A, and three network performance metrics, which include (1) the end-to-end latency L, (2) the battery lifetime B, and (3) the end-to-end reliability E. We consider $R \in \{0.7, 0.71, 0.72, ..., 0.90\}$[3], $C \in \{1,2,3,4,5,6,7,8\}$, and

[3]Emerson Process Management [22] recommends using a constant value

Table 1: Data flows.

Flow ID	Source	Destination	Period (ms)	Priority
1	147	146	500	1
2	144	143	500	2
3	105	104	500	3
4	149	118	1000	4
5	136	135	1000	5
6	137	108	1000	6

$A \in \{1,2,3\}$ as the possible parameter values, and combine them to create 744 $(31*8*3)$ network configurations. Please note that some network configurations make the network manager generate the same routes and transmission schedule. After removing all redundancy (the configurations leading to the same routes and transmission schedule), there are 88 distinct network configurations left under our experimental setup.

After deploying the data flows on the testbed, we implement the same network in the simulator[4], feed the PRR and noise traces, the routes, and the transmission schedule collected from the physical network into the simulator, and then run simulations to evaluate network performance under each network configuration. Specifically, the simulator generates simulated L, B, and E values under each network configuration (R,C,A). The network performance (L, B, and E values) is computed in every $50s$. 75 network performance traces are collected under each network configuration. In total, we collect 6,600 $(88*75)$ data traces from simulations. Then, we run experiments on our testbed and measure the network performance under each network configuration. Similarly, we collect 6,600 data traces from our testbed. The data gathered from the simulated network and the physical network is denoted as $\mathcal{D}^s$ and $\mathcal{D}^p$, respectively.

4.2 Network Configuration Prediction

The primary task in network configurations is to select the configuration (the selections of parameters R, C, and A), which allows the network to meet the performance requirements (L, B, and E) specified by the application. The parameter selection should be as *accurate* as possible with *minimal data collection overhead*. We formulate the network configuration prediction task as a machine learning problem. Let $\mathbf{x} = concatenation(L,B,E)$ denote the given network performance requirements and $\mathbf{y} = concatenation(R,C,A)$ denote the configuration, which allows the network to meet performance requirements. The goal is to learn a nonlinear mapping $f_\theta(\cdot) : \mathbf{x} \rightarrow \mathbf{y}$. Based on the specific application, the user can set the performance requirements ($\mathbf{x}$). The input features in $\mathbf{x}$ are selected by the feature selection study in Section 4.3.

(0.6 in general or 0.7 for control and high speed monitoring) for R [23]. We did not consider R lower than 0.7 because of the consistent low reliability we observed.

[4] We repeat our experiments using four simulators: TOSSIM, Cooja, OMNeT++, and NS-3.

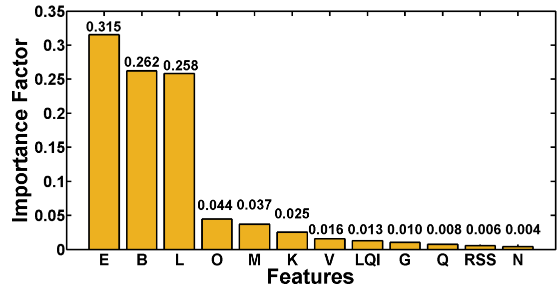


Figure 2: Importance factors of different features when using tree-based feature selection method. Under the tree-based method, the features that are selected at the top of the trees are in general more important than the features that are selected at the end nodes of the trees, as generally, the top splits lead to bigger information gains. We use the normalized importance factor generated by the tree-based method as a metric for feature selection.

We use θ to denote the model parameters that are learned from data in a data-driven manner. Given the fact that the network configuration values ($\mathbf{y}$) can be discretized without losing the generality, we further restrict f_θ as a discriminative model to solve a classification problem: an application can set its performance requirements ($\mathbf{x}$), and the classifier (f_θ) will predict the network configuration ($\mathbf{y}$) to satisfy the application requirements. This data-driven learning-based model can take advantage of a large amount of data to consistently improve its performance. Experimental results (See Section 6.2) show that it significantly outperforms traditional optimization-based methods such as Response Surface Methodology (RSM) [9] and Kriging surrogate modeling approach [78]. The latter usually suffers the issues that include limited predictive power and being vulnerable to uneven data distribution [15].

4.3 Feature Selection

In addition to the features (L, B, and E) that represent performance requirements, we consider nine other features, which include the received signal strength RSS [8], the link quality indicator LQI [6], the background noise G [6], the packet delay variation O, the power consumption variation K, the network reliability variation M, the received signal strength variation V, the link quality indicator variation Q, and the background noise variation N. Using all features that are relevant to the network configuration prediction problem may not necessarily achieve the best performance but rather increases computational cost and data collection overhead. We perform a study that employs three classic feature selection methods (the tree-based method [50], the univariate feature selection method [39], and the recursive feature elimination

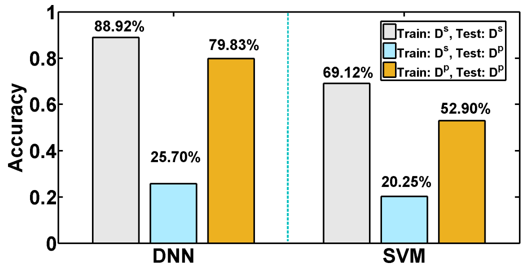


Figure 3: Modeling accuracy when model is trained and tested on different data sets ($\mathcal{D}^s$: the simulation data produced by OMNeT++ and $\mathcal{D}^p$: the physical data). The difference between the grey bar and the blue bar indicates the simulation-to-reality gap.

method [16]) to pick the most useful features. Figure 2 plots the importance factors of different features when we use the tree-based method. As Figure 2 shows, L, B, and E have much higher importance factors (0.315, 0.262, and 0.258) than the rest. Similar results are observed when we use other methods. Therefore, we use L, B, and E as the input features for WirelessHART networks. Please note that our method can accept more features for other networks.

4.4 Simulation-to-reality Gap

Our goal is to learn a classifier to predict network configurations on the physical data. However, it is a nontrivial task to learn the model from either the physical data ($\mathcal{D}^p$) or the simulation data ($\mathcal{D}^s$). Instead, we propose to use both $\mathcal{D}^p$ and $\mathcal{D}^s$ to learn the model as explained in the next section.

Using only physical data ($\mathcal{D}^p$): This would result in significant time and energy consumption due to the costly data collection process. We first leverage the physical data ($\mathcal{D}^p$) collected from the physical network to train machine learning models and explore its feasibility for our network configuration prediction problem. We employ two machine learning models, DNN and support vector machine (SVM), for classification. The input to the models is network performance requirements and the output is network configurations. We normalize the collected data ($\mathcal{D}^p$) into the $[0,1]$ range and split it randomly for training and testing. The yellow bars in Figure 3 show the modeling accuracy[5], when DNN and SVM models are used for the network configuration prediction, respectively. Both DNN and SVM models trained based on the physical data can provide high modeling accuracy when we test the models on the physical data (DNN: 79.83% and SVM: 52.90%), as the yellow bars show. This justifies the feasibility of our proposed machine learning method in Section 4.2 for the network configuration prediction and we may use the measurements collected from the physical network to train a good model. Unfortunately, relying on running experiments on a physical network to explore the configuration parameter space is impractical in many cases because running experiments on a physical network is very costly and time-consuming. The left side of Table 2 shows the modeling accuracy, data collection time, and device energy consumption when we train the DNN model with different sizes of the physical data (collected from a physical network). The modeling accuracy increases significantly from 19.39% to 79.83% with the size of the training set ($\mathcal{D}^p$) that increases from 88 traces to 3,960 traces. However, the time spent on collecting the training data ($\mathcal{D}^p$) increases from 1.22*hours* to 55.00*hours*. Moreover, the energy consumed by each field device for data collections on average increases from 310.61J to 13,974.26J, which represents 0.73% and 32.73% of its total energy capacity.

Using only simulation data ($\mathcal{D}^s$): This would result in low modeling accuracy due to the simulation-to-reality gap. The simulation data can be quickly and cheaply obtained from a simulator. As the right column of Table 2 show, the time spent on generating the simulation data varies from 27.41s to 1,231.40s and no energy is consumed by any field devices. However, a classifier that is trained based on the simulation data ($\mathcal{D}^s$) may suffer the following issue when applied on the physical data. As the grey bars in Figure 3 show, both models provide high modeling accuracy when we train based on the simulation data ($\mathcal{D}^s$) and test the models on the simulation data (DNN: 88.92% and SVM: 69.12%). However, the modeling accuracy drops rapidly when we test the models on the physical data ($\mathcal{D}^p$) as shown in blue bars (DNN: 25.70% and SVM: 20.25%). The differences on the modeling accuracy (DNN: 63.22% and SVM: 48.87%) clearly show the effect of the simulation-to-reality gap, a subtle but important discrepancy between reality and simulation that prevents the simulated experience from directly enabling effective real-world performance [12, 77]. The simulation-to-reality gap exists in network configurations because the theoretical models adopted by the simulator cannot capture all real-world performance-related factors. For example, the prerecorded noise traces and the probability-based prediction on packet reception cannot precisely capture the effects of packet failures caused by extensive uncertainties, variations, and dynamics in real-world wireless deployments (see Section 6.5). We observed similar discrepancy gaps when using Cooja, TOSSIM, OMNeT++, and NS-3. Because of the simulation-to-reality gap, the machine learning models trained based on simulation data ($\mathcal{D}^s$) for network configurations, no matter how large the data volume is, may not generalize well to a physical network.

4.5 Close the Gap by Domain Adaptation

The observations presented in Section 4.4 motivate us to explore the feasibility of using a substantial amount of inexpen-

[5] The modeling accuracy is defined as, given a set of input network performance requirements (L,B,E), the percentage of the testing set that a model can select the network configuration (R,C,A), which allows the network to meet performance requirements.

Table 2: Modeling accuracy (%), data collection time (s), and device energy consumption (J) when using the physical data ($\mathcal{D}^p$) or the simulation data ($\mathcal{D}^s$) produced by OMNeT++ for training. For comparison, our solution achieves 70.24% accuracy with only 440 data samples which are collected in 22,000s with 1,502.88J of energy (see Section 6.2).

# of Data Samples Used for Training	From a Physical Network (Train: $\mathcal{D}^p$, Test: $\mathcal{D}^p$)			From Simulations (Train: $\mathcal{D}^s$, Test: $\mathcal{D}^p$)		
	Accuracy (%)	Collection Time (s)	Energy (J)	Accuracy(%)	Collection Time (s)	Energy (J)
88	19.39	$4.40 * 10^3$	310.61	6.52	27.41	0
528	42.16	$2.64 * 10^4$	1,863.53	13.70	163.09	0
968	57.92	$4.84 * 10^4$	3,416.34	17.69	301.95	0
2,024	67.68	$1.01 * 10^5$	7,143.11	20.17	633.11	0
3,080	78.82	$1.54 * 10^5$	10,869.61	22.44	933.99	0
3,960	79.83	$1.98 * 10^5$	13,974.26	25.70	1,231.40	0

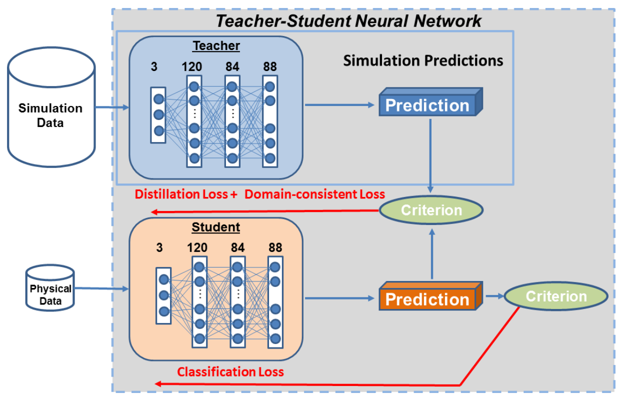


Figure 4: Our teacher-student neural network.

sive simulation data together with a small amount of costly physical data to train the model for network configuration prediction. To this end, our objective narrows down from solving a classification problem to using domain adaptation to address the domain discrepancy issue. Specifically, we first gather N^s data tuples by running simulations (source domain) and then acquire N^p data tuples by conducting experiments on the physical network (target domain). We assume $N^s \gg N^p$ due to the significant data collection overhead on the physical network (See Section 4.4). We assume that the source and target domains are characterized by different probability distribution q_1 and q_2, respectively. Our goal is to construct a deep learning model that can learn transferable features that bridge the cross-domain discrepancy and build a classifier $\mathbf{y} = f_\theta(\mathbf{x})$, which can maximize the target domain accuracy ($f_s \to f_p$) by using a small amount of physical data (N^p). The detailed design of our teacher-student neural network will be discussed in Section 5.

5 Teacher-Student Neural Network

In this section, we present our teacher-student neural network for domain adaptation. Our goal is to build a classifier that can maximize the target domain (physical network) accuracy by using a small amount of physical data (N^p) and adequate simulation data (N^s) where $N^s \gg N^p$ due to the significant data collection overhead (See Section 4.4) on the physical network. The teacher and student use independent parameters and the teacher generates the soft labels [4, 34] to transfer its knowledge to the student. Figure 4 shows our teacher-student neural network. The first stream (teacher) operates on the simulation data and the second stream (student) operates on the physical data. Classification loss, distillation loss, and domain-consistent loss are used in the training process for the student.

5.1 Teacher Neural Network

The teacher takes advantage of the large amount of simulation data for training and the training data ($\mathcal{D}^s$) consists of a total number of N^s data tuples. We follow Multilayer Perceptron (MLP) [71] to design the architecture of three layers: 120 and 84 neurons in the first two hidden layers, and 88 neurons in the output layer to represent the totally 88 distinct configuration categories. Rectified linear unit (ReLU) and softmax are employed to activate the hidden and output layers, respectively. The teacher's parameters (θ_1) are learned by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta_1) = -\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^s} \mathbf{y} log(f_{\theta_1}(\mathbf{x})), \quad (1)$$

where $\mathcal{D}^s$ denotes the training data generated from simulations, θ_1 denotes the teacher's parameters, y denotes the one-hot label, and $f_{\theta_1}(\mathbf{x})$ is the prediction made by the teacher. We use the Adam optimizer [41] with a learning rate of 0.01 to optimize the parameters of the teacher. A total number of 100 training epochs with a batch size of 128 have been used to train the neural network.

5.2 Student Neural Network

We train the student based on the N^p physical data with the help of the teacher. The student can be quickly learned using only a few shots of physical data ($N^p \ll N^s$). To achieve this, we leverage the teacher to facilitate the training of the student where knowledge is transferred from the simulation domain to the physical domain. The student shares the same architecture with the teacher but uses independent parameters. ReLU and softmax are employed to activate the hidden and output layers, respectively. The student's parameters (θ_2) are learned by minimizing the following loss:

$$\mathcal{L}(\theta_2) = \mathcal{L}_{cls} + \alpha\mathcal{L}_{dis} + \beta\mathcal{L}_{mmd} \tag{2}$$

where α, and β are weights. We empirically set $\alpha = 1$, and $\beta = 0.2$ which can provide good performance.

Classification loss $\mathcal{L}_{cls}$: This loss function allows the student to learn from the limited (N^p) physical data through employing the cross-entropy loss:

$$\mathcal{L}_{cls} = -\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^p} \mathbf{y} log(f_{\theta_2}(\mathbf{x})), \tag{3}$$

where $\mathbf{y}$ is the one-hot label and $f_{\theta_2}(\mathbf{x})$ is the prediction made by the student.

Distillation loss $\mathcal{L}_{dis}$: This loss function allows the teacher to transfer its knowledge to the student. The generalization ability of the student can be enhanced by the loss generated by the soft labels, which carry the information of probability distribution for each class [4, 34]. To compute $\mathcal{L}_{dis}$ with soft labels, we use the following formula:

$$\mathcal{L}_{dis} = -\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^s} \mathbf{q} log(f_{\theta_2}(\mathbf{x})), \tag{4}$$

where $f_{\theta_2}(\mathbf{x})$ is the prediction made by the student and $\mathbf{q}$ is the tempered softmax probability generated by the teacher. $\mathbf{q}$ is computed by:

$$\mathbf{q} = \frac{exp(z_i/T)}{\sum_j^k exp(z_j/T)} \tag{5}$$

where T is the temperature [34] and z_i is the pre-softmax output of the teacher. When T increases, the soft label q approaches a uniform distribution and the probability distribution generated by the softmax function becomes softer, which provides more information as to which class the teacher finds more similar to the predicted class, instead of giving a hard prediction that indicates which class is correct. We set $T = 10$ to generate soft labels for the student.

Domain-consistent loss $\mathcal{L}_{mmd}$: This loss function is employed to achieve domain-consistent representations between the source and target domains. Matching the distributions in the original input feature space is not suitable because some features may have been distorted by the domain shift. The key idea of domain-consistent regularization is to align two domains, the target (physical data) and the source (simulation data), in a latent embedding space. Our method uses the MMD [21] to achieve this goal. MMD is a hypothesis test that tests whether two samples are from the same distribution by comparing the means of the features after mapping them to a Reproducing Kernel Hilbert Space (RKHS) [68]. We calculate the loss as:

$$\mathcal{L}_{mmd} = || \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^s} f_{\theta_1}(\mathbf{x}) - \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^p} f_{\theta_2}(\mathbf{x}) || \tag{6}$$

where $f_{\theta_1}(\cdot)$ and $f_{\theta_2}(\cdot)$ denote the pre-softmax output of the teacher and the student, respectively. We use a learning rate of 0.01 with the stochastic gradient descent (SGD) optimizer to train the student. The momentum is set to 0.05 and the weight decay parameter is set to 0.003, which governs the regularization term of the student. A total number of 500 epochs have been trained on the student.

Table 3: Training and testing setups of different methods.

Method	Training		Testing	
	Physical Data	Simulation Data	Physical Data	Simulation Data
TPTP	√	×	√	×
TSTP	×	√	√	×
FT	√	√	√	×
CCSA	√	√	√	×
DaNN	√	√	√	×
RSM	√	√	√	×
Kriging	√	√	√	×
Ours	√	√	√	×

6 Evaluation

We perform a series of experiments to validate the efficiency of our method to identify good network configurations. We first evaluate the capability of our method to effectively improve the modeling accuracy and compare our method against seven baselines, which include five machine learning based methods: (i) Using the physical data for both training and testing (TPTP); (ii) Using the simulation data for training and the physical data for testing (TSTP) [75, 76]; (iii) Fine-tuning (FT) method [83]; (iv) CCSA: Unified deep supervised domain adaptation and generalization [58]; and (v) Domain adaptive neural network (DaNN) [28], and two non-machine learning methods: (vi) RSM method [9, 87] and (vii) Kriging method [11, 78]. Table 3 summarizes the training and testing data used by each method. All methods use L, B, and E as their input features. We then apply the network configurations selected by our method on our testbed and measure the network performance. We repeat our experiments with different network setups under various wireless conditions. Finally, we evaluate the effects of our method on closing the gap when employing different simulators and radio models.

6.1 Experimental Setup

As presented in Section 4.1, we configure six data flows on our testbed. On each data flow, sensor data is generated by

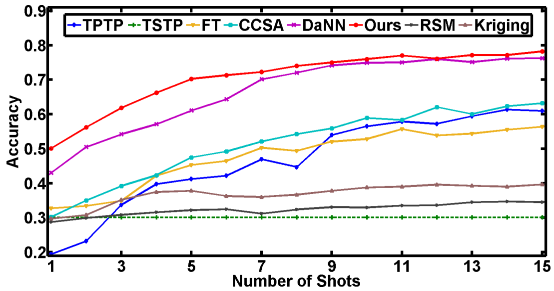


Figure 5: Modeling accuracy of our method and baselines when different number of shots of physical data are added into simulation data (3,960 data samples in total) for training. One shot includes 88 data samples (one data sample under each network configuration).

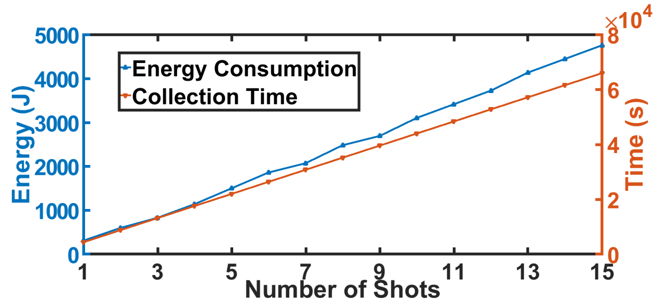


Figure 6: Time and energy consumption to collect different number of shots of data from a physical network. Using only physical data to train the model is infeasible due to unacceptable time and energy overhead.

the source node and forwarded to the access points (uplink) and then a corresponding control command is sent to the destination node (downlink). We calculate the latency, energy consumption, and reliability every 50*s*. We employ the same DNN architecture for the teacher and the student in our method with independent weights (see Section 5). Each neural network has 120 and 84 neurons in the first two hidden layers, and 88 neurons in the output layer. The weight β of MMD is 0.2 and the temperature T is 10. The learning rate is 0.01 with the SGD optimizer for the student. CCSA uses the cross-entropy loss and the semantic alignment loss between the source and target domains with the Siamese architecture. DaNN uses the standard classification loss and the MMD regularization for classification and domain adaptation. FT first uses the simulation data to train the initial model and then fine-tunes the neural network parameters to fit the target domain using a small amount of physical data. FT uses the learning rate of 0.001 to tune the parameters of the last layer in the DNN with the physical data. RSM and Kriging methods use simulation data and different amount of physical data to build RSM and Kriging models and use them to predict network configurations. Specifically, RSM is a black-box modeling technique and uses polynomial functions to approximate the model functions between the inputs and the outputs [9], while Kriging leverages spatial interpolation that uses complex mathematical formulas to estimate values at unknown points based on the values, which are already sampled [78].

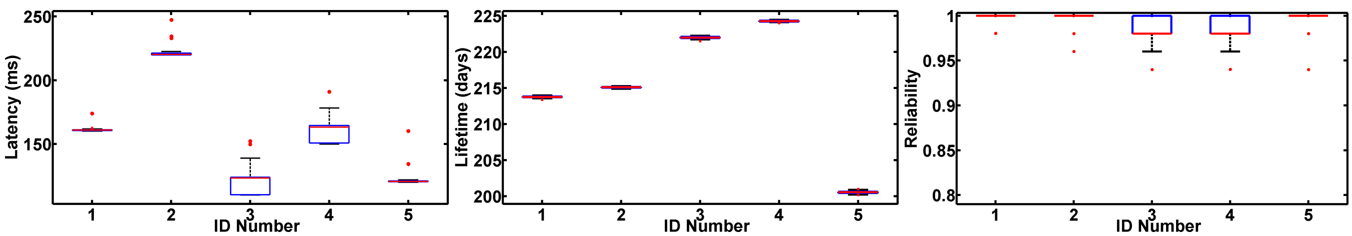
6.2 Performance of Our Method

We first evaluate the modeling accuracy of our method and compare its performance against seven baselines using the data traces presented in Section 4.1. 3,960 data samples from the simulation data are used for training under all methods except TPTP, which uses only the physical data for training. Figure 5 plots the modeling accuracy of all methods when different number of shots of physical data are added into the simulation data for training. As Figure 6 plots, collecting one shot of physical data (one data sample under each of 88 network configurations) takes 1.22 hours and consumes 310.61*J* of energy. Please note that TSTP uses only the simulation data for training (see Table 3) and provides the lowest accuracy (30.10%) due to the simulation-to-reality gap. The results clearly show that the model trained with the simulation data does not work well on the physical data. RSM and Kriging also provide poor performance with the maximum accuracy of 35.06% and 46.87%, respectively. Our method achieves the best performance. With only one shot of physical data (88 data samples), our method provides an accuracy of 50.12%. With four more shots of physical data, our method hits 70.24% accuracy. Using a small amount of physical data to provide a good model represents an important feature of our method because the data collection from a physical network is very time and energy consuming. As a comparison, without using the simulation data, TPTP provides only an accuracy of 19.39% and 41.21% at one shot and five shots, respectively. This highlights the importance of learning knowledge from simulations and transferring it to a physical network for network configurations.

We also observe that the accuracy improves slowly from 70.24% to 78.25% when the number of shots increases from 5 to 15. However, collecting 10 more shots of physical data from a physical network takes a long time and consumes much energy. As Figure 6 plots, the collection of five shots of physical data takes 6.11*hours* and consumes 1,502.88*J* of energy, while collecting 15 shots take 18.33*hours* and consumes 4,758.70*J* of energy. The improvement on the modeling accuracy is largely shadowed by the significantly increased data collection overhead. Therefore, we use five shots in the rest of our evaluation. Figure 5 and 6 also show that only using physical data to train the model is inefficient. It takes 18.33*hours* to collect enough data from a physical network, which allows TPTP to provide an accuracy of 60.95%. By comparing the accuracy provided by our method and TPTP, we can clearly see the effectiveness of our method on reducing the data collection time for training good mod-

Table 4: Six example network configurations selected by our method and TSTP. Figure 7 and 8 show the network performance after applying the configurations selected by our method and TSTP on our testbed, respectively. Our method can meet all performance requirements. The performance requirements that TSTP fails to meet are highlighted.

ID #	Input			Output (our method / TSTP)		
	Latency (*ms*)	Battery lifetime (*days*)	Reliability (%)	PRR threshold (%)	# of Channel	# of Tx Attempts
1	**170**	**210**	98	84 / 82	4 / 7	3 / 3
2	225	**214**	97	90 / 88	5 / 1	3 / 3
3	**130**	**220**	95	84 / 78	4 / 8	2 / 3
4	**165**	**224**	95	90 / 89	4 / 6	2 / 2
5	**130**	200	**98**	87 / 72	2 / 1	3 / 2



(a) Boxplot of latency. (b) Boxplot of battery lifetime. (c) Boxplot of reliability.

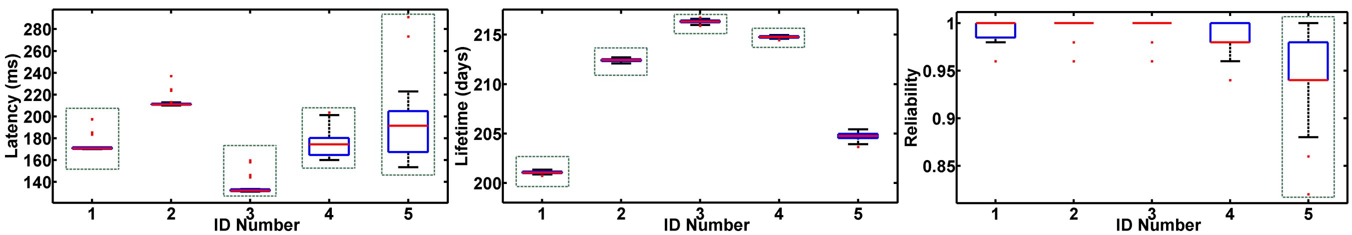
Figure 7: Network performance when employing the network configurations selected by our method (listed in Table 4). Central mark in box indicates median; bottom and top of box represent the 25th percentile (q_1) and 75th percentile (q_2); red dots indicate outliers ($x > q_2 + 1.5 * (q_2 - q_1)$ or $x < q1 - 1.5 * (q_2 - q_1)$); whiskers indicate the range that excludes outliers.

els for network configuration prediction. Our method consistently outperforms those two existing domain adaptation methods (DaNN and CCSA), which use the Siamese DNN model with different distance loss functions. For example, our method provides an accuracy of 70.24% when it uses five shots of physical data for training, while CCSA and DaNN provide 47.46% and 61.07% accuracy, respectively. The accuracy provided by FT increases from 32.73%, to 33.42%, and then to 56.40% when the number of shots increases from 1, to 2, and to 15 shots.

Our method can consistently outperform the baselines because it not only uses two different neural networks to learn two specific models for different but highly related domains with the soft labels but also employs the MMD regularization, while both DaNN and CCSA use same weights between the source and target domains for domain adaptation. Moreover, the distillation loss $\mathcal{L}_{dis}$ of our method provides a set of candidate network configurations for the student to choose and the student can quickly adapt to the target domain. The results also show that the domain-consistent loss, as a distribution distance measure, is effective for eliminating domain divergence between the source domain (simulated network) and the target domain (physical network). Our method also significantly outperforms FT. The low accuracy provided by FT shows that changing only the weight of the last layer in the DNN cannot produce a good adapted model.

We further validate the network configurations selected by our method on our testbed by examining the actual network performance. Specifically, we feed different network performance requirements to our method, employ the selected network configurations, and then measure the network performance. We repeat the experiments under each network configuration 108 times. Table 4 lists six example network configurations selected by our method and TSTP when facing different network performance requirements. Figure 7 plots the boxplots of latency, battery lifetime[6], and reliability when employing six network configurations selected by our method. As Figure 7 shows, our method always helps the network meet the network performance requirements posed by the application (listed in Table 4). For instance, the latency, battery lifetime, and reliability requirements are 170*ms*, 210*days*, and 98% in the first example ($ID = 1$). When employing the network configuration selected by our method (84% as PRR threshold, four channels, three transmission attempts for each packet), the network achieves a median latency of 161.00*ms*, a median battery lifetime of 213.76*days*, and a median reliability of 100%, which meet all given requirements. Similarly, the latency, battery lifetime, and reliability requirements are 165*ms*, 224*days*, and 95% in the fourth example ($ID = 4$). When employing the

[6]To compute the battery lifetime, we assume that each field device is powered by two Lithium Iron AA batteries with a total capacity of 42,700J. We compute the radio energy consumption based on the timestamps of radio activities and the radio's power consumption in each state according to the radio chip data sheet.



(a) Boxplot of latency.
(b) Boxplot of battery lifetime.
(c) Boxplot of reliability.

Figure 8: Network performance when employing the network configurations selected by TSTP (listed in Table 4). The dotted boxes highlight the network performance that fails to meet the requirements. Compared to Figure 7, our method always provides better network configurations than TSTR and help the network meet the application performance requirements.

network configuration selected by our method (90% as PRR threshold, four channels, two transmission attempts for each packet), the network achieves a median latency of 163.33ms, a median battery lifetime of 224.28$days$, and a median reliability of 98%, which meet all given requirements. Larger variations on latency are observed when the number of transmission attempts for each packet is small, which confirms the observations reported in our previous study [75, 76]

As a comparison, we also employ the network configurations selected by TSTP when facing the same network performance requirements. Table 4 lists the network configurations selected by TSTP and Figure 8 plots the resulting network performance. Due to the simulation-to-reality gap, the network configurations selected by TSTP cannot always meet all network performance requirements. The dotted boxes in Figure 8 highlight the network performance that fails to meet the application requirements listed in Table 4. For instance, the latency, battery lifetime, and reliability requirements are 130ms, 200$days$, and 98% in the fifth example ($ID = 5$). When employing the network configuration selected by TSTP (72% as PRR threshold, one channel, two transmission attempts for each packet), the network achieves a median latency of 191.40ms, a median battery lifetime of 204.74$days$, and a median reliability of 94.00%, which fail to meet the latency and reliability requirements.

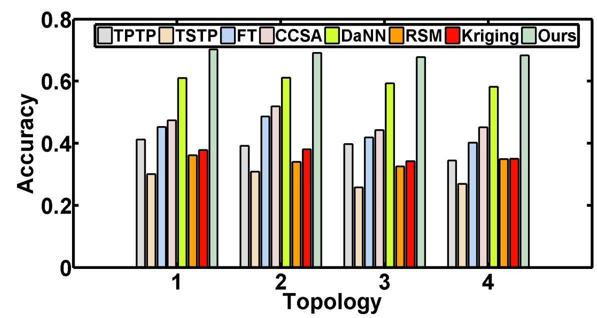


Figure 9: Accuracy comparison among different methods with different network topologies. All methods use five shots of physical data. Topology 1 is used for Figure 5.

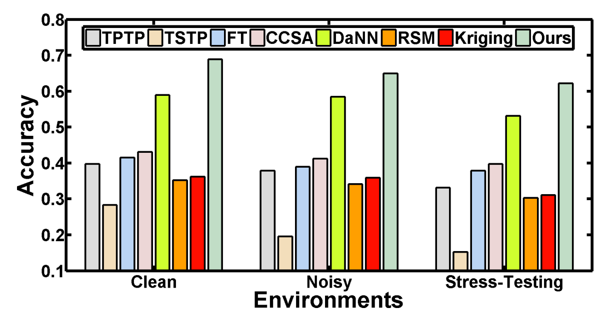


Figure 10: Accuracy comparison among different methods under different wireless conditions.

6.3 Performance with Different Network Topologies under Various Wireless Conditions

To examine the applicability of our method, we repeat our experiments with different network topologies under various wireless conditions. We first vary the number of data flows, the number of devices in the network, and the locations of source nodes, destination nodes, and access points and measure the performance of our method. Figure 9 plots the accuracy comparisons between our method and seven baselines under four example network topologies. Our method consistently provides the highest accuracy. For instance, our method achieves an accuracy of 67.09% under the third network topology, while CCSA and DaNN provide 44.23% and 59.37% accuracy, respectively. TPTP, TSTP, FT, RSM, and Kriging achieve 39.72%, 25.78%, 41.90%, 32.56%, and 34.26% accuracy, respectively. The results confirm the improvements presented in Section 6.2 and show our method can consistently outperform the state of the art.

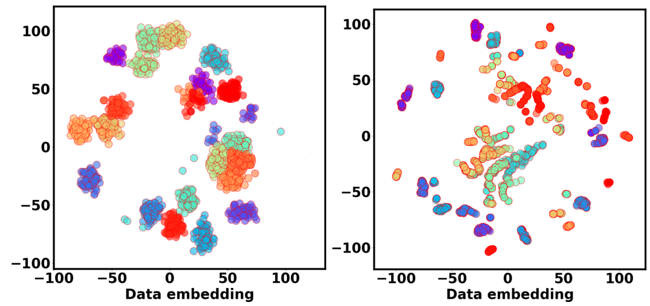
We also examine the performance of our method under different wireless conditions. We set up three jammers on our testbed (ID 116, 131, and 134 in Figure 1) and run Jamlab [10] on them to generate controlled WiFi interference with various strengths. We create three wireless conditions: a clean environment without controlled interference, a noisy environment with moderate controlled interference, and a stress-testing environment with strong controlled interference. We train the model again with different physical data under different wireless conditions. Figure 10 plots the modeling accuracy under three wireless conditions when employing our method and seven baselines. As Figure 10 shows, the accuracy provided by our method decreases from 68.89%, to 64.99%, and then to 62.20% when stronger interference is introduced. We observe similar trends when employing other methods.

This exposes a limitation of current wireless simulators, which cannot precisely simulate the effects of external interference and environmental dynamics. To better understand the physical data distribution, we visualize the data distribution of (L, B, E) collected from the physical data $(\mathcal{D}^p)$ using the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm [88], a dimension reduction tool for data visualization. Figure 11 shows the network performance visualization provided by t-SNE where different colors stand for different network configurations. Figure 11(a) and Figure 11(b) plot the data distributions when the network operates with and without the presence of strong controlled interference, respectively. Please note that those two figures include the same amount of data points. Many data points in Figure 11(b) overlap each other. These larger variations, result from the interference, significantly increases the difficulty on transferring knowledge learned from simulations to a physical network. With the presence of interference, our method still consistently outperforms all baselines. For instance, in the stress-testing environment, our method provides an accuracy of 62.20%, while other methods provide up to 53.21% accuracy.

To illustrate the differences between physical data and simulation data, Figure 12 plots the reliability measured from the physical network and simulated by TOSSIM under four network configurations. Because of the simulation-to-reality gap, the measured reliability is different from the simulated one. More importantly, the variations of the measured reliability values are much larger than the simulated ones. Such differences highlight the important of our method, which effectively closes the gap and increases the accuracy of predicting a good network configuration that allows the network to meet performance requirements.

6.4 Effects of Different Losses

To study the effects of different losses on the performance of our method, we repeat the experiments by disabling one or two losses among the classification loss $\mathcal{L}_{cls}$, the distillation loss $\mathcal{L}_{dis}$, and the domain-consistent loss $\mathcal{L}_{mmd}$. We conduct our experiments using Topology 1 in Figure 9 in a clean environment. Figure 13 plots the accuracy when our method uses different combination of loss functions. As Figure 13 shows, our method with a single loss provides very low classification accuracy ($\mathcal{L}_{dis}$: 28.22%, $\mathcal{L}_{mmd}$: 26.81%, and $\mathcal{L}_{cls}$:41.21%). The accuracy is also very low (36.84%) when our method uses $\mathcal{L}_{dis}$ and $\mathcal{L}_{mmd}$ due to the critical need of the classification loss on the target domain. The accuracy increases to 64.60% when our method combines $\mathcal{L}_{cls}$ with $\mathcal{L}_{dis}$, because the distillation loss $\mathcal{L}_{dis}$ provides a set of candidate network configurations for the student to choose and the student can quickly adapt to the target domain by combining the knowledge distillation loss and classification loss. The accuracy further increases to 70.24% when our method uses all three losses. The results show that the domain-consistent loss, as a distribution distance measure, is effective for eliminating domain divergence between the source domain (simulated network) and the target domain (physical network).



(a) In the stress-testing environment. (b) In the clean environment.

Figure 11: Data visualization provided by t-SNE [88]. Larger variations are observed in stress-testing environment, which significantly increase the difficulty on transferring knowledge learned from simulations to a physical network.

6.5 Effects of Simulators and Radio Models

Finally, we study the effects of different simulators and radio models on the performance of our method. Unit Disk Graph Medium (UDGM) [55] and Directed Graph Radio Medium (DGRM) [55] are the two most popular radio models supported by Cooja [17, 65]. UDGM in Cooja uses the disk communication model and assumes that the receiver inside the communication range of the sender can successfully receive its packets with a constant PRR (i.e., 90%). DGRM in Cooja allows its user to specify the PRR of each link and use it together with a random number to determine whether each packet can be delivered successfully. Closest-fit pattern matching (CPM) in TOSSIM allows its user to input ambient noise traces and specify the gain value (propagation strength) between each pair of devices on every channel and then generates statistical models based on the CPM algorithm to compute the packet delivery ratio for each pair of devices [43]. We create DGRM-E by extending DGRM by allowing an user to specify different PRRs on different channels for each link, and then integrate it with TOSSIM. DISTANCE in NS-3 allow its user to specify the locations of all wireless devices and use the shadowing model to determine packet receptions [62]. OMNET++ allows its user

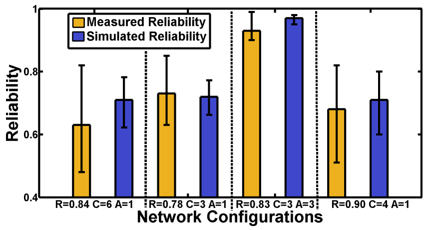


Figure 12: Reliability measured from the physical network and simulated by TOSSIM under four network configurations.

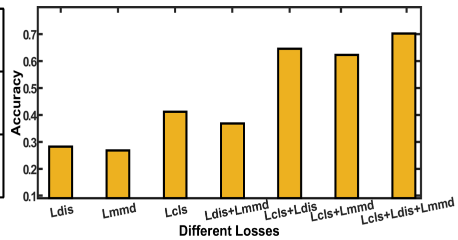


Figure 13: Accuracy when our method uses different loss functions.

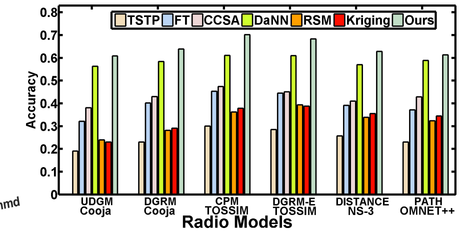


Figure 14: Accuracy comparison when using different simulators and radio models.

to specify device locations and background noise levels and uses the signal propagation model (path loss model) to compute the RSS values for packet reception prediction [64].

Figure 14 plots the accuracy of our method and our baselines when they use the simulation data generated from different simulators with various radio models. As Figure 14 shows, all methods achieve better performance when they use a more realistic model, which benefits from a smaller domain discrepancy. For instance, our method achieves 70.24% and 68.32% accuracy when it employs CPM and DEGRM-E in TOSSIM, respectively. The high accuracy results from the use of real-world noise or PRR traces in simulations. Our method provides a slightly lower accuracy (63.95%) when it uses DGRM in Cooja, which makes an unrealistic assumption that the PRRs of a link are the same on all channels. The worse performance (60.83%) appears when our method uses the simple disk model (UDGM) in Cooja. Similarly, the accuracy provided by TSTP decreases from 30.10% to 19.13% when it uses a less realistic radio model. More importantly, our method consistently provides the best performance and makes better use of more realistic simulations compared to other methods. The accuracy increases from 60.83% to 70.24% (a 9.41% increase) when our method uses CPM in TOSSIM instead of UDGM in Cooja, while the accuracy improvement offered by DaNN is 4.77% when making the same change.

The consistent low accuracy provided by TSTP shows that the simulation-to-reality gap is not tie up with a particular simulator or radio model. Although the theoretical models adopted by those simulators work satisfactorily in general, they cannot capture all real-world performance-related factors. For instance, the CPM approach in TOSSIM allows its user to input noise traces collected from a physical network and specify the gain value (propagation strength) between each pair of devices on every channel and then generates statistical models to predict packet receptions during simulations based on the CPM algorithm. Such an approach may introduce simulation inaccuracies because it has to use prerecorded noise traces and predefined gain values to simulate packet failures, and the probability-based prediction cannot precisely capture the effects of packet failures caused by extensive uncertainties, variations, and dynamics in real-world wireless deployments.

7 Conclusions

Over the past decade, WMNs have been widely used for industrial automation, military operations, smart energy, etc. Due to years of research, WMNs work satisfactorily most of the time. However, they are often difficult to configure as configuring a WMN is a complex process, involving theoretical computation, simulation, and field testing, among other tasks. Relying on field testing to identify good network configurations is impractical in many cases because running experiments on a physical network is often costly and time-consuming. Simulating the network performance under different network parameters provides distinct advantages when it comes to identifying a good network configuration, because a simulation can be set up in less time, introduce less overhead, and allow for different configurations to be tested under exactly the same condition. Unfortunately, out study shows that many network configurations identified in simulations cannot help physical networks achieve desirable performance because of the simulation-to-reality gap. To close the gap, We leverage a teacher-student deep neural network for efficient domain adaptation, which transfers network configuration knowledge learned from simulation to a physical network. Our method first uses the simulation data to learn a teacher neural network, which is then used to teach a student neural network to learn from a few shots of the physical data. Our experimental results show that our method consistently outperforms seven baselines and achieves a modeling accuracy of 70.24% with only 440 data samples collected from the physical network.

Acknowledgment

This work was supported in part by the National Science Foundation under grant CNS-2046538. The authors thank the anonymous reviewers and the shepherd Junchen Jiang at University of Chicago for their insightful comments.

References

[1] IEEE 802.15.4e WPAN Task Group. http://www.ieee802.org/15/pub/TG4e.html.

[2] E. Altman, T.Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynterc. A Survey on Networking Games in Telecommunications. *Computers and Operations Research*, 33(2):286–311, 2006.

[3] Irvan B Arief-Ang, Flora D Salim, and Margaret Hamilton. DA-HOC: Semi-Supervised Domain Adaptation for Room Occupancy Prediction using CO2 Sensor Data. In *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys)*, 2017.

[4] Taichi Asami, Ryo Masumura, Yoshikazu Yamaguchi, Hirokazu Masataki, and Yushi Aono. Domain Adaptation of DNN Acoustic Models Using Knowledge Distillation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[5] Sachin Ashok, Sai Surya Duvvuri, Nagarajan Natarajan, Venkata N. Padmanabhan, Sundararajan Sellamanickam, and Johannes Gehrke. iBox: Internet in a Box. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2020.

[6] Nouha Baccour, Anis Koubâa, Luca Mottola, Marco Antonio Zúñiga, Habib Youssef, Carlo Alberto Boano, and Mário Alves. Radio Link Quality Estimation in Wireless Sensor Networks: A Survey. *ACM Transaction on Sensor Network*, 8(4), 2012.

[7] Mihovil Bartulovic, Junchen Jiang, Sivaraman Balakrishnan, Vyas Sekar, and Bruno Sinopoli. Biases in Data-Driven Networking, and What to Do About Them. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2017.

[8] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej. Using RSSI value for distance estimation in wireless sensor networks based on ZigBee. In *International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2008.

[9] Marcos Almeida Bezerra, Ricardo Erthal Santelli, Eliane Padua Oliveira, Leonardo Silveira Villar, and Luciane Amélia Escaleira. Response Surface Methodology (RSM) as a Tool for Optimization in Analytical Chemistry. *Talanta*, 76(5):965 – 977, 2008.

[10] Carlo Alberto Boano, Thiemo Voigt, Claro Noda, Kay Romer, and Marco Zuniga. JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. In *ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, 2011.

[11] Gabriele Boccolini, Gustavo Hernández-Peñaloza, and Baltasar Beferull-Lozano. Wireless Sensor Network for Spectrum Cartography based on Kriging Interpolation. In *International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2012.

[12] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor Sampedro, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[13] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain Separation Networks. In *Advances in Neural Information Processing Systems (NIPS)*. Curran Associates Inc., 2016.

[14] Xianghui Cao, Lu Liu, Yu Cheng, and Xuemin Shen. Towards Energy-Efficient Wireless Networking in the Big Data Era: A Survey. *IEEE Communications Surveys & Tutorials*, 20(1):303–332, 2018.

[15] C. Caruso and F. Quarta. Interpolation Methods Comparison. *Computers and Mathematics with Applications*, 35(12):109 – 126, 1998.

[16] Xuewen Chen and Jong cheol Jeong. Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA)*, 2007.

[17] Source Code of Cooja. https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja.

[18] H. Dakdouk, E. Tarazona, R. Alami, R. Féraud, G. Z. Papadopoulos, and P. Maillée. Reinforcement Learning Techniques for Optimized Channel Hopping in IEEE 802.15.4-TSCH Networks. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2018.

[19] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. *CoRR*, abs/1605.09782, 2017.

[20] Wei Dong, Chun Chen, Xue Liu, Yuan He, Yunhao Liu, Jiajun Bu, and Xianghua Xu. Dynamic Packet Length Control in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications*, 13(3):1172–1181, 2014.

[21] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. Training Generative Neural Networks via Maximum Mean Discrepancy Optimization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

[22] Emerson Process Management. https://www.emerson.com/en-us/automation-solutions.

[23] System Engineering Guidelines IEC 62591 WirelessHART by Emerson Process Management. http://www2.emersonprocess.com/siteadmincenter/PM%20Central%20Web%20Documents/EMR%5fWirelessHART%5fSysEngGuide.pdf.

[24] WirelessHART Networks Deployed by Emerson Process Management. https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology.

[25] Songwei Fu, Yan Zhang, Yuming Jiang, Chengchen Hu, Chia-Yen Shih, and Pedro Jose Marron. Experimental Study for Multi-layer Parameter Configuration of WSN Links. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2015.

[26] Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. In *International Conference on International Conference on Machine Learning (ICML)*, 2015.

[27] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[28] Muhammad Ghifary, W. Bastiaan Kleijn, and Mengjie Zhang. Domain Adaptive Neural Networks for Object Recognition. In *Pacific Rim Conference on Artificial Intelligence Trends in Artificial Intelligence (PRICAI)*, 2014.

[29] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. *Covariate Shift and Local Learning by Distribution Matching*, pages 131–160. MIT Press, 2009.

[30] Dolvara Gunatilaka, Mo Sha, and Chenyang Lu. Impacts of Channel Selection on Industrial Wireless Sensor-Actuator Networks. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[31] HART Foundation (Now FieldComm Group). http://www.hartcomm.org/.

[32] Hongli He, Hangguan Shan, Aiping Huang, Qiang Ye, and Weihua Zhuang. Reinforcement Learning-Based Computing and Transmission Scheduling for LTE-U-Enabled IoT. In *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. In *Deep Learning and Representation Learning Workshop (NIPS)*, 2015.

[35] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[36] Liang Huang, Suzhi Bi, and Ying Jun Zhang. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. *IEEE Transactions on Mobile Computing*, Early Access, 2020.

[37] International Electrotechnical Commission (IEC). https://www.iec.ch/.

[38] International Society of Automation (ISA). https://www.isa.org/.

[39] Alan Jovic, Karla Brkić, and Nikola Bogunović. A Review of Feature Selection Methods with Applications. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015.

[40] Charles W. Kazer, Jo ao Sedoc, Kelvin K.W. Ng, Vincent Liu, and Lyle H. Ungar. Fast Network Simulation Through Approximation or: How Blind Men Can Describe Elephants. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2018.

[41] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[42] D.Praveen Kumar, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. Machine Learning Algorithms for Wireless Sensor Networks: A Survey. *Information Fusion*, 49:1–25, 2019.

[43] HyungJune Lee, Alberto Cerpa, and Philip Levis. Improving Wireless Simulation through Noise Modeling. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[44] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *International Conference on Embedded Networked Sensor Systems (Sensys)*, 2003.

[45] Feng Li, Kwok-Yan Lam, Zhengguo Sheng, Xinggan Zhang, Kanglian Zhao, and Li Wang. Q-Learning-Based Dynamic Spectrum Access in Cognitive Industrial Internet of Things. *Mobile Networks and Applications*, 23:10, 2018.

[46] Xin Li, Qiuyuan Huang, and Dapeng Wu. Distributed large-scale co-simulation for iot-aided smart grid control. *IEEE Access*, PP:1–1, 2017.

[47] Chi Harold Liu, Qiuxia Lin, and Shilin Wen. Blockchain-Enabled Data Collection and Sharing for Industrial IoT With Deep Reinforcement Learning. *IEEE Transactions on Industrial Informatics*, 15(6):3516–3526, 2019.

[48] Jane Liu. *Real-Time Systems*. Prentice Hall PTR, USA, 1st edition, 2000.

[49] Yongkang Liu, Richard Candell, Kang Lee, and Nader Moayeri. A Simulation Framework for Industrial Wireless Networks and Process Control Systems. In *IEEE World Conference on Factory Communication Systems (WFCS)*, 2016.

[50] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. Understanding Variable Importances in Forests of Randomized Trees. In *International Conference on Neural Information Processing Systems (NeurIPS)*, 2013.

[51] Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems. In *Proceedings of the IEEE, Special Issue on Industrial Cyber Physical Systems (ICPSs)*, 2016.

[52] Zhi Quan Luo and Wei Yu. An Introduction to Convex Optimization for Communications and Signal Processing. *IEEE Journal on Selected Areas in Communications*, 24(8):1426–1438, 2006.

[53] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys and Tutorials*, 21(4):3133–3174, 2019.

[54] Qian Mao, Fei Hu, and Qi Hao. Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials*, 20(4):2595–2621, 2018.

[55] Tayyab Mehmood. Cooja Network Simulator: Exploring the Infinite Possible Ways to Compute the Performance Metrics of IoT Based Smart Devices to Understand the Working of IoT Based Compression & Routing Protocols, 2017.

[56] Fan Meng, Peng Chen, Lenan Wu, and Julian Cheng. Power Allocation in Multi-User Cellular Networks: Deep Reinforcement Learning Approaches. *IEEE Transactions on Wireless Communications*, Early Access, 2020.

[57] Meshdynamics. https://www.meshdynamics.com/index.html.

[58] Saeid Motiian, Marco Piccirilli, Donald A. Adjeroh, and Gianfranco Doretto. Unified Deep Supervised Domain Adaptation and Generalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[59] Oshri Naparstek and Kobi Cohen. Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access. *IEEE Transactions on Wireless Communications*, 18(11):310–323, 2019.

[60] Yasar Sinan Nasir and Dongning Guo. Multi-Agent Deep Reinforcement Learning for Dynamic Power Allocation in Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 37(10):2239–2250, 2019.

[61] NS-3 Network Simulator. https://www.nsnam.org/.

[62] NS-3 Shadowing Model. https://www.nsnam.org/docs/release/3.10/doxygen/classns3_1_1_shadowing_loss_model.html.

[63] Source Code of OMNeT++. https://github.com/omnetpp/omnetpp.

[64] OMNeT++ INET Framework and Transmission Medium. https://inet.omnetpp.org/docs/users-guide/ch-transmission-medium.html.

[65] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-Level Sensor Network Simulation with Cooja. In *IEEE Conference on Local Computer Networks (LCN)*, 2006.

[66] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, Early Access, 2020.

[67] Stephen S. Oyewobi, Gerhard P. Hancke, Adnan M. Abu-Mahfouz, and Adeiza J. Onumanyi. An Effective Spectrum Handoff Based on Reinforcement Learning for Target Channel Selection in the Industrial Internet of Things. *Sensors*, 19(6):1–21, 2019.

[68] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain Adaptation via Transfer Component Analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[69] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual Domain Adaptation: A Survey of Recent Advances. *IEEE Signal Processing Magazine*, 32(3):53–69, 2015.

[70] Yang Peng, Zi Li, Daji Qiao, and Wensheng Zhang. I2C: A Holistic Approach to Prolong the Sensor Network Lifetime. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 2670–2678, 2013.

[71] Marius-Constantin Popescu, Valentina E. Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer Perceptron and Neural Networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[72] Xiaoyu Qiu, Luobin Liu, Wuhui Chen, Zicong Hong, and Zibin Zheng. Q-Learning-Based Dynamic Spectrum Access in Cognitive Industrial Internet of Things. *IEEE Transactions on Vehicular Technology*, 68(8):8050–8062, 2019.

[73] Mauricio G.C. Resende and Panos Pardalos. *Handbook of Optimization in Telecommunications*. Springer, 2006.

[74] Noga H. Rotman, Michael Schapira, and Aviv Tamar. Online Safety Assurance for Learning-Augmented Systems. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2020.

[75] Junyang Shi and Mo Sha. Parameter Self-Configuration and Self-Adaptation in Industrial Wireless Sensor-Actuator Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[76] Junyang Shi and Mo Sha. Parameter Self-Adaptation for Industrial Wireless Sensor-Actuator Networks. *ACM Transactions on Internet Technology*, 20(3), 2020.

[77] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[78] Timothy Simpson, Timothy Mauery, John Korte, and Farrokh Mistree. Kriging Models for Global Approximation in Simulation-Based Multidisciplinary Design Optimization. In *AIAA Journal*, 2001.

[79] Mariusz Slabicki, Gopika Premsankar, and Mario Di Francesco. Adaptive Configuration of LoRa Networks for Dense IoT Deployments. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2018.

[80] Zigbee Smart Energy. https://zigbeealliance.org/zigbee_products/smart-energy-monitor-2/.

[81] TelosB Datasheet Provided by Memsic Incorporation. http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf.

[82] Shun Tobiyama, Bo Hu, Kazunori Kamiya, and Kenji Takahashi. Large-Scale Network-Traffic-Identification Method with Domain Adaptation. In *Companion Proceedings of the Web Conference (WWW)*, 2020.

[83] Edna Chebet Too, Li Yujian, Sam Njuki, and Liu Yingchun. A Comparative Study of Fine-tuning Deep Learning Models for Plant Disease Identification. *Computers and Electronics in Agriculture*, 161:272 – 279, 2019.

[84] Source Code of TOSSIM. https://github.com/tinyos/tinyos-main/tree/master/tos/lib/tossim.

[85] IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH). https://tools.ietf.org/html/rfc7554.

[86] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep Domain Confusion: Maximizing for Domain Invariance. *ArXiv*, abs/1412.3474, 2014.

[87] K. K. Vadde, V. R. Syrotiuk, and D. C. Montgomery. Optimizing Protocol Interaction Using Response Surface Methodology. *IEEE Transactions on Mobile Computing*, 5(6):627–639, 2006.

[88] Laurens Van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE . *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[89] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (ICST)*, 2008.

[90] Jiliang Wang, Zhichao Cao, Xufei Mao, and Yunhao Liu. Sleep in the Dins: Insomnia Therapy for Duty-cycled Sensor Networks. In *IEEE Conference on Computer Communications (INFOCOM)*, 2014.

[91] Jingjing Wang, Chunxiao Jiang, Kai Zhang, Xiangwang Hou, Yong Ren, and Yi Qian. Distributed Q-Learning Aided Heterogeneous Network Association for Energy-Efficient IIoT. *IEEE Transactions on Industrial Informatics*, 16(4):2756–2764, 2020.

[92] Mei Wang and Weihong Deng. Deep Visual Domain Adaptation: A Survey. *Neurocomputing*, 312(27):135–153, 2018.

[93] Shangxing Wang, Hanpeng Liu, Pedro Henrique Gomes, and Bhaskar Krishnamachari. Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks. *IEEE Transactions on Cognitive Communications and Networking*, 4(2):257–265, 2018.

[94] WCPS Simulator. http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator.

[95] WirelessHART for Industrial Automation. https://fieldcommgroup.org/technologies/hart.

[96] Hansong Xu, Xing Liu, Wei Yu, David Griffith, and Nada Golmie. Reinforcement Learning-Based Control and Networking Co-Design for Industrial Internet of Things. *IEEE Journal on Selected Areas in Communications*, 38(5):885–898, 2020.

[97] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[98] Helin Yang, Arokiaswami Alphones, Wen-De Zhong, Chen Chen, and Xianzhong Xie. Learning-Based Energy-Efficient Resource Management by Heterogeneous RF/VLC for Ultra-Reliable Low-Latency Industrial IoT Networks. *IEEE Transactions on Industrial Informatics*, 16(8):5565–5576, 2020.

[99] Kok-Lim Alvin Yau, Peter Komisarczuk, and Paul D. Teal. Reinforcement Learning for Context Awareness and Intelligence in Wireless Networks: Review, New Features and Open Issues. *IEEE Journal on Selected Areas in Communications*, 37(10):2239–2250, 2019.

[100] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1277–1290, 2019.

[101] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287, 2019.

[102] Tianyu Zhang and Omid Ardakanian. A Domain Adaptation Technique for Fine-Grained Occupancy Estimation in Commercial Buildings. In *ACM/IEEE International Conference on Internet of Things Design and Implementation (IoTDI)*, 2019.

[103] Nan Zhao, Ying-Chang Liang, Dusit Niyato, Yiyang Pei, Minghu Wu, and Yunhao Jiang. Deep Reinforcement Learning for User Association and Resource Allocation in Heterogeneous Cellular Networks. *IEEE Transactions on Wireless Communications*, 18(11):5141–5152, 2019.

[104] Zigbee Alliance. https://zigbeealliance.org/.

[105] Marco Zimmerling, Federico Ferrari, Luca Mottola, Thiemo Voigt, and Lothar Thiele. PTunes: Runtime Parameter Adaptation for Low-Power MAC Protocols. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2012.